Research Proposal: Online Adaptive Content Personalisation

Supervisor: Benjamin Rosman

Author: Jonathan Wasson 463117

November 18, 2014

UNIVERSITY OF THE WITWATERSRAND

SCIENTIA ET LABORE

JOHANNESBURG

Abstract

Personalisation is starting to play a major role in modern software and hardware. Software that can adapt to the specific user's needs and preferences is more marketable as it has a broader range of uses. An example of this would be the current generation of computer games. Games that can provide new content to the player based on their skill level would target a larger audience, by targeting challenging game-play to skilled players and alternatively content which will not be frustratingly difficult to new or unskilled players. Content could also scale to match the player's enjoyment such that they remain engaged for as long as possible, perhaps through increasing difficulty which scales appropriately.

For example, a game such as 'Temple Run' (which is a popular game on smart-phones) may be able to infer the user's ability and then adapt the games overall difficulty accordingly. Perhaps even infer the user's current enjoyment in the game and actively provide different content, such as unique goals or events, to maintain the user's enjoyment.

This research proposal proposes an experiment which will involve human users playing a game that uses an adaptive algorithm to optimize the user experience and enjoyment. The results of the experiment will provide insight into the usage of an adaptive algorithm which generates content in real-time (online).

Contents

1 Introduction **1**

2 Background and Related Work **2**

2.1 Introduction . . . 2
2.2 Significance . . . 2
2.2.1 Uses of Adaptive Content . . . 2
2.2.2 Viability of Adaptive Content . . . 4
2.3 Markov Decision Processes . . . 5
2.4 Conclusion . . . 8

3 Research Method **9**

3.1 Introduction . . . 9
3.2 Motivation and Hypothesis . . . 9
3.3 Overview of Method . . . 10
3.3.1 Programming Languages . . . 10
3.3.2 Phase 1: Game Design . . . 10
3.3.3 Phase 2: Testing . . . 11
3.3.4 Phase 3: Algorithm Design . . . 12
3.3.5 Phase 4: Experimental Analysis . . . 13
3.4 Conclusion . . . 14

4 Results **15**

4.1 Introduction . . . 15
4.2 Phase 1: Data collection . . . 16
4.3 Phase 2: Adaptive algorithm . . . 16

5 Results **17**

5.1 Future Work . . . 19

6 Conclusion **20**

7 Bibliography **21**

1 Introduction

As computers begin to encroach on every aspect of everyday life, it has become more and more apparent that the diversity of people using computers is also increasing. This dependence on technology permeates most modern businesses and households. From doctors to store clerks, most will have to make use of a computer at some point in their everyday lives [Beaudry and Pinsonneault 2005]. Take doctors for example; Doctors must make use of software that can keep track of patient records as well as machinery that performs complicated procedures on patients. New technology has been developed that allows surgeons to operate over long distances through the use of the internet and augmented reality, allowing countries without the necessary skilled labour to import it from overseas almost instantaneously in cases of emergency.

However, this progress has left a few behind. Older people, the disabled and often those who are not technologically savvy are struggling to adapt to the rigours of modern life. Nowadays, we can see old people who would have once refused to use things such as email or phones are now being forced to use these services and devices on a daily basis. Disabled people are also being forced to adapt to this change since most devices are designed in a way that is not suitable for their use[Gajos *et al.* 2008]. Indeed, this ever increasing reliance on technology has forced people to adapt to modern times and it can be said that the modern worker must be able to use a computer or face the risk of unemployment.

The research described in this document will focus on content adaptation in computer games performed in real-time. As was said previously, content adaptation has a wide variety of uses but we have decided to use it in computer games. This was done due to the relative ease at which an experiment could be created and conducted in conjunction with a computer game. There are a wide variety of computer games that can be picked from which will give us room to mould the game into something that will yield better results with little room for confusion. In performing this research it is necessary to provide examples of where the research can be implemented, which there is an abundance of. In doing this we will show the impact and commercial aspect of this research as a whole.

To justify this as a research topic we must analyse all available media that provides insight into this topic. The following paper will discuss material relevant to this question and then use it to justify this as a research topic. We will begin by reviewing relevant research that has been conducted in the field of content adaptation and then tying this in to what we aim to achieve. Following this, we will lay out the experiment we performed and the results that we gained from said experiments. Then in conclusion we will discuss the overall impact of the results and any work that could follow.

The rest of this document is structured as follows: Chapter 2 will provide an overview of related work and its relevance to the work we aim to achieve. Section 2.2 provides the significance of the problem. Section 2.3 will give the reader an understanding of Markov Decision Processes and how this relates to the problem. Chapter 3 describes how the research will be done with section 3.2 providing the motivation for the choice in method. Section 3.3 is broken up into several smaller subsections, each explaining a step in the research process. Reasons and explanations will be given through out this chapter as to why each step of the experiment is conducted. Chapter 4 will then go on to explain what results we have obtained from this research, why this is relevant and hopefully what future work can be done in this field. Finally, chapter 5 concludes the report with a summary and a brief overview of the research.

2 Background and Related Work

2.1 Introduction

This chapter's first aim is to provide an insight into the work (as well as the works viability) which has been done in creating adaptive content. It is motivated by the need for future pieces of software to appeal to a wider audience as well as to provide this audience with a more personal and enjoyable experience.

Currently there are several computer games that provide adaptive content to its user base. However, these implementations are not very intricate. With most implementations requiring a calibration phase (i.e. the decision of the content is decided upon by direct input). Very few games actually using a real machine learning algorithm to determine the best content to present the user with. Thus, the second aim of this section is to expound upon and to improve current methods being used to generate adaptive content. Therefore this chapter will discuss in detail the viability of this research by scrutinizing relevant and related works.

2.2 Significance

2.2.1 Uses of Adaptive Content

As technology has improved, hardware is now capable of handling adaptive content. Previously, adaptive content has not been used as the hardware could not handle the extra computation required to adapt content in real-time and there were not many methods available to developers to successfully create adaptive content.

The uses of adaptive content are many. In almost any software field, adaptive content could be implemented to improve the software. To fully explain just how broad and monumental adaptive could be in the computing world, I'll provide some examples below of where it could be implemented and where it has been implemented in lesser ways.

Gaming

Most games nowadays try appeal to a larger audience. This is done so that computer game companies can sell more copies of their games and in doing so make more money. There is a downside to this though, adaptive content means more content has to be created. Since more content has to be created, some of which each player might not even use, more money and development time must be spent in creating this content. The methods to implement such content is not up to scratch either.

