Proceedings of the Fourteenth Artificial Intelligence and
Interactive Digital Entertainment Conference (AIIDE 2018)

Toward a Unified Understanding of Experience Management

David Thue
School of Computer Science
Reykjavik University
Reykjavik 101, Iceland
davidthue@ru.is

Vadim Bulitko
Department of Computing Science
University of Alberta
Edmonton, Alberta, T6G 2E8, Canada
bulitko@ualberta.ca

Abstract

We present a new way to represent and understand experience managers – AI agents that tune the parameters of a running game to pursue a designer's goal. Existing representations of AI managers are diverse, which complicates the task of drawing useful comparisons between them. Contrary to previous representations, ours uses a point of unity as its basis: that every game/manager pair can be viewed as only a game with the manager embedded inside. From this basis, we show that several common, differently-represented concepts of experience management can be re-expressed in a unified way. We demonstrate our new representation concretely by comparing two different representations, Search-Based Drama Management and Generalized Experience Management, and we present the insights that we have gained from this effort.

1 Introduction

Using an AI agent to tune parameters of a running game has become a common practice. The specific goals of doing so have varied widely, from balancing the competitiveness of differently skilled players in *Super Mario Kart* (Ohyagi and Satou 2005) to raising or lowering the tension that players feel in *Left 4 Dead* and *Rimworld* (Valve Corporation 2008; Booth 2009; Ludeon Studios 2013). In academia, this task of influencing a game's systems using AI has had many names: *dynamic difficulty adjustment* (Hunicke and Chapman 2004), *drama management* (Mateas and Stern 2005), *adaptive game AI* (Spronck et al. 2006), *adaptive game mechanics* (Lindley and Sennersten 2008), *experience management* (Riedl et al. 2008), *player-adaptive games* (Ha et al. 2011), and *procedural game adaptation* (Thue and Bulitko 2012). Following Riedl et al. (2008), we use the term *experience management* throughout this text, as it is the most general of the proposed descriptions; it is unbound from any specific goal (e.g., balancing difficulty or producing dramatic situations) and it is not specific to games as its target domain. We will refer to an AI agent that performs the task of experience management as an *AI manager*.

To ensure that the design and application of AI managers can benefit from prior work, it is necessary to support two kinds of comparisons. For the designers of new AI managers, it must be straightforward to compare alternative technologies for AI managers (e.g., two different ways to build a player model). Such comparisons can help determine which technologies should be used when designing a new manager. For practitioners wanting to apply an AI manager, it must be straightforward to compare existing managers in a particular domain with a particular goal. This sort of comparison could, for example, help decide which manager will best balance competitiveness in *Super Mario Kart*. While making both kinds of comparisons is important, little work has supported doing so in practice, and there are few published examples of such comparisons being made. With this work, we aim to simplify the comparison of AI managers and their related technologies by unifying their representations.

To compare different technologies for AI managers, one must be able to represent managers in a common way. A manager's *representation* is a set of terms and concepts that are used to explain its *design* (how it is intended to operate). This distinction between representation and design is essential when comparing manager technologies, because a manager's representation is what an analyst uses to reason about each technology's suitability, given the design. For example, if one represents an AI manager as an agent that chooses between trees of potential non-player character actions, one might view reactive planning as an attractive technology to use (Mateas and Stern 2005; Riedl et al. 2008). Alternatively, if one represents an AI manager as an agent that chooses between potential adjustments to the player's world state, one might prefer to use heuristic search or reinforcement learning instead (Weyhrauch 1997; Nelson et al. 2006b). Having a *shared* representation allows an analyst to find similarities and differences between related technologies that are used by different managers (Nelson and Mateas 2008). Although several sets of managers exist that share *similar* representations (Nelson et al. 2006a; Roberts et al. 2006), (Young et al. 2004; Riedl and Stern 2006; Riedl 2009), (Thue et al. 2007; 2011; Ramirez and Bulitko 2015), comparisons that use these representations as their basis remain rare (Nelson and Mateas 2008). We suspect that this rarity arises because the representations themselves often evolve in parallel with the design of each new manager, and re-representing the old manager requires "extra", time-consuming work (Thue 2015).

To compare different managers using a given domain and goal, one must be able to make them share a common *inter-*

face, which describes what information passes (and when) between a game, a manager, and a player; it can be considered part of a manager's representation. Having a shared interface allows two managers to operate in (and thus be compared in) the same domain. Examples of this sort of comparison are also rare (Ramirez and Bulitko 2015), since adapting a manager to a new domain can be quite challenging.

We hypothesize that the observed lack of comparisons among AI manager technologies and AI managers themselves is (at least partly) due to a pervasive, *representational gap*; as long as the representation of each manager remains coupled to that manager's design (either by genesis or by evolution), it will remain difficult to compare one manager to another. In this paper, we propose and demonstrate a concrete step toward addressing this situation: a representation for AI managers that unifies existing representations and supports direct comparisons between them.

