Proceedings of the Eighteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment
(AIIDE 2022)

Using Multi-Armed Bandits to Dynamically Update Player Models in an Experience Managed Environment

Anton Vinogradov, Brent Harrison

Dept. of Computer Science, University of Kentucky
Anton.Vinogradov@uky.edu, Harrison@cs.uky.edu

Abstract

Players are often considered to be static in their preferred play styles, but this is often untrue. While in most games this is not an issue, in games where experience managers (ExpMs) control the experience, a shift in a player's preferences can lead to loss of engagement and churn. When an ExpM makes changes to the game world, the game world is now biased in favor of the current player model which will then influence how the ExpM will observe the player's actions, potentially leading to a biased and incorrect player model. In these situations, it is beneficial for the ExpM to recalculate the player model in an efficient manner. In this paper we show that we can use the techniques used to solve multi-armed bandits along with our own idea of distractions to minimize the time it takes to identify what a player's preferences are after they change, compensate for the bias of the game world, and to minimize the number of intrusive elements added to the game world. To evaluate these claims, we use a text-only interactive fiction environment specifically created to be experience managed and to exhibit bias. Our experiments show that multi-armed bandit algorithms can quickly recalculate a player model in response to shifts in a player's preferences compared to several baseline methods.

Introduction

An Experience Manager (ExpM) is often used in games when one wants to provide a tailored experience for the player. An Experience Manager acts as an omniscient third party, monitoring player actions and preferences, and utilizes of these observations to deliver better content to the player (Sharma et al. 2007).

The ExpM has the ability to take actions while the player is playing the game, gently guiding the player towards an optimal gameplay path. In more recent advances in experience management the ExpM has access to a player model created from observations of the player, often times gathered while the game is being played. This allows the ExpM to provide content that is more personalized to the player (Yu and Riedl 2013) and allow for a balance between authorial intent and player agency. Modeling a player as they play and having the ExpM change the environment that the player is modeled in has an issue though. The environment is biased by past choices that the ExpM has made. If the ExpM thinks that the player enjoys combat, it will send monsters to the player. The player then kills those monsters, not because they prefer it, but because the monsters are there. The ExpM sees that the player is engaging with this task and decides that the player must really love killing. If a player is static in their preferences this issue may not appear, as this would just help narrow down the static preferences. Unfortunately, it has been shown that players often do change their play styles (Valls-Vargas, Ontañón, and Zhu 2015), and that a dynamic computation of the player model needs to be done (Khoshkangini et al. 2018). This improperly managed game may be more frustrating and less engaging than an unmanaged game as the environment is now biased and thus the environment is tailored towards a set of preferences that the player no longer holds.

Past approaches have not attempted to control for this bias. One might think that removing the changes that the ExpM introduced could be a viable solution to controlling the bias, but this leads to greater issues. While managing the narrative the ExpM must make changes to the world, some of which make lasting changes to the narrative. Attempting to remove these changes to remove the bias may be impossible without disrupting the narrative and thus the player. Thus, it is necessary to instead compensate for bias by making additions to the game world. These additions need to be small enough as to not disturb the narrative, but still be noticeable to player.

In this paper we propose a system to introduce new objects into the game world to gather information from the player, and to quickly find the shift in the player's preferences[1]. We model the ExpM's player profiling as a Multi-Armed Bandit (MAB) to be able to take advantage of these techniques to quickly find the player's current preferences. MAB's ability to balance between exploration and exploitation allows us to quickly discover what the player's current preferences are without serving them too many unwanted distractions. This is critical as providing too many unwanted distractions may cause the player to disengage from the experience.

We evaluate our system using a custom text-based in-

Copyright © 2022, Association for the Advancement of Artificial Intelligence (www.aaai.org). All rights reserved.

[1]The code is available at: https://github.com/garyyo/MAB-PM-Recovery

teractive fiction environment and a set of scenarios where the ExpM is focused solely on finding a new player model with the assumption that it has already detected a shift in player preferences. We implement several different MAB algorithms and player agents across 20 different player preference shift scenarios to evaluate our system. We find that MAB algorithms are capable of quickly finding what the player agent's preferences are despite the inherent bias of the environment while reducing the number of distractions needed compared to baselines.

Background and Related Works

In this section we will discuss some of the earlier work done on drama management and provide a brief overview of multi-armed bandits.