Games such as MMORPGs (Massively Multiplayer Online Role-Playing Games) are already starting to push out adaptive content. Most MMORPGs have monthly subscription fees as well as being story driven. The audience of these games expect to be able to make decisions that affect the outcome of the games storyline. This means that developers have to make multiple endings and storylines from which the player can pick from. Certain games, such as the Mass Effect $^{\text{TM}}$ series(Trademark of the Bioware company), tackled this problem by creating large trees representing decisions which the player the 'traversed' along. Here is the tree of the Mass Effect $^{\text{TM}}$ series.

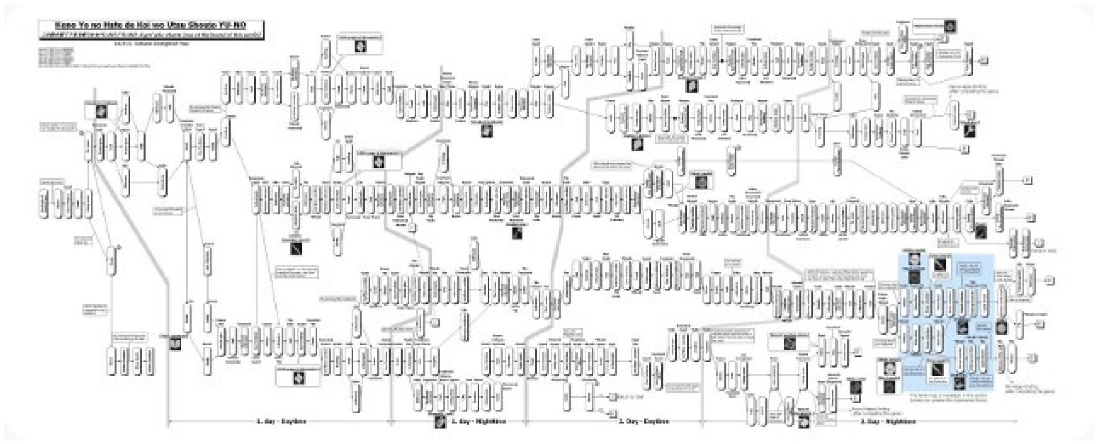


Figure 1: Mass Effect storyline graph

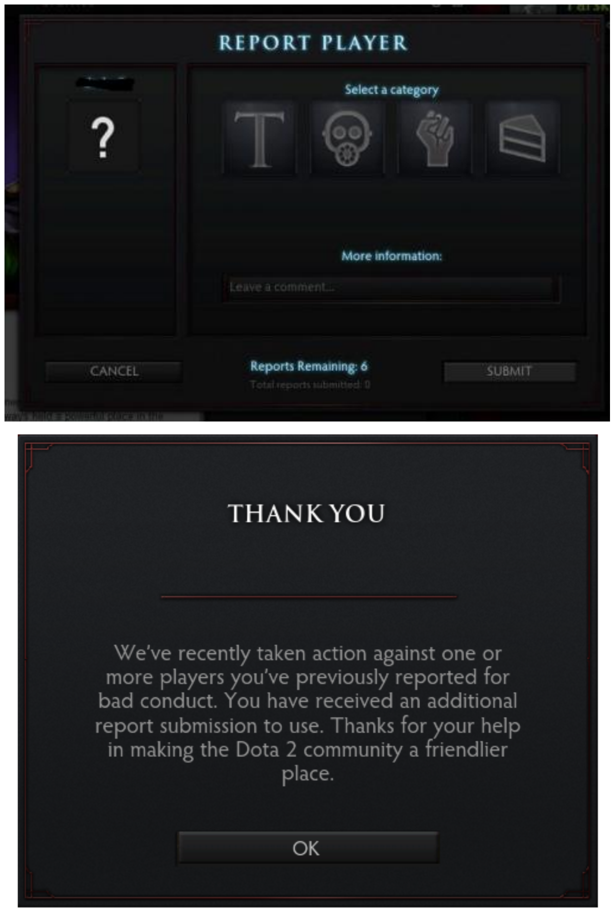
As you can see the tree is overly complicated and certain decisions will completely lock-out certain other events from happening. This is an ineffective solution as the player experience is not being optimised. The solution to this problem will be presented later on in the paper.

Online Gaming

As mentioned earlier games are increasingly starting to rely on adaptive content to appeal to larger audience. Another example of games using adaptive content is that of online games such as MOBAs (Multiplayer online battle arena) and online FPSs (First-person shooters) which rely heavily on competitive game-play between users from all over the world. Indeed companies such as Valve have even hosted competitions for their game Dota 2 $^{\text{TM}}$, with the prise money reaching 10 million dollars for the winning team in the last such competition. This show that these games are really starting to build traction and as such are starting to earn their respective owners a lot of money.

Since these games are online they have the very real problem of players harassing each other. Harassment in these games occur for a wide variety of reasons and punishment of problem player does not usually prevent further harassment. To combat this certain gaming companies have implement machine learning algorithms that monitor all online matches occuring and along with a few questions that each player must answer determine who players are matched up with when they play. These algorithms keep the obvious goal in mind of keeping problem players seperate from players who are well behaved. This tactic has been commonly referred to as the 'prison island' tactic by gamers. Another outcome is so that problem players are punished automatically without any other players having to complain.

Certain games also deploy tactics to team players up with other players who have the same skill level as they do. To do this, these companies monitor players scores and move them up or down in 'rankings' appropriately. This is not a perfect system though as scores do not perfectly reflect player ability and has in some cases incorrectly placed players with high skill levels in low skill divisions and vice versa. As these games are competitive, this can create quite a lot of strife within the communities, even forcing players to abandon the games out of frustration losing the gaming company any revenue they could have expected from these players. Better algorithms at placing players in the correct division could thus potentially save these companies a lot of money.



We've recently taken action against one or more players you've previously reported for bad conduct. You have received an additional report submission to use. Thanks for your help in making the Dota 2 community a friendlier place.