The remainder of this paper is organized as follows. We begin by explaining some relevant concepts and terminology, and follow with a review of related work. We then present our approach, in which a game, an AI manager, and a single player are represented in a Factored-state Markov Decision Process (Boutilier, Dearden, and Goldszmidt 1995). We demonstrate our approach by re-expressing and comparing two different representations for AI managers: Bates', Weyhrauch's, and later Nelson et al.'s Search-Based Drama Management (1992; 1997; 2006b; 2008) and Thue et al.'s Procedural Game Adaptation / Generalized Experience Management (2012; 2015). We conclude by discussing the benefits and limitations of our approach and offering ideas for future work. We focus on single player games throughout this paper, as an extension to multiplayer games remains as future work.

2 Concepts and Terminology

We begin by making our notion of a manager's representation more concrete. Formally, we view a *manager representation* (MR) as a set of definitions that explains how a game, its AI manager, and a player interact with one another during the course of any player experience in the given game. Given a manager representation, we assert that it should be possible to precisely determine: (i) how each element of the game/manager/player system depends on any other element, and (ii) both how and when each dependence is realized during gameplay. Thus it should be possible to use a given MR to precisely determine (i) how and when a manager or the player exert their influence over the game, (ii) how and when the manager or player learn about the current state of the game, and so on. Deciding which elements should be represented in an MR is part of designing that MR.

To explain our contributions in Section 4, we will present the game/manager/player system from two particular perspectives. The *disjoint perspective* views the game and the manager as separate entities, where the latter modifies the former as a player's experience in the game proceeds. This view of experience management is widely accepted, but there is little agreement about which elements should be used by each MR, nor how their dependencies should be defined. We will support this point further in Section 3. The *joint perspective* views the game and the manager as a single entity, which the player interacts with as they would under the disjoint perspective. Although this perspective is uncommon in the literature on experience management, it represents two realistic viewpoints: (i) that of a player who is unaware of the manager, and (ii) that of the computing infrastructure on which the game and manager are executed. It also represents a point of unity across all AI managers: fundamentally, every such manager/game pair can be re-represented as "just" a game (albeit a more complex game than before). As we will show in Section 4, this unified, joint perspective can be used as a starting point for bridging the representational gap between existing AI managers.

To support our explanations in Section 4, we will represent an arbitrary computer game using a deterministic, stationary, Factored-state Markov Decision Process (FMDP) similar to that used by Chakraborty and Stone (2011). Although Markov Decision Processes (MDPs) are traditionally used to represent the environment of an AI agent, we do not seek to solve any MDP. Instead, we use the representational capacity of MDPs in a more general way. Specifically, we use an FMDP to represent the environment of an *arbitrary* agent; the agent could be an AI system, a human player, or any agent that can observe states and perform actions. When we view a game as an FMDP, we reserve the right to consider both the manager and the player as potential agents therein.

A (non-factored) deterministic MDP is given by a tuple $\langle S, A, \mathcal{P} \rangle$, where S is a set of *states*, A is a set of *actions* that an agent can perform, and $\mathcal{P} : S \times A \to S$ is a *transition function*, where $\mathcal{P}(s, a) \to s'$ means that s' is the state that will deterministically occur when an agent performs action a in state s. The *current state* of an MDP (at *time step* $t \geq 0$) is given by s_t. Using language more common to games, s_t is an instantaneous snapshot of the game's state, and the transition function represents the game's mechanics.

In an FMDP, the state representation is factored across n *factors* $\boldsymbol{v} = \langle v_1, \ldots, v_n \rangle$, where each *state factor* v in $\boldsymbol{v}$ is a variable that can hold a value in some domain $\mathcal{D}(v)$. We use $s(v)$ to denote the value of variable v in state s. For example, Figure 1 shows some of the state factors for *Super Mario Kart*, where each node represents a different variable in the game state (e.g., racer 1's position, R1.pos).

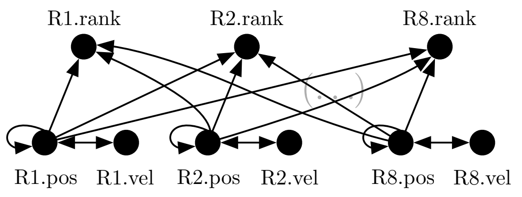

Figure 1: A partial view of the state factors (circles) and transition factors (arrows) for three of the eight racers in a *Super Mario Kart* race (R# means "racer #").

Further following Chakraborty and Stone (2011), we additionally factor the representation of our FMDP's transition function across n factors $\boldsymbol{\mathcal{P}} = \langle \mathcal{P}_{v_1}, \ldots, \mathcal{P}_{v_n} \rangle$, where each *transition factor* $\mathcal{P}_v$ in $\boldsymbol{\mathcal{P}}$ is a function that computes the value of variable v, given a state and an action: $\mathcal{P}_v$:

$S \times A \rightarrow \mathcal{D}(v)$. For example, $\mathcal{P}_v(s_t, a_t)$ would compute $s_{t+1}(v)$ (which is the value of v in s_{t+1}). Concretely, the two arrows pointing to variable R1.pos in Figure 1 indicate (by the source of each arrow) which values in any current state s_t will be used by $\mathcal{P}_{\text{R1.pos}}$ to compute $s_{t+1}(\text{R1.pos})$. These values are $s_t(\text{R1.pos})$ and $s_t(\text{R1.vel})$. For each agent action in the FMDP, we say that the next state is computed by *executing* all of the transition factors in $\boldsymbol{\mathcal{P}}$. Finally, we refer to the time during which $\boldsymbol{\mathcal{P}}$'s transition factors are executed as the FMDP's *execution phase*; this occurs once per time step and immediately follows the player's action for that step.

