2024

Detecting and Recovering From Player Preference Shifts In A Player Modeled Experience Management Environment

anton vinogradov
University of Kentucky, garyantonyo@gmail.com
Author ORCID Identifier:
https://orcid.org/0009-0000-4312-4896
Digital Object Identifier: https://doi.org/10.13023/etd.2024.394

anton vinogradov, Student

Dr. Brent Harrison, Major Professor

Dr. Simone Silvestri, Director of Graduate Studies

Detecting and Recovering From Player Preference Shifts In A Player Modeled Experience Management Environment

DISSERTATION

A dissertation submitted in partial
fulfillment of the requirements for
the degree of Doctor of Philosophy
in the College of Arts and Sciences
at the University of Kentucky

By
Anton V. Vinogradov
Lexington, Kentucky

Director: Dr. Brent Harrison, Professor of Computer Science
Lexington, Kentucky
2024

ABSTRACT OF DISSERTATION

Detecting and Recovering From Player Preference Shifts In A Player Modeled Experience Management Environment

An important challenge in game design is understanding and maintaining player engagement. This is particularly crucial in both entertainment and educational games, where the player's commitment to the game directly impacts their experience and learning outcomes. However, quantifying engagement proves challenging due to the diverse interests of players. This dilemma is addressed through adaptive game design techniques where the game world is personalized to suit player preferences. In computer games, this personalization is facilitated through Experience Management and Player Modeling, where an intelligent agent gathers information on the player, including their preferences and actions, and takes actions to modify the game world to better suit the player.

This approach to adapting games to players currently hinges on the assumption that player preferences remain relatively stable over time, allowing the intelligent agent to refine its player model accordingly. However, this assumption becomes problematic when player preferences undergo significant shifts during gameplay. It is difficult to model every aspect of a player's preferences and thus predict such shifts, but these shifts can be handled without being directly anticipated. In this dissertation, I create a system that allows for the detection and recovery of shifts in player preferences. This technique makes use of a two step process where the experience manager either watches the player for behavior that is anomalous compared to their previous behavior, or attempts to recover the player's preferences and form a new player model once it detects that the player's current preferences are no longer similar to their past model. This process makes use of a new type of game object that I call *distractions* which help with testing the player's preferences without relying on the existing environment as it may be limited by previous experience manager interventions.

My research contains three parts, the detection of player preference shifts, the recovery of the player model, and the verification of the system on human data. I use an artificial testing environment that mimics aspects of human behavior to evaluate these systems in an automated fashion and a human evaluation of the artificial testing environment to validate the claims made with the automated tests.

KEYWORDS: Adaptive Games, Experience Management, Player Modeling, Novelty Detection, Multi-Armed Bandits

Anton V. Vinogradov

September 16, 2024

Detecting and Recovering From Player Preference Shifts In A Player Modeled Experience Management Environment

By
Anton V. Vinogradov

Dr. Brent Harrison
Director of Dissertation

Dr. Simone Silvestri
Director of Graduate Studies

September 16, 2024
Date

CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1 Introduction

Games as a medium forms an important part of our cultural identity, with video games being one of the most recent additions. In 2024 it is estimated that 3.32 billion people actively play video games, from simple word puzzle games that help pass the time to realistic training simulations and everything in between. They form a massive industry that serves to entertain people, let them socialize comfortably across distances, and train and educate.

An important topic in games research, for both entertainment and training, is the idea of *engagement*. Engagement is a multifaceted concept that is related to a number of other concepts such as immersion, flow, fun, presence, and enjoyment [41]. It is the means by which the player *wants* to continue playing a game on some level, and serves as a means for the game's authors to keep players playing. A player can be completely engaged in a game enough to be in a flow state[11], but more often than not will still have awareness of their surroundings and needs of their surroundings[9]. This arguably extends past a gameplay session; a player can be engaged in a game, still playing a game without actively *playing* it, and can resume playing it with no interruption to their engagement with it. Staying engaged with a game helps players continue playing, increasing their entertainment and maximizing their enjoyment. In serious context such as educational games this serves as a means to help people learn faster[39], increase knowledge acquisition, and content understanding[10]. It is, then, unfortunate that engagement is so difficult to codify in games, itself being just the simple measure of how interesting the game is to a player. As of now, a large portion of games created are designed to cater to only engage specific interests that players may have, which may exclude a large number of possible players. A specific player will generally have multiple interests with each likely to deviate from the average, thus it is often considered difficult to make a game that is well suited for all players. There have been game developers that try to combat this by creating games that contain a large variety of interests that they cater to, with open-world games like Minecraft (2011) and Witcher 3 (2015) becoming very popular recently, giving unparalleled freedom and variety to players. While these do present a solution to the problem, ultimately the underlying issue of loss of engagement is just pushed further down and this approach creates a new set problems. This underlying problem is that players are asked to be the discriminator between the content that they want and the content that they do not want, all without necessarily having a clear picture of how to access and pursue that content. This can be tedious for the player as they have have to simultaneously learn how to play the game, decide which parts are interesting to them before actually experiencing the game, and find a way to access that content, all of which can impact their engagement with the game.

This problem is not unique to video games and has appeared in many other submediums, including tabletop role-playing-games (TTRPGs) which have a smaller amount of active players at only 50 million playing the most popular TTRPG, *Dungeons & Dragons*, as of 2020. TTRPGs themselves have an interesting solution to this engagement issue: While the game world may have a large spread of content for players to interact with much like open-world video games, they do not exclusively require that the player makes all the decisions to which kinds of content they want to engage in. Put plainly, the player's engagement on a session level is also being managed by a third party.

Games such as Dungeons & Dragons not only have several players playing the game, but have another member called the Game Master (GM) who controls the rest of the game environment outside of the player's characters. This is typically a single person and their task is to act as the rules underlying the world, though a more important goal is to work together with the players to craft an interesting narrative for all of them to enjoy. The GM has the ability to observe the players as they interact with the world, along with the responsibility of describing how the world reacts to those actions, making changes to suit the player's actions and their preferences for where the game goes. Thus, the GM can make decisions that adapt the world to the player, sometimes even going outside existing pre-planned content to better accommodate the wants and needs of the player and helping them stay engaged in the game for longer. This continues to bleed over between sessions too, players that are more interested in the game are more likely to play better, devoting more of their time preparing and planning for upcoming sessions.

This ability to adapt a game to a player can be done with computer games, games which are called adaptive games. Game personalization like this can be done through *Experience Management*, the process of optimizing a player's experience in an interactive environment by changing that environment while the experience is underway [35]. This generally invokes an Experience Manager (ExpM), an automated system that guides players through a more interesting and tailored experience than that which could normally be achieved. Much like a GM, the ExpM sits alongside the player and the game environment to observe the player and their actions, and, using the information gained by these observations, is able to take its own actions to change the environment. This has been used in educational games, where it is of the utmost importance to ensure that the learning process is well-suited for each student. One example of this is *Crystal Island* [38], which uses narrative adaptation to adapt the game to the students and give customized guidance and feedback based on their personal educational needs. In commercial games, you can find examples such as dynamic difficulty adjustment as seen in Resident Evil 4's (Capcom, 2005) hidden Difficulty Scale system which modifies the difficulty based on how well the player is performing and the Left 4 Dead *AI Director* (Valve, 2008) which use dynamic enemy placement to control player experience in reaction to their actions.

ExpMs need information about the player to be able to change the environment according to their needs, much like how a GM cannot do a good job of running a TTRPG without interacting and understanding the players. A good GM will naturally have a mental model of the their players, understanding their motivations, strengths, and weaknesses and making use of them to create tense, fun, and exciting gameplay moments. With an ExpM, some of this information comes from making assumptions about the player, such as in a rule-based system like dynamic difficulty adjustment or by aggregating previous player data to create a model of how an average player acts. This is akin to a GM making assumptions about how a player would play based on broad experience, players who want to play as a thief may want to be sneaky, or that new players may struggle to role-play. These assumptions can help inform some of the actions of an ExpM but often are not enough to personalize the game to specific players, limiting their ability. Instead, a more immediate and personal source of information comes from taking direct observations of the player's action, allowing the ExpM to directly react to what the player is doing in the game [26]. These direct observations can be used to form a player model (PM), a persistent model of the player that can be queried at a later point that enables the ExpM to decide what would be best for the player to do next, bypassing the need for the player to sift through the options themselves. This model can be quite simplistic while arguably still being considered player-modeled experience management. Even dynamic difficulty adjustment has a sort of persistent model (has the player died often recently) and uses that to then decide how to set the difficulty level (if so, lower the difficulty). It can also be significantly more advanced and model the player's preferences for various actions that exist in the game world and automatically select events that the PM predicts that the player would enjoy [33, 47]. In more modern times these have even used machine learning to model a player's specific preferences and for horror games even their fears[12]. ExpMs can work well in theory; however, as they stand, there are some limitations in how player models are used in modern experience management systems that make them difficult to apply to some game environments.

Currently, player models in experience-managed games make an assumption that the player's preferences do not change much and that any further observations will help refine the PM to better reflect the player's preferences. This assumption is generally considered necessary as without it there is little information that the ExpM can trust and this is reflective of how people act as they do tend to have some consistency. That being said, the strong assumption that people will never change their preferences, or at least not in the playtime of a game, is too limiting and has been shown to be false [48]. Given this, we run into a problem: if an ExpM is making changes to the world to limit what the player has access to and the player suddenly shifts their preferences, then the player may be in a situation where they do not have access to the sorts of things they prefer. In attempting to solve the problem of too much choice the ExpM necessarily biases the environment to the player's preferences seen in the PM, closing off certain paths and removing actions and objects from the world. When a player starts to prefer to do an action that currently does not exist the environment, there is no good way for the ExpM to notice this, it cannot observe the player interacting with what they want when those types of interactions have been removed. This in turn limits how quickly the ExpM can update the PM which slows its response and can lead to the player becoming no longer being engaged with game.

I expect that some players may only slightly change their preferences or do so very slowly as a sort of refinement as they come to understand the game better. These are likely the majority of cases, and due to their prevalence, they are already handled with experience management systems. Instead, I focus on preference shifts that are

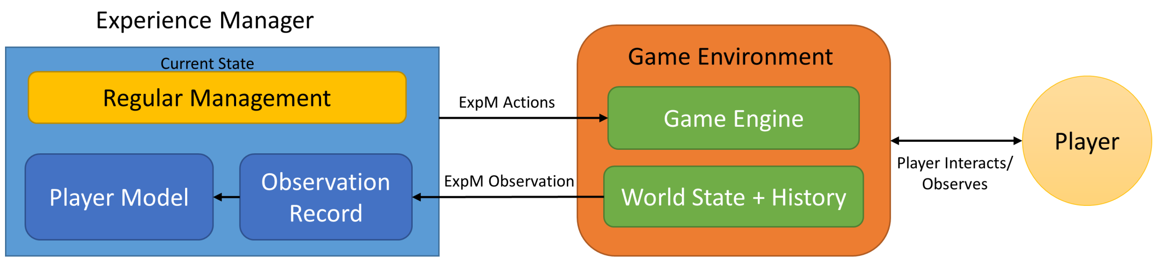


Figure 1.1: The ExpM observes the player's interaction with the environment to inform and incrementally refine the player model. Using this information, it makes changes to the game by adding or removing objects, locking or unlocking paths, or moving things around.

more sudden and may lead to completely different styles of play. While it may seem like an incredulous claim, consider a player who has opted for a very combat-suited style of play. As they continue to fight and engage well with the game in this aspect, they may be slowly growing tired of this play style without noticeably changing their actions in game. What was once considered interesting and fresh may be now tired and samey as they grow accustomed and familiar with the gameplay. They may not want to completely disengage with the game instead wishing for some variety, perhaps wanting to focus on crafting and building a base. This in effect looks like a sudden shift in preferences when in actuality the ExpM and PM are not able to model the player with that level of specificity. In a traditional ExpM that makes the static preference assumption, the ExpM may have removed the player's preferred interaction for the sake of personalizing the game to the player. Fundamentally this is a limitation in the modeling process, ExpMs and PMs do not currently have the capabilities of modeling a human in the same way a human DM can, and even so are largely limited in how they observe the player, generally only having access to the player's actions. With a more sophisticated modeling process where the intent of the player can be read and understood this may not be an issue and but this is currently not done.

To be able to handle a preference shift situation with the current methods of experience management they would need to be able to detect that a preference shift has occurred and correct for it by quickly recovering the player's preferences. To this end I have devised the following three research tasks to tackle these questions in experience managed games:

- RT1: Recognize when a player's preferences have shifted and that the current environment may no longer be well suited.
- RT2: Recover a model of the player's preferences after a player preference shift has been detected.
- RT3: Verify that these methods work in gameplay data from human subjects.

1.1 RT1: Preference Shift Detection

My first limitation is the assumption that player preferences are static. In removing this assumption, I need to handle the possible scenarios that are presented, namely detecting that a player's preferences no longer match the player model. If the environment is equally balanced to any possible play style, this would not present a problem as the player would seek out their preferred experience themselves, but in an experience managed game this may not be possible. If the environment is biased towards a player's previous preferences, then the player cannot interact in their preferred way. Furthermore, since the player can only interact with the current environment, observations that an ExpM takes will show that the player is still interacting with things in the environment and thus may not be well suited for recovering the player's preferences and adjusting the PM. This feedback loop where the bias of the environment interferes with the accuracy of the ExpM's observations requires a different system to take over.

My solution is to silently and continually test the player. As the player continues to take actions they form a stream of data, one which can be monitored for novel patterns. When the player takes actions they are likely to take ones that are consistent with their already established player model. In the rare case that they shift their preferences, the player is likely to take start taking actions that are no longer consistent. This could be in the form of trying out the other sorts of actions that are available in the environment to see if they are interesting or simply taking less meaningful actions such as wandering around aimlessly. In my research I find that by just watching for this difference in how the player behaves, it is possible to gain some metric of how engaged the player is in the game.

These observations are not completely accurate though, there are unknown factors that may lead to fluctuations in the data, complicating the matter. This variability can be a problem when identifying preference shifts as small deviations in the observations may be misconstrued and act as false positives. For this I take inspiration from novelty detection [32, 23, 31] which attempts to distinguish random fluctuations present in noisy data from true parameter shifts 2.1. I formulate the PM as a distribution over possible actions and by using novelty detection find when a departure has been made from the previously known distribution to a new one, irrespective of what the new distribution has become. In addition I create a different method which is inspired by the pattern that a preference shift may take.

These methods are tested with artificial agents that are meant to mimic certain aspects of how humans behave, and are deployed in a text-based game environment written in Inform7. I find that they work with mixed success with my own method working the best overall. All of these methods, mine included, are capable of recognizing when a shift has occurred but all but my own tend to struggle when there is no shift.

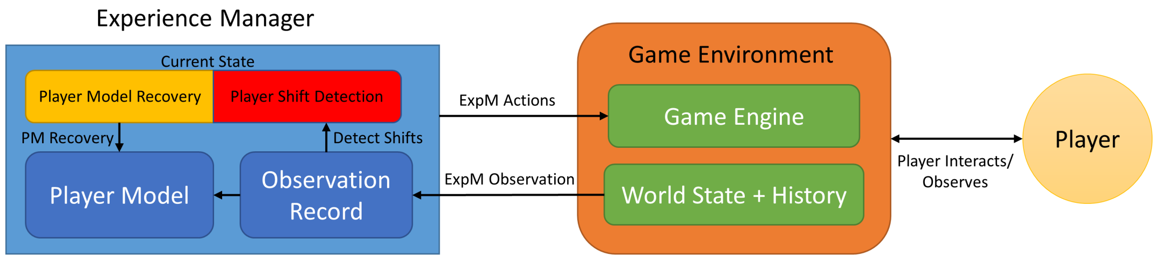


Figure 1.2: After adding my system, if the ExpM detects that the player's preferences have shifted it will switch to the Player Model Recovery state. In this state it can test the player via ExpM distraction actions and quickly update the player model. Once the player model is recovered it will switch back to the Player Model Shift Detection state, where it will continue to watch for preference shifts.

1.2 RT2: Player Model Recovery

Given that a player shift is detected I still need to find what the player's new set of preferences are to recover the PM. While this detection is useful, there is still the issue of environmental bias as preference shift detection makes no modifications to the environment, opting only to watch the player's behavior. Removing some of the elements from the environment may help balance the environment, but these elements may have significance for the narrative and this does not help if any aspects are completely missing, thus is not viable. Instead I opt for the alternative, adding new elements to the environment in the form of *distractions*.

Distractions are a new type of game object, which serve to test the player on their preferences. These distractions will be introduced into the environment with the purpose of trying to *distract* the player from their current task to gain information about their preferences and to help balance the environmental bias towards certain actions. Players whose preferences are adequately provided for will ideally not be drawn to interact with a distraction, but players with preferences that do not match the environment should seek them out as they match their preferred means to interact with the game. Being in complete control of the ExpM I do need to be careful about how to design these distractions as they have the potential to cause harm to both the player experience and the author's intent for the game. Thus I place restrictions on their design: distractions need to be largely irrelevant to important parts of the game like quests as to not tread on any authorial goals and distractions need to be used minimally as to not overwhelm the player with a sudden deluge of options to interact with. While there are other limitations in designing distractions for use with human subjects that will be discussed later, these are the most important and what influences the design of the player model recovery system.

Since player model recovery is a different task, we need to switch the internal state of the ExpM to be focused on solely recovering the PM, as shown in Figure 1.2. This PM recovery state will no longer try to detect that the player's preferences have shifted nor attempt to change the environment to suit the current PM, opting to instead throw out its previous notions on the player. In this state we need to be able to reduce the bias that is present in the environment and to allow the ExpM to take action. This is done through a series of ExpM actions that dynamically add and remove distractions from the environment as the ExpM deems necessary. By observing how the player interacts with the environment after the ExpM has added distractions, the player model can be recovered.

To do this quickly and with minimal interruptions I have found that I can represent this problem as a Multi-Armed Bandit (MAB). Multi-Armed Bandits (MABs) are a class of sequential decision-making problems where an agent is tasked with maximizing the amount of reward from iteratively taking one of k actions (often called arms) [44]. Algorithms for solving these problems are well researched and have been found to be good at balancing between gaining information (exploring) and making use of this gained information (exploiting). Since a player will shift to an unknown set of preferences, it is necessary to explore to gather this data, but I also want to minimize the impact on the player and thus recover the PM as quickly as is possible.

By modeling this system as a MAB and testing with artificial agents in a text based game environment much like RT1, I find that there is some limited success in their ability to find the preferences of the player and recover the player model. To combat this I loosen the restrictions due to MABs by using Combinatorial Multi-Armed Bandits (CMABs), a combinatorial extension on MABs, which allows for significantly better performance when using the same agents and environment. With these systems developed and shown to work with artificial agents, the only thing left is to test them with human subjects.

1.3 RT3: Human Study Verification

My third research task is to verify the validity of both Preference Shift Detection and Player Model Recovery with human data. It is difficult to develop these methods with human data as they often require continuous testing, thus they were developed with artificial agents that mimic various aspects of human-like behavior. These artificial agents are significantly simpler in their behaviors than humans generally are, and thus begs the question as to whether they are valid representations of humans for these tasks. To verify this, I collected data of human subjects as they interacted with a modified version of the environment and related systems that were created for the previous automated agent tests. As previously stated human subjects are significantly more complex than the automated agents thus simplifications that were appropriate to the automated agents needed to be expanded on and changes were needed to the size, the setting, and the content of the environment as well as the distractions themselves.

Since I am measuring metrics that are related to how engaged a player is in their play, I have had to modify the environment such that it may actually hold their interest, or at least allow them to feign interest. Thus the environment is expanded out to roughly four times the size, the actions available to the player are changed to ones more fitting to a game (as opposed to being inspired by the default set of actions in Inform7), and these actions are grounded with quests for the player to accomplish. Likewise the requirements on distractions are shifted, instead of focusing exclusively on minimizing the effects on players while still serving as useful tools, distractions additionally need to be well suited to the actions that they are representing and recognizable to the player as belonging to that action. This additional requirement allows distractions to accurately communicate the intentions of the player to the ExpM.

I wish to test the validity of both Player Model Recovery and Preference Shift Detection, but only Player Model Recovery needs the ExpM to take active intervention in the game. Preference Shift detection itself is silent to the player and thus can be used on traces of the game instead of requiring to be done online. Thus I have opted to run a single human study, one that includes the active intervention of the ExpM from Player Model Recovery, and then reuse relevant parts of the data to create synthetic data for Preference Shift Detection.

I find that both Player Model Recovery and Preference Shift Detection work with the data gathered from this human study, and that the behaviors that the agents were designed to exhibit are somewhat present within the participants sampled. With Preference Shift Detection I find that the methods work considerably better, being able to detect a true shift 100% of the time with only a 5% false positive rate, but this verification had to make use of synthesized data, making it difficult to make hard conclusions with. Player Model Recovery also works well and seems to indicate that a player model can be recovered in less MAB rounds and less game turns than previously thought.

Chapter 2 Literature Review

2.1 General lit review

2.1.1 Experience Management

The broad goal of experience management is to do more than just entertain (as with a drama manager); it is also to coerce the game so that the player's experience conforms to a set of provided properties [35]. This essentially extends the purpose of interactive experiences in the first place, which is to balance the goals of the author in crafting the experience while simultaneously keeping the player's sense of agency. With an ExpM the major change is that this balance is managed actively while the player is playing, allowing it to have a finer degree of control over the player's experiences.

The ExpM achieves this balance by having its own set of actions that it can take, going through the game alongside the player. These actions can be locking or unlocking paths for the player, adding or removing objects and NPCs, or modifying what information the player has access to and leading them to the author's narrative goals. Early examples of works that became known to be experience management also included changing underlying rules of the game, such as the process of dynamic difficulty adjustment [18]. While holding academic interest [17, 20, 45, 19], experience management (sometimes referred to as AI directors) have had trouble breaking into commercial games. These processes are not cheap to integrate often requiring different methods and paradigms to develop the games that include them, but have seen some success[53, 22]. Popular commercial examples that use some form of experience management include Valve's Left4Dead 1 and 2 [49, 50] which changes the locations and intensity of enemy spawns to follow an author defined intensity level to maximize drama, and Capcom's Resident Evil 4 which silently adjusts the difficulty of enemies and ammunition drops based on the player's performance [5] among others [36].

A notable early academic work is *Façade* [26], where story content was structured into beats. These beats are typically small, changing out every minute or so, and are representative of the smallest unit of dramatic action. These beats are partially ordered, through the use of pre- and post-conditions, and thus allow the player to influence where the story goes through their actions. Their ExpM, called a beat manager, chooses which beats based on several factors including which preconditions have been met (by player actions or past beats), the narratives planned tension, and internal (sometimes conditional) priorities. This allows for a balance between author goals and player agency but does so without forming an explicit model of the player.

Similarly to Façade, other early works in experience management (often under the more narrow term drama management) did not make use of a player model. This includes *planning-based drama management* [34, 6] which allow players to take actions that break the narrative. The game is not interrupted or broken as the DM can repair these breaks by introducing new narrative elements to fill the gaps. *Search based drama management* could leverage a model of the player and use it to search the space of DM actions and likely player actions to deliver a better story [25, 28, 29]. These methods did not try to model the player as they played but instead reacted to the player as they played.

My 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 personalized ExpM actions [42, 43, 51]. A player model can model any aspect of the player, be it of the player's preferences [47] (fighter, power gamer, method actor, etc.) or the player character [13] (a player's heroism, self-interest, cowardice, etc.). These have more recently been expanded into multiplayer environments [16, 52]. In each of these works, player preferences are assumed to be static. In my works, I explore the methods which would allow for an ExpM to recalculate a player model in response to shifts in player preferences.

2.2 RT1 - Player Model Shift Detection

To detect shifts in player preferences I split the ExpM observations into two different distinct parts: observations that are likely to be before a shift, and unclassified observations. This aligns well with the idea of novelty detection, or the task of recognizing and differentiating between data that is similar to what has been seen in the past and data that differs [32]. This is similar to outlier detection, where data needs to be classified as an outlier or not, but novelty detection requires that past data is free from outliers. Both outlier and novelty detection present themselves as one-class classification problems [27], where normal data (i.e. data similar to that used in training) is distinguished from the outlier data. This is different than a two-class classification problem as it does not make the assumption that there are two distinct classes of data, instead it only identifies a single normal class which must be distinguished from all other possibilities.