Experience Management and Player Modeling

The goal of experience management, and the subset field of drama management, is to balance the player's sense of agency while also preserving the goals of the author. A Drama manager (DM) can achieve this by making changes to the game outside of the player's control through a series of actions. These actions may include things like locking or unlocking certain paths, adding or removing some characters or objects from the game, or modifying the information that is given to the player to guide them towards the author's narrative goals. Early works in drama management did not make use of a player model (Riedl and León 2008; Cavazza et al. 2009; Lamstein and Mateas 2004; Nelson and Mateas 2005, 2008).

Our approach works within the space of player modeling focused experience management as having an accurate and useful model of the player can allow for more intelligent ExpM actions (Sharma et al. 2007, 2010; Yu and Riedl 2013). These have more recently been expanded into multiplayer environments (Gray, Zhu, and Ontañón 2021; Zhu and Ontañón 2019). In each of these works, player preferences are assumed to be static. In our work, we explore the methods which would allow for an ExpM to recalculate a player model in response to shifts in player preferences.

There have been works recently that have acknowledged the need for dynamic player models. These works have focused on trying to dynamically recalculate a player model according to a pre-existing set of playstyles (Khoshkangini et al. 2018; Valls-Vargas, Ontañón, and Zhu 2015). The difference in their work is that they make an assumption of pre-existing categories for a player to be placed into. Our method does not make this assumption and seeks to learn a player model described purely by player behavior. Since our method is incorporated into an experience managed environment, we also need to deal with the left-over biasing of the environment from past ExpM actions.

MABs have been used in player modeling before with the focus of adapting games to their players (Gray et al. 2020; Gray, Zhu, and Ontañón 2021), but these are concerned with finding player models from scratch, rather than correcting one in live gameplay. As such while many of their goals are the same, such as finding a player model quickly, they have more freedom in how to modify the environment and can make more assumptions. Due to the issues our method is attempting to solve we cannot make these assumptions about a player's play style and must actively intervene to discover them.

Background on Multi-Armed Bandits

Multi-Armed Bandits (MABs) are a class of sequential decision-making problems where an agent is tasked maximize the amount of reward from iteratively taking one of k actions (often called arms) (Slivkins 2019). Since we are able to model the act of discovering the player model as a MAB, we can use the techniques used to solve a MAB for this task.

They take their name and inspiration from slot machines in a casino, which are informally referred to as one-armed bandits. The problem can be framed using this analogy as follows. Assume the existence of a slot machine with many arms that a player could pull. Each arm has a different payout probability, but the user does not know which arm will likely result in the largest payout. As such, the person needs to find a strategy (policy) to maximize their own return on playing time while minimizing their own losses from the cost associated with pulling an arm. Rewards are often assumed to be drawn from fixed (yet independent) distributions. We make the same assumptions in this work, with the added assumption that the environment is fully observable. In other words, the user receives an accurate observation of the reward they receive after pulling an arm.

The difficulty associated with the multi-armed bandit problem is in balancing the act of gathering information about the payout associated with each arm (exploration) and maximizing reward given the current known information (exploitation). In this work, we make use of three different algorithms for our MAB policy learning: ϵ-greedy, UCB1-Tuned, and Thompson sampling. The simplest is ϵ-greedy, which at each turn will either randomly choose an arm to pull with a probability ϵ or will select the arm with the highest expected payout given current information with a probability $1 - \epsilon$. Random actions serve as a way to gather information about the arms while the greedy actions take advantage of this information. We also consider a variant of ϵ-greedy called ϵ-decreasing in which the ϵ value decreases over time according to a decay function.

UCB1 is a technique for learning MAB policies that involves estimating the upper confidence bound on rewards for each arm and selecting the one that maximizes this value. We use a variant of UCB1 called UCB1-Tuned which uses a more finely tuned upper confidence bound for the variance of an arm for a Bernoulli random variable (Auer, Cesa-Bianchi, and Fischer 2002). This method performs substantially better than UCB1 in all of our tests, so we have opted to only include this tuned variant. UCB1-Tuned (like UCB1) requires that one of each arm is played before more intelligently selecting the arm with the highest upper confidence bound.

$$\arg\max\left(Q_a + \sqrt{\frac{\ln n}{n_a} min(1/4, V_a(n_a))}\right) \quad (1)$$

$$V_a(n) = \left(\frac{1}{n}\sum_{t=1}^{n} R_{a,t}^2\right) - \bar{R}_{a,n}^2 + \sqrt{\frac{2\ln N}{n}} \quad (2)$$