We view a game as Factored-state MDP (FMDP) with state factors in $\boldsymbol{v}$ and transition factors in $\boldsymbol{\mathcal{P}}$. Under the joint perspective, we can also view a game/manager pair in the same way: everything about both a game and its manager can be represented as a single FMDP. Note that the player is the only agent in this FMDP – the manager is *not* an agent, as its operations have been completely "built-in" to the FMDP. By representing a game/manager pair as an FMDP and distinguishing between different types of state factors and transition factors, we will show how two distinct (disjoint-perspective) representations can be unified under a common framework.

2.1 Problem Formulation

We aim to develop and demonstrate a manager representation that will unify existing manager representations, toward ultimately facilitating new comparisons: both between AI managers and between their related technologies. We will judge our attempt to be successful if the new representation allows us to (i) re-express common concepts of experience management that we find in any given manager representation, and (ii) compare given manager representations in a way that yields new insights.

3 Related Work

The study of experience management dates back at least to Bates (1992), with early contributions by Laurel (1986) and Weyhrauch (1997), all of which focused on managing interactive narrative experiences. Nelson et al. (2006b) and Thue and Bulitko (2012) used Markov Decision Processes to represent different aspects of how AI managers work. Thue and Bulitko (and later Thue (2015)) additionally sought to generalize their work across two kinds of experience management: drama management and dynamic difficulty adjustment. Although they successfully represented one of each kind of manager, they did not compare them, and they did not compare the representations of the two managers. Nelson and Mateas (2008) performed a direct comparison of two managers that used different representations: Search-based Drama Management (Nelson et al. 2006b) versus Targeted-Trajectory Distribution MDPs (Roberts et al. 2006). However, their comparison was focused on the relative performance of different variants of the two managers; they did not compare the (relatively similar) representations of those managers. Ramirez and Bulitko (2015) also compared two managers that used different representations: PaST (which they made) and the Automated Story Director (Riedl and Stern 2006). Similarly to Roberts et al. (2006), they compared the two managers in terms of their performance, rather than their representations.

Magerko (2005) presented a set of five requirements for interactive story representations. These included enabling sufficient author expression, ensuring coherence between story events, supporting varied sequences of events, enabling player action prediction, and allowing authors to dictate when certain events will happen. We view the problem of representing an interactive story as a subproblem of representing an experience manager.

Roberts and Isbell (2008) presented a set of ten metrics for assessing the behaviours and affordances of experience managers, and used these metrics to categorize a wide range of existing managers. While these metrics provide some basis for comparing different managers, they treat each manager as an abstract "black box"; the managers' representations cannot be compared. Roberts and Isbell (2008) also claimed that all experience managers are based on four components: a set of plot points, a set of manager actions, a model of authorial intent, and a model of player responses to manager actions. They did not provide precise representations for these components, but we will discuss how they can be represented precisely in Section 5. To the best of our knowledge, no work has sought a way to directly compare any two representations for experience management.

4 Proposed Approach

At a high level, our approach begins by viewing a game/manager pair as "just" a game (using the joint perspective); doing so ensures our ability to represent existing AI managers, as we argued in Section 2. Then, using the language of Factored-state MDPs, we propose a mapping from the joint perspective to common concepts of the disjoint perspective (where the game and manager are viewed as separate entities). From this mapping, we gain the ability to understand and re-express existing representations that use the disjoint perspective, with the extra guarantee that any new manager can also be represented using the joint perspective. We demonstrate our approach in Section 5, where we re-express and compare two different representations.

4.1 Common Concepts of the Disjoint Perspective

We begin with some high-level assumptions and concepts regarding an arbitrary AI manager, which are common across many disjoint-perspective representations. (i) A manager has a *policy* that uses some part of the *game's state* and the player's actions to compute a manager action. (ii) A *manager action*, when applied, changes one or more of a game's *tuneable parameters*. (iii) At least once during a player's experience in the game, the manager gets an *opportunity to act*, during which its policy is used to compute a new manager action, and after which the new action is applied. (iv) The game might have *mechanics* that compute parts of the game's state independently from the manager.

For example, the manager in *Super Mario Kart* gets a new opportunity to act whenever a player activates a question block by driving their kart over it. It then uses its policy to consider the relative ranks of that player and the other players, and computes a manager action that will tune the probability distribution over potential power-ups accordingly. This manager action is then applied to the game, and the tuned probability distribution is used to select a power-up for the player who activated the block. This behaviour is often referred to as "rubber-banding", as it gives more powerful power-ups to players who have worse ranks in the race.