Novelty detection also takes a step further by distinguishing the data over time. Whereas outlier detection can be applied to a static dataset, novelty detection is focused on separating the data into previously known normal data and new unknown data. This distinction means that novelty detection often has a significant imbalance in training examples, with only normal data being well represented. This is useful when training machine learning models to identify whether or not a given input is likely to be well handled by the model, as a model is more likely to give a correct output when its input is similar to the data used to train it. This also makes it well suited for my task of finding player preference shifts, as my data has a time component and thus I only have access to known normal data and no known abnormal data.

I make use of three different novelty and outlier detection methods: Elliptic Envelope [37], Local Outlier Factor [4], and One Class SVMs [40]. One Class SVMs function by finding an optimal separating hyperplane between the classes by taking into account the boundary points. To better separate the data a kernel function is applied, for my tests with automated agents I use a degree 4 polynomial kernel though for my human subjects experiments find that a sigmoid kernel is more optimal. Local Outlier Factor method on the other hand calculates to what degree each point in the space is an outlier. Each point is considered local as it takes into account

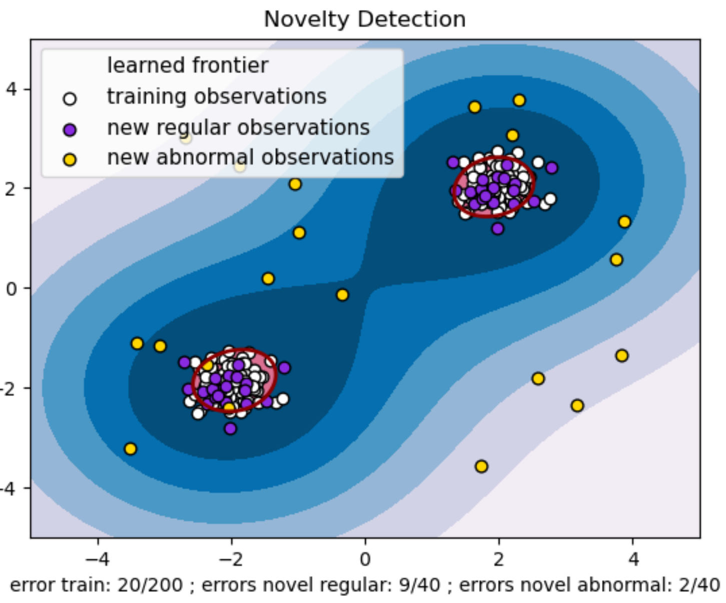


Figure 2.1: Novelty detection is tasked with finding abnormal new observations given a set of normal training observations. Pictured here is the decision boundary of a One-class SVM with non-linear kernel.

its surrounding neighborhood as opposed to the data as a whole. Elliptic Envelope works by defining an elliptic boundary separating outliers from normal data and is designed for Gaussian distributed data. Both Local Outlier Factor and Elliptic Envelope methods are generally used for outlier detection, but since my data already has a distinction between known normal and unknown data, I treat them all as novelty detection methods.

2.3 RT2 - Player Model Recovery

2.3.1 Dynamic Player Modeling

There have been works recently that have acknowledged the need for dynamic player models like in Valls-Vargas et al.'s work on *Exploring Player Trace Segmentation for Dynamic Play Style Prediction* [48] and an extension by Khoshkangini et al. [21]. These works explore the problem of play-style shifts by predicting the play style from previously collected gameplay traces. This work focuses on recovering the play style but does so in traces, where they cannot make changes to the game as the player plays. This does not allow them to directly test the player only observe what their actions are. They found that trying to predict play style information on a too fine (minute-by-minute) level does not provide enough information to give accurate results. My work differs in that it can directly introduce new elements into the environment to test the player, which should allow me to use a much finer level. Since my method is incorporated into an experience managed environment, I also need to deal with the left-over biasing of the environment from past ExpM actions.

I have also found that MABs have been used in player modeling before with the focus of adapting games to their players [15, 16], 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 my system is attempting to solve I cannot make these assumptions about a player's play style and must actively intervene to discover them.

2.3.2 Multi-Armed Bandit Overview

Multi-armed bandits (MABs) are single state Markov Decision Processes where agents choose to take one of several actions (called arms) [24, 44]. Each action has a reward distribution and the task of solving comes in the form of finding an optimal policy that maximizes the total reward while minimizing losses. This policy needs to be able to balance between exploring each of the arms while gaining information on their underlying reward distribution and exploiting this information to maximize the reward from all the arms.

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 [44]. I 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.

More formally defined, MAB problems consists of a set of m arms, each associated with a random variable $X_{i,t}$ for $1 \leq i \leq m$ and $t \geq 1$, which denotes the outcome for arm i at round t. The set of random variables $\{X_{i,t} | t \geq 1\}$ associated with arm i are i.i.d. with some unknown expected mean value μ_i. The vector $\boldsymbol{\mu} = \{\mu_1, \mu_2, \ldots, \mu_m\}$ represents the expected mean of all arms. In each round t, a single arm i_t is chosen to be played, and its outcome $X_{i_t,t}$ is observed. At the end of a round the reward is revealed and associated with the arm. The goal is to maximize the total reward over a sequence of rounds by selecting arms in such a way that the cumulative expected reward is maximized.

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 Chapter 4, I make use of three different algorithms for my 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 greedy actions take advantage of this information. There are many variants on ϵ-greedy, like ϵ-decreasing in which the ϵ value decreases over time according to a decay function. Many of these were tested but found to perform worse in preliminary tests and were not included.

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 [2], which has the effect of selecting high-variance options more often. 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.

$$\arg\max\left(Q_a+\sqrt{\frac{\ln t}{t_a}min(1/4,V_a(t_a))}\right) \tag{2.1}$$

$$V_a(t)=\left(\frac{1}{t}\sum_{t=1}^{t}R_{a,t}^2\right)-\bar{R}_{a,t}^2+\sqrt{\frac{2\ln T}{t}} \tag{2.2}$$

UCB1-Tuned calculates the upper confidence interval for every arm using equations 2.1, 2.2 where Q_a is the sample mean for arm a, t_a is the number of times arm a is given, $R_{a,t}$ is the reward of arm a at time t, $\bar{R}_{a,t}$ is the average reward, and T 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.

$$\arg\max\left(B\left(\alpha_a+1,\beta_a+1\right)\right) \tag{2.3}$$

Thompson sampling is the first bandit algorithm in literature first appearing in 1933 [46]. 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 2.3, 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 from primarily exploration early on to being more exploitation as they gain more information.

2.3.3 Combinatorial Multi Armed Bandit Overview

Combinatorial Multi-Armed Bandits (CMABs) are an extension on MABs [8], and here I will define them in a manner mirroring the MAB overview in the previous subsection. Similarly to MABs, a CMAB contains a set of m base arms that are played over some number of rounds. Each arm is associated with a random variable $X_{i,t}$ for $1 \leq i \leq m$ and $t \geq 1$ which denotes the outcome for arm i at round t. The set of random variables $\{X_{i,t} | t \geq 1\}$ associated with base arm i are i.i.d. with some unknown expected mean value μ_i. $\boldsymbol{\mu} = \{\mu_1, \mu_2, ..., \mu_i\}$ is the expected mean of all arms. Instead of playing a single arm, as one would do with a MAB, a *super arm* S is played from the set of all super arms $\boldsymbol{S}$. I consider $\boldsymbol{S}$ to be the subsets of the set of all arms m of size k, for $k \in \{2, 3\}$. At the end of the round, the reward $R_t(S)$ is revealed for the super arm and is given to the contributing arm $i \in S$. The reward is observed per arm played and only a single arm is rewarded, a type of feedback belonging to *semi-bandits* [1]. $T_t(i)$ is the number of times arm i has been played up to round t.

To learn an optimal policy, I make use of the CUCB algorithm [8], which extends the UCB algorithm described previously. This algorithm first takes an initial m rounds of playing a super arm that contains one of each of the arms. For my implementation of the algorithm, I choose to always play a super arm that contains the least played arms thus far, ensuring that each arm is played k times during this phase. Afterwards, I calculate the adjusted means using the formula $\bar{R}_a + \sqrt{\frac{3 \ln T}{2 t_a}}$, where $\bar{R}_a$ is the sample mean of arm a, t_a is the number of times arm a has been played, and T is the total number of rounds. The super arm with the highest adjusted mean is then selected.

Chapter 3 Preference Shift Detection

3.1 Introduction

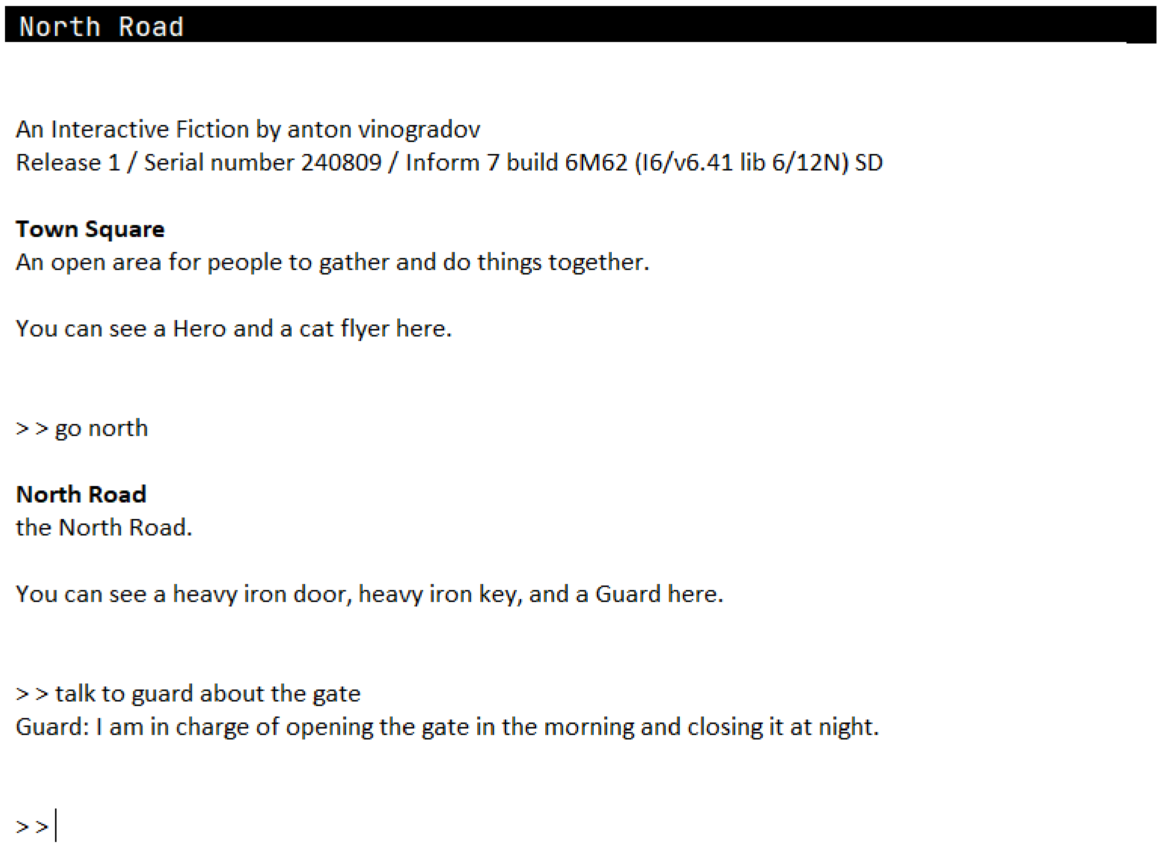
I cannot continuously try to recover the player's preferences without first detecting that they have shifted, as this process would be too disruptive to the player and may affect the intended experience enough to significantly depart from the author's intention. Thus, to start, I have to detect that a player's preference has shifted, and I should do so in a way that is invisible to the player. Since the ExpM is not able to see into the mind of the player it must make do with the information that it does have, namely observing what actions the player takes.

To do this, I treat the player's actions as streaming data, assuming that actions taken in the past correspond to some previous set of preferences. Actions done more recently then can be tested against these and the differences between them can be used to detect whether a preference shift has occurred. This is generally known as novelty detection, which, given some sequential data, is capable of distinguishing incoming observations as either *inliers*, observations similar to data seen before, or *outliers*, observations that have not been seen [32]. Novelty detection is itself a subset of anomaly detection, and some of the methods used to distinguish between non-temporally oriented inliers and outliers can be used for novelty detection. For this chapter, I make use of three existing methods for detecting novel and outlier data: Elliptic Envelope [37], Local Outlier Factor[4], and One Class SVM [40].

The three novelty detection methods are designed to detect if incoming data is similar to the previous data, but with my text based gameplay data it may be too noisy and trigger false positives. There is normally only a small chance to detect a false positive, but since these methods apply on every turn with the data stream this effect is compounded. To combat this I have created a method that is more resistant to this effect based on how preference shifts are likely to present themselves in player actions and test it against the existing methods. My method makes use of more of the data to compare against the initial baseline and only take into account the action types that are likely to belong to the preference shift. I evaluate these methods by examining two key factors: the accuracy of detecting whether a shift has or has not happened and how soon a shift is detected. I find that the custom method is more effective in detecting shifts and significantly more effective at avoiding false positives. It is also quicker to detect shifts compared to existing novelty and outlier detection methods.

3.2 Methods

In this section I will discuss the various parts of the player shift detection system. This will include information about the environment in how I define my player model, and the preference shift detection methods that I use.


```
North Road

An Interactive Fiction by anton vinogradov
Release 1 / Serial number 240809 / Inform 7 build 6M62 (I6/v6.41 lib 6/12N) SD

Town Square
An open area for people to gather and do things together.

You can see a Hero and a cat flyer here.


>> go north

North Road
the North Road.

You can see a heavy iron door, heavy iron key, and a Guard here.


>> talk to guard about the gate
Guard: I am in charge of opening the gate in the morning and closing it at night.


>>|
```

Figure 3.1: A screenshot of the interface for a text based adventure game. User input follows a >>, and the text description of the outcome of that action follows. The location of the player is shown in the header at the top.

3.2.1 Environment and Player Model

I make use of a custom environment that will be reused for other sections. This environment is built with Inform7 [14], a language designed to create text-based interactive fiction games for the Z-code or Glulx virtual machines. While not an entirely accurate description, it can be thought of as a text-based game engine. Inform is well suited for this project as it interfaces well with artificial agents. Games written using Inform are played in rounds, called turns, that alternate between user input and the outcome of that input. This is demonstrated in Figure 3.1 which shows the player going north, with the room change description being printed afterwards, then talking to a guard about the gate. Since the action space is discrete it is possible to easily enumerate all actions that can be taken in the environment if desired. Games made using Inform are also typically played with natural language commands, using simple sentences or fragments such as "pick up shovel," making them suitable for humans, even those who are not well versed in traditional video games.

In this environment, I use ten areas called rooms (Figure 3.2), which are used

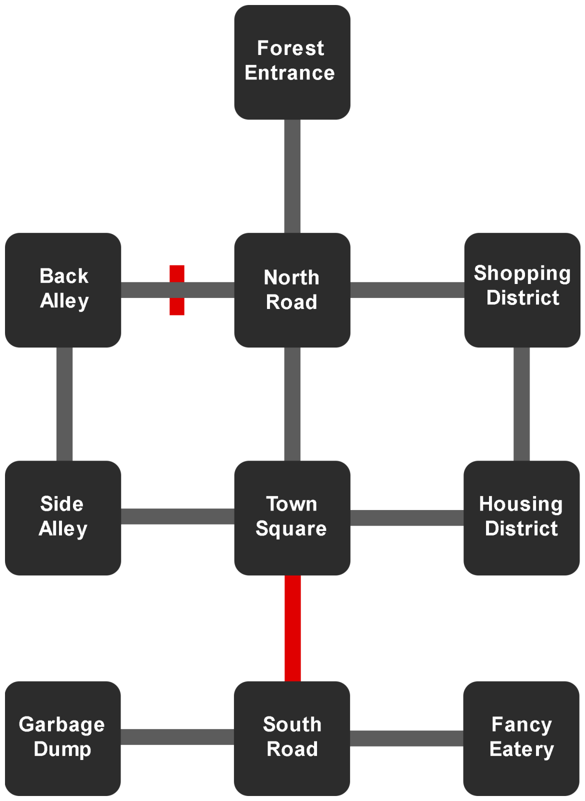
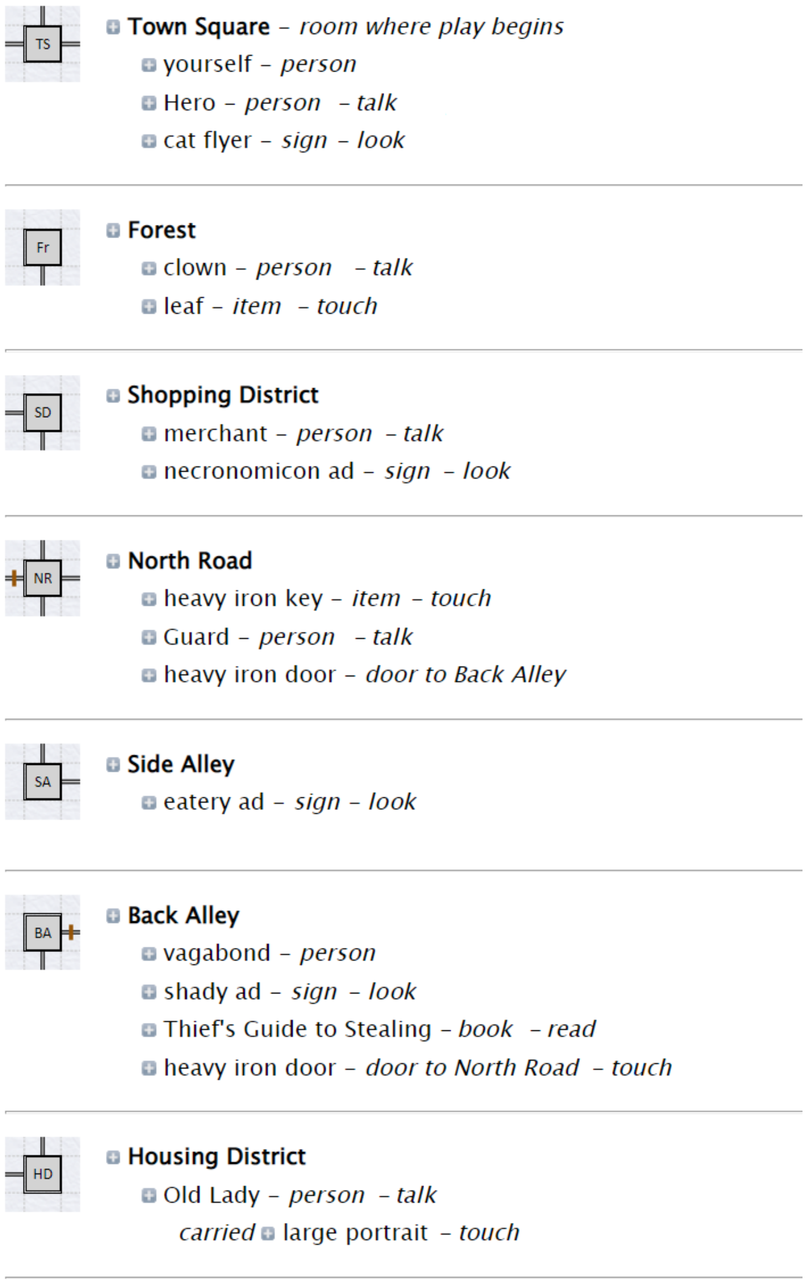


Figure 3.2: A map showing the layout of room for the environment for the artificial agents.

to represent a standard fantasy medieval town, though to simulate the effects of the ExpM biasing the environment, I only make use of seven of these. In other words, this environment assumes that the current player is unlikely to want to visit three of the available ten rooms in the game. These seven rooms can be traversed using only the commands for the cardinal directions: north, south, east, and west. Other directions are possible within Inform, including inter-cardinal directions such as north-east, as well as up, down, and even going inside or outside nested rooms. While possible, we do not use these more complex directions for navigation in order to simplify the number of actions considered by the artificial agents used to test the methods.

Within these rooms are a number of objects; for example, the *North Gate* room contains a guard and a locked path to a different room, among other things. A full list of these objects and their locations can be found in Figure 3.3. These objects all have a property called the *action type*, which represents what I think is the primary means of interacting with that object. The action type for the guard is *talk*, and in the quest, I expect the player to ask the guard for information in their search for the missing cat. Likewise, the action type for the locked path is *touch* as to go through it, the player is required to interact with it by touch. There are five of these action types: *look*, *talk*, *touch*, *read*, and *eat*. These action types are inspired by the actions built into Inform7 and are found in many other games.



Town Square – *room where play begins*
- yourself – *person*
- Hero – *person* – *talk*
- cat flyer – *sign* – *look*

Forest
- clown – *person* – *talk*
- leaf – *item* – *touch*

Shopping District
- merchant – *person* – *talk*
- necronomicon ad – *sign* – *look*

North Road
- heavy iron key – *item* – *touch*
- Guard – *person* – *talk*
- heavy iron door – *door to Back Alley*

Side Alley
- eatery ad – *sign* – *look*

Back Alley
- vagabond – *person*
- shady ad – *sign* – *look*
- Thief's Guide to Stealing – *book* – *read*
- heavy iron door – *door to North Road* – *touch*

Housing District
- Old Lady – *person* – *talk*
  - *carried* large portrait – *touch*

Figure 3.3: A list of objects (referred to as *things* in Inform7 parlance) that exist in the environment and their action types, organized by room.

Figure 3.4: The verb coin from *Monkey Island 3: Curse of Monkey Island*, which contains a simplified set of verbs for touching, looking, and talking, reduced from 12 and 9 verbs for the first and second game in the series, respectively.

Since I am focused on detecting shifts in an experience managed environment, I simulate the effects that the ExpM might have on the environment. These effects come in the form of biasing the environment so that only certain actions are possible. In my medieval town, I consider the *talk* action to be the primary action type, as the quest is centered around doing actions of this type. I also include the *look* and *touch* action types in the environment, but these are not the focus of the quest and thus are less present. The *read* and *eat* action types are largely missing from the environment and are largely limited to three rooms locked off from the player that are not included in the environment, though there is a single book to read in the back alley. The *look*, *talk*, and *touch* action types together are considered Environmental action types since they appear in the environment, while *read* and *eat* are considered *Missing* action types since they are missing from the environment. These categorizations of actions as Environmental or Missing is important for how I group together preference shift scenarios later.

These five action types are also the basis for my player model, which measures the probability that the player will perform a particular action type. This is a common way of representing the player model that is found elsewhere in the literature [47, 43], and thus, I have chosen to adopt it. I expect that even when a player prefers only a single type of action, they will not be able to complete a quest without doing at least some others, simply because quests are often designed in this way. Thus, when using this type of player model, the probability of doing any specific action type is never set to zero. Since I am focused on detecting a player preference shift, I do not calculate a player model for my agents and instead assume that the system would have an accurate measure of the player model beforehand.

3.2.2 Shift Detection

To utilize shift detection methods, I interpret a player's actions as streaming data. This means that the data is treated as continuous and observed at somewhat regular intervals. By treating the data like this, I can use these methods to identify shifts in player action observations in real time while a player is playing. The observations that the ExpM makes are based on the action type that the player took during the last turn, represented as a one-hot vector. However, I apply a 10-turn rolling average to these vectors to make the data more suitable for the methods I test. In addition to the five action types I have described previously, a sixth action type is included: the move action type. This action type is not present in the model and objects cannot have it as a primary means of interaction with the entire environment. Moving around the environment is a necessary action to play the game and while a player can prefer to explore, it is not one of the preferences that the ExpM can use to optimize the player's experience in this text based game. For my shift detection methods, I hold on to a sliding window of data rather than considering a single observation at a time, and my operations will be applied to this buffer. In other words, my shift detection methods will apply to a set of previous actions rather than to the most recent action.