Figure 2: Dota 2 report system

2.2.2 Viability of Adaptive Content

As shown in the introduction the need for this research was justified by saying that more people are using technology on a daily basis but not all of these people are capable of using technology efficiently. How then can this issue be tackled to help these people improve their performance when using computers? It was shown in [Beaudry and Pinsonneault 2005] how users of varying technological ability responded to a new technology being introduced in to their work environment. The paper found that although peoples reactions and abilities to adapt vary widely for each individual, it is possible for these adaptation 'Strategies' to be grouped. The research concluded by observing actual people in real life scenarios that there are four broad adaptation strategies that cover all individuals coping mechanisms to new technology. This is highly important since it shows that if it is possible to label an individuals coping mechanism and behaviour then it is possible to develop a method to help any individual cope with new technology.

How though can we implement this? How is it possible to create something that will automatically fit a person to effectively minimise the time they take to cope with new technology? There are several papers that have provided a solution to this problem. They propose that to optimise a users performance, one must simply adapt the interface of said program to perfectly suit the user [Lavie and Meyer 2010]. Whether or not this is applicable is another problem, as adaptive interfaces might be to hardware intensive or the creation of different interfaces as opposed to a single broader interface might be cost ineffective. “Benefits and costs of adaptive user interfaces" by Lavie Talia and Meyer Joachim gives an outline of an experiment where this is explored. Through the use of an experiment involving people of different ages operating vehicles with different interfaces and then measuring their response times, they came to the conclusion that adaptive interfaces are beneficial in situations where computation time is not an issue. They also came to the conclusion that older peoples' performances improved drastically when using adaptive interfaces but younger people's performance either remained unchanged or dropped drastically. In these cases the interfaces actually impeded user performance.

This shows that it is possible to create an adaptive interface which can improve user performance but it was noted in Lavie and Meyer [Lavie and Meyer 2010] that further research was required to explore and expound upon this topic.

2.3 Markov Decision Processes

How then can an adaptive user interface be generated? There are several methods to accomplish this. The method that has the lowest apparent computation time and most apparent usefulness makes use of what is called a Markov decision process [Sutton and Barto 1998]. "Reinforcement learning: An introduction" [Sutton and Barto 1998] explains in detail the applications of a field of computer science called reinforcement learning. Reinforcement learning is a machine learning process whereby the environment learns from the users interaction with it. The book also covers a very broad range of applications, seeing use in topics such as genetic algorithms, psychology, control engineering etc. However there are several challenges with reinforcement learning. The major challenge is the trade-off between what is referred to as exploitation and exploration. To achieve a high reward, reinforcement learning must exploit as many decisions it has observed in the past. However, to obtain these decisions it must explore as much as possible, which might lead to massive training data. The paper then goes on to explain that there is no easy way to pick an optimal route between exploitation and exploration.

A Markov decision process is a mathematical framework that can easily represent decision making when some decision may be random. Its most common representation is that of a directed graph where states and actions are nodes and the connections between these are probabilities of certain events occuring. Through the use of an MDP, we can easily create an iterative method that updates states and actions so that an optimal policy is possible. In conclusion the paper shows that reinforcement learning can generate agents whose performance improves over time through computational effort.[White III 1991]

These MDPs have been used in [Ramamoorthy *et al.* 2013] and [Rosman *et al.* 2014] to generate interfaces. The MDPs were represented using the list data structure with look-up times increased using hash tables. In [Andrade *et al.* 2005] a representation of a basic MDP is shown. They used a list structure with each state mapping to several actions as well. Using this, the program can measure user's actions and then store them over several 'games'. This data can be averaged to show the behaviour of the user. Knowing which actions have better results will allow us to tweak the user interface so that the user picks better action more often.

Formally, an MDP can be defined as follows.

- Let S be the set of states in the MDP
- Let A be the set of actions in the MDP where As are the set of actions available at state s

- Let Pa(s, s') be the probability of an action a being taken at state s to the state s'
- Let Ra(s, s') be the reward of taking action a from state s to state s'

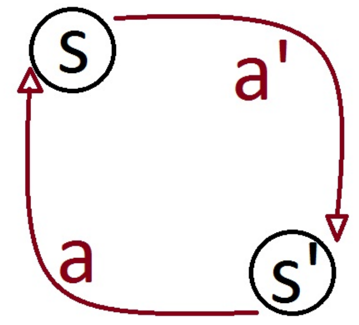
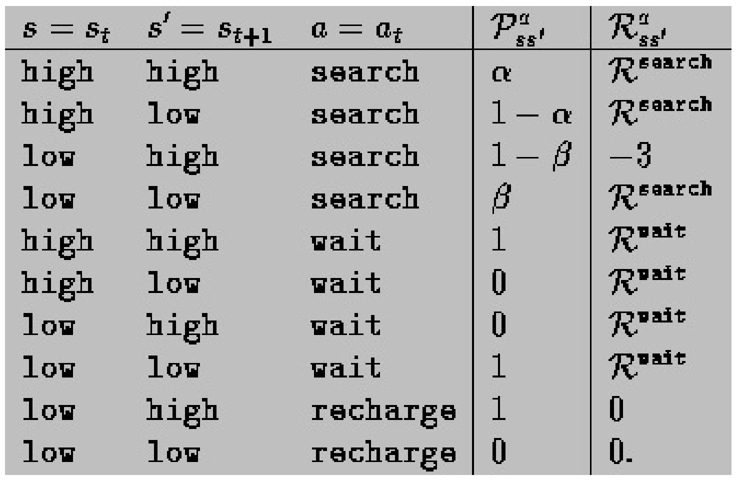


Figure 3: Basic Representation of an MDP



$s = s_t$	$s' = s_{t+1}$	$a = a_t$	$\mathcal{P}^a_{ss'}$	$\mathcal{R}^a_{ss'}$
high	high	search	α	$\mathcal{R}^{\text{search}}$
high	low	search	$1-\alpha$	$\mathcal{R}^{\text{search}}$
low	high	search	$1-\beta$	-3
low	low	search	β	$\mathcal{R}^{\text{search}}$
high	high	wait	1	$\mathcal{R}^{\text{wait}}$
high	low	wait	0	$\mathcal{R}^{\text{wait}}$
low	high	wait	0	$\mathcal{R}^{\text{wait}}$
low	low	wait	1	$\mathcal{R}^{\text{wait}}$
low	high	recharge	1	0
low	low	recharge	0	0.