UCB1-Tuned calculates the upper confidence interval for every arm using equations 1, 2 where Q_a is the sample mean for arm a, n_a is the number of times arm a is given, $R_{a,n}$ is the reward of arm a at time n, $\bar{R}_{a,n}$ is the average reward, and N is the total number of rounds that have occurred. The $1/4$ constant is an upper bound on the variance of a Bernoulli random variable, where V_a is the calculated variance.

$$\arg\max\left(B\left(\alpha_a + 1, \beta_a + 1\right)\right) \quad (3)$$

Thompson sampling is the first bandit algorithm in literature first appearing in 1933 (Thompson 1933). It builds a probability model of the expected rewards, which it then samples an arm pull from. It draws samples from a beta distribution of the number of successes and failures for each arm, choosing the sample with the maximum probability, as seen in equation 3, where α and β represent the number of successes and failures for arm a. Initially, both α and β are set to 1 to establish a uniform prior distribution. Thompson and UCB1 both are able to naturally shift between primarily exploration early on to being more exploitation as they gain more information.

Methods

In this section we will go over the various parts of our system and how they are adapted to fit the framework of a MAB. We will also discuss the implementation decisions of our interactive narrative game environment.

Multi Arm Bandit Adaptation

In an experience managed system, the experience manager changes the game world based on expected player behavior as defined by a player model through the use of ExpM actions. If player's preferences change during gameplay the player model can be considered outdated. Any ExpM actions made based on this now outdated player model will not have the desired effect. In these situations, the player model should be recalculated; however, the ExpM cannot use past observations to perform this recalculation as these player observations were gathered in a biased setting caused by the ExpM.

To recalculate the player model, we propose an additional set of ExpM actions that focus on offering a new set of tasks to the player, which we call distraction actions. The sole purpose of these new ExpM actions is to gather information about the player and their preferences. This means that they will not attempt to further authorial goals nor make the environment well suited to the player.

One of the core concepts to our method is the idea of a *distraction*. A distraction is an object that can be added to the game environment via an ExpM distraction action that the player can interact with, much like a regular game object. Distractions differ from other game objects in that their purpose is not to contribute to the game, but rather to offer the player alternative gameplay options that the ExpM can use to update the player model. Distractions can also be removed from the environment when they are no longer needed, as they should be designed to not be important enough to the narrative to require a consistent presence.

All objects in the game, including distractions, have a primary means that the player would interact with that object which we call the *action-type*. For example, a book object may have the action-type *read*. These action-types correspond to the axes of our player model, and thus are related to regular ExpM actions. This is common way of representing the player model that we see elsewhere in literature (Thue et al. 2007; Sharma et al. 2007) In our system a regular ExpM action would consist of the ExpM noticing that the player model has a high affinity to a specific action-type or types and then customizing future content to include objects with that action-type. On the other hand, an ExpM distraction action consists of simply adding a distraction corresponding to an action-type (which it wants to test the player on) to the environment.

In this work, we propose to model this process as a multi-armed bandit problem. There are several benefits to doing this, but the most important one is that existing MAB methods work to balance the exploration/exploitation trade-off in order to quickly learn behavior policies. The ExpM has to balance between providing distractions for the purposes of gathering information critical to recalculating the player model (exploration) and taking actions that the player is interested in so that they do not disengage too much from the experience (exploitation). ExpM distraction actions are one of the primary means that the ExpM and player are modeled as a MAB, as these are used as an arm pull, with one arm for each action-type. Before each player turn, the ExpM can take a distraction action and add a distraction with a specific type to the environment. When a player interacts with a given distraction, a reward of one is given to its type, and when they do not a reward of zero is given. We consider both the player interacting with a different object and the player moving to a different area as not interacting with that distraction. Average reward is then calculated as the total number of times a player interacts with a distraction of a given action-type over the number of times that action-type has been given.

While the experience manager is providing distractions, the player is constantly making choices as to which object they want to interact. We observe these actions to form an *action history*, which contains a turn-by-turn record of what objects the player interacted with (if any), if these objects are distractions, and what the action-type of the object is. Previous observations of the player may be biased since the player's actions are limited by what is available in the environment, and the environment has been influenced by the ExpM's past actions. Since we want our observations to be as unbiased as possible, we only consider a truncated version of the action history, only including the actions that

happened after we are sure that the player's preferences have shifted. We then calculate a dynamic player model by finding the frequencies of each action-type within the truncated history.