Since I represent my observations as streaming data, I have taken inspiration from novelty detection for this task. Novelty detection is a type of anomaly detection that attempts to classify whether incoming data is an outlier. In this context, these outliers are called novel data. It is well suited for streaming data as it makes a distinction between data used to train the classifier, which it assumes to be uncontaminated, and incoming data, which may be contaminated with outliers. This assumption is necessary for novelty detection, but I also consider it safe as the ExpM already makes it. Since the ExpM makes changes to the environment that bias it towards a player model, it already makes the assumption that its previously held player model was correct. Any observations generated to create the player model must already be uncontaminated, and thus, I can assume that before a preference shift has happened, the player should be taking actions in accordance with their player model. This assumption is broken when a preference shift happens, but the purpose of the shift detection process is to detect a shift in the preferences before the observations of the preference shift are used as training data for the classifiers.

I explore the use of three existing classifiers for shift detection: Elliptic Envelope [37], Local Outlier Factor [4], and One Class SVMs [40]. Elliptic envelope makes the assumption that the data is normally distributed, then attempts to learn an ellipse an ellipse as a boundary between inliers and outliers. Local Outlier Factor detects anomalies by computing the local density deviation of a given data point respective of its neighbors. A point is then considered an outlier when it had substantially lower density than its neighboring points. One Class SVMs learn a optimal hyperplane to separate the data much like SVMs, but with only a single class tries to separate the known data from the origin instead of two distinct classes. Points that like below this hyperplane and are closer to the origin are considered novel. Elliptic Envelope and Local Outlier Factor are considered outlier detection methods and thus do not necessarily require that the training data is uncontaminated, while a One Class SVM is considered a novelty detection method and therefore does.

I use a gap between the training and testing data to ensure that the training data is uncontaminated by keeping it further from the testing data. Each of these methods was given a 50-turn window to use as training data, followed by a 50-turn gap and, finally, a variable-size testing window ranging from 1 to 20 turns. The training and gap windows were chosen to be 50 turns each to be sufficiently larger as to account for the agents interacting in multiple rooms, while the test window was chosen to be

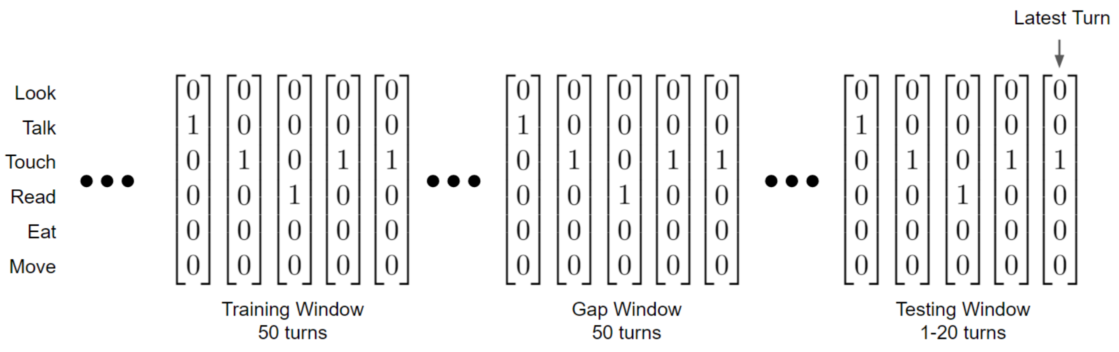


Figure 3.5: An illustration of how the data is formatted and organized before it is processed and utilized by the shift detection methods.

smaller as the larger test window size is the longer it would take to fill it and thus delay any shift detection by that amount of turns. For the testing window, I also introduced a threshold parameter that governs what percentage of the turns in the testing window need to be considered outliers before I consider a shift to be detected. This is necessary to account for any potential noise in the data, which may lead to an anomalous detection of a shift without a preference shift ever happening.

In preliminary tests, I found that novelty detection techniques were sensitive to the data and thus encountered challenges associated with excessive false positives or false negatives. The existing methods have the advantage of being well suited for the general task of detecting novel data, but my task is focused on finding a specific type of shift in the data produced by a player shifting their preferences. I reasoned that I could take inspiration from how this shift might look and create my own method with the game environment in mind. I expect that a shift in player preference likely consists of a sharp difference in their actions over some period of time and then staying consistent in those new actions. This expectation comes from the nature of a preference shift, if a player suddenly becomes disinterested in what they were previously doing they are likely to stop doing that action. A player that no longer wants to socialize with NPCs would be unlikely to talk to more NPCs on their way to their preferred action. Likewise the sharp difference is expected regardless of if the player finds what they prefer. If they do not, the player will likely continue to search, and this would be likely observed as taking a wider variety of actions, but consistently over the next turns. If they do, the player will engage with that action and likely consistently switch to it.

Thus, my method takes 50 turns as the baseline window (P_1) and compares that to another 50-turn comparison window (P_2) to see if the player's actions differ, ignoring the move action type. These window sizes were chosen to match the test and gap windows for the novelty detection techniques. This creates a mask where 0 represents no increase, and 1 represents an increase for a given action type. I amend this mask by setting the highest known preferred action type (according to the player model) to be 0, regardless of its previous value. I do this to ensure that the method does not detect that the player has shifted their preference to an action type that they already prefer. This mask is then multiplied by the averaged frequency of action types in the test window (P_3) to get a shift score for each action type, which is then compared to the threshold (equation 3.1). If the shift score for any action type is greater than the threshold, I consider a shift to be detected.

$$ShiftScore = \left(\sum P_1 > \sum P_2\right) * \overline{P_3} \tag{3.1}$$

I fine-tuned the test window size and threshold parameters for each of these methods and any other available parameters for the existing methods. To score the fine-tuned parameters, I calculated the f1 score for a single trial and picked the parameters that resulted in the highest score. I found that my method works best with a test window of 10 and a threshold of 40%. For the Elliptic Envelope method, I tuned the test window to 10, the threshold to 90%, and the support fraction to 0.6. For the Local Outlier Factor method, I tuned the test window to 10, the threshold to 20%, and the number of neighbors to 30. For One-Class SVMs, I tuned the test window to 13 and the threshold to 90%. I also found that the degree 4 polynomial kernel works best.

3.3 Experiments

In my experiments, I use 3 artificial agents and test 45 scenarios, which I group into 10 categories to examine their differences. The 45 scenarios are: 20 fast preference shifts from primarily one action type to another, 20 slow shifts matching the fast ones, and 5 no shift scenarios with one for each action type. I measure two different criteria: the percentage of the time that a shift was correctly detected (Table 3.1) and the turn count when a shift is detected (Table 3.2). The first criteria help me find the method that detects a shift the quickest, while the second allows me to measure its accuracy. Note that for detection, I report both detecting a shift when a shift has occurred (true positive rate) and detecting no shift when no shift has occurred (true negative rate) together.

3.3.1 Agents

For my experiments here and in Chapter 4, I will be making use of artificial agents in place of human subjects. These artificial agents provide much more flexibility and rapid iteration and allow for these methods to be designed quickly. I make use of three agents that all work on a similar basis, having a set of internal preferences that dictate the probability that they take certain types of actions. Where they differ is how they utilize their internal preference distribution. Some first select a type of action before trying to find a matching object with that type, and others decide first if they wish to interact with objects in the room before using the distribution to select which object is available. In some rooms, there will be no objects to interact with, and the agent will default to moving to a different room, though the details of how are specific to each agent.

I use three agents: the goal-focused agent, the exploration-focused agent, and the novelty-focused agent. I chose to use three different agents because I believe it is unlikely that a single agent can accurately model the complexities of human behavior, and with three, I can represent a wider range of player behaviors. Two of these agents, the Goal and Exploration-focused agents, come from the Bartle Taxonomy of player types [3], which details 4 different player type quadrants: Explorers, Achievers, Socializers, and Killers. Of these four player types, I only try to model Explorers and Achievers (corresponding to Exploration and Goal focused agents, respectively) as these types are focused on how the player interacts with the world on the players-world axis. Since I am using a single player environment, I cannot model the Killers and Socializers, so they are left unrepresented. The third artificial agent is a departure from this taxonomy and is instead based on findings in the literature that state that novelty is a key component in user engagement [30]. These three agents allow me to better model a wider variety of how players act in a game but do not cover all possible players and may have overlap between them.

Goal Focused Agent: I will first talk about the Goal focused agent, the achiever, whose algorithm is detailed in Algorithm 1. This agent first selects an action type that it wishes to interact with (line 2), sampling from its internal preference distribution, then attempts to interact with an object with that action type. If an object with that action type is not present within the current room, the agent defaults to its fallback behavior shown in Algorithm 2. This fallback behavior attempts to act either by taking an action towards the goal or by moving to a neighboring room, each with equal probability. When it chooses to take action toward the goal, it references an ordered set of steps required to complete the quest. As it continues to complete the quest, these steps are removed, though each step often requires the agent to be in a specific room first. If the agent is not in the correct room it will first find a valid path to that room and take a single move action towards. This list of steps to complete the quest and paths to the correct room is maintained by Inform7 and is represented on line 4 as taking from a list, though Inform7 only refers returns the next step.

The effect of this behavior is that when the agent is in a room with objects that it prefers, it will tend to stay there, with only a rare chance of choosing an action that is not present in the room. On the other hand, if the agent is not in a room with objects it prefers, or if there are no objects that it prefers in the environment, it will tend to wander toward the next step in the quest, where it will tend to have access to objects of environmental action types. In these cases, the agent is also likely to take the environmental type actions that the quest is centered around, even when it does not prefer to do that action.

This agent is considered goal focused because it has a higher likelihood of taking actions that contribute towards the goal, especially when it does not have access to the types of actions that it prefers. I expect that players that are exhibiting this kind of behavior would normally also attempt to complete the goal when they have access to their preferred types of actions, but this is not distinguishable with the agents thus I have chosen not model it for simplicity. A goal focused player would also likely fall

Algorithm 1 Select Action for Goal Focused Agent

```
1: procedure GOAL.SELECTACTION(preference)
2:     $action\_type \leftarrow$ random_choice_with_weights($preference$)
3:     $objects \leftarrow$ environment.get_objects_of_type($action\_type$)
4:     if len($objects$) $> 0$ then
5:         $object \leftarrow$ random_choice($objects$)
6:         return ($action\_type, object$)
7:     else
8:         return goal_or_move()
9:     end if
10: end procedure
```

back to completing goals even if they are not preferable, if only to move to the next goal which may be more to their liking. It is this behavior that I believe is represented in this agent.

Algorithm 2 Goal or Move

```
1: procedure GOAL_OR_MOVE
2:     $random\_number \leftarrow$ random$(0, 1)$
3:     if $random\_number > 0.5$ then
4:         $next\_goal \leftarrow$ environment.next_quest_goal.pop()
5:         return $next\_goal$
6:     else
7:         return random_choice($environment.room.exits$)
8:     end if
9: end procedure
```

Exploration Focused Agent: Next, I will examine the Exploration focused agent, whose algorithm can be seen in Algorithm 3. This agent does not use the quest steps, instead starting with a 10% chance of wandering to an adjacent room (line 3). The other 90% of the time, it will first survey what objects it has access to in the current room (line 5). From these objects, it will always choose to interact with the object with the action type that it has the highest preference for, randomly choosing an object if there are multiple for a given action type. For example, if a room only has objects for *talking* and *eating*, and the agent has a preference of 50% for *talking* and 10% for *eating*, the agent will always choose to talk, though may talk to different NPCs if there are multiple. The environment has been designed so that initially each room has an object to interact with, but due to the actions of the agent this can change as they play, thus if there are no objects in the room it will opt to go to an adjacent room.

This behavior has the effect of this agent wandering randomly, with no regard to the quest or its goal. When the agent finds a room with objects in it, it will tend to interact with those objects, even if it does not prefer those objects. If the room has objects that it does prefer, it will lead to its actions tending to heavily favor its top preference as it will always interact with its most preferred available action type, never interacting with ones that it does not prefer. Even when it does find a room with objects that it prefers, it will still have the same likelihood of moving to a different room, though, causing it to be unlikely to stay in the same room for long. This agent is considered exploration focused because it always has an inherent chance to move away from its current room in a random direction. While human players are unlikely to move completely randomly, with the smaller size of this test environment the likelihood that this agent full explores the map is quite high, where as other agents will only do so after the goal is complete.

Algorithm 3 Select Action for Exploration Focused Agent

```
1: procedure EXPLORATION.SELECTACTION(preference)
2:     if random(0, 1) < 0.1 then
3:         return random_choice($environment.room.exits$)
4:     end if
5:     $action\_types \leftarrow$ environment.get_available_action_types()
6:     if len($action\_types$) = 0 then
7:         return random_choice($environment.room.exits$)
8:     end if
9:     for each action_type $a$ in preference do
10:        if $a \notin action\_types$ then
11:            $preference[a] \leftarrow -\infty$
12:        end if
13:    end for
14:    $best\_action \leftarrow \arg\max(preference)$
15:    $objects \leftarrow$ environment.get_objects_of_type($best\_action$)
16:    $object \leftarrow$ random_choice($objects$)
17:    return ($best\_action, object$)
18: end procedure
```

Novelty Focused Agent: The final type of agent is the Novelty focused agent, the pseudo-code for which can be seen in Algorithm 4. This agent is more complex than its peers in that it remembers which objects it has interacted with and uses that information to alter its likelihood of interacting with them again. First, to ensure it does occasionally move, it has a 10% chance of wandering to an adjacent room, much like the Exploration agent (line 3). Afterward, it calculates a novelty score for each object in the room based on how often and how recently it has interacted with it in the past 50 turns, which can be seen in Algorithm 3.3.1. This is calculated for the object o as $NoveltyScore(o) = \sum_{i=0}^{49}(\frac{1}{i+1} * interacted_{i,o})$, a weighted sum for the past 50 turns where the turn i is only added as when the object was interacted on that turn. I use $\frac{1}{i+1}$ to ensure that turns that happened longer ago have less influence.

The minimum valued novelty score (which corresponds to the most novel object) for each action type is then averaged with the agent's preference for that action type $\frac{1}{novelty\_score+1} + preference$ to form a novelty score for that action type as an *action_novelty*. These are then used as the new probabilities for each action type after normalizing to sum to 1, with a *best_action* type being chosen from the resulting distribution. The object that is acted upon (the *best_object*) is chosen by finding the minimum scoring object with the *best_action* type. The resulting object and action type are used to construct the final command. If it is in a room with no objects, it will opt to act like a goal-focused agent and randomly select between moving to an adjacent room or taking a step toward the goal.

The behavior of this agent is difficult to describe at a glance, but the effect is that the agent seeks to interact with objects with which it has not interacted much before. This means that it may override its own preferences to interact with objects it does not prefer but has seldom interacted with. It also inherits some behavior from the other agents, with the chance to wander before it takes novelty into account like the Exploration focused agent, and potentially taking steps towards completing the goal if it finds itself in a room with no objects like the Goal focused agent. These effects are small, and the agent is unlikely to find itself in an empty room, but do serve to make it less likely to fall into familiar patterns. This agent is considered novelty focused as it attempts to mimic the behavior of seeking out new objects to explore how they function. Objects that are well explored by a player are likely to left alone until they can be used in a new context, which I attempt to model by discounting the novelty score based on how many turns ago that object was interacted with, before falling out of the 50 turn window.

3.3.2 Scenarios

To better test my methods I have made use of several different kinds of preference switch scenarios. Like the agents, these will also be used for Player Model Recovery in Chapter 4 though with some modification. Each of these starts with 100 turns of the game with the agent exhibiting a certain preference before the shift (pre-preference) and 100 turns of the agent exhibiting a different preference after the shift (post-preference). This allows me to make 50 observations by sliding the train, gap, and test windows by one turn until the train window starts to overlap with the post-preference. The pre and post-preferences each selected a single action type and weighted it at 11/15, while the other action types were set to 1/15. These values were selected so as not to set any action type to 0, ensuring that agents will tend to have a chance to choose action types that they do not most prefer, though rarely.

As an example consider the *fast shift talk to touch* scenario with the goal focused agent. In the pre-preference this agent would first randomly select an action type that it wishes to interact with, which would most likely land on talk. If it is in the Town Square then may have 3 actions for talk and 1 for look, and it would randomly choose one from the 3 available talk actions, all of which would be a dialogue option from the Hero NPC who is in that room. After the 100 turns of preferring talk actions the agent is in the post-preference or touch and it may find itself in this room again. In this case it is still likely to randomly select the action types that it wishes to interact with, but this time landing on touch. Since there are a lack of things to touch the

Algorithm 4 Select Action for Novelty Focused Agent

```
1: procedure NOVELTY.SELECTACTION(objects, preference)
2:     if random(0, 1) < 0.1 then
3:         return random_choice($environment.room.exits$)
4:     end if
5:     Initialize $novelty\_scores \leftarrow \{\}$
6:     for each object $o$ in objects do
7:         $novelty\_scores[o] \leftarrow$ NoveltyScore($o$)
8:     end for
9:     Initialize $min\_novelty \leftarrow \{\text{look} : \infty, \text{talk} : \infty, \ldots, \text{eat} : \infty\}$
10:    for each object $o$ in room_objects do
11:        $action\_type \leftarrow o.action\_type$
12:        if $novelty\_scores[o] < min\_novelty[action\_type]$ then
13:            $min\_novelty[action\_type] \leftarrow novelty\_scores[o]$
14:        end if
15:    end for
16:    Initialize $action\_novelty \leftarrow \{\}$
17:    for each action_type $a$ do
18:        $novelty\_score \leftarrow min\_novelty[a] + 1$
19:        $action\_novelty[a] \leftarrow novelty\_score^{-1}$
20:        $action\_novelty[a] \leftarrow \frac{action\_novelty[a]+preference[a]}{2}$
21:    end for
22:    $best\_action \leftarrow \arg\max(action\_novelty)$
23:    $best\_object \leftarrow$ min_novelty_object($best\_action, novelty\_scores, objects$)
24:    if $best\_object$ is not found then
25:        return goal_or_move()
26:    else
27:        return ($best\_action, best\_object$)
28:    end if
29: end procedure
```

agent will take a step closer to the goal, which if it is at the beginning of the quest would correspond to going east to the Housing District to talk to the Old Lady about rescuing her cat.

I tested three different rates at which a preference switch happened: an instant switch where the pre-preference switches to the post-preference at turn 100, a slow shift where the preference is linearly interpolated between the pre-preference and post-preference over the course of 20 turns centered on turn 100, and no shift where the preference is not changed. I chose to test both fast and slow shifts to better represent the variability of how a person may play the game. The *slow shift talk to touch* scenario would be represented in the previous example as smoothly transitioning from 11/15 preference for talk actions to 11/15 preference for touch actions. The other 3 action type preferences would not change as they are both 1/15 before and after. The no shift case was included to measure how likely my system is to detect

Algorithm 5 Calculate Novelty Score for an Object

```
1: procedure NoveltyScore(object)
2:     Initialize $score \leftarrow 0$
3:     for each turn $i$ from 0 to 49 do
4:         if object was interacted with at turn $i$ then
5:             $score \leftarrow score + \frac{1}{i+1}$
6:         end if
7:     end for
8:     return $score$
9: end procedure
```

a shift when there is none; thus, it represents the true negative rate. In the example this would be represented as no change in the preference, with the preference staying at 11/15 for talk for a *no shift talk* scenario The fast and slow shifts have 20 different scenarios, and the no shift only has 5, for a total of 45 different scenarios.

These scenarios were then categorized into four groups: *Environment to Environment*, *Missing to Environment*, *Environment to Missing*, and *Missing to Missing*. These correspond to how I group the action types into either *Environment* or *Missing*, where *look, talk,* and *touch* are considered Environment action types as they are present in the environment, and *read* and *eat* are considered Missing action types since they are missing from the environment. Some of these scenarios are unlikely to occur when a human plays a game, for example switching from *Missing to Environment*, but these are included anyway to get a complete picture of how my system behaves in all circumstances. These are considered unlikely as a player that starts with a preference for a missing action type would most likely be already detected by such a system in the past. This totals four grouped fast shifts, four grouped slow shifts, and two grouped no shifts (only Environment to Environment and Missing to Missing apply to no shift) for ten total groups.

3.4 Results and Discussion

For my experiments, I ran 100 trials for each scenario, agent, and shift speed combination. These were then run through each method to generate the two metrics: if a shift was detected and what turn that shift was detected on. The average number of turns to detect a shift is reported in Table 3.2, with the best in each category highlighted. Likewise, the average likelihood that a shift is detected is shown in Table 3.1, along with the F_1-score to compare the overall performance of each method.

3.4.1 Scenario Differences

While I have included an exhaustive set of preference shifts, as stated earlier, I do not expect that all of these are equally likely to occur in actual gameplay. In particular, are the *Missing to-* and the *Missing No Shift* scenarios, which represent a state where the environment is already ill-suited for the player's preferences, which I consider to be unlikely to find a player in. Despite this, I have included them to evaluate how the system can handle these rare circumstances, and I have found that, in some cases, it can.

Due to how I have set up the environment, the differences between the two missing action types are largely indistinguishable by the shift detection methods, and this is reflected in the similarity of results in *Missing No Shift* and *Missing to Missing* scenario groups. These two groups largely mirror each other, with the accuracy of detecting a shift in *Missing to Missing* being the inverse of detecting no shift in *Missing No Shift*. It is unlikely that these can be truly distinguished without getting more information from the player, but since this is unlikely, this is not a big concern.

For the *Missing to Environment* scenario group, I found that each method performs similarly to the *Environment to Environment* group. The observations made when the player prefers a missing action type are often more random, and when used as training data this more uniform distribution of actions taken is easy to distinguish for the novelty and outlier methods. In my method, this would show up as a change in the frequency that a specific action is taken between the baseline and comparison windows, as the action type for the post-preference is available in the environment. This contrasts to the *Environment to Missing* group, where my method would not see such a rise as there are no actions with the type that the player prefers in their post-preference. While this is a weakness of my method, I still find that it outperforms the others.

I found little difference between the fast and slow groups, with the slow shift being only marginally more difficult to detect. The gap compensates for the shift speed, and it seems that having a gap larger than the time it takes for a preference to shift removes the effect of a slower shift. Since the gap does not seem to have a significant effect on the performance, it may be more useful to eliminate the sliding training window in favor of using some set of the past data that is known to adhere to the current player model. The sliding training window requires constant retraining of all the methods except my own, which may be prohibitively expensive to run in real time, depending on the nature of the game that this system is implemented in. My method does not have this same drawback, but it may benefit from a smaller comparison window, which would decrease the number of turns needed to detect a shift.

3.4.2 Agent and Method Differences

I found that different agents pose different challenges to different methods. For the *-to Environment* groups, the Goal focused agent is the the most predictable for each method leading to better performance for those methods, but this reverses in the *-to Missing* groups. The Goal focused agent's behavior causes it to tend to complete the goal when it does not have its preferred action type. Since the quest's goal requires a specific type of environmental action, a shift in preference will likely cause this agent to change its behavior, which is noticeable to the novelty and outlier detection methods. Since my method works differently and only looks at the frequency of actions of each type, the frequency of environmental action types may not change significantly enough to trigger a detection. This suggests that a combination of my method and existing novelty detection methods may be necessary to cover all types of players.

The Exploration and Novelty focused agents are less likely to follow the quest, with the Exploration focused agent only doing so accidentally and the Novelty focused agent only doing so purposefully when it does not have any objects to interact with. The likelihood that the agent follows the quest seems to be the main contributing factor to how effective each method is detecting shifts in each agent rather than their complexity. Detecting whether the player is engaging with the environment because they are interested in that environment instead of just finishing the quest solely for the sake of completion may be useful to improve the ability to detect a preference shift.

I have included two versions of my method for comparison: one with a test window of one turn and another of ten turns. While other methods require a larger test window as they only categorize each observation in the test data as an outlier or not, my method outputs a score that I can compare against the threshold. I found that attempting to increase the window past 10 significantly decreased the F_1-score in my method, but decreasing the window did not have such an impact. This allows my method to have an average detection time that is 6 turns sooner than other methods, but also leads to a slight performance hit.

My method outperforms others primarily because it is significantly better at not detecting a shift when no shift has occurred. This can be seen with the *Environment No Shift* group, where my method can correctly identify that no shift has occurred up to 90% of the time for the Goal focused agent, though the results for the other two agents are significantly worse. Compared to the Local Outlier Factor method, which scores a maximum of 47% only in the novelty-focused agent, my method scores only slightly worse at 46%, but all of these have very high variability. Overall, all methods are rather poor and unreliable when it comes to ignoring a shift, but my method shows promise, and this can be further improved with future work.