Figure 4: Example table

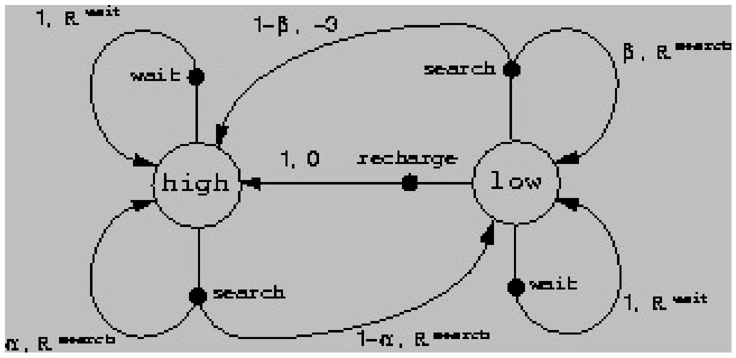


Figure 5: Corresponding graph

So it has now been shown that adaptive interfaces can indeed be made and that through the use of an MDP we can generate a user profile for each user. All that needs to be shown is how to actually create and decide on interfaces for users. The method for doing this quite clear cut [Dessart *et al.* 2011]. All that must be done to adapt the interface is to create a transition function which is done as follows:

- Find what action has been changed by the user.
- Find which element of the user interface is affected
- Perform one of the following operations on the element: Resize, Relocate, Image transformation, Widget transformation, Widget splitting.

There is one more thing that needs to be shown. Once a user profile is selected and an appropriate interface selected, the user is then given the newer interface to use. After this the program will hone in on an optimal interface until the user is always performing his/her optimal actions [Dessart *et al.* 2011] [Andrade *et al.* 2005].

An adaptive interface isn't exactly easy to implement due to computation constraints. Further study can be done in creating adaptive interface algorithms more efficiently. A computer game usually requires a large amount of computation power. This has large implications for the tech industry since several devices require a calibration stage before usage. An adaptive interface can be used to circumnavigate this and instead perform the 'calibration' on the fly. Take for example the computer game system called the xbox. Users upon first starting the console up must perform several basic adjustments to alter the interface to suit themselves. Most users find this highly annoying and often write complaints on this very issue to manufacturers as a result.

2.4 Conclusion

Overall, this research has never been fully explored and has only been implemented a few times in small contained experiments such as those in [Andrade *et al.* 2005], [Ramamoorthy *et al.* 2013] and [Rosman *et al.* 2014]. Adaptive content has been implemented in computer games but not to the large extent that is being proposed with the use of MDPs. Since this approach does not simply have to be used in the generation of user interface but can be used to map user preferences and behaviours. Due to this fact, there is a lot of room for improvement and exploration. Thought has been given to use this approach in the generation of RSS feeds on the internet. Even advertisements can use this method to create user profiles and then a mapping from one advert to another. This research can, quite literally, be used in almost every aspect of computer systems since most computer systems are used by people with varying skill levels and understanding of computers. This research will explore and tackle this in a way that can be implemented anywhere.

Therefore it has been proposed that an interface can be created which will increase the users performance. Which can be summarised as follows: The content and interface of a computer program can be updated or changed to automatically improve the users performance.
The commercial and societal impact of this research has been shown to have merit by showing that several companies would benefit directly from this research. Lack of research in this area of computer science indicates that more research is required to further this field. Many new and novel ways have also been suggested to generate adaptive interfaces but have not been fully explored.

3 Research Method

3.1 Introduction

The research being discussed in this section has two primary goals. Firstly the research aims to verify that adaptive content in computer games improves the users' performances. Namely research done by Subramanian Ramamoorthy, MM Hassan Mahmud, Pushmeet Kohli and Benjamin Rosman [Rosman *et al.* 2014], [Ramamoorthy *et al.* 2013]. Secondly it aims to implement and expound upon the method provided in these papers in a more complicated gaming environment. This chapter provides the basis by which the research aims to achieve this.

3.2 Motivation and Hypothesis

Rosman and Ramamoorthy [Rosman *et al.* 2014] have shown that it is possible to create an interface which is better suited to the user. Even though this interface might actually initially impair performance, Rosman and Ramamoorthy have shown that the users will reach a higher performance with the newer interface over time. In addition it was shown that the time taken for the user to adapt to the new interface will decrease if intermediate interfaces are provided to the user. Where intermediate interfaces are interfaces that incrementally add changes to the existing interfaces such that the user is not overwhelmed by the new changes. Previous research thus allows us to make three assumptions that we will base our work on:

- Skill level can be quantified
- Skill level can be inferred from user behaviour
- Aspects of the game can be adjusted to affect the user experience

In all other research, experiments were done with relatively simple games (such as navigating mazes). In these experiments the actions the player made could be very easily linked to a change in the MDP. However, if this idea is to be implemented in other real-world scenarios, the changes in the MDP might not be as obvious. Take for example a game where a player could pick from 100 possible actions as opposed 4. The representation of the MDP we are familiar with would not be helpful in this scenario and it would be difficult for us to visualise such a MDP. Therefore, we our research aims to contribute to this existing research by focusing on a more complex game, where there might not be an optimal choice to be made.

Which comes to my research hypotheses (Note that these both apply to complex games):

Hypothesis 0: The content and interface of a computer program can be updated or changed automatically based on skill level.

Hypothesis 1: Adaptive content improves the users' experience.

3.3 Overview of Method

3.3.1 Programming Languages

All programming done in performing this research will be done in python using the PyGame and NumPy libraries. MDPs will be represented as a linked list. The reasons are as follows:

- Python is relatively simple to use
- We have experience in using Python as well as the similar language Java
- Python has several useful libraries

3.3.2 Phase 1: Game Design

Although we aim to extend the game to more complex games, this does not necessarily mean we should become over-ambitious in our game choice. Thus a game must be chosen that has several choices available to the user at each stage, but not too many choices, as we want to be able to finish the research within the current time constraints.

