Towards Online Adaptive Content Personalisation

Jonathan K. Wasson
School of Computer Science
University of the Witwatersrand
South Africa
wassonjonathan@gmail.com

Benjamin Rosman
School of Computer Science
University of the Witwatersrand
South Africa
benjros@gmail.com

Abstract

In computer systems it is desirable to have systems which infer the skill level and preferences of user. So that the system can present content to that user appropriately. Existing systems require the user to either go through calibration, training or even force the user to adjust the systems settings to suit himself. By changing the type or difficulty of the content presented to the user, we may improve the systems ease of use. In this paper we will discuss a method with which a systems content can be changed on the fly to improve the users performance and enjoyability. The method shows how to infer the class of user with which to make the changes by observing the user's action in real-time. We begin by modelling the user as a Markov Decision Process (MDP), where the transitions within the MDP are indicative of the user's behaviour or type (the actions made by the user are associated with a user type). We present a novel approach to provide content adaptation. We present an algorithm that observes the user's actions and update a distribution over the user type. We evaluate the accuracy of the algorithm in its ability to identify the true user type and determine whether or not content adaptation is indeed useful to the user through the use of experiments in a Tetris domain.

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 [1]. 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. [2] 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 [2]. 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 focuses on content adaptation in computer games performed in real-time. As stated previously, content adaptation has a wide variety of uses but we have decided to use it in computer games. There are a wide variety of computer games that can be picked from which give us room to mould the game into something that 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.
This paper discuses material relevant to this question and then uses it to design and implement an algorithm capable of online content personalisation. We 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 lay out an experiment used to determine the feasibility and effectiveness of our algorithm.

1.1. Related Work

The concept of programs being able to pitch different content to specific users is beginning to gain more attention from researchers and companies. Several computer games are already starting to provide adaptive content to its users. Most of these implementation require a calibration phase (i.e. the decision of the content is decided upon by direct input). Very few games actually use an automated system to determine the best content to present to the user.
The game Left 4 Dead [3] presents a way for the content of the game to remain exciting throughout the game by matching player actions to a curve. Too much inactivity will spawn a wave of monsters and too much activity will decrease spawn rate to give the players time to recover. This method is different to our proposed model, in that it does not explicitly model play style but rather measures in-game activity. Games such as MMORPGs (Massively Multiplayer Online Role-Playing Games) are already starting to deliver adaptive content. Many 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. Certain games, such as the Mass Effect ^{TM} series(Trademark of the Bioware company), tackled this problem by creating large trees representing decisions which the player the ‘traversed’ along. This makes no use of the player’s skill level or behaviour but rather looks at what decisions they made in-game. 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.
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.

1.2. Viability of Adaptive Content

As has been discussed 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? Studies have been conducted into how users of varying technological ability responded to a new technology being introduced in to their work environment [1]. The paper found that although people's reactions and abilities to adapt vary widely between individuals, 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 important since it shows that if it is possible to label an individual's coping mechanism and behaviour then it is possible to develop a method to help any individual cope with new technology.

How then can we implement this idea? 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 lines of research that have suggested a solution to this problem. These propose that to optimise a user's performance, one must simply adapt the interface of said program to perfectly suit the user [4]. Whether or not this is directly applicable is another problem, as adaptive interfaces might be too 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" [4] 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 by Lavie and Meyer [4] that further research was required to explore and expound upon this topic.

2. Markov Decision Processes

A Markov Decision Process (MDP) [5]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 occurring. 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. Such reinforcement learning can generate agents whose performance improves over time through computational effort. [6]

- Let S be the set of states in the MDP
- Let A be the set of actions in the MDP where A*s* 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.

$$B' = argmax|B| \sum f(s1:k,Y) > |B| - 1 + \beta \quad (1)$$

These MDPs have been used in [7] and [8] to generate adaptive interfaces. The MDPs were represented using the list data structure with constant look-up times via the use of hash tables. An MDP stored as a list will look as follows { s,a,s' } where s represents the current state and s' the next state resulting from the action a being taken [9]. We can then match an action set A with each state to get the following: { s, A }. The action set stores all the probabilities for each action being made at that state. An MDP is useful since we can look at a given state and see the probability of any action being taken from that state.