3.5 Conclusion

Experience managed games allow for the player to have a more personalized experience with the game, but can fall short when modeling the player. If a player model cannot account for how a player's preferences will shift as they play a game, the ExpM will not be able to make accurate observations of the player due to the bias in the environment. In this chapter, I show that detecting a shift is possible with a fairly high degree of accuracy and that a lack of shift can be ignored. I developed a means to detect such a shift and show that it is better than existing novelty detection methods for my data. For future work, this system could be expanded to include a second stage to confirm a shift exists, as recovering the player's preferences too often may be disruptive to the player. This would act as a hybrid of preference shift detection and player model recovery, combining elements from both. As such, a system that can confirm that a shift has happened with less disruption will allow for a significantly better playing experience while still being able to adapt to unexpected shifts in the player's preferences.

Table 3.1: The mean and standard deviations of the likelihood that a shift is correctly detected. Bolded are the best performing for each category.

Agent	EE	LOF	SVM	Mine (1)	Mine (10)
			Fast		
		Environment to Environment Action Types			
Exploration	0.84±0.36	0.91±0.29	0.93±0.25	**0.94±0.23**	0.94±0.25
Goal	**1.00±0.00**	**1.00±0.00**	1.00±0.04	**1.00±0.00**	**1.00±0.00**
Novelty	0.98±0.13	0.95±0.21	0.97±0.16	**1.00±0.06**	**1.00±0.06**
		Missing to Environment Action Types			
Exploration	0.89±0.31	0.89±0.32	0.83±0.37	**0.99±0.09**	**0.99±0.09**
Goal	1.00±0.04	**1.00±0.00**	1.00±0.06	**1.00±0.00**	**1.00±0.00**
Novelty	0.97±0.17	0.89±0.32	0.91±0.29	**1.00±0.00**	**1.00±0.00**
		Environment to Missing Action Types			
Exploration	0.79±0.41	0.84±0.37	**0.97±0.18**	0.90±0.30	0.87±0.33
Goal	0.99±0.11	**0.99±0.08**	0.96±0.19	0.42±0.49	0.36±0.48
Novelty	0.96±0.19	0.85±0.36	**0.99±0.10**	0.79±0.41	0.75±0.43
		Missing to Missing Action Types			
Exploration	0.84±0.36	0.77±0.43	0.92±0.27	**1.00±0.00**	**1.00±0.00**
Goal	0.65±0.48	0.67±0.47	**0.81±0.40**	0.61±0.49	0.53±0.50
Novelty	0.80±0.40	0.58±0.49	0.86±0.34	**0.95±0.22**	0.94±0.23
			Slow		
		Environment to Environment Action Types			
Exploration	0.84±0.36	0.91±0.28	0.91±0.29	**0.93±0.25**	0.93±0.26
Goal	1.00±0.04	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**
Novelty	0.98±0.15	0.91±0.28	0.95±0.21	**0.99±0.09**	**0.99±0.09**
		Missing to Environment Action Types			
Exploration	0.86±0.34	0.87±0.33	0.77±0.42	**0.99±0.08**	0.99±0.10
Goal	1.00±0.06	**1.00±0.00**	0.99±0.08	**1.00±0.00**	**1.00±0.00**
Novelty	0.98±0.13	0.88±0.32	0.88±0.32	1.00±0.04	**1.00±0.00**
		Environment to Missing Action Types			
Exploration	0.77±0.42	0.83±0.37	**0.96±0.20**	0.86±0.34	0.87±0.34
Goal	0.98±0.15	**0.99±0.10**	0.94±0.24	0.43±0.50	0.39±0.49
Novelty	0.96±0.19	0.83±0.37	**0.98±0.12**	0.81±0.40	0.77±0.42
		Missing to Missing Action Types			
Exploration	0.83±0.37	0.80±0.40	0.91±0.29	**0.99±0.07**	**0.99±0.07**
Goal	0.68±0.47	0.67±0.47	**0.73±0.45**	0.60±0.49	0.54±0.50
Novelty	0.76±0.43	0.58±0.49	0.84±0.37	**0.94±0.23**	0.94±0.25
			No Shift		
			Environment No Shift		
Exploration	0.28±0.45	0.14±0.34	0.08±0.27	0.43±0.50	**0.45±0.50**
Goal	0.31±0.46	0.34±0.47	0.24±0.43	0.90±0.30	**0.91±0.28**
Novelty	0.21±0.41	**0.47±0.50**	0.21±0.41	0.37±0.48	0.46±0.50
			Missing No Shift		
Exploration	**0.19±0.39**	0.19±0.39	0.06±0.24	0.02±0.12	0.01±0.07
Goal	0.38±0.49	0.31±0.46	0.21±0.41	0.43±0.50	**0.47±0.50**
Novelty	0.19±0.39	**0.42±0.49**	0.18±0.38	0.05±0.22	0.06±0.25
			F_1-Score		
–	0.69	0.69	0.67	0.71	**0.72**

Table 3.2: The mean and standard deviations of the turn the shift was detected on. Note that the only cases where a shift was detected are included. Bolded are the best performing for each category (lowest value for Shift, and highest for No Shift).

Agent	EE	LOF	SVM	Mine (1)	Mine (10)
			Fast		
		Environment to Environment Action Types			
Exploration	129.4±11.1	125.2±09.5	127.9±09.3	**116.8±09.3**	126.3±09.8
Goal	120.4±01.9	120.3±02.0	123.2±01.8	**112.3±03.7**	121.0±03.3
Novelty	124.0±07.0	125.3±09.0	126.3±07.5	**115.7±07.1**	124.6±07.2
		Missing to Environment Action Types			
Exploration	129.2±11.1	127.8±11.4	131.5±11.4	**115.1±07.1**	124.6±07.6
Goal	121.3±04.1	120.5±03.0	123.9±04.5	**112.1±03.6**	121.1±03.6
Novelty	124.6±08.0	125.1±09.3	128.6±10.2	**113.3±04.8**	122.2±05.0
		Environment to Missing Action Types			
Exploration	130.2±11.2	126.2±10.2	126.2±07.4	**118.3±10.3**	127.3±10.3
Goal	122.7±06.3	**121.8±05.5**	125.9±06.7	123.5±12.4	133.4±12.6
Novelty	125.8±08.9	126.5±09.8	126.0±07.1	**121.3±11.1**	130.4±11.4
		Missing to Missing Action Types			
Exploration	129.6±10.7	128.3±11.1	128.6±09.3	**114.9±06.6**	123.8±06.4
Goal	133.5±11.7	129.8±12.1	134.0±12.1	**122.3±11.7**	131.3±11.3
Novelty	131.3±11.4	130.3±11.8	131.0±10.6	**116.7±08.4**	125.5±09.6
			Slow		
		Environment to Environment Action Types			
Exploration	127.4±10.2	125.4±09.9	127.4±08.6	**116.8±09.2**	125.8±09.4
Goal	120.6±02.5	120.5±02.4	123.2±01.9	**113.1±04.2**	121.6±03.9
Novelty	124.7±07.8	126.4±10.1	126.3±07.2	**116.0±07.7**	125.0±08.3
		Missing to Environment Action Types			
Exploration	128.0±10.9	126.8±10.5	131.0±11.1	**114.7±06.9**	124.0±07.3
Goal	121.8±04.6	120.5±02.6	123.7±03.3	**112.9±03.8**	121.4±03.5
Novelty	125.4±08.3	126.9±10.1	128.3±09.3	**113.6±04.8**	122.4±05.0
		Environment to Missing Action Types			
Exploration	131.2±11.7	126.4±10.4	126.6±07.7	**121.3±10.4**	129.2±10.9
Goal	122.8±06.0	**122.1±05.6**	126.3±07.2	125.6±11.6	134.7±12.2
Novelty	126.0±08.9	128.0±11.1	125.6±06.1	**121.5±11.0**	130.0±11.3
		Missing to Missing Action Types			
Exploration	130.0±10.9	129.5±11.6	129.0±09.5	**115.0±07.2**	123.8±06.6
Goal	133.9±12.0	132.6±12.3	135.1±12.8	**121.5±11.4**	131.3±12.5
Novelty	132.7±12.2	132.6±12.4	131.0±11.0	**116.2±08.2**	125.7±09.0
			No Shift		
		Environment No Shift			
Exploration	131.1±11.5	129.4±12.0	131.5±11.3	122.3±12.3	**131.7±12.3**
Goal	134.3±11.7	131.2±11.5	134.2±11.8	127.2±13.2	**137.2±14.1**
Novelty	131.6±11.9	131.3±12.8	132.0±10.8	124.6±12.4	**133.4±12.4**
		Missing No Shift			
Exploration	**130.8±12.0**	129.1±11.9	127.9±08.3	115.0±07.5	124.5±08.1
Goal	**132.8±11.0**	129.6±12.3	132.7±11.0	121.3±11.3	131.0±11.6
Novelty	131.5±12.1	**131.5±12.4**	131.4±11.3	117.3±09.4	125.7±09.1

Chapter 4 Player Model Recovery

4.1 Introduction

Once a player preference shift has occurred and is detected, we still need to update the player model to reflect the preference shift. Detecting a preference shift does not help with understanding which way the player's preference has shifted, and the underlying environmental bias is still not addressed. Even if a shift is detected, we cannot recover the player model by simply observing what actions the player is performing in the game. Since the ExpM has altered the world's possible actions, the player is still selecting actions from a biased set. Thus, it is difficult to determine the player's new preferences without access to unbiased observations. To tackle this, I introduce a new type of game object called a distraction, a special game object that the ExpM can use to subtly query the player to recover the player model.

Distractions are meant to enable the ExpM to gather unbiased observations of player behavior. As such, it is necessary that distractions can be placed in any location and be associated with any action type. This means that distractions are very flexible in how they can be inserted into the game world. At the same time, the ExpM must limit how distractions are used because overuse could be problematic. First, distractions should not be relevant to any quests that exist in the world. This is to prevent users from interfering with quest progress by taking distraction actions. Second, distraction actions should be used sparingly. Given that distraction actions are very flexible, it is possible to simply overload the environment with distractions. This allows the user to take essentially any action type, making it easy for the ExpM to determine the player's new preferences. The issue with this is that the sudden influx of available actions may overwhelm the user and, in the worst case, result in the user disengaging from the experience due to being presented with too many choices. As such, distractions need to be delivered to the player more carefully and in smaller amounts in an attempt to maximize the amount of information the ExpM gathers with each distraction presented.

To do this, I reformulate the problem of the ExpM recovering the player model as a Multi-Armed Bandit (MAB) to take advantage of the existing pool of techniques to quickly find the player's current preferences. The problem discussed above is similar to the classic problem of exploration versus exploitation. If an agent explores an environment, they will gather valuable information about said environment. This, however, likely comes at the cost of not taking the best action at any given time since exploration is typically random. This is similar to providing the player with a large amount of distractions. The ExpM is, essentially, overexploring the space to determine the player's preferences. MAB's ability to balance exploration and exploitation allows me to quickly discover the player's current preferences without causing too many unwanted distractions. Distractions serve an important role here as they enable this process, acting as the arms of the MAB.

In this chapter, I create and test a system based on MABs that introduces new objects in the game world as distractions to gather information from the player and quickly detect a shift in the player's preferences. To evaluate this system, I reuse the custom text-based interactive fiction environment and the artificial agents that were created in Chapter 3. I also use a similar set of preference switch scenarios that are modified to better fit this method.

I find promising results in modeling the system as a MAB, but the experiments show weaknesses in the process stemming from restrictions in MABs. So, in addition to the MAB adaptation, I also extended this adaptation to Combinatorial Multi-Armed Bandits (CMAB), making modifications to how the system is represented. These are likewise evaluated in the text-based environment with the same agents and show considerable improvements over MAB methods. The rest of this chapter is separated into the first section, which details the methods, experiments, and results of the MAB adaptation, and the second section, which discusses the changes made for CMABs and their experiments and results.

4.2 Multi-Armed Bandit Based Recovery

For my experiments with MABs, I make use of the same environment and player model as in Chapter 3. The environment contains 7 rooms that an artificial agent inhabits, and there is a single quest that can be completed, though only one agent is focused and likely to complete it. As before, that environment is meant to simulate the previous involvement of an ExpM and thus has actions that are very prevalent (*talk* type actions), those that are still present (*look* and *touch* type actions), and those are missing (*read* and *eat*). Action types that are missing are still capable of being used in the environment through distractions injected, but they are not present within the environment initially due to the ExpM's simulated biasing of the environment.

The rest of this section will describe how I adapt the ExpM to MAB and give the details on the experiments and their results.

4.2.1 Multi-Armed Bandit Adaptation

In an experience managed system, the experience manager changes the game world using ExpM actions based on expected player behavior as defined by a player model. If a player's preferences change during gameplay without updating the player model, the player model is considered outdated. Any ExpM actions made based on this now outdated player model will not necessarily have the desired effect. In these situations, the player model should be recovered; however, the ExpM cannot use past observations to perform this recovery process as these player observations were gathered in a biased setting caused by the ExpM.

To recover the player model, I propose an additional set of ExpM actions that focus on offering the player a new set of tasks, which I call distraction actions. These distraction actions add and remove a new type of game object to the environment, a *distraction*. Distractions are a type of game object that the player can interact with. They are similar to regular game objects but differ from other objects in that their purpose is not to contribute to the game. Instead, they serve to test the player's interest in the various types of actions so that the ExpM can gain information on the player's preference without relying solely on what is available in the environment.

As stated before, every object in my environment has a primary means of interaction called the *action type*, and this includes distractions. These action types form the basis of the player model and how I represent the player's preferences. Thus, having distractions for each action type allows the ExpM to test the player's preference independently of the environment. In this 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, a distraction action consists of simply adding or removing a distraction corresponding to an action type to the environment.

To model this process as a MAB problem, I use these distraction actions as arm pulls, with one arm for each action type available in the environment. At the start of each turn, the ExpM can pull an arm, taking a distraction action to add a specific type of distraction to the room that the player is currently in. Afterwards, the player is allowed to act, and the outcome is whether the player interacts with the distraction or not. This determines the reward, or payout, associated with that arm for a given pull. For my system, I consider the player interacting with a distraction to give a reward of 1 to the arm associated with the action type of that distraction. If the player does not interact with the distraction, then no reward is given for that turn, even if the player interacts with a game object of the same type. The mean 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 provides distractions, the player constantly makes choices about which object they want to interact with. The ExpM observes 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 I want the observations to be as unbiased as possible, I only consider a truncated version of the action history, only including the actions that happened after the ExpM is sure that the player's preferences have shifted. For my experiments, this shift point is simulated and known to the ExpM instead of being detected. The ExpM then calculates a dynamic player model by finding the frequencies of each action type within the truncated history.

By formulating player model recovery as a MAB problem, this allows me to utilize algorithms designed to efficiently learn the expected payout of each arm (here, essentially the probability that the player interacts with this distraction). Formally, each algorithm does this by minimizing the total regret over an execution episode, where regret is defined as the difference between the maximum total payout that could be achieved by pulling the best arm each turn and the payout that the agent actually received. This ensures that the ExpM will be offering as few distractions as is necessary to gather the information it needs to be able to effectively provide distractions that the player is likely to take.

4.2.2 Distractions

Distractions themselves come with some restrictions on how they should be designed, which may make them difficult to develop. I have stated before that distractions need to not contribute towards the game, only serving as a means for the player model recovery system to take unbiased observations of the player's actions. To put this more formally I expect that distractions:

- will need to be largely irrelevant to important parts of the game like quests so as not to interrupt any authorial goals
- will need to be carefully designed to not entice an engaged player while distracting unengaged players
- should represent a type of action or style of play in the game (it's action-type)

The first item is inherited from one of the purposes of experience management: to balance the sense of player agency and needs of authorial goals. Given the nature of distractions as something that could be potentially distracting I suspect that they are likely to interfere with authorial goals, either due to being mistaken as quest objects (which may lead the player to carry them looking for a use) or just by shifting the player's focus away from what the author wishes for them to be engaged in for too long. To combat this all the distractions are designed to resolve themselves in a single round, and then be removed from play. This is safe to do with the agents as they do not mind objects disappearing unexpectedly, but I will revisit this later for human subjects. I also ensure that distractions are only used when they are needed for this system.

The second item is largely meant as a safety measure in the case of a false positive for a shift detection. Part of this is accomplished by limiting the amount of distractions used, for which the MAB methods is a single distraction per round. Players that are engaged in their current task are likely to continue engaging if there are minimal distractions, but if the environment is full of them the likelihood that the player will notice at least one is increased. Since the MAB methods get a reward when the player interacts with distractions it will then give more distractions of that type, which may throw an engaged player off their previous task. By limiting the amount of distractions I can make this possibility less likely to happen and preserve authorial intent and not disturb engaged players.

The third item is meant to ensure the distractions use as a measurement tool. To be able to give the MAB a reward for an action type, the distraction must match the action type that is being rewarded. For artificial agents this is simple as it only requires that the distraction is internally labeled as a distraction with a specific type, but for a human subject this may be more of a challenge. For the agents distractions are generated programmatically with a script as the agents do not look or judge the method of interaction, the name, nor the outcome of the distractions.

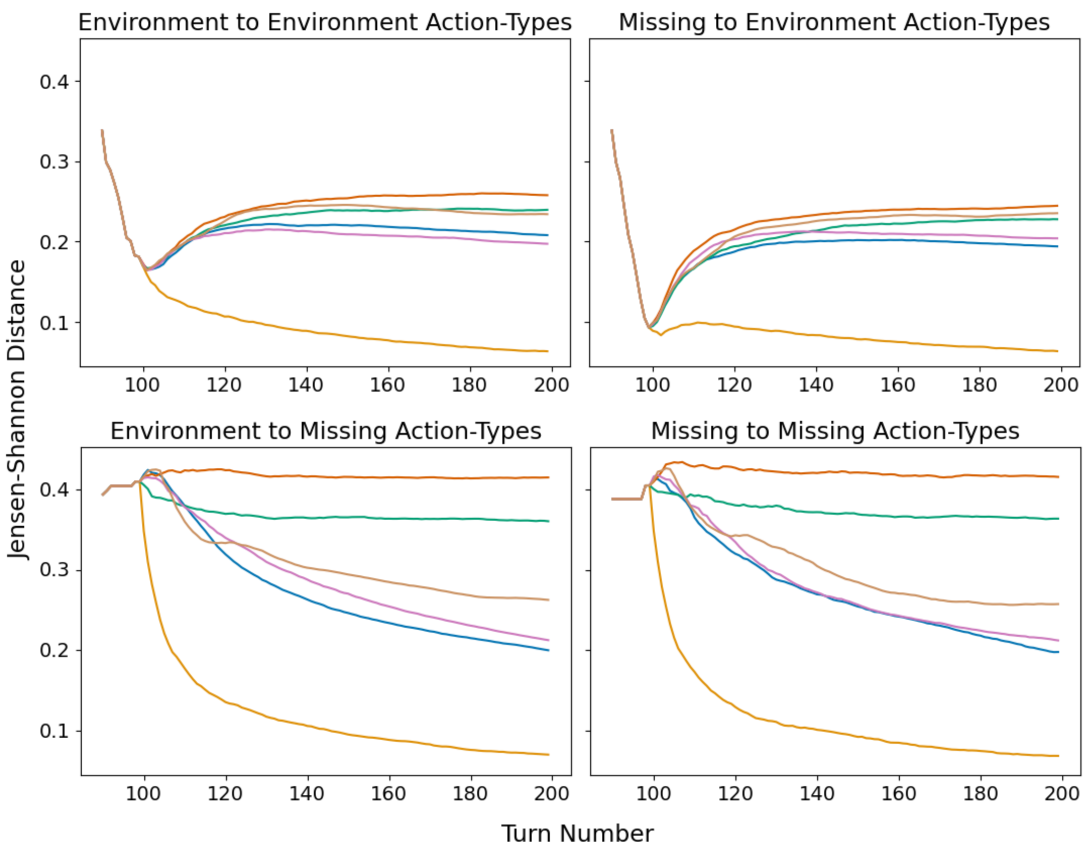


Figure 4.1: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Exploration Focused Agent**

4.2.3 Experiments

With these experiments I intend to show that using MABs I can quickly recover a player model, which in conjunction with Chapter 3 have a complete proof of concept system that can detect a preference shift and recover the new preferences in the form of a player model. I will demonstrate this by measuring the rate at which the calculated player model converges to a new value and how close it can get to the true player preferences. To do this I use the same 3 agents as in Chapter 3: the Exploration Focused Agent, the Novelty Focused Agent, and the Goal Focused Agent. Each of these agents has an underlying preference distribution that describes the likelihood to engage with certain types of actions which is dictated by the scenario. While the agents remain unchanged, the preference shift scenarios are slightly different to better suit player model recovery. I still consider all possible preference shifts from preferring primarily a single action type to shifting to a different action type, but I no longer consider the no-shift category. Likewise, since the methods explicitly ignore any data before the preference shift has occurred, the speed of the shift is not considered, reducing the total number of preference shift scenarios to 20. These 20

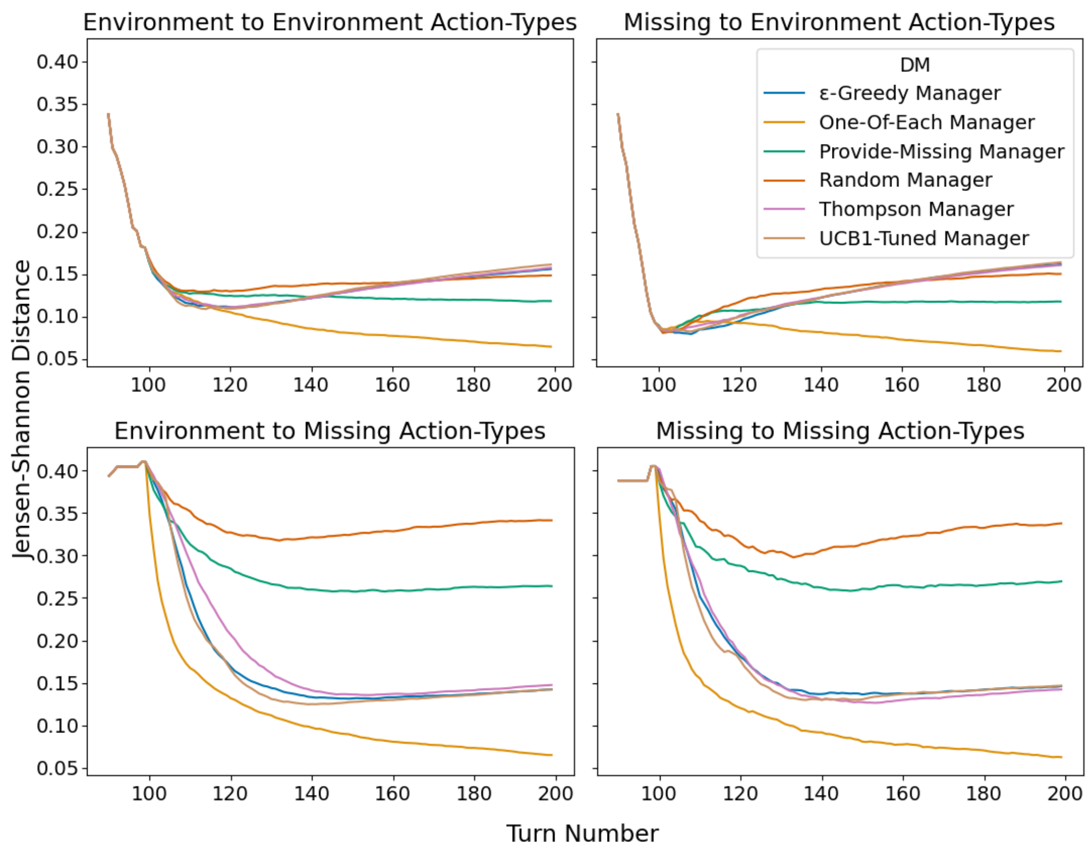


Figure 4.2: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Goal Focused Agent**

scenarios are also categorized into 4 group scenarios: *Environment to Environment*, *Missing to Environment*, *Environment to Missing*, and *Missing to Missing*.