4.2 Joint Perspective to Disjoint Perspective

When we make a manager part of an FMDP, it must be the case that each of the common concepts of the disjoint perspective (a manager policy, the game's state space, tuneable parameters, game mechanics, manager actions, and the manager's opportunities to act) will somehow manifest within the elements or operation of the FMDP. We identify each such manifestation in this section.

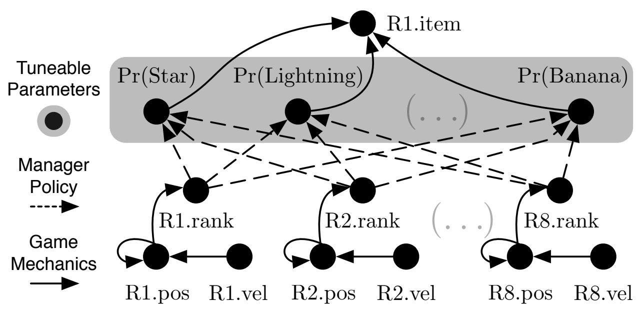

Figure 2: An expanded view of Figure 1 (under the joint perspective), showing how *Super Mario Kart*'s tuneable parameters and manager policy manifest as part of the joint-perspective FMDP. Pr(x) means "probability of getting power-up x". We omit many mechanic arrows for clarity.

Manager Policy. How does a manager's policy manifest, under the joint perspective? The policy performs computation, and the only elements that can perform computation in a joint-perspective FMDP are the transition factors. Therefore, the manager's policy must manifest as some subset of the FMDP's transition factors, $\boldsymbol{\mathcal{P}}^{\mathrm{M}} \subseteq \boldsymbol{\mathcal{P}}$.[1] As shown in Figure 2, this corresponds to viewing "rubber-banding" as part of the mechanics of *Super Mario Kart*.

Game State Space and Tuneable Parameters. Each transition factor depends on the FMDP's state and a player action ($\mathcal{P}_v : S \times A \to \mathcal{D}(v)$), and the manager's policy depends on the *game's* state and a player action. From this we conclude that part of the FMDP's state space S (i.e., some state factors in $\boldsymbol{v}$) must represent the game's state space, S^{G}. Furthermore, each transition factor computes the value of a state factor (i.e., a value in $\mathcal{D}(v)$ for factor v), and the manager's policy computes the value of some tuneable parameters. From this we conclude that some state factors in $\boldsymbol{v}$ (i.e., part of S) must also represent these tuneable parameters, S^{T}. Without loss of generality, we assume that S **is** the game's state space ($S = S^{\mathrm{G}}$, and $S^{\mathrm{T}} \subseteq S$).

[1]Although $\boldsymbol{\mathcal{P}}$ is a vector, we abuse set notation to treat it as a set that contains all of its transition factors. The same applies to $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$.

As shown in Figure 2, S^{T} in *Super Mario Kart* would contain variables whose values determine the game's probability distribution over power-ups, while the other variables in S^{G} (i.e., in $S^{\mathrm{G}} \setminus S^{\mathrm{T}}$) represent the rest of the game (including, e.g., a position, velocity, and rank for every racer's kart). In summary, the manager's policy ($\boldsymbol{\mathcal{P}}^{\mathrm{M}}$) uses the game's state S^{G} (and a player action) to compute new values for the game's tuneable parameters (state factors in S^{T}).

Game Mechanics. To complete our definition of the FMDP's transition factors, $\boldsymbol{\mathcal{P}}$, we assume that every transition factor that is *not* part of the manager's policy ($\boldsymbol{\mathcal{P}}^{\mathrm{M}}$) is instead used to represent the mechanics of the game: $\boldsymbol{\mathcal{P}}^{\mathrm{G}} \subseteq \boldsymbol{\mathcal{P}}$. This implies that the game's mechanics are separate from the manager's policy ($\boldsymbol{\mathcal{P}}^{\mathrm{G}} \cap \boldsymbol{\mathcal{P}}^{\mathrm{M}} = \emptyset$). In *Super Mario Kart*, $\boldsymbol{\mathcal{P}}^{\mathrm{G}}$ would represent all of the game's computations (including computing new kart positions) *except for* the "rubber banding" computations. As shown in Figure 2, withholding these computations from *Super Mario Kart*'s mechanics (i.e., considering only the solid-line arrows) still leaves the game fully playable: the (initial) values of the power-up probability distribution can still be used to determine which power-ups should appear as the value of R1.item.

Manager Actions. We now consider how a manager's actions and its opportunities to act manifest in the joint-perspective FMDP. The actions of the FMDP are already defined to be the actions that the *player* performs (e.g., steering $R1$ in *Super Mario Kart*), and this is common across both joint and disjoint perspectives (Section 2). What, then, does it mean for a manager to act? Given our assumptions above, a manager action is the result of computing its policy, and we have equated this policy with $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$, part of the FMDP's transition factors. A manager *acting* thus manifests as a part of $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$ *executing* while the FMDP operates. A manager action manifests as an assignment of new values to the variables in S^{T}. The execution of $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$ also applies each manager action, changing each variable in S^{T} to its newly computed value.