Taking all of this into account, we have decided on implemented a simple 'Tetris' game. The reasoning in this decision is as follows. Tetris is a very simple game and is widely recognised. It is also relatively simple to program. The popularity and familiarity associated with this game will aid in testing as it will be easy to round up testers who are skilled. The 'gameworld' in Tetris looks as follows:

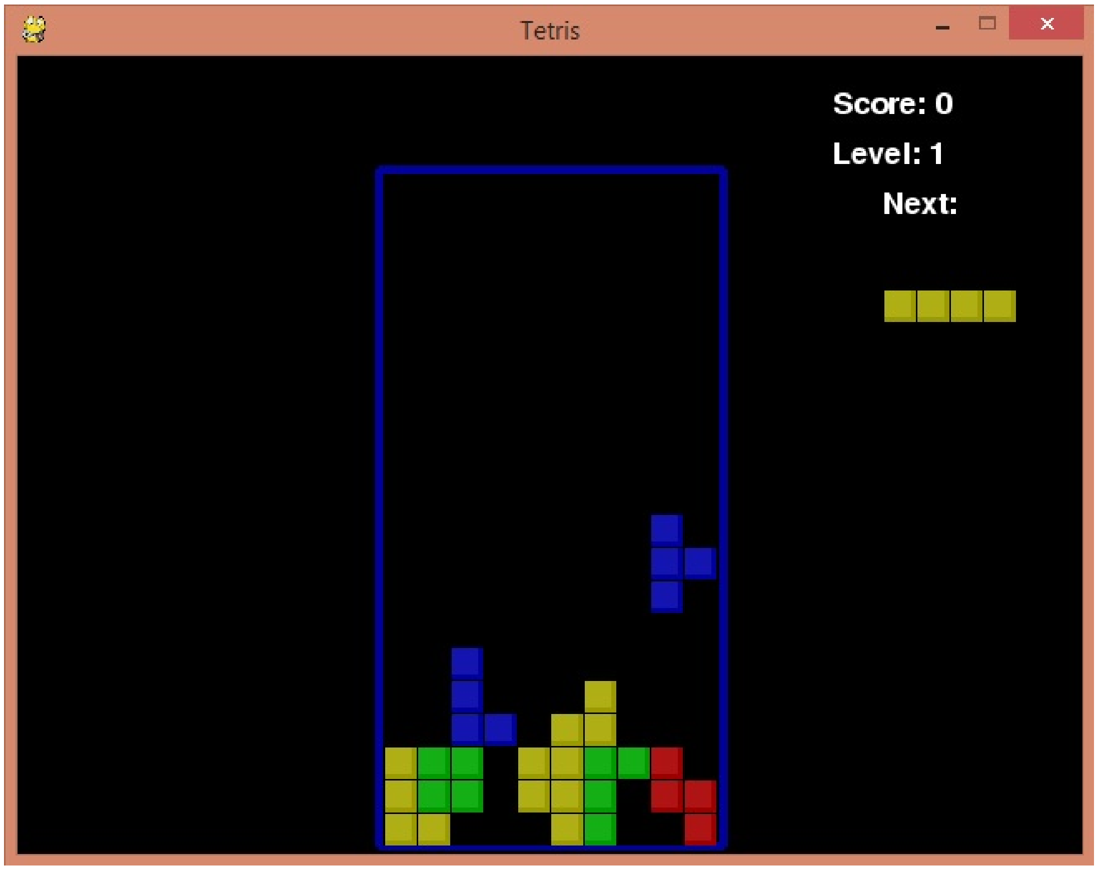


Figure 6: The Tetris game world

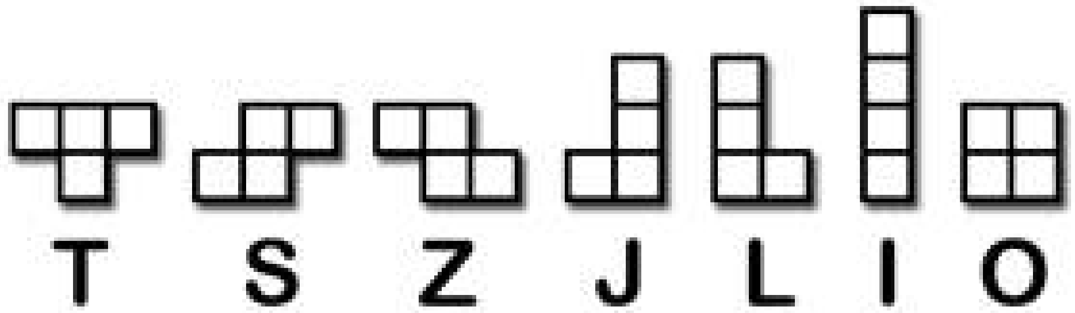


Figure 7: The blocks players must place

This basic grid layout will help keep programming the game simple. The goal of Tetris is to fill an entire row with blocks. Players are given blocks to place and form stacks, there are several strategies to place these blocks optimally. In the event a row is completed, it will disappear and all other blocks will move down by 1 spot. If the board becomes saturated with blocks and the next block that spawns cannot be moved then the game terminates. Your score will correspond to the amount of rows you cleared in the entire game.

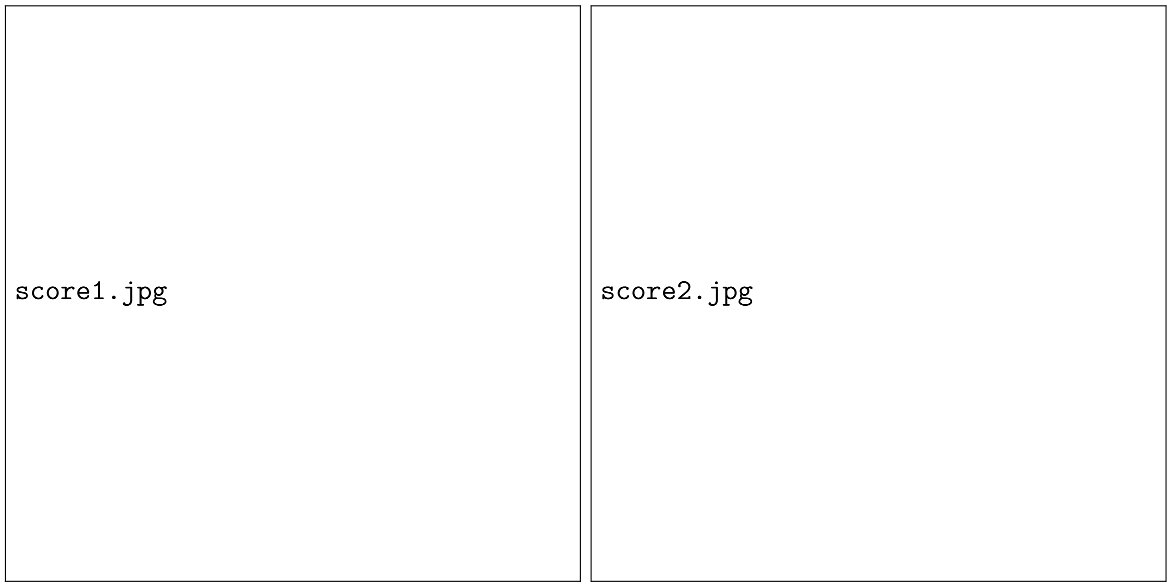


Figure 8: The player has scored

The main reason we have picked Tetris as the game we will test our method on is the distinctive strategies that players can pick and the relative ease at which we can get data from the world.