3. Method

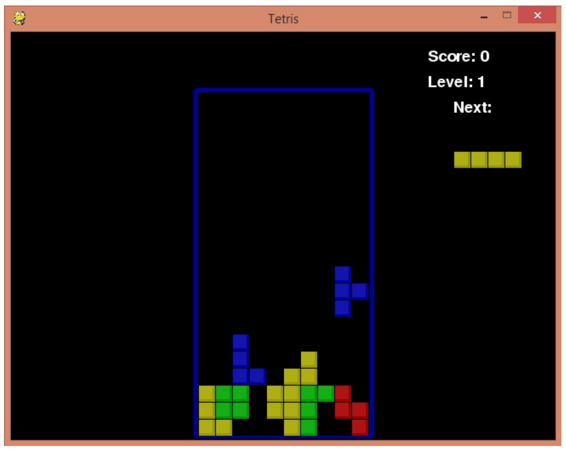

Figure 1. Example of the game world

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.

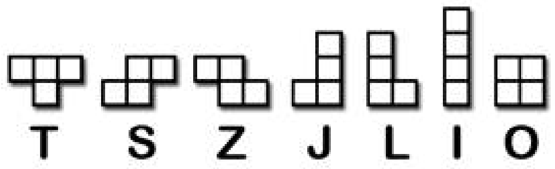

Figure 2. The different types of blocks a user is presented with and its associated label

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 agent that places blocks on the lowest point of the board with an average of lines cleared.
- **Moderate:** An agent 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).

3.1. Phase 1: Data collection

Every user that plays Tetris has a skill level $\tau \in [0, 2]$, where 0 represents Unskilled, 1 Moderate and 2 Skilled. With each skill level there is an associated probability $p \in [0, 1]$ that the current

Algorithm 1 Training Phase

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

agent corresponds to that skill level. There is also a probability that an agent will make an action given the state $P(a|s,\tau) \in [0,1]$. 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.

3.2. Phase 2: Adaptive algorithm

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

Algorithm 2 Adaptive Algorithm

```
procedure ADAPTIVE GAME ($\tau$)
    Set the probability for each agent
    to be equally likely
    Probabilites = [1/numAgents,...]
While Game Running:
    Get a, r and s at t
    Get all possible s and r
    after taking action a at state s
     and call these s1:k and r1:k
    Observe the action made by the agent $\tau$
    Compute B'
    B(t+1) = B(t) - B'
    Update Probabilities:
    if thenif B(t+1) is empty then
        B(t+1) = S(k)
    Set A(t+1) = X(B(t+1))
    t = t + 1
    Calculate: logP($\tau$|a) = logP(a|$\tau$) + logP($\tau$)
    if any Probabilities $\tau \approx 1$ then
        Make changes to game based on probabilities
```

allow for more thought at block placement. The skilled player would be presented with a greater drop frequency and random blocks. The adaptive algorithm has only been applied after a threshold has been reached for a number of reasons. Firstly, we do not want to present content to a user that is incorrect for that user. Secondly, by allowing the content to be presented after a certain threshold we can be certain that the content is meant for that user. This has the added benefit that if the probability goes below the threshold then we can remove the adaptive content till we are certain again that the user is of a certain type.

4. Experiment

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?

4.1. Results

Accuracy

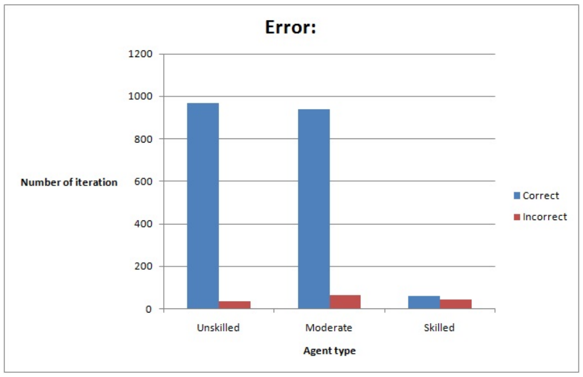

Figure 3. The accuracy at predicting each type

		Predicted Agent		
		Skilled	Moderate	Unskilled
Actual Agent	Skilled	60	26	14
	Moderate	13	938	49
	Unskilled	0	33	967

Figure 4. Confusion matrix of predictions

We ran the adaptive algorithm 1000 times for each agent and 100 times for the Skilled player. We found that the Unskilled agent was easy to identify, with 97% of the guesses correctly identifying the agent. The Moderately skilled agent 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 agents 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 unskilled and moderate players with distinct behaviour 95% of the time. A representative confusion matrix is included to illustrate this.