Opportunities to Act. If a manager action occurs in the FMDP when (i) $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$ is executed and (ii) the results are used to update S^{T}, then each of the manager's opportunities to act must occur during the execution phase of the FMDP. During a player's experience in the FMDP, one execution phase will occur for each time step that occurs.

From the disjoint perspective, an important consideration regarding the manager's opportunities to act concerns when they occur *in relation to* the execution of the game's transition function. Since they are viewed as distinct processes from that perspective, one is typically modelled as happening after the other. From the joint perspective, a manager's opportunities to act and the game's mechanics both occur during the execution phase of the FMDP. To avoid altering the definition of an FMDP, we choose to represent the relative ordering of action opportunities and game mechanic execution by considering two sequential time steps instead of one. We will explain this representation in terms of one ordering (manager action, then game mechanics), but both orderings are supported. On the first time step, $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$ computes new values for the tuneable parameters (S^{T}). On the next time step, $\boldsymbol{\mathcal{P}}^{\mathrm{G}}$ computes new values for the game's state factors, using the values in S^{T} from the previous time step.

The choice of this order is important. To understand why, consider each order in the context of when player actions occur and when new game states are computed. Since each game state is the immediate result of computing the game mechanics, there are only two possible orders (where M = manager, G = game mechanics, P = player, and S = state): . . . M G S P M G S P . . . and . . . G S M P G S M P . . . .

With the first order, whenever the player acts, the manager has an opportunity to intervene before the next state is computed. This capacity is essential for implementing the most powerful form of narrative mediation (intervention) (Riedl, Saretto, and Young 2003), which prevents an undesirable game state from ever occurring. With the second order, this capability is lost; the manager must choose an action before the player chooses their action, and both of these steps happen before the next state is computed. This leaves no opportunity for the manager to prevent undesirable game states.

Summary. We have shown that our joint perspective FMDP can be used to re-express six common concepts of disjoint-perspective representations. The *game's state space* manifests as S^{G}, the state space of the FMDP ($S = S^{\mathrm{G}}$). The *tuneable parameters* of the game manifest as S^{T}, state factors in the FMDP's state space ($S^{\mathrm{T}} \subseteq S$). A *manager policy* manifests as $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$ – the part of the FMDP's transition function ($\boldsymbol{\mathcal{P}}$) that computes new values for the state factors in S^{T}. The *game's mechanics* manifest as $\boldsymbol{\mathcal{P}}^{\mathrm{G}}$ – the part of the FMDP's transition function that computes new values for the state factors *outside* of S^{T} (i.e., in $S^{\mathrm{G}} \setminus S^{\mathrm{T}}$). A *manager action* manifests as an assignment of values to the state factors in S^{T}. Finally, a manager's *opportunities to act* manifest in each execution of $\boldsymbol{\mathcal{P}}^{\mathrm{M}}$, which occurs once per time step during the execution phase of the FMDP.

5 Demonstration

We now demonstrate our approach by using it to re-express two manager representations: Search-Based Drama Management (SBDM) (Weyhrauch 1997; Nelson et al. 2006b; Nelson and Mateas 2008) and Generalized Experience Management (GEM) (Thue and Bulitko 2012; Thue 2015). We have chosen these representations because they both use the language of MDPs to offer precise definitions of at least some of their concepts. The structure of this section follows the six common concepts of the disjoint perspective. For each concept, we will explain how it arises in both SBDM and GEM, using our joint-perspective approach from Section 4 to establish a common view of both.

Game State Space. SBDM defines a state as a temporally-ordered record of which sequence of *plot points* (important game events that the manager should reason about) have occurred and which manager actions have been performed in any current player experience. Using the joint perspective, we can re-express this record as a set of state factors (S^{+}_{SB}). The literature on SBDM does not give a precise representation for a plot point, but their examples from the interactive fiction *Anchorhead* (Gentry 1998) suggest that each plot point is a set of possible sequences of game states and player actions in the managed game. Using the joint perspective, we can re-express the possible states of SBDM's target game as another set of state factors ($S^{\mathrm{G}}_{\mathrm{SB}}$), where $S_{\mathrm{SB}} = S^{\mathrm{G}}_{\mathrm{SB}} \cup S^{+}_{\mathrm{SB}}$.

GEM uses an MDP to represent the game that is to be managed, where the player is an agent that performs actions inside that MDP. Similarly to SBDM, GEM also represents some information in addition to the game's state (e.g., a vector of features of the game's state and the player's actions, such as a player model). Using the joint perspective, we can express GEM's state space directly as one set of state factors ($S^{\mathrm{G}}_{\mathrm{GEM}}$), and its additional information as an additional set of state factors (S^{+}_{GEM}), such that $S_{\mathrm{GEM}} = S^{\mathrm{G}}_{\mathrm{GEM}} \cup S^{+}_{\mathrm{GEM}}$.