Environment

To test our methods, we use a custom text-based Inform7 environment. This environment consists of a game world made of 7 areas, called rooms, that the player agent can move around in, with game objects that the agent can interact with, with each object having a set of methods of interacting with that correspond to our action types. The rooms that are in our environment are set up to all be traversable using only the commands north, south, east, west as they represent the outdoors portions of a standard video game medieval town.

We use simulated agents in place of human subjects as it allowed us to make direct measurements of the agent's internal preferences and compare them to the computed player model. These simulated agents have the ability to interact with objects in the town through a list of valid actions provided by the environment that are currently available to the agent. These interactions are split across five different action-types: *look*, *talk*, *touch*, *read*, and *eat*. Each of these action-types has a corresponding distraction, and some number of objects scattered across the game world.

Our environment is set up to simulate the effects of a previously managed player who is already familiar with the game. As such, we start in the middle of a quest that is centered around the *talk* action-type. Other action-types that the agent did not have a preference towards are still included as it is often not possible to design a narrative with only a single action-type, but these are fewer in number, which we call *environmental* action-types. Action-types that are currently not present in the environment at all are grouped under the name *missing* action-types. We consider *look* and *touch* along with *talk* to be *environmental* action-types and *read* and *eat* to be *missing* action-types. This setup forms a biased environment where a shift in the player's preferences to *read* and *eat* may not be adequately provided for.

A typical playthrough of the environment consists of the player starting in the middle of town square with the task to find an old lady's cat. This would lead a human player to ask around town for the location (using the *talk* action-type), and eventually stumble upon some clues for it's location. The player can also *look* around for physical clues and pick up various objects to give to NPCs and to unlock doors (using the *touch* action-type) but these are not the focus of the quest. The player does not have access to the parts of the game that would allow for reading (like a library) or eating (a potions shop) and thus we consider these missing action-types. Distractions would be spawned in as an object similar to regular game objects but does not contribute to the quest. For example, this could be a NPC spawned in the town market that the player can talk to, but they would give no clues to the cat's location. The game world is often filled with objects that are not directly related to the main quest so this hopefully does not detract from the experience of the game, but we expect that it might and thus try to minimize adding distractions.

Experiments

In our experiments we use a single environment to test the effects of 3 different agents, 6 ExpMs, and 20 scenarios. In this section we go over the differences between all of these. We aim to maximize the rate at which our calculated player model converges to a new value and how close we are to the true player preferences, without giving too many distractions.

Player Agents

We used simulated players to test our environment. In order to better model player variability, we use three different types of agents which make decisions in different ways: *Exploration Focused Agent*, *Novelty Focused Agent*, and *Goal Focused Agent*. All agents have an underlying preference distribution which describes their likelihood to engage with certain types of actions which is dictated by the scenario. The agents first make a decision of what type of object they wish to interact with then selects an object of that type. If the selected type does not have an object in the current area the agent will fall back on to a default behavior, detailed below.

The *Exploration Focused Agent* attempts to interact with objects in line with its internal preferences with a 10% chance each turn to randomly move to a different area. Its default behavior is to also randomly wander and thus it completely ignores the quest and any of the required actions needed to complete it but may still achieve quest relevant actions incidentally.

The *Goal Focused Agent* also primarily attempts to interact with objects in line with its internal preferences, much like the previous agent, but changes its default behavior to be equally split between taking a single action to achieve the goal of the quest and wandering. These actions are predetermined for the quest for the sake of the experiment but often require that the agent moves to the correct area first.

The *Novelty Focused Agent* likewise attempts to interact with objects in line with its internal preferences but puts equal importance on how novel the object is. This has the effect that objects that this agent has not seen before or has not interacted with in a while will be more likely to interacted with even if the agent's preferences do not reflect this. This agent's default behavior is the same as the Goal Focused Agent's, split between achieving the quest and wandering.

The Exploration and Goal Focused Agents are inspired by the Bartle taxonomy of player types (Bartle 1996) representing explorers and achievers respectively. The other two groups in the taxonomy (killers and socializers) require a more social representation and are not applicable to our environment. On the other hand the Novelty Focused Agent is not inspired by any taxonomy, but rather is based on findings in literature on user engagement across many different technology platforms, including games, which states that novelty (or variety) can increase engagement (O'Brien and Toms 2010).