For the sake of consistency, each of these preference shifts is structured the same, consisting of 100 turns of a pre-generated history that is shared between agents for a given scenario. This history exists to reduce the computation time needed to run these experiments, as the addition of ExpM intervention would otherwise take a prohibitively large amount of time. The 100-turn history consists of 90 turns of the pre-preference followed by 10 turns of the post-preference. These 10 turns are introduced to simulate the time it takes to detect a preference shift. It is unlikely that a preference shift can be detected this quickly or consistently, but as stated earlier, the methods that I use ignore data before the preference shift, so I consider it valid to use. This pre-generated history is created using the Goal focused agent.

After these 100 turns of history the ExpM is activated and is allowed to take actions. These are not full ExpMs, only implementing the necessary parts to recover the player model, and will be referred to as *managers* to distinguish this. I include three managers as baselines: the One-of-Each manager, which provides one of each

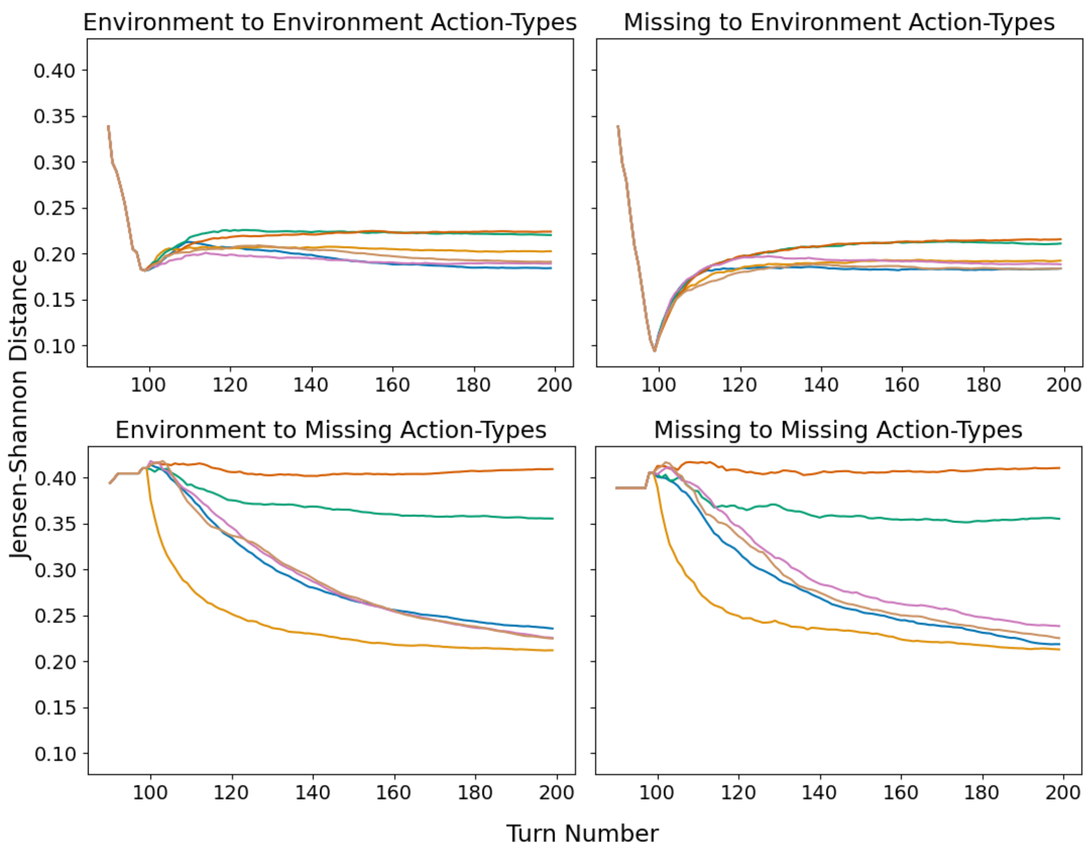


Figure 4.3: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Novelty Focused Agent**

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. I expect the One-of-Each manager to perform the best as it breaks from my requirement of minimizing the number of distractions given. Thus, it serves as an upper bound on the performance of how fast a manager can expect to reconstruct a player model. The MAB managers each implement a different MAB algorithm, of which I use 3: ϵ-greedy ($\epsilon = 0.2$), UCB1-Tuned, and Thompson. The details of these algorithms are present in the background section for MABs in 2.3.2. These were chosen after testing several different algorithms including ϵ-decreasing, various ϵ values for ϵ-greedy, and UCB1. Of these, I 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. As stated in Section 2.3.2, UCB1 generally underperforms compared to UCB1-Tuned due to the tuned variant selecting selecting high-variance options more often.

I measure the agent's preferences against the calculated player model by treating each as a probability distribution over action-types and use Jensen–Shannon (JS)

Agent	Random	Provide-Missing	ϵ-Greedy	UCB1-Tuned	Thompson	One-Of-Each
			Environment to Environment Action-Types			
Exploration	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	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	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	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	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	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	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	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	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	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	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	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 4.1: 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.

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, the manager starts at turn 90 of the pre-generated history as this is the point at which the agent switches to the post-preference, though the manager does not take distraction actions until turn 100. For each run I first average the models over each trial and use that averaged model to compare against the agent's preferences. I run 50 trials for each test of agent, manager, and scenario.

4.2.4 Results and Discussion

For each test I have created a graph that measures the JS distance between the player agent's internal preferences and the measured player model in Figures 4.1, 4.2, and 4.3 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 turns 90 and 100 are all the same, I include them to show the effect of switching between the different groups of preferences. I 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, I 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 4.1 and 4.3. This is especially prevalent in the Novelty Focused Agent, as it prefers objects that it deems novel. Since the manager has not taken any distraction actions before turning 100, the sudden addition of distractions means that it will likely start to interact with them 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. I 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 I will be focusing on these the *-to Missing* scenario groups unless noted otherwise.

Manager and Agent Differences The 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 simultaneously. Multiple distractions prevent 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 the other baseline, and it serves as a pure exploration option. I find that for the *-to Missing* scenario groups, all of the managers are statistically better than this manager, as shown in Table 4.1. 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 the 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 the other baselines as it is informed, but also differently than the 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 environmental bias is a factor and that observing the environment should play a part in how a manager chooses a distraction. This will be addressed in the next section detailing the transition to combinatorial bandits.

While the MAB managers perform very similarly, 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.

In some rare specific cases I 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 4.1. 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 4.2) 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 the performance is promising, there are limitations to this approach. Only being able to play a single arm does help with reducing the number of distractions used per turn, but it also only allows the algorithm to observe whether the player prefers the distraction over the environment, requiring more turns to deduce the best distraction. The MAB methods also do not take into the environment, only rewarding the MAB for environmental actions when the player interacts with a environmental distraction. These limitations both require more distractions to be played at a time, which can be constructed by considering each pair of distractions as an arm, but the CMAB framework provides a better solution.

4.3 Combinatorial Bandit Based Recovery

As stated, while the results from the MAB experiments are promising, there is a limitation inherited from adapting the system as an MAB: Only a single distraction can be used per turn. While it is my goal to minimize the number of distractions used, using too few may be too restricting in the amount of information that can be gained per turn. To combat this I have extended the MAB framework to a combinatorial MAB framework and have leveraged of the CUCB algorithm [7] to recover the player's preferences.

4.3.1 CMAB Adaptation

CMABs are an extension of MABs which adapts the problem to a combinatorial framework. This allows for the use of multiple arms, but more notably allows for the algorithm to gain more information with more arms played, making observations not only on if the player prefers the distraction over the environment, but also which of the distractions are preferred. Similarly to MAB adaptation, for CMABs I still make use of the same set of ExpM actions to add distractions to the environment, though this time allowing for more than a single distraction to be added. Multiple distractions are played with each being a single arm, the collection of multiple of these arms played in a round forms a super arm, which is played. Based on how the agent interacts, a reward is given, with a reward of 1 given to the arm that is interacted with in that round. This means that the individual arms within the super arm are not rewarded together; only a single arm within the super arm gets a reward. If the agent has not interacted with any distractions in that turn then no reward is given to any arms that round.

In the previous section, I discussed how there is some limited success in solely reducing the bias of the environment in the form of the Provide-Missing manager, and this was used as inspiration for an additional improvement: *replace-with-environment-action*. With this improvement active, a super arm containing a distraction with an action type already present within the current area will have the distraction replaced with the matching environment action. Instead of adding a distraction of that overlapping type, the algorithm considers any action taken of that overlapping type in that round to count towards that distraction. This reduces the number of distractions added early on to around 1.7 per round in the first 10 turns, though as the algorithm gains more information, it starts to rarely need to take an environment action, and thus, this decreases to an average of 1.95 after the first 10 turns.

To illustrate this through an example, consider an agent that prefers to interact with talk actions, which are the most prevalent action types in the environment. Without replace-with-environment-action if the agent is given a distraction for talk and read, the agent may still not interact with the talk distraction, instead opting to interact with a talk action that is already present in the room. With the replace-with-environment-action option set the talk distraction is never added to the room, instead all talk actions in the room for that round will be considered as talk distractions. If the agent then interacts with a talk action, the CMAB algorithm will count this as a reward towards talk distractions, and no reward for read distraction.

4.3.2 Experiments and Results

This second round of experiments mirrors the first in that they use the same agents and the same preference shifts. The preference shifts in the scenarios are the same, but the new managers are run with a newly generated history for each scenario. The main difference is the introduction of three new managers, which I compare against the previous best, ϵ-greedy ($\epsilon = 0.2$), and the optimal possible baseline, One-of-Each manager. The three new managers are dedicated to testing variants on CUCB: one where a super arm consists of 2 distractions ($k = 2$), another with a super arm of 3 distractions ($k = 3$), and the last where a super arm only has 2 or fewer distractions using my replace-with-environment-action strategy ($k = 2$, rwea). All of these ExpMs calculate the player model the same way, by measuring the frequency of types of actions that the player takes starting when the ExpM thinks that the player has shifted their preference.

For each combination of the 5 ExpMs, 3 agents, and the preference switch scenarios, I ran 100 trials, up from the 50 for the MAB experiments, and measured the JS distance, comparing the agent's internal preference to the ExpMs calculated player model. I present and discuss only the results from the previously identified most realistic scenario *Environment to Missing* (Figure 4.4, but include a full set of results for completeness at the end of the chapter in Figures 4.5, 4.6, and 4.7 . I also report the JS distance on the final turn in Table 4.2 to match the MAB results, and the

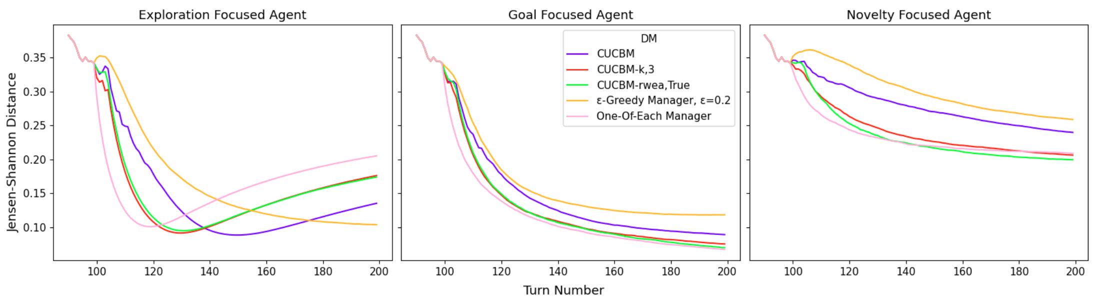


Figure 4.4: Mean JS Distance between Player Model and Agent Preferences vs. Turn in the Environment to Missing Scenario

turn that a minimum JS distance was found in Table 4.3 as the new results require more context to explain their performance.

Manager Differences While the best performing manager still is the One-Of-Each manager, I do find that the differences between it and the CUCB based managers are much closer than before, and giving only two distractions at a time already provides significant benefits over the previous best, ϵ-greedy ($\epsilon = 0.2$). This is expected as giving more distractions allows the manager to gain more information, taking advantage of not just if the player agent interacts with a distraction but also which of the distractions it chooses to interact with.

For CUCB, both $k = 3$ and $k = 2$ when replacing a distraction with an environment action are capable of getting surprisingly close to One-Of-Each in all agents. Replacing a distraction with an environment action was developed to reduce the number of distractions shown each turn. This reduced the number of distractions from 2 to an average of 1.93, though this value is around 1.3 for the first couple of turns. Later, the manager has enough information about actions in the environment that it does not need to play them as often, so replacement only occurs rarely. I expected that this would negatively affect this strategy's ability to recover the new preferences, but I found that it actually increased its ability. Without replacement, when the agent interacts with the environment, no reward is given for any of the actions in the environment. In replacing a distraction with an environment action, I allow any object that the agent interacts with to count as a reward for the CMAB algorithm. This allows it to watch for more actions than before and naturally fill in action types that are underrepresented due to the environment's biasing. This indicates that replacing an arm with an existing environment action may be a good strategy for MAB-based managers, but without an alternative distraction to test alongside, it may unintentionally cause issues with more goal-focused players.

Agent Differences I find that the difference between agents largely mirrors the MAB experiments, including the previously seen effect where the distance between the agent's internal preference distribution and the measured preference distribution hits a minimum value and then starts to increase. This is attributed to the MAB gaining enough information on the preferences that it started to give the highest valued action type almost exclusively, though was often only seen in the other scenarios. In these tests, we see that this effect is also present, even in the Environment to Missing scenario in the Exploration-focused agent for everything but the ϵ-greedy manager. This suggests that for this agent, the strategies that exhibit this effect are capable of identifying the new preference within 20 turns of the ExpM activating, and this is likewise because one of the distractions given is almost always the preferred distraction. Other agents do not exhibit this effect, which I take to mean that they are significantly more difficult to recover the player's preferences for. For both the Goal and Novelty focused agents this is likely because they will default to environment actions when they are not given an action that they wish to interact with, thus skewing the data in favor of environment action types.

This overshooting effect does skew the data, no longer making Table 4.2 an accurate representation of the overall performance of the managers. For the Goal and Novelty focused agents, I find that both ϵ-greedy and the CUCB managers continue to trend downwards, never reaching their maximum, but for the Exploration agent, the CUCB managers recover the preference too quickly and then overshoot, causing the distance on the final turn to be lower than for ϵ-greedy. To remedy this I have also included average turn on which the minimal distance value was found for each. Making use of the average turn a minimal distance was found in Table 4.3 we can see that for the Environment to Missing section the Exploration focused agent measures the best, beating out ϵ-greedy. This table likewise does not paint the full picture, though, as it shows that ϵ-greedy seems to outperform the Goal focused agent, but this is due to the other managers trending downwards till the final turn, never hitting a minimum.

Overall the need to analyze these results on which turn they reached a minimum value indicates that the CUCB methods far outperform the non-combinatorial methods earlier. The previous methods additionally were far from the optimal baseline of the One-Of-Each manager while the CUCB based ones start to overlap with it. The replace-with-environment-action option, which itself is only possible with the CMAB approach, solves one of the previous issues with the MAB approaches in that it can make use of how the agent interacts with the environment and considerably improves performance, occasionally past what the CUCB can do with 3 distractions played per round.

4.4 Conclusion

In the previous chapter, I have shown a method to detect the presence of a preference shift, but finding a preference shift is only part of the process. In this chapter, I have created a method to recover the player model after a detected preference shift. This method uses active intervention by the experience manager and a new type of game object called a distraction. This method has been shown to work with artificial agents taking the place of human players. This method is based on MABs and allows the ExpM to use distractions to both gain information on the player's preferences and use that information to allow the player to interact in their preferred way. This method was then further refined to use Combinatorial MABs, which increased its performance. This allowed for a modification to reduce the number of distractions used while increasing performance.

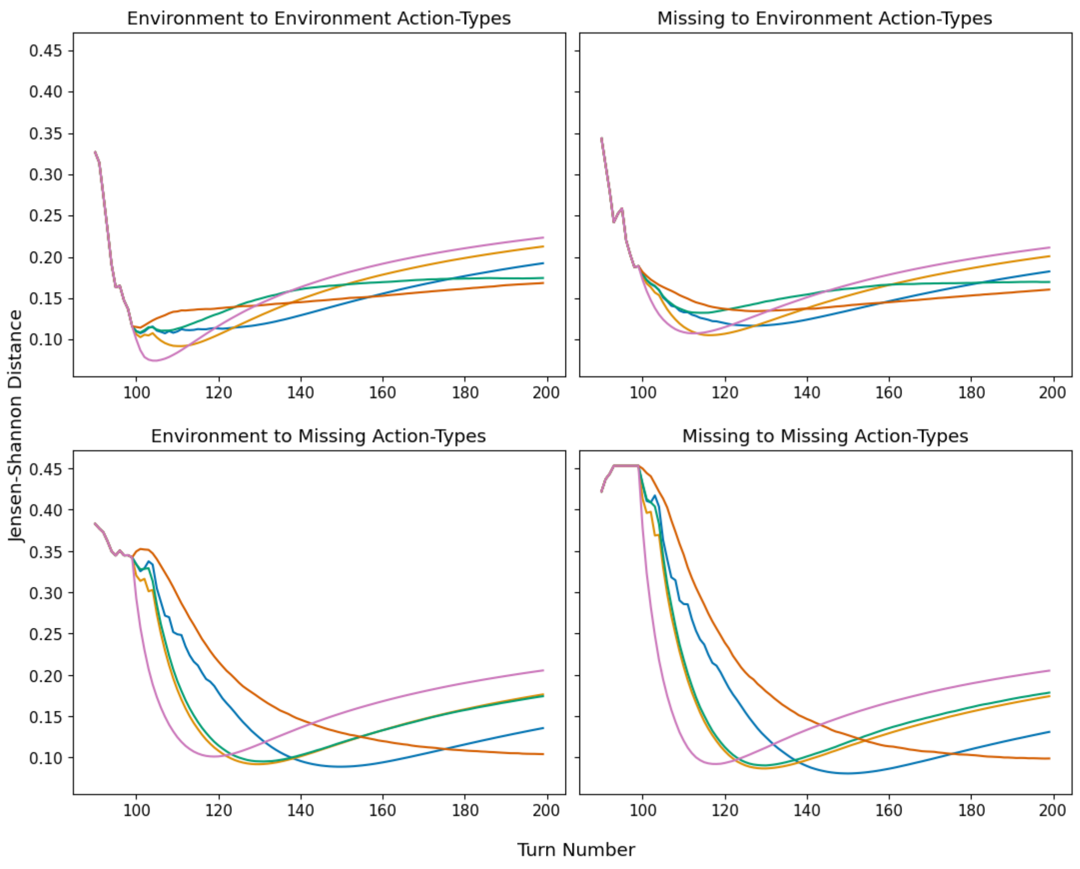


Figure 4.5: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Exploration Focused Agent**

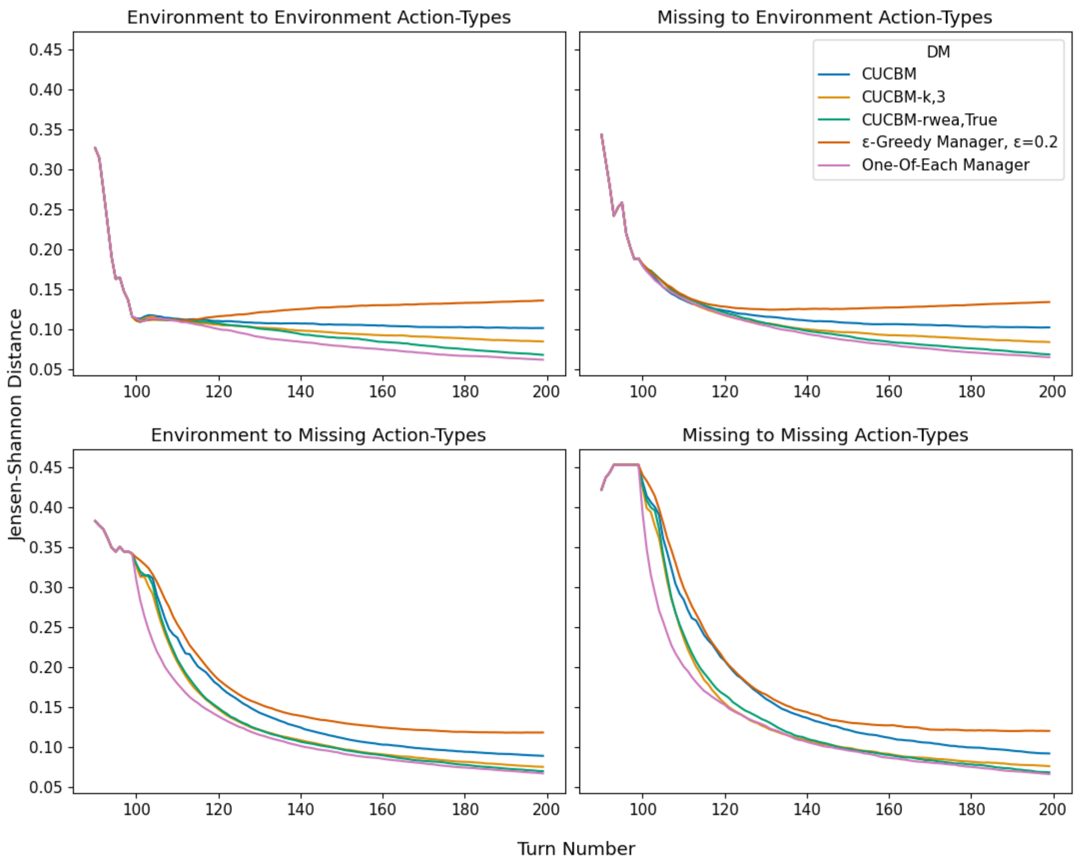


Figure 4.6: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Goal Focused Agent**

Agent	ϵ-Greedy , ϵ=0.2	CUCBM	CUCBM-rwea	CUCBM-k,3	One-Of-Each
			Environment to Environment Action-Types		
Exploration Focused	**0.169±0.045**	0.192±0.033	0.175±0.072	0.213±0.016	0.223±0.010
Goal Focused	0.136±0.025	*0.102±0.029*	***0.068±0.025***	*0.085±0.027*	*0.062±0.022*
Novelty Focused	0.189±0.037	*0.177±0.035*	0.208±0.034	***0.173±0.033***	0.190±0.029
			Missing to Environment Action-Types		
Exploration Focused	**0.161±0.038**	0.183±0.029	0.170±0.073	0.201±0.014	0.211±0.009
Goal Focused	0.134±0.025	*0.103±0.030*	***0.069±0.025***	*0.084±0.027*	*0.065±0.023*
Novelty Focused	0.198±0.039	*0.182±0.036*	0.212±0.034	***0.179±0.035***	0.195±0.030
			Environment to Missing Action-Types		
Exploration Focused	**0.104±0.039**	0.135±0.025	0.174±0.038	0.176±0.018	0.205±0.018
Goal Focused	0.119±0.029	*0.089±0.028*	***0.070±0.025***	*0.076±0.027*	*0.067±0.023*
Novelty Focused	0.259±0.076	*0.240±0.047*	***0.200±0.037***	*0.206±0.037*	*0.209±0.036*
			Missing to Missing Action-Types		
Exploration Focused	**0.099±0.030**	0.131±0.020	0.178±0.048	0.174±0.014	0.205±0.013
Goal Focused	0.120±0.032	*0.092±0.031*	***0.069±0.024***	*0.076±0.026*	*0.066±0.023*
Novelty Focused	0.263±0.074	0.247±0.048	***0.209±0.033***	*0.213±0.037*	*0.218±0.033*

Table 4.2: Mean and Std. of the JS distance between player model and agent preferences on the final turn for all CMAB tests. Bolded entries represent the best performing manager (excluding One-Of-Each) and italicized entries are statistically significant and better than ϵ-greedy with $p < 0.001$ according to a Student's T-Test.