Game Mechanics. SBDM does not directly represent the mechanics of the game (e.g., in *Anchorhead*), but it does allow gameplay to proceed even when the manager performs a null action (doing nothing). This implies that the game must have mechanics that operate independently from the manager's policy. We can re-express these mechanics as $\boldsymbol{\mathcal{P}}^{\mathrm{G}}_{\mathrm{SB}} : S^{\mathrm{G}}_{\mathrm{SB}} \times A_{\mathrm{SB}} \to S^{\mathrm{G}}_{\mathrm{SB}}$, where A_{SB} is the set of actions that the player can perform in the game.

GEM represents the mechanics of the game as the transition function of its MDP. We can re-express this transition function as $\boldsymbol{\mathcal{P}}^{\mathrm{G}}_{\mathrm{GEM}} : S^{\mathrm{G}}_{\mathrm{GEM}} \times A_{\mathrm{GEM}} \to S^{\mathrm{G}}_{\mathrm{GEM}}$. A_{GEM} is the set of actions that the player can perform in the game.

Manager Actions and Tuneable Parameters. SBDM defines a manager action as a tuple containing a command (*cause*, *deny*, *temporarily deny*, *re-enable*, or *hint*) and a plot point. Each tuple describes a variety of changes to the game's state (e.g., moving non-player characters or initiating dialogue) that collectively affect the given plot point as the given command describes. The parts of the game's state that are changed by manager actions represent the game's tuneable parameters ($S^{\mathrm{T}}_{\mathrm{SB}}$), meaning that we can re-express them as state factors in an FMDP: $S^{\mathrm{T}}_{\mathrm{SB}} \subseteq S_{\mathrm{SB}}$. Consequently, any SBDM manager action can be re-expressed as an assignment of new values to these state factors.

GEM defines a manager action as an assignment of a new transition function to the MDP that it uses to represent the game (i.e., a GEM manager changes the game's mechanics). To re-express this kind of manager action using the joint perspective, we need to express [changing mechanics] as [assigning new values to some state factors]. This challenge can be approached most directly from the observation that game mechanics can be adapted by changing their parameters. For example, the difficulty of *Super Mario Kart* can be adapted by choosing between three different race modes (50cc, 100cc, and 150cc). This one parameter (cc) affects many of the game's transition factors, including computations that govern the speeds of the karts and how the non-player racers steer. Based on this observation, a GEM manager action can be re-expressed as an assignment of new values to particular state factors in the game – those that are used as parameters by the game's mechanics. These are the tuneable parameters of our GEM FMDP: $S^{\mathrm{T}}_{\mathrm{GEM}} \subseteq S_{\mathrm{GEM}}$.

	SBDM	GEM
Game State Space	$S^{\text{G}}_{\text{SB}} \subseteq S_{\text{SB}}$	$S^{\text{G}}_{\text{GEM}} \subseteq S_{\text{GEM}}$
Game Mechanics	$\boldsymbol{\mathcal{P}}^{\text{G}}_{\text{SB}} : S^{\text{G}}_{\text{SB}} \times A_{\text{SB}} \to S^{\text{G}}_{\text{SB}}$	$\boldsymbol{\mathcal{P}}^{\text{G}}_{\text{GEM}} : S^{\text{G}}_{\text{GEM}} \times A_{\text{GEM}} \to S^{\text{G}}_{\text{GEM}}$
Tuneable Parameters	$S^{\text{T}}_{\text{SB}} \subseteq S_{\text{SB}}$	$S^{\text{T}}_{\text{GEM}} \subseteq S_{\text{GEM}}$
Manager Policy	$\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{SB}} : S^{+}_{\text{SB}} \times A_{\text{SB}} \to S^{\text{T}}_{\text{SB}}$	$\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{GEM}} : (S^{\text{G}}_{\text{GEM}} \cup S^{+}_{\text{GEM}}) \times A_{\text{GEM}} \to S^{\text{T}}_{\text{GEM}}$
Opportunities to Act	. . . G S M P G S M P . . .	. . . M G S P M G S P . . .

Table 1: Summary of comparing SBDM and GEM. The last row shows execution orders from Section 4.2.

Manager Policy. SBDM defines a manager policy as function which, given a game modelled by states and a transition function, specifies an action to perform in each state. We have already re-expressed SBDM states using state factors in an FMDP (S^{+}_{SB}) and SBDM manager actions as assignments of new values to state factors in the game (i.e., to S^{T}_{SB}). We can therefore re-express an SBDM manager policy as part of an FMDP's transition function ($\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{SB}} \subseteq \boldsymbol{\mathcal{P}}_{\text{SB}}$) that computes new values for S^{T}_{SB} based on S^{+}_{SB}. Formally, $\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{SB}} : S^{+}_{\text{SB}} \times A_{\text{SB}} \to S^{\text{T}}_{\text{SB}}$. For example, based on the sequence of plot points that have occurred (s^{+}_{SB}) and the player's last action (a_{SB}), an SBDM manager could decide to lock a door to a nearby room (by modifying s^{T}_{SB}).