3.3.3 Phase 2: Testing

The second phase of the experiment will require testing to be done on players of varying skill levels. The testing will require players who are familiar with tower defence games, people who are familiar with computer games in general and people who are not familiar with computer games in general. These groups of players will correspond to skilled, intermediate and unskilled levels respectively. They will then be required to complete the game. Certain variables will be recorded and then used to determine each users play style respectively. Players will also be required to rate the games enjoyability.

All independent variables available will be recorded. Giving us as much information as possible to connect player levels to MDPs. For example, when a player places a tower, the stage at which he placed the tower, its position and the towers type will all be recorded. This would tell us what actions each type of player would make given possible options. However, the most informative variables and the unimportant variables will not be obvious at this point so as much data as possible must be collected. After all the data is collated and sorted, we can then move onto the algorithm design phase.

3.3.4 Phase 3: Algorithm Design

After all the data is collected regarding each players 'play style' we now have information with which to create our MDPs. Where the state of the game is the current wave number and the action is the tower placement. The following definition are required:

- Let S be the set of states in the MDP
- Let A be the set of actions in the MDP where As are the set of actions available at state s
- Let Pa(s, s') be the probability of an action a being taken at state s to the state s'
- Let Ra(s, s') be the reward of taking action a from state s to state s'
- Let M be the set of all MDPs
- B is the updated MDP
- N represents the function by which B is being updated in algorithm 1. N is usually a Q learning algorithm.
- $$B' = argmax|B| \sum f(s1 : k, Y) > |B| - 1 + \beta \qquad (1)$$

Listing 1: Algorithm 1: MDP update

```
        Initialize all variables
        B(0) = S(k)
        t = 0
        while true do
                Learner choose action A(t)
                Learner obtains some reward R(A(t))
                Nature returns set B(t+1) = N(A(t),B(t),T)
                t = t + 1
        end
end
```

Listing 2: Algorithm 2: Online Interact

```
        Input is the set of all MDPs M and X the selection policy
        As well as the constant beta, f and term where beta is a constant
        term is the task termination condition
        Initialize all variables
        B(0) = S(k)
        Let Y be the uniform set over S for each a and s
        Set all r to be 0 for each a and s
        t = 1
        while term =false do
                Get a, r and s at t
                Get all possible s and r after taking action a at state s and call
                Compute B'
                B(t+1) = B(t) - B'
                if B(t+1) is empty then
                        B(t+1) = S(k)
                Set A(t+1) = X(B(t+1))
                t = t + 1
        end
end
```

3.3.5 Phase 4: Experimental Analysis

Similarly to phase 2, the game with the machine learning algorithm now applied, will be presented to skilled, intermediate and unskilled players. Their performances are then recorded as before with all in-game variables necessary such as tower position and type as well as wave number. A direct comparison will then be done between their performances and those of the players in phase 2 to map the performance change in each group. Appropriate graphs and other visual aids can then be created to aid in this comparison. This phase will allow us to decide on the best and most optimal MDPs. Players will also be asked to rate the enjoyability of the game as before in the no adaptive version. These two results will be taken into account to determine whether or not players prefer the normal game are the version with the adaptive content.

3.4 Conclusion

The aim of this research is to provide empirical data to verify adaptive content as a viable option in computing. The method for accomplishing these goals was laid out in this chapter with supporting motivation and hypotheses. The method was broken up into 4 phases, two of which were development phases and two of which were testing phases. The method was taken from Rosman and Ramamoorthy [Ramamoorthy *et al.* 2013] and then improved upon to fit a more complex scenario. The algorithm to do so was provided. The time that we expect all of these tasks to be done in was provided as well. The next chapter will describe what results we hope to gain from conducting this research.

4 Results

4.1 Introduction

It is necessary to explain exactly what results we expect to obtain from our research are. In this section we will explain what results we expect to see and exactly what we hope to achieve with this. As well as explaining how unexpected results would affect our research. The consequences of our research will also be discussed. Due to the nature of this section, an overview of what future research could be conducted.

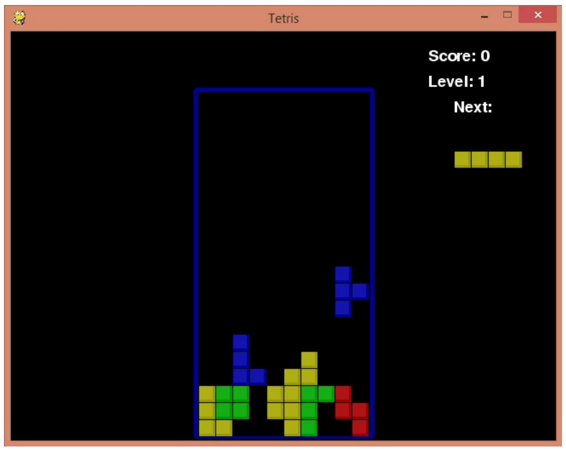


Figure 9: Example of the game world

We consider a game called Tetris. Tetris is 2D game world where users stack blocks in order to make complete rows of blocks (complete rows disappear). The user receives a score point for each row cleared and loses if the stack of blocks reaches the top of the board. There are seven block shapes that are presented to the user for stacking. The user can then decide to move the piece to the left or right, rotate it and then place it.

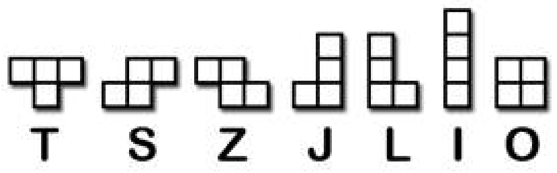


Figure 10: The different types of blocks a user is presented with and its associated label

We suggest collecting data of 3 different behaviours of a user playing Tetris. Each behaviour has a corresponding skill level indicated by how many lines can be cleared before losing. We have created three behaviour types with which to play the game:

- **Unskilled:** An AI that places blocks on the lowest point of the board with an average of lines cleared.
- **Moderate:** An AI that minimizes the amount of empty spaces and places pieces in stacks with an average of 2 lines cleared.
- **Skilled:** A human player that simply filled the board from left to right with an average of 26 lines cleared.