Experience Managers

We use many different Experience Managers to both establish our baselines and to test our methods. Each ExpM takes

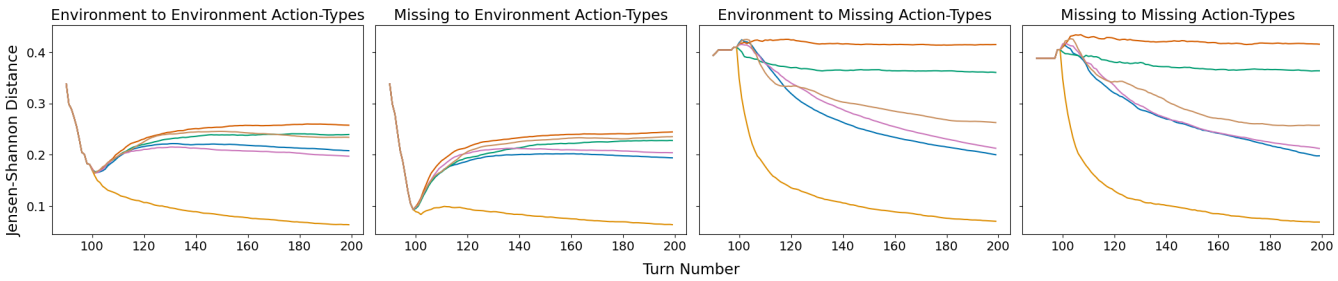

Figure 1: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Exploration Focused Agent**

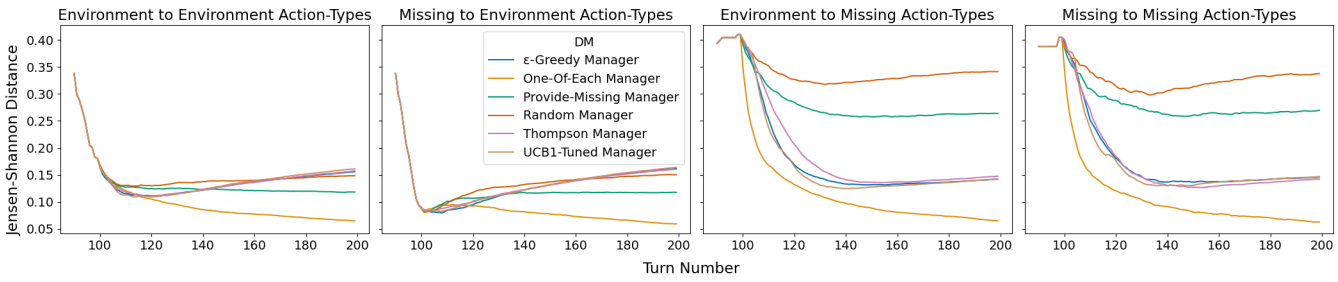

Figure 2: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Goal Focused Agent**

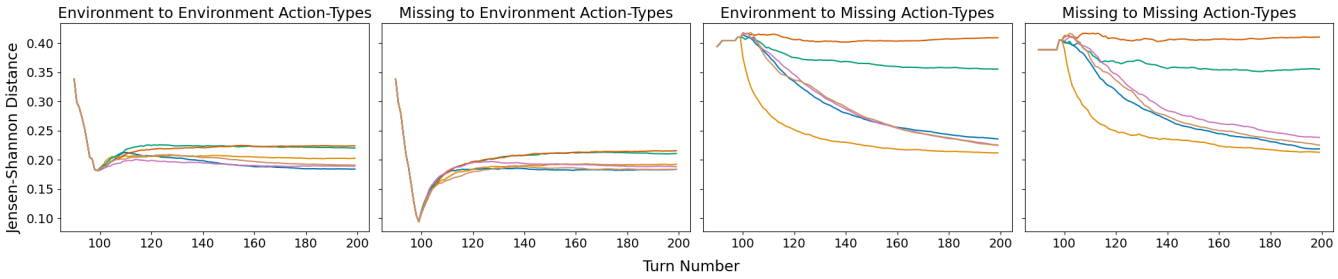

Figure 3: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Novelty Focused Agent**

a single distraction action each turn which generally creates a single distraction object. These distractions objects are removed after a single turn for these experiments.