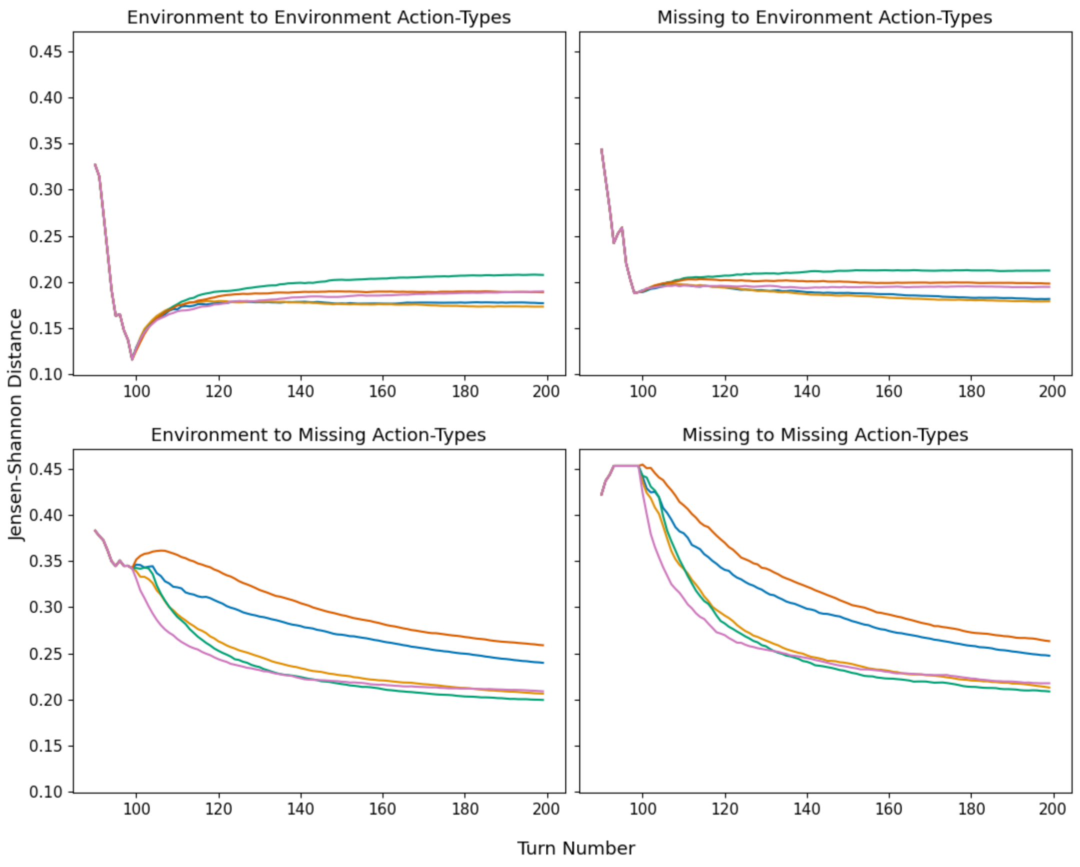


Figure 4.7: Mean JS Distance between Player Model and Agent Preferences vs. Turn for **Novelty Focused Agent**

Agent	ϵ-Greedy , ϵ=0.2	CUCBM	CUCBM-rwea	CUCBM-k,3	One-Of-Each
			Environment to Environment Action-Types		
Exploration Focused	115.45±6.79	112.84±3.08	120.98±9.44	***108.51±1.27***	*106.82±0.39*
Goal Focused	**117.52±7.83**	129.78±10.26	149.47±12.99	138.23±11.25	148.90±11.03
Novelty Focused	107.64±7.83	110.80±9.44	**104.91±5.99**	112.02±9.30	104.24±4.70
			Missing to Environment Action-Types		
Exploration Focused	124.43±9.33	120.56±4.48	130.81±11.16	***114.06±2.83***	*112.58±0.48*
Goal Focused	**131.62±9.02**	150.17±9.95	171.21±9.82	158.90±12.00	172.54±10.56
Novelty Focused	117.50±8.69	122.88±10.10	**113.27±8.48**	125.05±10.86	118.87±9.29
			Environment to Missing Action-Types		
Exploration Focused	164.66±6.85	*141.76±2.81*	*128.62±4.52*	***125.44±1.02***	*116.08±0.57*
Goal Focused	**156.63±8.83**	162.83±7.92	165.66±7.80	164.17±7.89	167.42±7.93
Novelty Focused	**148.89±10.96**	153.68±9.89	155.99±8.89	153.67±10.55	148.35±11.21
			Missing to Missing Action-Types		
Exploration Focused	178.81±14.96	*149.60±5.02*	*133.21±7.12*	***130.49±1.88***	*119.04±1.07*
Goal Focused	**168.15±15.50**	178.43±15.63	179.84±14.81	174.53±16.97	182.43±13.48
Novelty Focused	**171.25±20.31**	174.34±18.47	171.26±16.16	171.73±18.14	164.10±20.70

Table 4.3: Mean and Std. of the average turn a minimum JS distance was measured between player model and agent preferences for all CMAB tests. Bolded entries represent the best performing manager (excluding One-Of-Each) and italicized entries are statistically significant and better than ϵ-greedy with $p < 0.001$ according to a Student's T-Test.

Chapter 5 Human Study Verification

5.1 Introduction

In the previous chapters, I have demonstrated that an ExpM can automatically detect preference shifts and recover the player model afterward, but thus far, this has only been tested on automated agents. These agents are designed to mimic several aspects of human-like behavior but do not represent the full breadth of complexity that humans are capable of. For this reason, I have opted to use a human subject study to ensure their validity in humans as well.

Unfortunately, this process is not simple as the methods developed thus far have been prototyped with the artificial agents in mind, often with no regard to the parts of human behavior that are explicitly not modeled by the agents. Artificial agents do not require a cohesive environment, do not need quests that follow a logical progression, and are able to prefer any set of actions that they are instructed to prefer. Humans, on the other hand, are less likely to seriously engage with the environment when it contains arbitrary unexplained elements, need to have enough information to reason out the next step in a quest, and may have trouble preferring not to read in a text-based game. Thus, these elements need to be reworked to be compatible with human subjects.

In addition to the rework of the environment, I also have limited resources which restricts the design of the experiment. Since I am interested in validating the results of the artificial agents experiments, I have opted to perform a single experiment on human subjects to test the Player Model Recovery (PMR) process. Unlike Preference Shift Detection (PSD), PMR requires active intervention by an ExpM, so the initial experiment will be used to validate PMR, and a portion of the data that the ExpM does not act upon will be amalgamated and resampled to create synthetic human-like data to validate PSD.

Furthermore, the PMR verification study works around the limitation of resources and participant availability, taking on a somewhat limited scope. Instead of designing an entire game and waiting for the player to exhibit a preference shift—a likely rare occurrence—I have instructed participants to take on a given preference and shift to a new preference after a certain number of turns. This allows me to gather more data from a larger group of participants while having an accurate understanding of where a preference shift is allowing for the ExpM to be able to handle it without confounding factors.

The rest of the chapter will lay out these parts. First, I will discuss the necessary changes to the environment and the distractions to make them compatible with human players. Then I will outline the experimental setup for PMR study on human subjects, and will include a part on how I reuse a portion of the data to test PSD. Finally, I will report on the results and discuss the conclusions that can be drawn from the data and the participants, along with the limitations of the study.

5.2 Human Player Modifications

Since the managers and method are built around text based interactive fiction games, I have opted to continue using Inform7 as the basis of the human study experiments. This allows for easy expansion and prototyping and is suitable for playing even by people with little experience with video games. In this section, I will go over the modifications made to the test environment to make it human-playable.

5.2.1 Environment

Early tests showed that the environment used with the agents was too small to be used for human players, which may lead a player to grow bored and disengaged and make it difficult to validate the performance of the methods. The agent's version of the environment only contains 7 usable rooms, 3 of which are locked off to simulate environmental biasing by the ExpM, and a single task that can be done in a handful of turns if the player focuses on it. To expand this to be human-compatible, I have instead decided to start with a different set of rooms. For the human study version, the map (Figure 5.1) now contains an increased 36 rooms arranged in a 6 by 6 grid. These 36 rooms are organized into 4 regions each containing 9 rooms: The forest region (NW), caves region (NE), village region (SW), and castle region (SE). The player starts near the center of the map, in the northeastern part of the village region with Merchant Hargrove, a quest giver NPC.

Alongside these 36 new rooms I have opted to change the action types. The previous set of action types was heavily inspired by the built-in set of verbs that are possible in Inform7 and other adventure games (Figure 3.4). While these were suitable for artificial agents, it would be difficult to claim that a human would prefer to interact with an object by either looking at it, speaking to it, or touching it, as these are often necessary to do in tandem to complete any given task or are very context-sensitive, like reading and eating. Instead, the new action types represent more abstract ideas and methods of playing a game and are designed to match the types of playstyles that can be found in modern games like *The Elder Scrolls V: Skyrim*. The new action types are: *Diplomacy*, *Crafting*, *Combat*, *Stealth*, and *Magic*.

These action types are also distributed differently than before, as I have access to fewer gameplay traces. Artificial agents can be made to run through the environment any number of times, with the combined number of trials, scenarios, and agents reaching into the thousands for previous experiments. For human players, I can only ask them to participate in the experiment once, which limits which action types take the focus in the experiment, and thus only diplomacy and combat are considered *environmental* action types, with the other three considered *missing*, only being present within distractions.

In addition, the quests available to the player have been expanded. These are designed to take multiple steps each and exceed the approximate number of turns it takes for the player to shift to their second preference. The first quest involves a merchant in the starting room asking the player to retrieve a gem from the caves. By continuing to talk to the merchant, the player is informed where to go, which

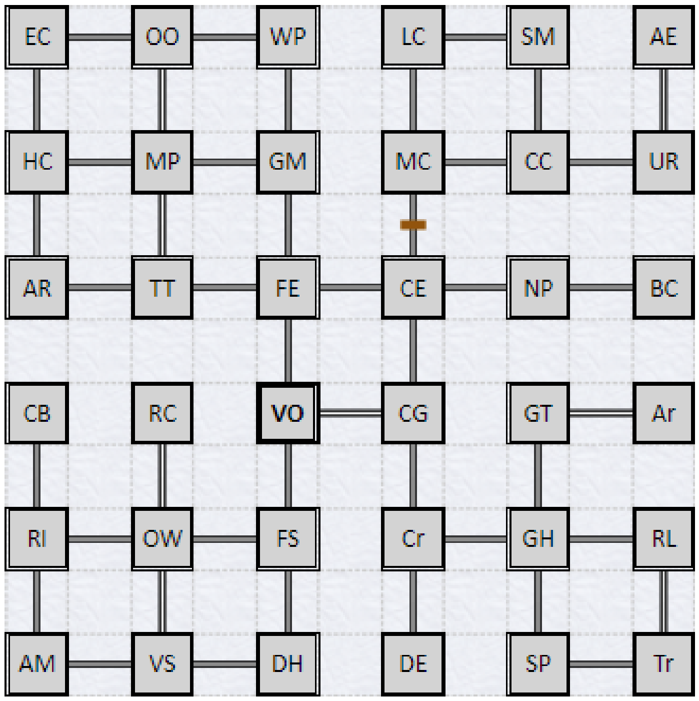


Figure 5.1: The expanded map for the human environment. The participant starts in the Village Outskirts (VO) and is tasked with retrieving a gem from the Secret Mining Shaft (SM) in the upper right, finding a way to bypass a group of miners blocking the path.

takes them past the second NPC quest giver on their way to the caves region. If the player tries to go there early, some miners block the path, but talking to them after accepting the quest unlocks the path to the rest of the caves, allowing the player to explore, retrieve the gem, and bring it back to the merchant. The second quest is similar in length and exists in case some players finish the first quest too quickly. The second NPC quest giver, Eldrin Leafwhisper, is in the beginning of the forest region and asks the player to settle a dispute between two factions of Forest Fae: the Sylvan Sylphs and the Verdant Sprites. The two factions are located in separate rooms, and the player can ask for directions from Eldrin to help navigate. The player can choose to follow up by talking to to the leader of either (or both) of the factions and is tasked with uncovering some information about a sacred idol in a nearby room. The minimum amount of turns needed to complete both quests is 40 turns, twice the amount of turns before they are asked to switch their preference, but with extra dialog to explore the player is likely to only complete one of the quests.

5.2.2 Distractions

Since I have a new set of action types, I create a new set of distractions to match. For each type, I created at least 20 distractions, a sample of which can be seen in Table 5.1. Some action types have more distractions, with crafting having 40 because it was the action type the participants were tasked with preferring after the preference shift. These are more likely to be used as the manager learns that the player prefers crafting and I wanted to ensure that the player is unlikely to see a repeat.

In preliminary experiments, I found that the previous requirements I outlined for distractions were not suitable for human players and led to players often learning to ignore them or never noticing them in the first place. I attempted to keep distractions as unimportant as possible and have them resolve themselves in a single turn. This attempt to not interfere with regular gameplay as much as possible led to the game being designed to end in a soft failure outcome, not punishing the player for interacting with a distraction but never rewarding the player either. For example, a combat object might be a piece of equipment that the player can take, but for a distraction, the object would be old, and attempting to pick it up would have the player character either reject it for being useless or disintegrating when they try to pick it up. Constant negative outcomes for the player were discouraging, so to rectify this, I rewrote the distractions so that most include a soft success, rewarding the player with some resource like experience points or crafting materials, though due to the short length of the experiment, these values are never tracked which can be seen in Table 5.1 with "+1 iron" and "+1 cloth" for crafting. To not make the distractions too enticing, some distractions still end in a soft failure, like attacking a "strange goblin" as seen in Table 5.1, which ends up being just some branches.

Distraction type	Distraction Action	Resolution
Combat	Attack Strange Goblin	As you approach the figure it turns out to be several branches vaguely in the shape of a goblin.
	Attack Deer	You quickly take down the deer. **+10xp**, **+3 rations**.
Crafting	Take Meteoric Iron	You have acquired **+1 iron**.
	Take cotton cloth	You have acquired **+1 cloth**.
Diplomacy	Help Hungry Beggar	You give some food to the beggar, who eats it gratefully and blesses you for your kindness. **+1xp**.
	Help Trapped Frog	You carefully free the frog from the crevice, and it hops away with a thankful croak. **+2xp**.
Magic	Pray at idol statue	Your heart is warmed by the god's blessing and you are filled with peace and courage. **+2mp**.
	Touch mystical vine	The vine's light pulses stronger, and you feel a rush of vitality. **+1hp**.
Stealth	Steal coin purse	You decide to take the whole thing **+10gp**, though you discard the coin purse itself after you extract the contents.
	Pickpocket snoozing man	The man reeks of alcohol and is absolutely conked out, but unfortunately you do not find anything of use on him.

Table 5.1: An example of distractions used in the human study. Distractions are designed to resolve themselves after a single interaction and to sometimes give the player a small reward. These rewards are bolded in the resolution and were displayed as bolded for the player though their values are not tracked for this experiment.

I also found early on that human players can have difficulty distinguishing between types of distractions. For example, some early distractions for stealth included items that the player could pick up or NPCs that the player could steal from, which led to a wide variety of different looking actions that all were considered stealth. For something like a *smoke bomb*, it is not readily apparent to a player who prefers stealth that this is related to stealth and was occasionally mistaken for a combat item. The solution I found was to restrict how the player would interact with each of the distractions, modifying it so that the only way to interact with combat actions was to attack, for crafting to take crafting materials, diplomacy to help animals and humans, and stealth to steal and pickpocket. Along with these, the player is hinted as to how to best interact with the distraction within its description when it is not clear, which was helpful for magic distractions as they had more methods of interaction.

These preliminary tests have led me to update the requirements for distractions. Now distractions need to be:

1. largely irrelevant to important parts of the game like quests so as not to interrupt any authorial goals

2. carefully designed to not entice an engaged player while distracting unengaged players

3. represent a type of action or style of play in the game (the distractions action type)

4. *readable*: clearly recognizable by the player as belonging to their given action type

Another problem with the use of distraction for the artificial agents was that I was able to add distractions each turn and remove them before the start of the next turn. For human players this could be jarring if they were alerted to the addition of new objects to the room, or the distractions would be ignored if they were not and the distractions were added silently. To avoid this, I have opted to add distractions on a more natural border when the player moves between rooms. The player would already need to read the room description to understand what is in the room when they move, allowing them a chance to be informed about any distractions added during that time.

This change requires that I modify the CMAB adaptation. Now, a round is counted as a variable number of turns delimited by the player's moving to a different room. This also increases the possibility that the player can interact with multiple distractions or with the same distraction multiple times. To prevent the possibility that the player can interact with the same distraction more than once, distractions are resolved within a single turn, removing them from play afterwards. However, the ability for the player to interact with multiple distractions is kept intact. Instead, the reward for the super arm is modified to give a reward of 1 for every distraction that was interacted with in that round.

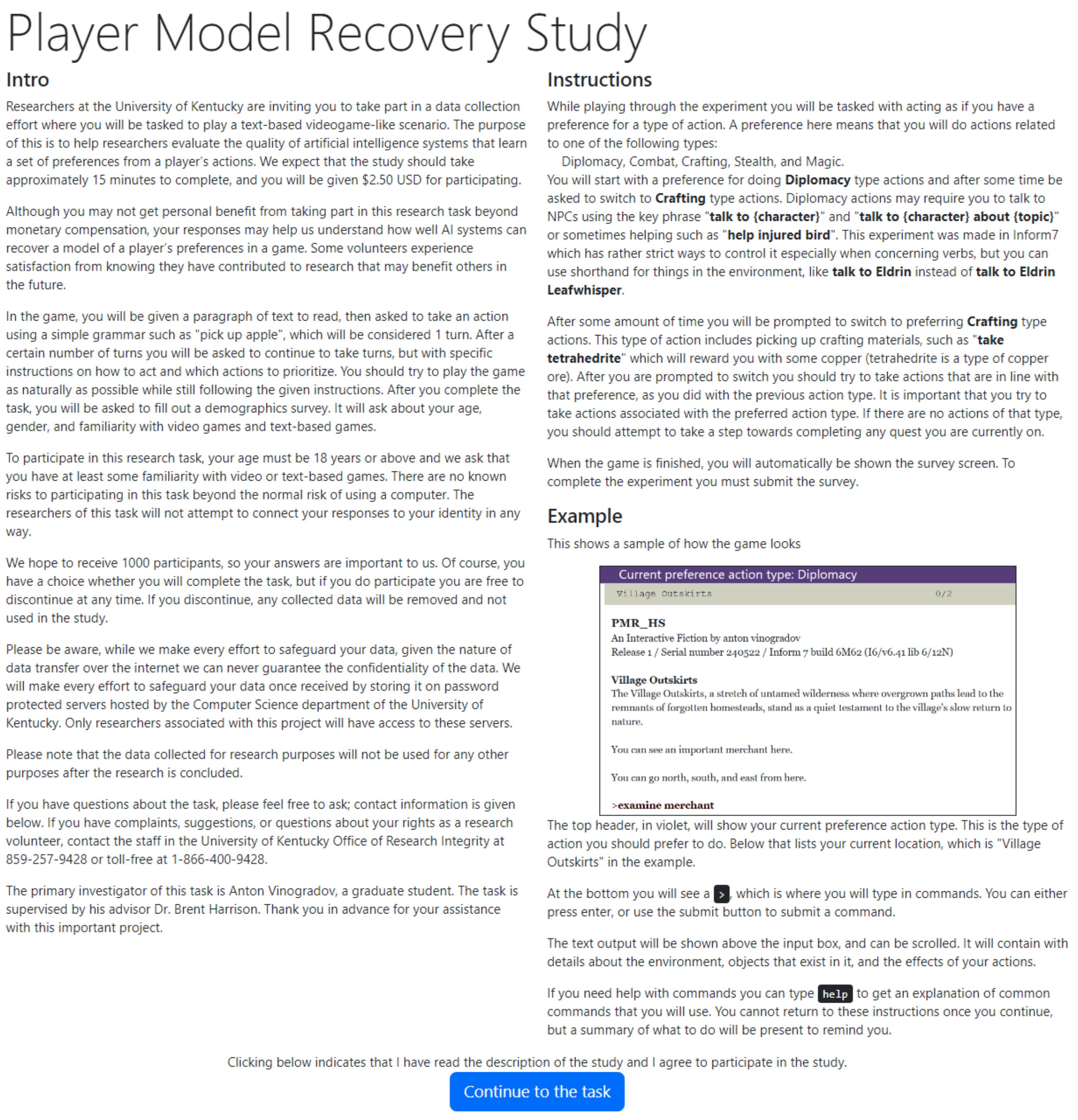
5.3 Experimental Setup

Since the experiment's goal is to verify that the same trends visible in the automated agents reflect the behavior found in human players, it largely focuses on the specific point where those trends are visible. Instead of waiting for a preference shift to happen naturally, I task participants to mimic a single Environment to Missing preference shift, with the CUCB (k=2) manager intervening after the shift. This study was reviewed and approved by the University of Kentucky's Institutional Review Board (IRB). I recruited 30 anonymous human participants over the course of 2 days on Prolific to take part in this study, only restricting the location to English speakers in the United States. Of these 30 participants, one had to be removed. This participant was able to finish the task, but the majority of actions they attempted to take caused errors as they did not use the syntax necessary for Inform7 games. This left only 22 valid actions, all but one of which was just moving around the map. The entire task, beginning with reading the instructions and ending with completing the exit survey, was designed to take 15 minutes, but the actual median time was 17.5 minutes. The shortest amount of time taken was 9 minutes and 59 seconds, and the longest was 40 minutes.

5.3.1 Task

The participant is first given a set of instructions to read (Figure 5.2) before they move on to the gameplay portion of the task. In these instructions, the participant is told to start with a preference for diplomacy type actions, with an explanation of what kinds of interactions are considered diplomacy. Afterwards, they are instructed to switch to prefer crafting type actions, though here they are not told when the switch will occur. I found that not introducing the participants to which action they should switch to would lead to some confusion on how to interact as if they had that preference, likely due to not having enough time to consider what that sort of interaction would entail. Afterward, the participant is told in more detail how to interact with the game in general, and a button leads them to the game. The participants are never informed about distractions or the existence of the manager, but they are informed that the purpose of the study is to "evaluate the quality of artificial intelligence systems that learn a set of preferences from a player's actions."

When the participant continues to the game, they are presented with the interface seen in Figure 5.3. A panel on the left provides a short summary of the instructions, which was included after some preliminary feedback. This summary and the header at the top inform the participant which action type they should currently prefer. When the participant is asked to switch their preference, an alert box appears that must be dismissed before continuing. The task itself consists of the participant playing through the game for 75 in-game turns. On turn 20, the participant is asked to switch their preference, and the manager activates five turns later, on turn 25. Since the CMAB rounds are now delimited by room changes, these 50 turns do not correspond to rounds, with a median of 17 rounds for all the valid participants. After turn 75, the screen switches to the exit survey, where the participants are asked their age group and gender and to rate their experience with video games and interactive fiction separately.



Player Model Recovery Study

Intro

Researchers at the University of Kentucky are inviting you to take part in a data collection effort where you will be tasked to play a text-based videogame-like scenario. The purpose of this is to help researchers evaluate the quality of artificial intelligence systems that learn a set of preferences from a player's actions. We expect that the study should take approximately 15 minutes to complete, and you will be given $2.50 USD for participating.

Although you may not get personal benefit from taking part in this research task beyond monetary compensation, your responses may help us understand how well AI systems can recover a model of a player's preferences in a game. Some volunteers experience satisfaction from knowing they have contributed to research that may benefit others in the future.

In the game, you will be given a paragraph of text to read, then asked to take an action using a simple grammar such as "pick up apple", which will be considered 1 turn. After a certain number of turns you will be asked to continue to take turns, but with specific instructions on how to act and which actions to prioritize. You should try to play the game as naturally as possible while still following the given instructions. After you complete the task, you will be asked to fill out a demographics survey. It will ask about your age, gender, and familiarity with video games and text-based games.

To participate in this research task, your age must be 18 years or above and we ask that you have at least some familiarity with video or text-based games. There are no known risks to participating in this task beyond the normal risk of using a computer. The researchers of this task will not attempt to connect your responses to your identity in any way.

We hope to receive 1000 participants, so your answers are important to us. Of course, you have a choice whether you will complete the task, but if you do participate you are free to discontinue at any time. If you discontinue, any collected data will be removed and not used in the study.