Then using the data created by these player we have created corresponding MDPs to represent each behaviour. Using these MDPs we create a new Tetris game called Adaptive Tetris that monitors the users actions in order to predict the users behaviour. Changes are then made to the game in order improve performance based on behaviour. We suggest two phases to implement this: the training phase (where the MDPs are created) and the adaptive phase (which checks user behaviour and determines whether changes made are beneficial).

4.2 Phase 1: Data collection

Algorithm 1 Training Phase

```
 1: procedure CREATE MDP ($\tau$)
 2:     MDP =$\phi$
 3:     While Game Running:
 4:     Observe the action made by the agent $\tau$
 5:     and then record the state, action
 6:     and the resultant state { s,a,s' }
 7:      in the set { S,A,S' }
 8:     for s in S:
 9:     Update each action set with the
10:      probability P(a |$\tau$) accordingly
11:     MDP[s] = A
```

Every user that plays Tetris has a skill level $\tau \ni [0, 2]$, where 0 represents Unskilled, 1 Moderate and 2 Skilled. With each skill level there is an associated probability $p \ni [0, 1]$ that the current agent corresponds to a skill level. There is also a probability that an agent will make an action given the state $P(a|\tau) \ni [0, 1]$. Our training phase is given as Algorithm 1. The algorithm runs indefinitely until shut down in order to create as much data as possible. Firstly allow the training user to play the game. After each action is made, record the action and the state of the game as well as the resultant state. In Tetris, the user can chose from the following actions: [Left, Right, Up, Down, Space, NoAction]. The state of the game is represented by the height of each column and the block's position, shape and rotation. [0,0,2,2,3,4,5,12,15,3 L 0 3 4] is an example of a possible state.

4.3 Phase 2: Adaptive algorithm

Algorithm 2 Adaptive Algorithm

```
 1: procedure ADAPTIVE GAME ($\tau$)
 2:     Set the probability for each agent
 3:     to be equally likely
 4:     Probabilites = [1/numAgents,...]
 5:     While Game Running:
 6:     Observe the action made by the agent $\tau$
 7:     For each element of Probabilities:
 8:     Calculate: logP($\tau$|a) = logP(a|$\tau$) + logP($\tau$)
 9:     Update Probabilities
10:     if any Probabilities $\tau \approx 1$ then
11:         Make changes to game based on $\tau$
```

We initially set the likelihood to be equal for each agent τ such that it is equally possible the user is any agent. Then while the game is running we observe each action made and fetch the probability of the action from each MDP. We then update the probabilities for each agent. If the probability that the user is a particular agent reaches some threshold such as 98% then we change the content based on which agent the user is likely to be. In the Tetris game we change the probability of certain pieces dropping as well as the fall frequency of the blocks based on what skill level the user is determined to be. For example: an unskilled player will get more squares and the fall frequency will drop to allow for more thought at block placement. The skilled player would be presented with a greater drop frequency and random blocks.

5 Results

We ran three experiments with each of the three users. In each experiment, we had each user type play the game Adaptive Tetris. The game was essentially the same for each experiment. However, we recorded different types of data to help us understand three important questions:

- Are we correctly able to identify a user?
- Is the time it takes to identify the user reasonable?
- Does the adaptive algorithm improve user performance?

Accuracy

We ran the adaptive algorithm 1000 times for each Ai and 100 times for the Skilled player. We found that the Unskilled AI was easy to identify, with 97% of the guesses correctly identifying the AI. The Moderately skilled AI was also easy to identify with an accuracy of 94%. However, the Skilled human player was more difficult to identify, with only 70% of the guesses being made correctly. We interpret that the Skilled player made actions that could be found in either of the other two Ai's action set. Since a skilled player in Tetris does not necessarily follow a single distinguishable behaviour but rather makes use of many. Overall, the algorithm made correct guesses as to the identity of the AI players with distinct behaviour 95% of the time.

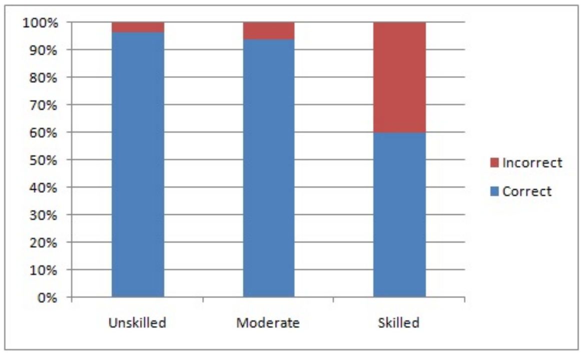


Figure 11: The accuracy at predicting each type

Convergence time

To calculate the time it took to predict agent type, we simply recorded how many steps were required to make a prediction. Since an action made corresponds to an update in the algorithm, we can instead refer to how many actions the user had to make. We recorded the actions taken in the same manner as we did when finding accuracy. However, the actions taken for each agent differed wildly without any real pattern. This can be attributed to the fact that an agent may make an action indicative of one agent then make another indicative of another, causing the probabilities to fluctuate. We noticed that sometimes the user could make 6 actions before a prediction was made and sometimes 400 actions could be made before a prediction was made. These times were in now way unique to each agent. So, we have rather taken an average prediction time from all the above data which we recorded as 38 actions from a set of 2100 games.

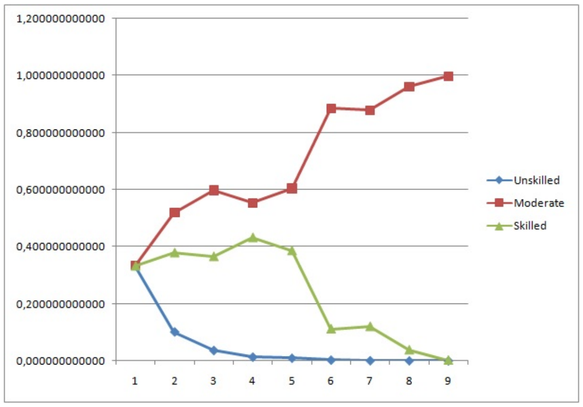


Figure 12: Example of probabilities converging on guess

User Performance

We first calculated the average score of each player type by observing all the games they played for the training phase of the MDPs. We calculated that the average score for a skilled player was 2 6 lines cleared, 0 lines for the unskilled and 2 lines cleared for the moderately skilled player. Using this we can directly check to see if any of the adaptations made in the game improve player score. To adapt the game we changed the type of pieces given to them and changed the block fall frequency. As before we ran the game 1000 times for each Ai and 100 times for the skilled player. We found that in 71% of the games played by the Ai players score was above the average for his behaviour type. The performance of the skilled player decreased due to the adaptive game making the game more difficult for the skilled players.