We include 3 managers as baselines: the One-of-Each manager which provides one of each of the distraction objects each turn, the Random Manager which provides a random distraction each turn, and the Provide-missing manager which provides a distraction that is least represented in the current area. The One-of-Each manager is expected to perform the best as it breaks from our requirement of minimizing the number of distractions given. Our MAB managers each implement a different MAB algorithm, of which we use 3: ϵ-greedy ($\epsilon = 0.2$), UCB1Tuned, and Thompson. The details of these algorithms are present in the background section for MABs. These were chosen after testing several different algorithms including ϵ-decreasing, various ϵ values for ϵ-greedy, and UCB1. Of these we found that UCB1 performed significantly worse than other baselines, ϵ-decreasing performed similarly to the others but was slightly worse, and that the optimal value for ϵ-greedy is 0.2.

Scenarios

We test 20 different scenarios, one for each transition from a preferred action-type before (pre-preference) to a preferred action-type after (post-preference) the preference switch. The preferred action-type has a weight of 11/15 and the rest have a weight of 1/15. This allows us to look at all the possible preference shifts and see how our system performs on each. These 20 scenarios are then categorized into 4 group scenarios, *Environment to Environment*, *Missing to Environment*, *Environment to Missing*, and *Missing to Missing*. These correspond to our grouping of the action-types into environmental and missing action-types, where the *look*, *talk*, *touch* are considered environmental and the *read* and *eat* action-types are considered missing. Each group contains 6 scenarios except for *Missing to Missing* which only contains 2 scenarios.

For the sake of consistency between trials we pre-generate and reuse the first 100 turns of each scenario. This pre-generated history consists of 90 turns where the agent is using the pre-preference, and 10 turns where the agent is us-

Agent	Random	Provide-Missing	ϵ-Greedy	UCB1-Tuned	Thompson	One-Of-Each
Environment to Environment Action-Types						
Exploration Focused	0.354±0.249	0.292±0.205	0.269±0.339	0.295±0.282	***0.211±0.250***	*0.019±0.013*
Goal Focused	0.097±0.073	***0.064±0.055***	0.087±0.024	0.092±0.025	0.088±0.025	*0.019±0.013*
Novelty Focused	0.255±0.162	0.232±0.136	***0.163±0.118***	*0.171±0.126*	*0.169±0.108*	*0.181±0.055*
Missing to Environment Action-Types						
Exploration Focused	0.327±0.250	0.270±0.194	*0.234±0.323*	0.291±0.260	***0.229±0.267***	*0.018±0.013*
Goal Focused	0.102±0.088	***0.062±0.051***	0.093±0.026	0.094±0.024	0.091±0.025	*0.016±0.011*
Novelty Focused	0.238±0.161	0.212±0.123	*0.162±0.125*	***0.156±0.105***	*0.171±0.126*	*0.163±0.050*
Environment to Missing Action-Types						
Exploration Focused	0.747±0.118	*0.562±0.111*	*0.273±0.433*	*0.362±0.275*	***0.266±0.340***	*0.023±0.014*
Goal Focused	0.585±0.190	*0.347±0.135*	*0.079±0.040*	***0.073±0.024***	*0.081±0.030*	*0.019±0.013*
Novelty Focused	0.735±0.138	*0.551±0.109*	*0.303±0.331*	*0.257±0.231*	***0.254±0.187***	*0.197±0.054*
Missing to Missing Action-Types						
Exploration Focused	0.747±0.107	*0.568±0.119*	***0.260±0.378***	*0.350±0.287*	*0.268±0.367*	*0.022±0.015*
Goal Focused	0.570±0.200	*0.363±0.146*	*0.083±0.039*	*0.079±0.023*	***0.075±0.029***	*0.018±0.012*
Novelty Focused	0.737±0.133	*0.548±0.108*	***0.251±0.245***	*0.258±0.214*	*0.299±0.281*	*0.199±0.055*

Table 1: Mean and Std. of the JS distance between player model and agent preferences on the final turn for all tests. Bolded entries represent the best performing manager (excluding One-Of-Each) and italicized entries are statistically significant and better than Random with $p < 0.001$ according to a Student's T-Test.

ing the post-preference. We include the 10 turns of the post-preference to simulate the time between when the player's preferences shift and when the ExpM starts to take action. Since we reuse this pre-generated history across multiple agents, this history uses the Goal Focused Agent.