GEM defines a manager policy as a function which, given (i) a game with states, actions, and a transition function, and (ii) a *history* of the game up to the current time step, specifies a transition function to use in each state. A history consists of a temporally-ordered record of which states have occurred, which actions the player has performed, and which transition functions the manager has previously selected. Similarly to SBDM's state representation, GEM's history can be re-expressed as a set of state factors in S^{+}_{GEM}. We have already re-expressed GEM's states as states in an FMDP ($S^{\text{G}}_{\text{GEM}}$), and we have explained how changing a transition function can be re-expressed as assigning new values to particular, tuneable parameters in an FMDP (i.e., to $S^{\text{T}}_{\text{GEM}}$). We can thus re-express a GEM manager policy as part of an FMDP's transition function: $\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{GEM}} \subseteq \boldsymbol{\mathcal{P}}_{\text{GEM}}$. Formally, $\boldsymbol{\mathcal{P}}^{\text{M}}_{\text{GEM}} : (S^{\text{G}}_{\text{GEM}} \cup S^{+}_{\text{GEM}}) \times A_{\text{GEM}} \to S^{\text{T}}_{\text{GEM}}$. For example, based on the game's current state $s^{\text{G}}_{\text{GEM}}$, its current model of the player (in s^{+}_{GEM}), and the player's most recent action (a_{GEM}), a GEM manager could decide to increase the probability of the player finding something valuable in the next container they search (by modifying $s^{\text{T}}_{\text{GEM}}$).

Opportunities to Act. SBDM allows a manager to act after each SBDM state that occurs, where the full execution order is stated is follows. Following a manager action, the player acts, and then both actions are used to transition SBDM MDP from its current SBDM state (S^{+}_{SB}) to a subsequent one. The manager then receives a new opportunity to act. Since S^{+}_{SB} varies with the game state, we assume that S^{G}_{SB} is computed at the same time as S^{+}_{SB}. This sequence of events (manager action, player action, game mechanics, new game state) corresponds to the second of the two execution orders that we discussed in Section 4.2.

GEM allows a manager to act immediately following each player action, but before the game's transition function executes to compute the game's state. This sequence of events (player action, manager action, game mechanics, new game state) is the first execution order we discussed in Section 4.2.

6 Discussion and Future Work

The contributions of this work are threefold. First, we formally specified the joint perspective of experience management, in which a game/manager pair is viewed as a Factored-state Markov Decision Process. We argued how every manager can be viewed from this perspective, and that it thus presents a unified basis for developing a way to understand different manager representations. Second, we described a way to re-express the common, disjoint perspective of experience management (where a game and manager are viewed as separate entities) using only our specification of the joint perspective FMDP. This provides a language into which any manager's representation can be translated, opening the potential to directly compare different representations. Third, we demonstrated this potential concretely by comparing two different manager representations: SBDM and GEM.

From our comparison of SBDM and GEM (Table 1), we have gained the following insights: (i) Both representations required extra state factors beyond what were needed solely for the game (S^{+}_{SB} and S^{+}_{GEM}). Although these factors can be represented as part of the game itself, the fact that considering them separately helped our discussion suggests that "manager-internal state" might be added as a seventh concept of the disjoint perspective. (ii) Some manager policies depend on the entire game state (like GEM's), while others depend on only features of the game state (like SBDM's); this may seem obvious in retrospect, but the difference only became apparent after translating SBDM's plot points into our precise language. (iii) The order in which the manager's policy and the game's mechanics execute is important. With one order (as in GEM), narrative mediation by intervention is possible, but with the other order (as in SBDM), it is not.

Our approach relies on the common assumptions and concepts that we stated in Section 4.1. While we expect that they will be easy to accept for most researchers in the field, we cannot guarantee that we have not missed one or more concepts that should be included; searching for such missing concepts is one avenue for future work, and pursuing this avenue might be simplified by applying our approach. It would also be interesting to compare more manager representations using our approach, and to explore whether our findings about SBDM and GEM hold across their variants.

References

Bates, J. 1992. Virtual reality, art, and entertainment. *Presence: The Journal of Teleoperators and Virtual Environments* 1(1):133–138.

Booth, M. 2009. The AI systems of Left 4 Dead. Presentation at the 5th AIIDE Conference.

Boutilier, C.; Dearden, R.; and Goldszmidt, M. 1995. Exploiting structure in policy construction. In *Proceedings of the 14th International Joint Conference on Artificial Intelligence*, 1104–1111. AAAI Press.

Chakraborty, D., and Stone, P. 2011. Structure learning in ergodic factored MDPs without knowledge of the transition function's in-degree. In Getoor, L., and Scheffer, T., eds., *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, 737–744. New York, NY: ACM.

Gentry, M. S. 1998. *Anchorhead*. Interactive Fiction Archive.