Please be aware, while we make every effort to safeguard your data, given the nature of data transfer over the internet we can never guarantee the confidentiality of the data. We will make every effort to safeguard your data once received by storing it on password protected servers hosted by the Computer Science department of the University of Kentucky. Only researchers associated with this project will have access to these servers.

Please note that the data collected for research purposes will not be used for any other purposes after the research is concluded.

If you have questions about the task, please feel free to ask; contact information is given below. If you have complaints, suggestions, or questions about your rights as a research volunteer, contact the staff in the University of Kentucky Office of Research Integrity at 859-257-9428 or toll-free at 1-866-400-9428.

The primary investigator of this task is Anton Vinogradov, a graduate student. The task is supervised by his advisor Dr. Brent Harrison. Thank you in advance for your assistance with this important project.

Instructions

While playing through the experiment you will be tasked with acting as if you have a preference for a type of action. A preference here means that you will do actions related to one of the following types:

Diplomacy, Combat, Crafting, Stealth, and Magic.

You will start with a preference for doing **Diplomacy** type actions and after some time be asked to switch to **Crafting** type actions. Diplomacy actions may require you to talk to NPCs using the key phrase "**talk to {character}**" and "**talk to {character} about {topic}**" or sometimes helping such as "**help injured bird**". This experiment was made in Inform7 which has rather strict ways to control it especially when concerning verbs, but you can use shorthand for things in the environment, like **talk to Eldrin** instead of **talk to Eldrin Leafwhisper**.

After some amount of time you will be prompted to switch to preferring **Crafting** type actions. This type of action includes picking up crafting materials, such as "**take tetrahedrite**" which will reward you with some copper (tetrahedrite is a type of copper ore). After you are prompted to switch you should try to take actions that are in line with that preference, as you did with the previous action type. It is important that you try to take actions associated with the preferred action type. If there are no actions of that type, you should attempt to take a step towards completing any quest you are currently on.

When the game is finished, you will automatically be shown the survey screen. To complete the experiment you must submit the survey.

Example

This shows a sample of how the game looks

```
Current preference action type: Diplomacy
Village Outskirts                                   0/2

PMR_HS
An Interactive Fiction by anton vinogradov
Release 1 / Serial number 240522 / Inform 7 build 6M62 (I6/v6.41 lib 6/12N)

Village Outskirts
The Village Outskirts, a stretch of untamed wilderness where overgrown paths lead to the
remnants of forgotten homesteads, stand as a quiet testament to the village's slow return to
nature.

You can see an important merchant here.

You can go north, south, and east from here.

>examine merchant
```

The top header, in violet, will show your current preference action type. This is the type of action you should prefer to do. Below that lists your current location, which is "Village Outskirts" in the example.

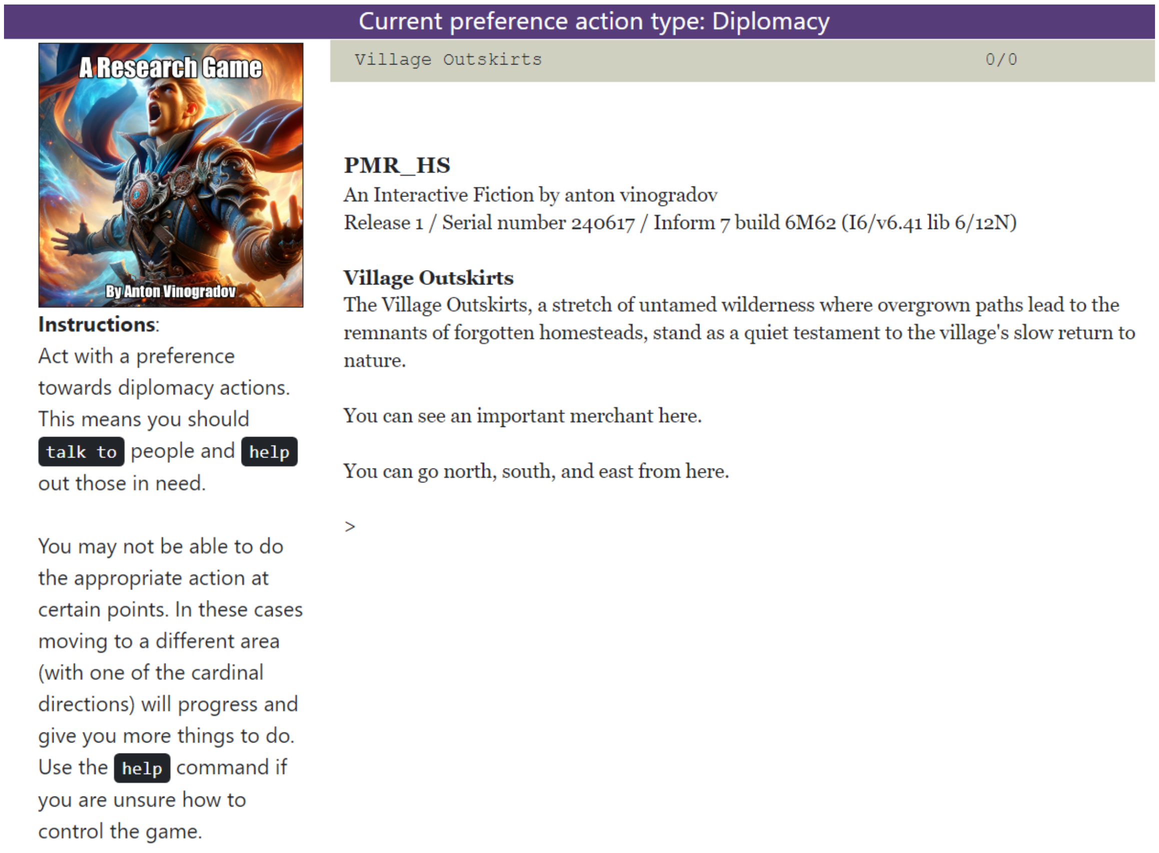
At the bottom you will see a `>`, which is where you will type in commands. You can either press enter, or use the submit button to submit a command.

The text output will be shown above the input box, and can be scrolled. It will contain with details about the environment, objects that exist in it, and the effects of your actions.

If you need help with commands you can type `help` to get an explanation of common commands that you will use. You cannot return to these instructions once you continue, but a summary of what to do will be present to remind you.

Clicking below indicates that I have read the description of the study and I agree to participate in the study.

Figure 5.2: The instructions given to the participants before they start the task. Included is a cover letter required by the IRB on the left side, the instructions for the task on the top right, and a more detailed explanation on how to play such a game on the lower right.


Current preference action type: Diplomacy

Instructions:
Act with a preference towards diplomacy actions. This means you should `talk to` people and `help` out those in need.

You may not be able to do the appropriate action at certain points. In these cases moving to a different area (with one of the cardinal directions) will progress and give you more things to do. Use the `help` command if you are unsure how to control the game.

```
Village Outskirts                                        0/0
```

PMR_HS
An Interactive Fiction by anton vinogradov
Release 1 / Serial number 240617 / Inform 7 build 6M62 (I6/v6.41 lib 6/12N)

Village Outskirts
The Village Outskirts, a stretch of untamed wilderness where overgrown paths lead to the remnants of forgotten homesteads, stand as a quiet testament to the village's slow return to nature.

You can see an important merchant here.

You can go north, south, and east from here.

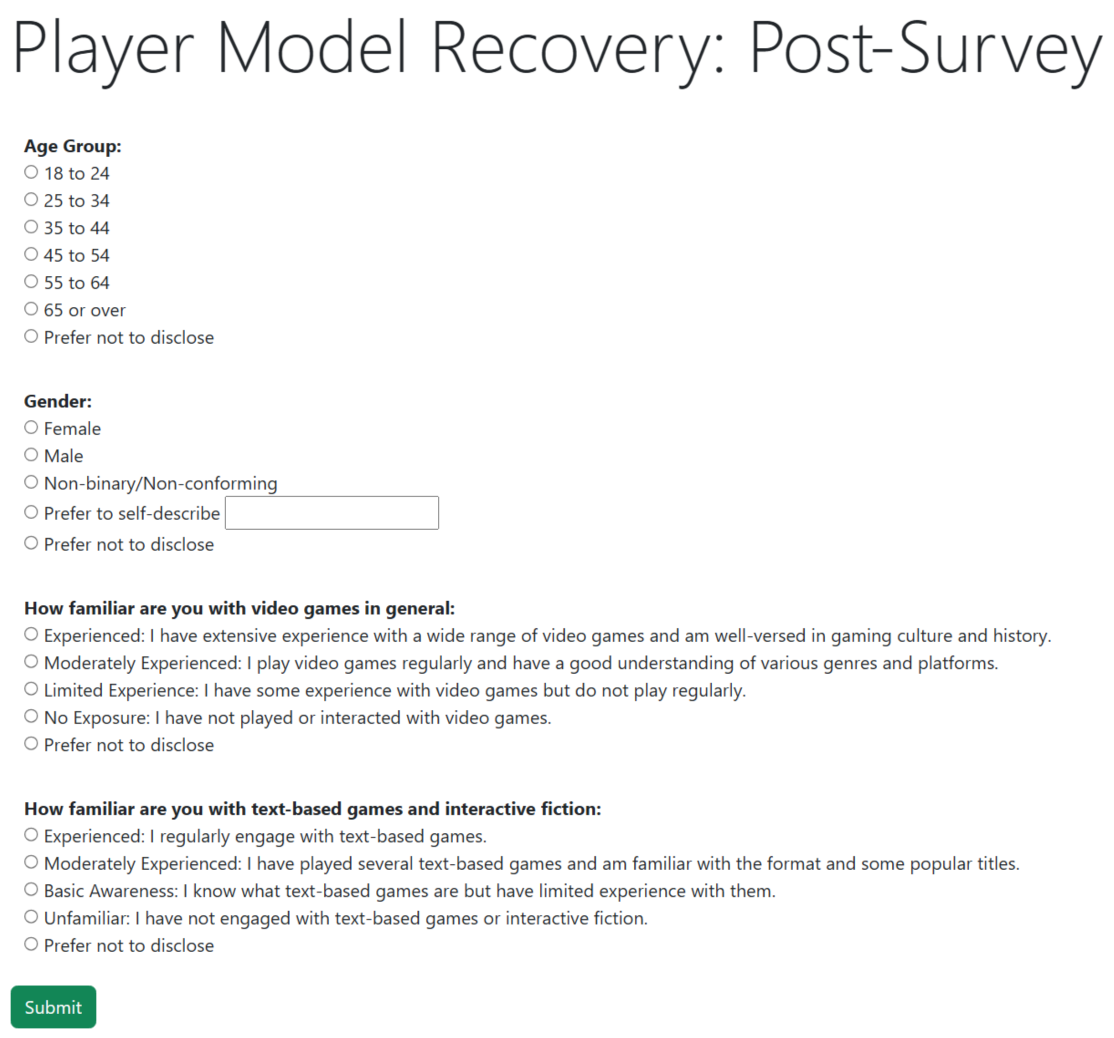
>

Figure 5.3: The interface for the game which includes an instruction summary to the left, and a header at the top reminding the participant which action their current preference should be.

5.3.2 Processing

During the task, the CMAB manager that I used was the CUCB (k=2) manager since it performed well in the automated tests previously. I did not use a larger super arm due to the worry that it may give the player too many distractions, especially without the ability to use the *replace-with-environment-action* improvement, which was left out due to technical constraints. To be able to host the experiment on the web and allow easy remote access, I had to switch from using the TextWorld[1] library to communicate with game in python to the Quixe[2] interpreter to communicate with the game in JavaScript and then communicate back to python with Pyodide[3] to interface with the manager. This limited what parts of the internal game state I could access, which no longer allowed me to automatically classify all actions for their respective types. This was worked around for distractions, as the manager could infer their type

[1]https://www.microsoft.com/en-us/research/project/textworld/
[2]https://eblong.com/zarf/glulx/quixe/
[3]https://pyodide.org/en/stable/



Player Model Recovery: Post-Survey

Age Group:
- ○ 18 to 24
- ○ 25 to 34
- ○ 35 to 44
- ○ 45 to 54
- ○ 55 to 64
- ○ 65 or over
- ○ Prefer not to disclose

Gender:
- ○ Female
- ○ Male
- ○ Non-binary/Non-conforming
- ○ Prefer to self-describe
- ○ Prefer not to disclose

How familiar are you with video games in general:
- ○ Experienced: I have extensive experience with a wide range of video games and am well-versed in gaming culture and history.
- ○ Moderately Experienced: I play video games regularly and have a good understanding of various genres and platforms.
- ○ Limited Experience: I have some experience with video games but do not play regularly.
- ○ No Exposure: I have not played or interacted with video games.
- ○ Prefer not to disclose

How familiar are you with text-based games and interactive fiction:
- ○ Experienced: I regularly engage with text-based games.
- ○ Moderately Experienced: I have played several text-based games and am familiar with the format and some popular titles.
- ○ Basic Awareness: I know what text-based games are but have limited experience with them.
- ○ Unfamiliar: I have not engaged with text-based games or interactive fiction.
- ○ Prefer not to disclose

Figure 5.4: A screenshot of the exit survey that is given to the participant. All fields are optional.

by knowing which distractions it had added.

This lack of access to some of the internal state information also meant that the actions needed to be classified afterward to calculate the recovered player model. Instead of manually going through each of the 2000+ actions, these were classified based on keyword matching, generally based on the verb used, and iteratively continuing to add keywords until every command was classified. I classified these into several categories, one for each of the five action types and additionally *move* for movement actions, and *none* for all the rest. The five action type categories mostly used the verbs related to the proper way of interacting with the distraction, but I also added extra keywords when the intent was clear (e.g. the invalid verbs “repair” and “craft” which are clearly an attempt to craft). The largest of these categories is *none* and accounts for all invalid verbs due to spelling mistakes, commands without verbs, and valid verbs like “look” and “examine” which simply do not correspond to an action type. For the player model calculation, I only used the 5 action type categories.

5.3.3 Shift Detection

The data collected from the human study was intended for Player Model Recovery, but there are portions of the data that can be used to validate the shift detection methods. The 20 turns before the participant is asked to shift their preference can be for the baseline of the shift detection methods, and the 5 turns afterward can be used to test the methods. There are not enough turns in either category, though, to fill the 50 turn windows, of which I use three: one for training, another for the gap, and the last for testing. Thus I have opted to instead to resample these to form 200 turn amalgamated human-like gameplay traces.

To do this, I combined all the turns taken by participants during the pre-preference shift into a single group and conducted the same post-preference shift. From these groups, I then uniformly randomly draw 100 turns, with replacement, to form a pre-preference section and likewise for the post-preference section. This is done 20 times to create 20 synthetic traces with 200 turns rather than 30 human traces with only 25 viable turns. I also include a modification to create no-shift traces by sampling from the pre-preference shift for both sections, adding an extra 20 traces. These human-like traces are then passed on to the preference shift detection methods as normal.

These traces were tested with the same methods as in Chapter 3: Elliptic Envelope [37], Local Outlier Factor [4], One Class SVMs [40], along with my own from Chapter 3. Each of these methods had free parameters that were fine-tuned by calculating the f1-score across all trials and choosing the parameters that resulted in the highest score. I found that my method works best with a threshold of 10% and a test window of 16. For Elliptic Envelope, I tuned the test window to 20, the threshold to 90%, and the support fraction to 0.6. For Local Outlier Factor, I tuned the test window to 20, and the threshold to 70%, and the number of neighbors to 26. For One Class SVMs, I tuned the test window to 16, and threshold to 95%, and also found that the sigmoid kernel with a kernel coefficient of auto ($\gamma = \frac{1}{n \cdot \sigma^2}$) works best.

5.4 Results and Discussion

The human study results for the PMR experiment can be seen in Figure 5.6. These results are graphed differently than the artificial agent's results, as I wanted to better demonstrate how this system may be used to decide on a recovered player model. Since it is not possible to extract a human's internal preferences with complete accuracy, I have instead compared the measured player model against 5 different primary preferences. These take the form of the agent's preferences, following the same pattern of assigning 11/15 to primary preference action types and 1/15 to the other 4 action types. Additionally, since the system is sensitive to which turn the player model measurement starts, I have opted to show it from 3 different starting points: the beginning of the experiment, the preference shift turn, and the distraction start turn.

The results of the PSD study can be seen in Table 5.2. This table shows the results in a format similar to Chapter 3's Tables 3.1 and 3.2, though are combined into a single table. I report the turn when a shift is detected and the standard deviation for each method when a shift is detected. I bold the best-performing turns, the earliest turn when there are shifts, and the latest turn when there is no shift. I also report the accuracy and standard deviation for each method split on shift and no shift, along with an F_1-Score, also bolded to show best performing. The rest of this section will discuss these results, starting with an analysis of the exit survey.

5.4.1 Survey

The survey results, summarized in Figure 5.5, show that all the participants had some experience with video games, though 5 participants had no experience with text games. The participants overall had much less experience with text games, but women were more likely to have experience in them than men. Surprisingly, more female participants rated themselves as experienced in video games than male participants, while the male participants tended to rate themselves as moderately experienced. In general, there were more female participants than males, with only a single participant identifying as non-binary. The age range was heavily skewed towards the 25-34 range, but over a third of the participants were older than 35.

While not shown in the survey results figure, I found that gender identity and video game experience had little to no effect on how easy the study was for the participant. While I do not have definitive evidence for this claim, by analyzing the individual traces of participants, I found that around half of the participants struggled with the game regardless of gender and game experience, with only those who rated more experience with text games having less trouble. This is most likely due to some of these participants already being familiar with the fairly restrictive grammar that the game requires for proper play. However, this advantage only lasted the first couple of turns, after which there were few mistakes due to the grammar.

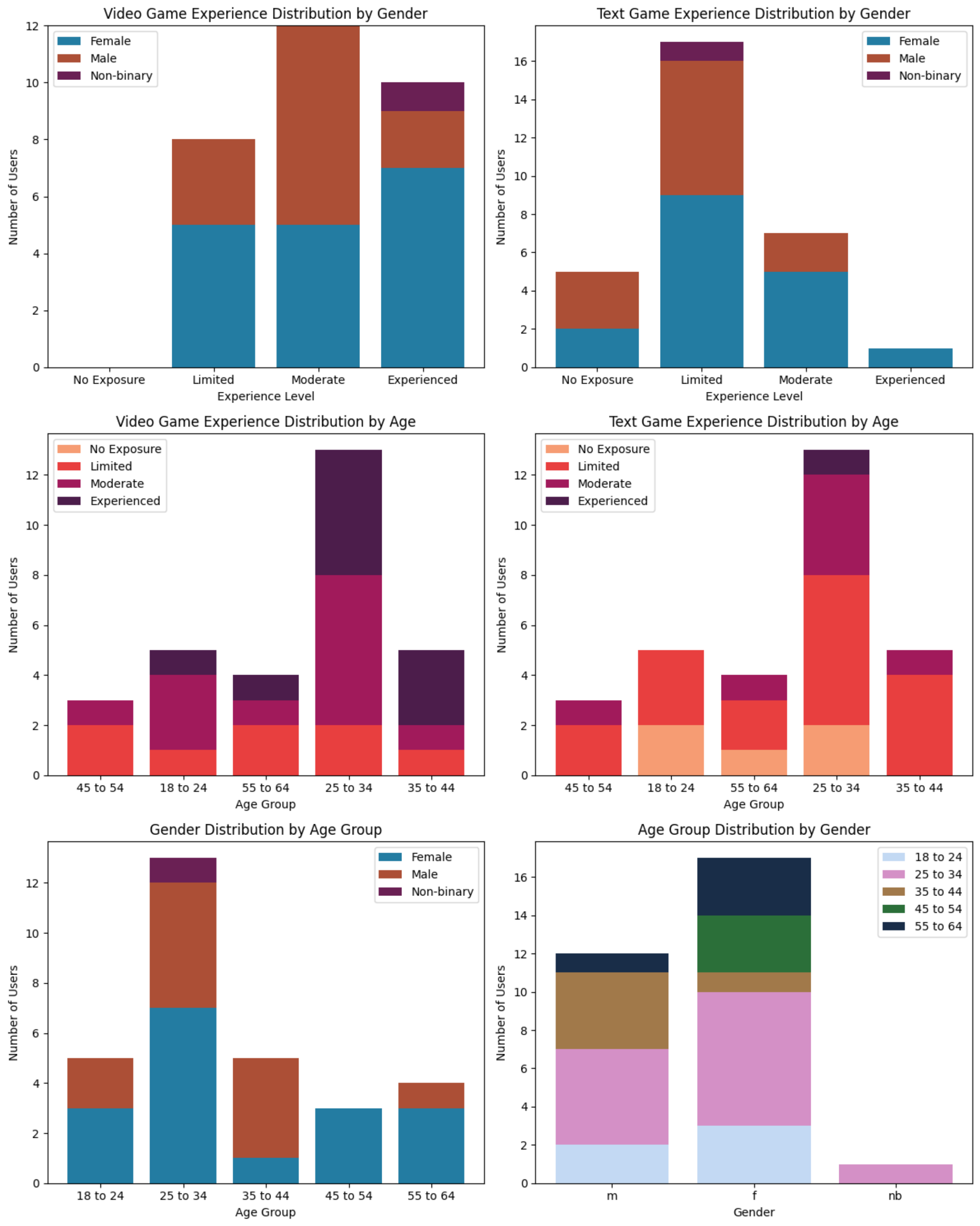


Figure 5.5: Various views of the survey results

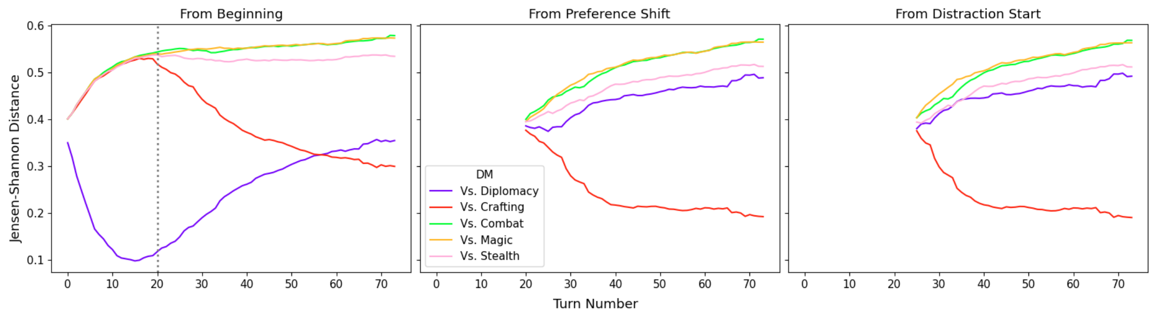


Figure 5.6: Mean JS Distance between Player Model and 5 Primary Preference Models. A vertical dotted line marks where the player is asked to shift their preferences.

5.4.2 Player Model Recovery

The results indicate that human subjects behave similarly to the artificial agents. The shift from diplomacy to crafting is considered an Environment to Missing preference scenario as diplomacy is the focus of the two quests that are present in the game and crafting is missing, which means I can compare these results to the previous chapter's Environment to Missing Figure 4.4. This scenario was specifically chosen for the human study to match what I had previously identified as the most likely scenario to be applicable to this system and allow for this comparison.

I expect that as soon as distractions are available (turn 25), I should see a sharp drop in the distance between the measured player model and a primarily Crafting model, while the distance to other models steadily rises starting from when the preference shift occurred (turn 20). This trend exists when measuring from the beginning, but there is an additional small immediate drop. This is due to some players attempting to take crafting actions immediately, even though the environment does not allow for it. I also expect the measured distance trend to eventually flatten out, as observed in all three plots.