We measure the agent's preferences against our calculated player model by treating each as a probability distribution over action-types and use Jensen–Shannon (JS) distance to measure the distance between the two. JS distance is used because it is symmetric and is guaranteed to have a finite value. For the purpose of this calculation, we start at turn 90 of the pre-generated history as this is the point at which the agent switches to the post-preference, though the ExpM does not take distraction actions until turn 100. For each run we first average the models over each trial and use that averaged model to compare against the agent's preferences. We run 50 trials for each test of agent, ExpM, and scenario.

Results and Discussion

For each test we have created a graph that measures the JS distance between the player agent's internal preferences and the measured player model in Figures 1, 2, and 3 for the *Exploration Focused*, the *Novelty Focused*, and the *Goal Focused* Agents respectively. Each graph starts at turn 90 when the preferences are switched and ends on turn 199 for a total of 110 turns. While the actions taken between turn 90 and 100 are all the same, we include them to show the effect of switching between the different groups of preferences. We will discuss these figures by first addressing the performance of the baselines, then the MAB based managers, and finally some of the effects of different agents and scenarios.

Scenario Differences

In the *-to Environment* scenario groups we find that since the environment already provides the agent with objects that match its preferences there is not much information that can be gained. This is especially true when going from Missing Actions to Environment Actions where this trend is followed by a quick increase in the distance between the player model and the agent's preferences seen in Figures 1 and 3. This is especially prevalent in the Novelty Focused Agent as it prefers objects that it deems novel, and since the manager has not taken any distraction actions before turn 100, the sudden addition of distractions means that it will likely start to interact with the distractions instead of solely in accordance with its preferences. These two scenario groups represent a situation that does not need ExpM intervention as the current environment is still well suited for the player despite their preference shift.

The scenarios going *-to Missing* represent the cases that are more appropriate for intervention via ExpM distraction actions. In these cases, the environment is not well suited for the types of actions the agent prefers, and thus without intervention the user's calculated player model will continue to drift further from their preferences. We do not consider the existence of the former scenario group to be an issue as going to an environment action-type would not trigger the need for the ExpM to take distraction actions. Going forward we will be focusing on these the *-to Missing* scenario groups unless noted otherwise.

Manager and Agent Differences

Our best performing manager is the One-Of-Each manager. This manager serves as a lower baseline, with the unfair advantage of being able to provide multiple distractions at the same time. Giving multiple distractions has the effect of preventing the agents from moving, as they will never fall back to their default behavior, with the exception of the Exploration Focused Agent which still has a small chance to move. This manager serves as the lower bound on JS distance and outperforms ever other manager in nearly every case.

The random manager is our other baseline, and it serves

as a pure exploration option. We find that for the *-to Missing* scenario groups all of our managers are statistically better than this manager as shown in Table 1. This shows that attempting to only explore and gather information does not lead to better results, and that a balance must be made to be able to quickly recalculate the player model. For our Novelty and Exploration Agents giving the player agent completely random distractions just serves to distract them without gaining any information. The Novelty Agent will focus on the distractions since they are new, while the Exploration Agent may choose to wander instead of interacting, lowering its chance that it would interact with the distraction.

The provide-missing manager works differently than our other baselines as it is informed, but also differently than our MAB managers in that it does not try to gain information from the player. It serves to test the importance of only compensating for bias in the environment as it tries to provide a distraction for the least represented action-type. While it performs statistically better than random, even sometimes in the *-to Environment* scenario groups, it does significantly worse than the MAB managers. This suggests that the role of the environmental bias is a factor, and that observing the environment should play a part in how a manager chooses a distraction. We will be integrating the behavior of this manager with MAB managers in future work.

We find that while the MAB managers perform very similarly to each other there are some slight differences between them. UCB1-Tuned often has a slight lead but this does not persist, and Thompson is slower to achieve the same performance as the rest but sometimes beats out UCB1-Tuned. The most consistent is ϵ-greedy, but this does not result in the lowest score at the end. This end is currently arbitrarily set, so being able to decrease the distance quickly may be more important in the future.