Convergence time

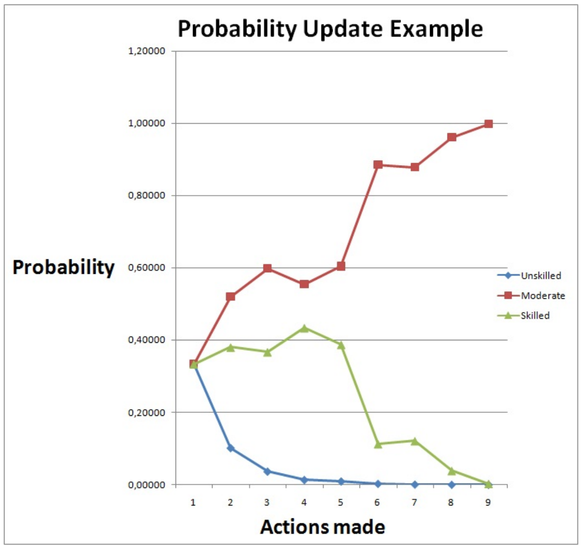

Figure 5. Example of probabilities converging

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.

User Performance

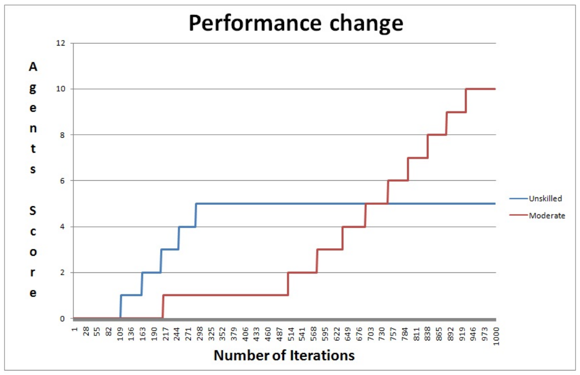

Figure 6. The Average score of the agents several games

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 26 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 agent and 100 times for the skilled player. We found that in 71% of the games played by the moderate and unskilled 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.

4.2. Discussion

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.

As we noted, the algorithm had a difficult time pointing out the Skilled player since its behaviour could be seen in the other two agents. The current experiments could be conducted on agents of ambiguous behaviour. This would be done to see how we could split similar behaviours apart from one another so as to make the algorithm more accurate.

5. Conclusion

This paper presented a method 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 developing 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 based on an existing method [7] to determine user type. We then showed the viability of this algorithm in providing content to the user through the use of three experiments in the Tetris domain. 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.

References

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

[2] K. Z. Gajos, J. O. Wobbrock, and D. 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*. ACM, 2008, pp. 1257–1266.

[3] M. Booth, "The ai systems of left 4 dead," in *Keynote, Fifth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE09)*, 2009.

[4] T. Lavie and J. Meyer, "Benefits and costs of adaptive user interfaces," *International Journal of Human-Computer Studies*, vol. 68, no. 8, pp. 508–524, 2010.

[5] R. S. Sutton and A. G. Barto, *Introduction to reinforcement learning*. MIT Press, 1998.

[6] C. C. White III, "A survey of solution techniques for the partially observed markov decision process," *Annals of Operations Research*, vol. 32, no. 1, pp. 215–230, 1991.

[7] S. Ramamoorthy, M. H. Mahmud, B. Rosman, and P. Kohli, "Latent-variable mdp models for adapting the interaction environment of diverse users," 2013.

[8] B. Rosman, S. Ramamoorthy, M. H. Mahmud, and P. Kohli, "On user behaviour adaptation under interface change," 2014.

[9] G. Andrade, G. Ramalho, H. Santana, and V. Corruble, "Challenge-sensitive action selection: an application to game balancing," in *Intelligent Agent Technology, IEEE/WIC/ACM International Conference on*. IEEE, 2005, pp. 194–200.

[10] C.-E. Dessart, V. Genaro Motti, and J. Vanderdonckt, "Showing user interface adaptivity by animated transitions," in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems*. ACM, 2011, pp. 95–104.