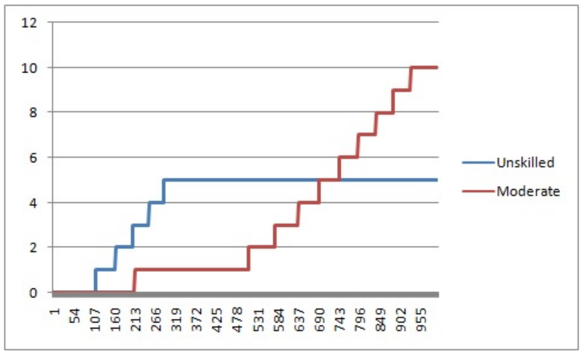


Figure 13: The Average score of the Ai's several games

5.1 Future Work

This algorithm can be implemented in real world scenarios as well as being modified to perform better. As shown earlier in the literature review, I described examples of real world instances of content adaptation. In the first example I showed that computer games create alternate storylines in their games usually represented as a complicated graph. With the use of this algorithm, this entire structure can be scrapped and the developers can create entirely unlinked quests. This pool of story points or quests can then be drawn from by the algorithm to best suit the individual user. The developers will now be able to structure the quest system of the game in a way that can keep each player optimal. There are additional side-effects though, each player could possibly have a unique experience which will keep player replaying and immersed in the game. Developers can now add content to the game in small updates rather than developing massive expansions by simple designing single quests that can be added to the list of quests the algorithm can pick from.

In competitive games, the algorithm could be applied to better keep players of similar skill-levels playing together. Which could potentially reduce frustration amongst the community and thus keeping the amount of moderation by the company down to a minimum. Work could also be done in testing the algorithm for other scenarios than gaming. As in most sciences, there is always room for improvement, research could be done into developing a more sophisticated algorithm that has lower computation times and more accuracy. It might be possible to refine the algorithm so that it consumes less system resources such as memory. As shown previously by Krzystof Z Gajos, Jacob O Wobbrock and Daniel S Weld [Gajos *et al.* 2008], adaptive interfaces may require too much computing power for them to be useful in real-life/ real-time scenarios. Thus much work can be done into improving this algorithm.

6 Conclusion

The goal of this paper was to identify user type so that we could provide individualised content to that user. In this paper we explored the use of MDPs as a novel approach at providing adaptive computer games. We provided an algorithm that can be used to generate MDPs representing user types. We also provided an algorithm with which to interpret these MDPs and provide appropriate changes based on this. This method was first present in us in "Latent-Variable MDP Models for Adapting the Interaction Environment of Diverse Users" Ramamoorthy *et al.* [2013]. We provided a method based off of theirs in which we could determine user type. We then showed the viability of this algorithm in providing content to the user through the use of three experiments. In which we were correctly able to identify user types in real time and then provide the users with individualised content. The possibilities regarding future work based off of this algorithm are many. As we have shown, there are a multitude of ways in which this algorithm could be used in real world applications. We hope that further work will be done in implementing the algorithm in more complex systems with a larger array of diverse users thus showing the algorithm efficiency. The viability of this research was conducted by several scien-

tists using several different experiments to show that it is indeed possible and useful. Rosman and Ramamoorthy [Rosman *et al.* 2014] developed an algorithm that could be implemented to alter the interface of the game to improve user performance. Their research showed that such an adaptive interface did improve the users experiences with the aid of staggered transition interfaces to ease the user into the new interface. The algorithm to do so was presented in their papers. It was determined that it was necessary for further research to be conducted to explore this algorithms use in more complex scenarios. This research aims to expand on this algorithm by implementing it in a more complex scenario. This proposal provided a groundwork for creating this algorithm. It also provided a literature review that effectively explains the process by which we have reached this idea as possible research. Future work can be done into refining

this method into a more robust method. That can be implemented quickly and effectively into other situations. Situations such as basic software and perhaps web-development could see use of this algorithm in their implementation. Existing game developers could make use of it on a broader scale by incorporating it into existing games titles. Altogether, the algorithm has a wide range of uses in computing and could possibly reach many markets. As such, all these factors contribute to us thinking of it as a highly sophisticated approach to the way developers present content to their users and as such we see it as a viable research topic.

7 Bibliography

[Andrade *et al.* 2005] Gustavo Andrade, Geber Ramalho, Hugo Santana, and Vincent Corruble. Challenge-sensitive action selection: an application to game balancing. In *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*, pages 194–200. IEEE, 2005.

[Beaudry and Pinsonneault 2005] Anne Beaudry and Alain Pinsonneault. Understanding user responses to information technology: A coping model of user adaptation. *Mis Quarterly*, pages 493–524, 2005.

[Dessart *et al.* 2011] Charles-Eric Dessart, Vivian Genaro Motti, and Jean Vanderdonckt. Showing user interface adaptivity by animated transitions. In *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*, pages 95–104. ACM, 2011.

[Gajos *et al.* 2008] Krzysztof Z Gajos, Jacob O Wobbrock, and Daniel S Weld. Improving the performance of motor-impaired users with automatically-generated, ability-based interfaces. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 1257–1266. ACM, 2008.

[Lavie and Meyer 2010] Talia Lavie and Joachim Meyer. Benefits and costs of adaptive user interfaces. *International Journal of Human-Computer Studies*, 68(8):508–524, 2010.

[Ramamoorthy *et al.* 2013] Subramanian Ramamoorthy, MM Hassan Mahmud, Benjamin Rosman, and Pushmeet Kohli. Latent-variable mdp models for adapting the interaction environment of diverse users. 2013.

[Rosman *et al.* 2014] Benjamin Rosman, Subramanian Ramamoorthy, MM Hassan Mahmud, and Pushmeet Kohli. On user behaviour adaptation under interface change. 2014.

[Sutton and Barto 1998] Richard S Sutton and Andrew G Barto. *Introduction to reinforcement learning*. MIT Press, 1998.

[White III 1991] Chelsea C White III. A survey of solution techniques for the partially observed markov decision process. *Annals of Operations Research*, 32(1):215–230, 1991.