The results are still a bit misleading in their comparison as they are graphed against the turn count, which in the artificial agents is synonymous with the CMAB round but not so in the human study. This means that when combining the measurements between different participants, the boundary of the rounds is not well defined. Given that each participant played a total of 75 turns and that there is a variable amount of rounds, the only way to collate these results would be on turn but this does mean that a more proper comparison would be to compare the human results to the first 20 or so turns for automated agents. When put into this perspective, the human results seem to be considerably better, especially when the measurement starts from the preference shift turn. The Vs. Crafting measurement flattens out around turn 40, only 15 turns after the manager activates, compared to the agents, who take many more turns and even more rounds to show similar results. Since CUCB requires a 5 round start up portion to play a super arm for each of the 5 possible arms, this flattening is likely due to the participant trying to take crafting actions before they

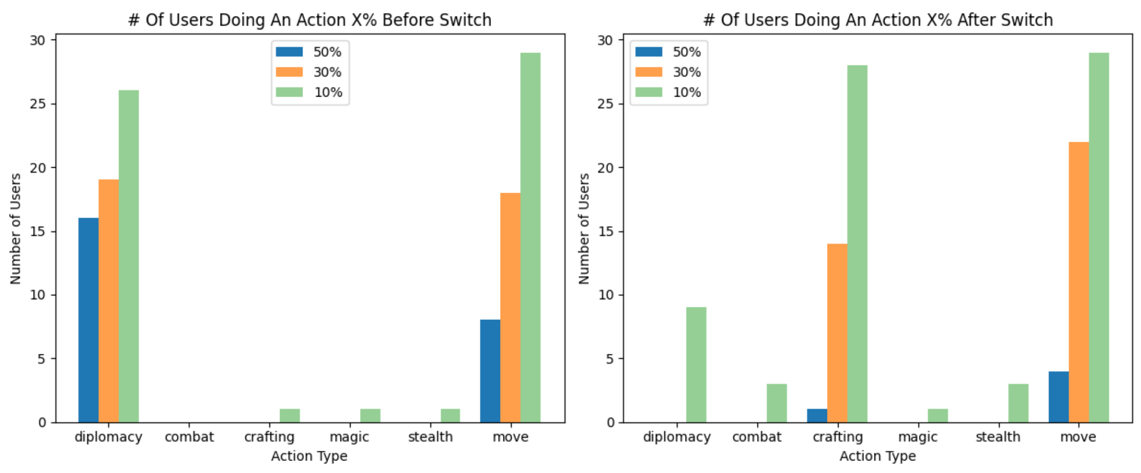


Figure 5.7: The number of participants that did an action type X% of the time, before and after the preference shift.

are possible.

The behavior of participants attempting to do actions that are not possible is due to how I have opted to manually classify what type of action a human commands. Instead of directly examining the type of action as reported by Inform, as I can for my automated tests, I classify actions based on the keyword that was used in the command. This allows me to classify actions that are not actually possible in the game, so it still counts as a crafting action when a participant attempts to "craft" something. This was done because I found in preliminary tests that many participants struggled to navigate the interface early on in the experiment but still wanted to capture the intent of the commands entered instead of throwing them out. It is unlikely that this sort of action intention interpretation is possible in non-text based games, but it is well suited to Inform7. Instead, for games that have more limited means of interacting with the game, other methods of observation may be possible, like measuring the time between actions or if any meaningful progress is made towards a goal.

This form of measuring the player's model is also sensitive to when the system starts measuring, as can be seen in the difference between the graphs in Figure 5.6. Starting before the preference shift happens makes it quite difficult to distinguish when a model can be considered recovered, as the actions taken before the shift pollute the measurement. In the From Beginning graph, this shows as the *Vs. Crafting* line only falling below the *Vs. Diplomacy* line around turn 55, already 30 turns after the manager has activated. Compared to the From Preference Shift graph, the difference is immediately noticeable, while starting even later, From Distractions Start seems not to affect this pattern. This suggests that to be able to accurately decide what the new player model should be, contaminated data is especially harmful, and measurement should err on the side of starting later.

There is some evidence that some participants act in manners similar to the artificial agents. Informed by Figure 5.7, most participants that show some similarity to the agents are likely to be classified close to exploration or goal oriented, though one participant seemed similar to the novelty focused agent. Participants who acted in an exploration focused manner preferred to move around the map rather than interact with the environment. The number of participants performing move actions is high, and this is expected as it is required not only to progress in the quests but also to find new distractions. Thus, I only consider those who moved more than 50As per the figure, 8 participants chose a move action more than 50% of the time before the preference switch, though only 4 did so afterwards. This value is lower after the switch, likely due to the introduction of distractions. A distraction offers the player something to interact with, even in rooms that only serve as passageways to objectives and have no objects. A few of these participants sometimes found their way to some of the far corners of the map, such as the castle, where there are few objects to interact with outside of any generated distractions and a couple combat items in the castle armory.

I would categorize participants as similar to the goal oriented agent when they continue to try to complete the quests, thereby performing diplomacy actions, even after being asked to switch their preferences. A total of 9 participants continued to do a noticeable amount of diplomacy actions after the preference switch, but only doing that action type 10% of the time, or greater than 5 times in the 55 turns after the preference switch. These participants may not have been enticed enough with the crafting distractions, choosing instead to interact with available environment actions, or may have been more attracted to the diplomacy distractions. One of these participants was identified as even ignoring nearly all distractions, opting to instead continue completing the quests. Some of the diplomacy distractions were about helping animals and there were cases of participants helping them suggesting that perhaps these distractions are too enticing.

Of all the participants, only one could be considered very novelty-focused. This participant reported in the survey that they were very experienced with video games and moderately experienced with text games. In their trace, they interacted with many of the distractions that they encountered. I suspect that this is due to the distractions having unique outcomes and all giving some reward when interacted with. There are 129 different distractions, making it impossible to see them all, and the novelty of their interactions and outcomes likely contributed to this participant's preference to interact with them. This does mean that this kind of participant is more difficult for the system and the recovery of their player model may be less accurate or take more time, but this is also true of the Novelty focused agent.

5.4.3 Preference Shift Detection

The results of the preference shift detection indicate that the method works even on human like data. In fact it looks like the synthetic human-like data works considerably better with all methods, with the highest scoring overall F_1-score in Table 3.1 of 0.72 being beat out by a 0.98 score in Table 5.2, with both belonging to my method. The tests using in Chapter 3 are done on all possible scenarios, while here I am focused on environment to missing, so this comparison is not completely accurate. Regardless, comparing the individual performance of the environment to missing and the environmental no shift categories shows that on synthetic human-like data these methods perform much better.

Part of this is likely due to the way that these methods work, establishing a baseline in the pre-preference and deciding whether the post-preference data is different. Since I had to increase the size of pre and post-preference turns from 25 and 5 to 100 and 100 respectively, they are no longer strictly ordered temporally with only the pre and post-preference sections being in order. Thus the novelty detection methods are comparing the entirety of the pre-preference turns (upsampled to 50 to fit the entire pre-preference window) to the entirety of the post-preference turns (upsampled to up to 20 turns) since the order of turns is ignored when synthesizing the human-like data. Since these methods works as a sliding window this means that there is little difference when the method slides forward a turn, with the sliding just giving that method another to detect the shift. This increase in the number of chances to detect without significantly changing the underlying data gives them a higher chance to detect a shift.

For my method works a bit differently as it does make use of the gap window. Instead of completely ignoring it like the novelty detection methods it uses it as a comparison to the first 50 turn window, creating a mask that lets it ignore action types that did not increase in the comparison window. Since the comparison window will contain more of the post-preference actions it will show that the types of actions present in the post-preference have increased, and likewise see that those actions are steady in the test window leading it to detect a shift. In more standard human data this is not necessarily a pattern that we expect to continue for longer periods of time and as such this may mean that my method has an unfair advantage compared to the rest.

Despite these potential flaws in the synthetic data, I also see better performance in the no shift category as well, meaning that these have a better true negative rate. This further suggests the difference between the pre and post-preference turns is significant in the shift category even though the post-preference draws from 29 different participants over 5 turns. When drawing from the same 20 pre-preference turns for the no shift for both the pre and post-preference sections then the actions taken do not have enough variability to be registered as a shift in most cases. This is promising since it indicates that there is enough difference in human behavior before and after a preference shift, and this difference seems common enough between human participants that it is able to be found even when the data from multiple participants is combined together.

A notable difference between the artificial agents and the synthetic human-like traces are that the amount of turns needed to find a preference shift is higher in the human-like traces compared to the agents for the Environment to Missing scenario group. With the artificial agents the maximum average turn to find a preference shift did not exceed 133.4, though for many methods it was lower. With the human-like traces the minimum average turn was 129.6, with the EE method taking an average of 139.8 turns. I suspect this is largely due to the optimization of the parameters,

	EE	LOF	SVM	Mine
Shift Turn	139.8±12.9	132.2±04.4	**129.6±08.5**	133.4±06.0
No Shift Turn	156.0±00.0	139.7±10.7	142.2±12.5	**168.0±00.0**
Shift Accuracy	0.95±0.22	**1.00±0.00**	**1.00±0.00**	**1.00±0.00**
No Shift Accuracy	0.90±0.31	0.85±0.37	0.75±0.44	**0.95±0.22**
F_1-Score	0.93	0.93	0.89	**0.98**

Table 5.2: The average turn that a shift is detected on when it is detected, and the accuracy with which it is detected. Bolded are the best performing for each category.

specifically the size of the test window. Previously I have identified that with a minimally sized test window of 1 my method could decrease the amount of time it took detect a shift at a slightly detriment to its accuracy. For the current optimized parameters it seems the inverse holds, since the test window size is higher the amount of turns needed is also higher.

One of my previous conclusions was that the size of the gap window may be decreased or that the sliding training window may be replaced with a trusted non-sliding set of turns to compare against. While it may still be advantageous to change the nature of the training window, in the current formulation a larger gap window may actually serve an advantage because of the increased amount of turns needed to detect a preference shift. With a smaller window there would be less chances to test for a preference shift before some observations start contaminating the training window, which may suggest tuning the size of the gap window may be necessary to fully optimize the performance.

The fact that these optimized parameters are so different with the synthetic data compared to those optimized for the artificial agents is itself an indication that the underlying data that is gathered may be very different. Part of this could be due to using human-like data but additionally the environment for the human study has only passing resemblance to that which was used by the agents. It may be that for different player bases or different games may be dissimilar enough to require re-optimization. I further suspect that small changes to game balance or feature improvements may require that these parameters be re-optimized on a sample of the player base, which may be costly for a developer.

Overall this does suggest that these methods work, but I can only draw limited conclusions on their efficacy given the limitations of the synthesized data. This data was reused from the PMR experiment so serves as a good addition, but it is comprised from fairly small set of turns, with only 5 turns per participant for the post-preference section. To be able to properly conclude that the PSD methods work on humans another experiment would have to be run with an additional human study.

5.5 Conclusion

In this chapter I have shown how I have adapted my environment to be compatible with human players, and the discoveries that were made in that process. I have added the additional requirement on distractions that they must be *readable*, or being recognizable as belonging to a specific preference to be useful to the ExpM to recover the player model. Afterwards I have also deployed this environment in the form of a human study to collect data to validate its efficacy. Previously this was only shown on artificial agents but with this human study it has also been shown to work with humans.

Additionally I have reused part of the data to create synthesized human-like traces, which allowed me to further show that the preference shift detection also works with humans. Since this is synthesized *human-like* data, I cannot claim for certain that this will continue to be the case with true human traces, but the system has been shown to be successful with both artificial agents and this data. For future work human gameplay traces can be used to further validate these conclusions.

Chapter 6 Conclusion

Traditionally video games have asked for the player to adapt to the game, but this need not be the case. Experience managed games can adapt themselves to the player, allowing for a better sense of player agency and lead to more personalized experiences, while still maintaining authorial intent. This adaptation is driven by its knowledge of the player and their preferences, but it has been shown that players may shift their preferences over time [48]. This shift in preferences can show up as a sudden shift in behavior but conventional experience management systems work under the faulty assumption that the player's preferences stay relatively static. The consequences of this assumption are that experience management systems that model the player cannot adapt when the player shifts their preferences, which could lead the player to become less engaged with the game. To solve this issue I have devised 3 research tasks:

- RT1: Recognize when a player's preferences have shifted and that the current environment may no longer be well suited.
- RT2: Recover a model of the player's preferences after a player preference shift has been detected.
- RT3: Verify that these methods work in human subjects.

I will go over how these were completed and what conclusions I can draw from each below.

6.1 RT1: Preference Shift Detection

To tackle RT1 I have created a system that is capable of detecting player preference shifts, and tested 3 existing methods and 2 variations of my own on data gathered from artificial agents. The existing techniques are based on novelty detection and are a good fit as I can model observations of the player's action as streaming data. This allows for some of the data to be considered clean, using it as a baseline to compare against incoming action observations and classifying them as novel (indicating a preference shift) or not. In addition I have created my own method which takes inspiration from how a preference shift should look. I find that my method works the best but that its advantages are slight.

As a purely information gathering technique this system is safe to use as it does not make any changes to the environment, instead opting to make observations on the player's behavior and using that as a proxy for their internal, and thus normally unmeasurable, preferences. When tested only on artificial agents I found that there was some success in its ability to detect preference shifts, but the methods were farm from perfect in all scenarios.

Some scenarios are likely rare to be found in games and are largely possible to test due to the use of artificial agents. One group of scenarios that are particularly difficult for this system are when switching from an action type that is missing from the environment to another missing action type. These scenarios are indistinguishable from each other due to the lack of preferred actions and are functionally the same as scenarios where no preference shift occurs. Another of these, the missing no shift scenario group, if found in a game would likely require that the following process of player model recovery occur regardless as it is likely that the player is already unengaged.

When compared to more common scenarios there was no one method that could be considered best, though some had advantages over others. This suggested that an ensemble of these methods would be ideal, but later human subjects experiments may suggest that this is not necessary. Overall these methods are reasonably capable of detecting when a shift occurs and can do so quickly enough to be considered a success.

6.2 RT2: Player Model Recovery

For RT2 I have developed another system, this time taking advantage of Multi-Armed Bandits (MABs) to balance between finding information on the player and making use of that information. This system takes active intervention in the form of adding and removing new game objects called *distractions* to the environment, no longer just observing the player silently and allowing it to find what the player's new preferences are. In the first half of the chapter I show that with normal MABs the manager can quickly recover the player model, though the restriction of only adding a single distraction per round is very limiting. This formulation of the system still performs considerably better than the comparison baseline and even performs better than the manager that solely acts to remove bias from the environment.

In the second half I introduce Combinatorial Multi-Armed Bandits (CMABs) into the system, making slight modifications on the MAB adaptation to do so. These perform significantly better, even approaching the performance of my unrealistic theoretical best baseline, the One-Of-Each manager which removes the restriction on how many distractions can be added per round. Furthermore I find that a small modification, called replace-with-environment-action, allows for not only fewer than an average of 2 distractions per round, but also increases the performance of the 2 distraction variant to that of the 3 distraction variant. Overall this method are incredibly capable of recovering a players preferences, but so far had only been tested in artificial agents.

6.3 RT3: Human Study Verification

RT3 consisted of a human study which was used to verify the conclusions in the previous two research tasks. To be able to validate these conclusions the previous test environment needed to be retooled to be accessible for human participants, which required extensive modification and the formulation of extra requirements on distractions to make them suitable with human use. The human study was centered around testing the player model recovery system from RT2 and tasked the human participants to simulate a preference shift from an environmental action type to a missing action type. This data was then analyzed and I found that indeed player model recovery does work with human data. In addition there was evidence that the human subjects, without being prompted, took on behaviors that were similar to the artificial agents.

In addition part of the human data was reused and combined to create synthesized human-like gameplay traces that could be used with the preference shift detection methods. I found that some of the previous conclusions on preference shift detection were called into question due to the methods working significantly better than on the artificial agents. After optimizing the parameters for each of the 3 novelty detection methods and my own method I found that preference shifts could be detected on this data with a 100% true positive rate, and a 95% true negative rate, which showed a massive improvement from the artificial agents. Despite these excellent results the synthesized data is not a true representation of how humans would act in this situation and thus only serves to show that the method is promising.

6.4 Summary

In this work I have developed new methods that allow for the recovery from and accommodation for player preference shifts in a player modeled experience management environment. Each step in this two step process has been shown to work with artificial agents and later with human subjects and human-like player traces. In future work this system can be integrated together and deployed with an existing player modeled experience manager to show its validity. Additionally there are improvements that can be made to preference shift detection as it is further tested with human generated data.

Bibliography

[1] J.-Y. Audibert, S. Bubeck, and G. Lugosi. Minimax policies for combinatorial prediction games. In *Proceedings of the 24th Annual Conference on Learning Theory*, pages 107–132. JMLR Workshop and Conference Proceedings, 2011.

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

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

[4] M. M. Breunig, H.-P. Kriegel, R. T. Ng, and J. Sander. Lof: identifying density-based local outliers. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, pages 93–104, 2000.

[5] Capcom. Resident Evil 4, 2005.

[6] M. Cavazza, D. Pizzi, F. Charles, T. Vogt, and E. André. Emotional input for character-based interactive storytelling. 2009.

[7] W. Chen, W. Hu, F. Li, J. Li, Y. Liu, and P. Lu. Combinatorial multi-armed bandit with general reward functions. *Advances in Neural Information Processing Systems*, 29, 2016.

[8] W. Chen, Y. Wang, and Y. Yuan. Combinatorial multi-armed bandit: General framework and applications. In *International conference on machine learning*, pages 151–159. PMLR, 2013.

[9] M.-T. Cheng, H.-C. She, and L. A. Annetta. Game immersion experience: its hierarchical structure and impact on game-based science learning. *Journal of computer assisted learning*, 31(3):232–253, 2015.

[10] T. M. Connolly, E. A. Boyle, E. MacArthur, T. Hainey, and J. M. Boyle. A systematic literature review of empirical evidence on computer games and serious games. *Computers & education*, 59(2):661–686, 2012.

[11] M. Csikszentmihalyi. The flow experience and its significance for human psychology. *Optimal experience: Psychological studies of flow in consciousness*, 2:15–35, 1988.

[12] E. S. de Lima, B. M. Silva, and G. T. Galam. Adaptive virtual reality horror games based on machine learning and player modeling. *Entertainment Computing*, 43:100515, 2022.

[13] M. S. El-Nasr. Interaction, narrative, and drama: Creating an adaptive interactive narrative using performance arts theories. *Interaction Studies*, 8(2):209–240, 2007.

[14] Graham Nelson. Inform 7, 2006. Interactive fiction development system.

[15] R. C. Gray, J. Zhu, D. Arigo, E. Forman, and S. Ontañón. Player modeling via multi-armed bandits. In *Proceedings of the 15th international conference on the foundations of digital games*, pages 1–8, 2020.

[16] R. C. Gray, J. Zhu, and S. Ontañón. Multiplayer modeling via multi-armed bandits. In *2021 IEEE Conference on Games (CoG)*, pages 01–08. IEEE, 2021.

[17] B. Harrison and D. Roberts. Analytics-driven dynamic game adaption for player retention in a 2-dimensional adventure game. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 10, pages 23–29, 2014.

[18] R. Hunicke. The case for dynamic difficulty adjustment in games. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 429–433, 2005.

[19] M. Jennings-Teats, G. Smith, and N. Wardrip-Fruin. Polymorph: dynamic difficulty adjustment through level generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, pages 1–4, 2010.

[20] I. Khaliq and Z. Watson. The omni framework: a destiny-driven solution to dynamic quest generation in games. In *2018 IEEE Games, Entertainment, Media Conference (GEM)*, pages 306–311. IEEE, 2018.

[21] R. Khoshkangini, S. Ontanón, A. Marconi, and J. Zhu. Dynamically extracting play style in educational games. *EUROSIS proceedings, GameOn*, 2018.

[22] K. Y. Kristen, M. Guzdial, N. R. Sturtevant, M. Cselinacz, C. Corfe, I. H. Lyall, and C. Smith. Adventures of ai directors early in the development of nightingale. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 18, pages 70–77, 2022.

[23] S. Kulinski, S. Bagchi, and D. I. Inouye. Feature shift detection: Localizing which features have shifted via conditional distribution tests. *Advances in neural information processing systems*, 33:19523–19533, 2020.

[24] T. L. Lai and H. Robbins. Asymptotically efficient adaptive allocation rules. *Advances in applied mathematics*, 6(1):4–22, 1985.

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

[26] M. Mateas and A. Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game developers conference*, volume 2, pages 4–8, 2003.

[27] M. M. Moya, M. W. Koch, and L. D. Hostetler. One-class classifier networks for target recognition applications. *NASA Sti/Recon technical report N*, 93:24043, 1993.

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

[29] M. J. Nelson and M. Mateas. Another look at search-based drama management. In *AAMAS (3)*, pages 1293–1298, 2008.

[30] H. L. O'Brien and E. G. Toms. 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, 2010.

[31] P. Oza, H. V. Nguyen, and V. M. Patel. Multiple class novelty detection under data distribution shift. In *Computer Vision–ECCV 2020: 16th European Conference, Glasgow, UK, August 23–28, 2020, Proceedings, Part VII 16*, pages 432–449. Springer International Publishing, 2020.

[32] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko. A review of novelty detection. *Signal processing*, 99:215–249, 2014.

[33] A. Ramirez and V. Bulitko. Telling interactive player-specific stories and planning for it: Asd+ passage= past. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 8, pages 173–178, 2012.

[34] M. O. Riedl and C. León. 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*, pages 8–10, 2008.

[35] M. O. Riedl, A. Stern, D. Dini, and J. Alderman. Dynamic experience management in virtual worlds for entertainment, education, and training. *International Transactions on Systems Science and Applications, Special Issue on Agent Based Systems for Human Learning*, 4(2):23–42, 2008.

[36] Rockstar Studios. Max Payne 3, 2012.

[37] P. J. Rousseeuw and K. V. Driessen. A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, 41(3):212–223, 1999.

[38] J. P. Rowe, L. R. Shores, B. W. Mott, and J. C. Lester. A framework for narrative adaptation in interactive story-based learning environments. In *Proceedings of the intelligent narrative technologies III workshop*, pages 1–8, 2010.

[39] J. L. Sabourin and J. C. Lester. Affect and engagement in game-basedlearning environments. *IEEE transactions on affective computing*, 5(1):45–56, 2013.

[40] B. Schölkopf, R. C. Williamson, A. Smola, J. Shawe-Taylor, and J. Platt. Support vector method for novelty detection. *Advances in neural information processing systems*, 12, 1999.

[41] H. Schønau-Fog and T. Bjørner. "sure, i would like to continue" a method for mapping the experience of engagement in video games. *Bulletin of Science, Technology & Society*, 32(5):405–412, 2012.

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

[43] M. Sharma, S. Ontanón, C. R. Strong, M. Mehta, and A. Ram. Towards player preference modeling for drama management in interactive stories. In *FLAIRS*, pages 571–576, 2007.

[44] A. Slivkins et al. Introduction to multi-armed bandits. *Foundations and Trends® in Machine Learning*, 12(1-2):1–286, 2019.

[45] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma. Adaptive game ai with dynamic scripting. *Machine Learning*, 63:217–248, 2006.

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

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

[48] J. Valls-Vargas, S. Ontanón, and J. Zhu. Exploring player trace segmentation for dynamic play style prediction. In *Proceedings of the AAAI conference on artificial intelligence and interactive digital entertainment*, volume 11, pages 93–99, 2015.

[49] Valve Corporation. Left 4 Dead, 2008.

[50] Valve Corporation. Left 4 Dead 2, 2009.

[51] H. Yu and M. Riedl. Data-driven personalized drama management. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, volume 9, pages 191–197, 2013.

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

[53] M. Zohaib. Dynamic difficulty adjustment (dda) in computer games: A review. *Advances in Human-Computer Interaction*, 2018(1):5681652, 2018.

Vita

Anton V. Vinogradov

Place of Birth:

- St. Petersburg, Russia

Education:

- University of Kentucky, Lexington, KY
  M.A. in Computer Science, Dec. 2022
- University of Kentucky, Lexington, KY
  B.S. in Computer Science, Dec. 2016

Professional Positions:

- Graduate Teaching Assistant, University of Kentucky Fall 2019
- Graduate Research Assistant, University of Kentucky Spring 2020 - Spring 2024

Publications & Preprints:

- Vinogradov, Anton, and Brent Harrison. "Using multi-armed bandits to dynamically update player models in an experience managed environment." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 18. No. 1. 2022.
- Vinogradov, Anton, and Brent Harrison. "Using multi-armed bandits to dynamically update player models in an experience managed environment." Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. Vol. 18. No. 1. 2022.
- Vinogradov, Anton, Andrew Miles Byrd, and Brent Harrison. "Mistake Captioning: A Machine Learning Approach For Detecting Mistakes and Generating Instructive Feedback." Proceedings of the International Conference on Recent Advances in Natural Language Processing (RANLP 2021). 2021.