In some rare specific cases we found that the MAB managers are actually capable of performing better than One-Of-Each. This can be seen with the Novelty Focused Agent in the *-to Environment* scenario groups on most MAB managers in Table 1. The MAB managers have an average reward value for each arm which often corresponds to the agent's preferences, but in rare cases the average reward is higher for the preferred action-type than the agent's preferences. While exploiting its knowledge, the manager (especially ϵ-greedy) only pulls the arm with the maximum average reward which causes the calculated player model to overshoot the player's preferences if not correctly tuned. This overshooting effect is also why we see the performance of the MAB managers in the Goal Focused Agent (Figure 2) start to get worse around turn 140. Here the distance lowers quickly because the average reward distribution has already overshot the player preference distribution and continuing to give distractions to the player actually gives them too many. While this may serve as an occasional advantage this effect is inconsistent, often harmful, and difficult to diagnose and in future work we will attempt to limit its influence.

Conclusion

While personalized experience management and dynamically updating player models has been explored before, both have not been implemented together. We believe that the prevalence of player preferences shifts requires that ExpMs have a means to quickly recover the player model when a preference shift occurs. The simple solution of just observing the player is not enough as there is bias left over from the ExpM's previous actions. In this paper we implement and compare several methods used to compensate for the bias and allow the experience manager to dynamically update the player model by modeling the system as a multi-armed bandit. We find that these methods can quickly recalculate a player model in response to preference shifts.

References

Auer, P.; Cesa-Bianchi, N.; and Fischer, P. 2002. Finite-time analysis of the multiarmed bandit problem. *Machine learning*, 47(2): 235–256.

Bartle, R. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research*, 1(1): 19.

Cavazza, M.; Pizzi, D.; Charles, F.; Vogt, T.; and André, E. 2009. Emotional input for character-based interactive storytelling. In *AAMAS '09: Proceedings of the 8th International Conference on Autonomous Agents and Multiagent Systems, Budapest, Hungary, 10-15 May 2009*, 313–320.

Gray, R. C.; Zhu, J.; Arigo, D.; Forman, E.; and Ontañón, S. 2020. Player modeling via multi-armed bandits. In *International Conference on the Foundations of Digital Games*, 1–8.

Gray, R. C.; Zhu, J.; and Ontañón, S. 2021. Multiplayer Modeling via Multi-Armed Bandits. In *2021 IEEE Conference on Games (CoG)*, 01–08. IEEE.

Khoshkangini, R.; Ontañón, S.; Marconi, A.; and Zhu, J. 2018. Dynamically extracting play style in educational games. *EUROSIS Proceedings, GameOn*.

Lamstein, A.; and Mateas, M. 2004. Search-based drama management. In *Proceedings of the AAAI-04 Workshop on Challenges in Game AI*, 103–107.

Nelson, M.; and Mateas, M. 2005. Search-based drama management in the interactive fiction Anchorhead. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 1, 99–104.

Nelson, M. J.; and Mateas, M. 2008. Another Look at Search-Based Drama Management. In *AAAI*, 792–797.

O'Brien, H. L.; and Toms, E. G. 2010. The development and evaluation of a survey to measure user engagement. *Journal of the American Society for Information Science and Technology*, 61(1): 50–69.

Riedl, M. O.; and León, C. 2008. Toward vignette-based story generation for drama management systems. In *Workshop on Integrating Technologies for Interactive Stories-2nd International Conference on INtelligent TEchnologies for interactive enterTAINment*, 8–10.

Sharma, M.; Ontañón, S.; Mehta, M.; and Ram, A. 2010. Drama management and player modeling for interactive fiction games. *Computational Intelligence*, 26(2): 183–211.

Sharma, M.; Ontañón, S.; Strong, C. R.; Mehta, M.; and Ram, A. 2007. Towards Player Preference Modeling for

Drama Management in Interactive Stories. In *FLAIRS Conference*, 571–576.

Slivkins, A. 2019. Introduction to Multi-Armed Bandits. *Foundations and Trends® in Machine Learning*.

Thompson, W. R. 1933. On the likelihood that one unknown probability exceeds another in view of the evidence of two samples. *Biometrika*, 25(3-4): 285–294.

Thue, D.; Bulitko, V.; Spetch, M.; and Wasylishen, E. 2007. Interactive storytelling: A player modelling approach. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 3, 43–48.

Valls-Vargas, J.; Ontañón, S.; and Zhu, J. 2015. Exploring player trace segmentation for dynamic play style prediction. In *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

Yu, H.; and Riedl, M. O. 2013. Data-driven personalized drama management. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Zhu, J.; and Ontañón, S. 2019. Experience management in multi-player games. In *2019 IEEE Conference on Games (CoG)*, 1–6. IEEE.