Ha, E. Y.; Rowe, J. P.; Mott, B. W.; and Lester, J. C. 2011. Goal recognition with markov logic networks for player-adaptive games. In *Proceedings of the 7th AIIDE Conference*, 32–39. Palo Alto, California: AAAI Press.

Hunicke, R., and Chapman, V. 2004. AI for dynamic difficulty adjustment in games. In *Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence*, 91–96. San Jose, California: AAAI Press.

Laurel, B. K. 1986. *Toward the design of a computer-based interactive fantasy system*. Ph.D. Dissertation, The Ohio State University.

Lindley, C. A., and Sennersten, C. C. 2008. Game play schemas: From player analysis to adaptive game mechanics. *Int'l Journal of Computer Games Technology* 7 pages.

Magerko, B. 2005. Story representation and interactive drama. In *Proceedings of the 1st AIIDE Conference*, 87–92. Marina del Rey, California: AAAI Press.

Mateas, M., and Stern, A. 2005. Procedural authorship: A case-study of the interactive drama Façade. In *Digital Arts and Culture (DAC)*.

Ludeon Studios. 2013. Rimworld. rimworldgame.com.

Valve Corporation. 2008. Left 4 Dead. www.l4d.com.

Nelson, M. J., and Mateas, M. 2008. Another look at search-based drama management. In *Proceedings of the 23rd AAAI Conference on Artificial Intelligence*, 792–797. Chicago, IL: AAAI Press.

Nelson, M. J.; Mateas, M.; Roberts, D. L.; and Isbell, C. L. 2006a. Declarative optimization-based drama management in interactive fiction. *IEEE Computer Graphics and Applications* 26(3):33–41.

Nelson, M. J.; Roberts, D. L.; Isbell, C. L.; and Mateas, M. 2006b. Reinforcement learning for declarative optimization-based drama management. In *Proceedings of the 5th Conference on Autonomous Agents and Multiagent Systems*, AAMAS '06, 775–782. New York, NY: ACM.

Ohyagi, Y., and Satou, K. 2005. Racing game program and video game device. Technical Report US7278913B2, U.S. Patent and Trademark Office.

Ramirez, A., and Bulitko, V. 2015. Automated planning and player modelling for interactive storytelling. *IEEE Transactions on Computational Intelligence and AI in Games* 7(4):375–386.

Riedl, M. O., and Stern, A. 2006. Believable agents and intelligent story adaptation for interactive storytelling. In *3rd International Conference on Technologies for Interactive Digital Storytelling and Entertainment (TIDSE 2006)*, 1–12. Darmstad, Germany: Springer.

Riedl, M. O.; Stern, A.; Dini, D.; and Alderman, J. 2008. Dynamic experience management in virtual worlds for entertainment, education, and training. In Tianfield, H., ed., *International Transactions on Systems Science and Applications*, volume 4, 23–42. Glasgow: SWIN Press.

Riedl, M. O.; Saretto, C. J.; and Young, R. M. 2003. Managing interaction between users and agents in a multi-agent storytelling environment. In *Proceedings of the 2nd Conference on Autonomous Agents and Multiagent Systems*, 741–748. Melbourne, Australia: ACM.

Riedl, M. O. 2009. Incorporating authorial intent into generative narrative systems. In *AAAI Symposium on Intelligent Narrative Technologies II*, 91–94. Palo Alto, California: AAAI Press.

Roberts, D. L., and Isbell, C. L. 2008. A Survey and Qualitative Analysis of Recent Advances in Drama Management. *International Transactions on Systems Science and Applications* 4(2):61–75.

Roberts, D. L.; Nelson, M. J.; Isbell, C. L.; Mateas, M.; and Littman, M. L. 2006. Targeting specific distributions of trajectories in MDPs. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*, 1213–1218.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive game AI with dynamic scripting. *Machine Learning* 63(3):217–248.

Thue, D., and Bulitko, V. 2012. Procedural game adaptation: Framing experience management as changing an MDP. In *Proceedings of the 5th Workshop on Intelligent Narrative Technologies*. Palo Alto, California: AAAI Press.

Thue, D.; Bulitko, V.; Spetch, M.; and Wasylishen, E. 2007. Interactive storytelling: A player modelling approach. In *Proceedings of the 3rd AIIDE Conference*, 43–48. Palo Alto, California: AAAI Press.

Thue, D.; Bulitko, V.; Spetch, M.; and Romanuik, T. 2011. A computational model of perceived agency in video games. In *Proceedings of the 7th AIIDE Conference*, 91–96. Palo Alto, California, USA: AAAI Press.

Thue, D. 2015. *Generalized Experience Management*. Ph.D. Dissertation, Department of Computing Science, University of Alberta, Canada.

Weyhrauch, P. 1997. *Guiding Interactive Drama*. Ph.D. Dissertation, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA.

Young, R. M.; Riedl, M. O.; Branly, M.; and Jhala, A. 2004. An architecture for integrating plan-based behavior generation with interactive game environments. *Journal of Game Development* 1(1):54–70.