Player Modeling for Intelligent Difficulty Adjustment*

Olana Missura and Thomas Gärtner

Fraunhofer Institute Intelligent Analysis and Information Systems IAIS,
Schloss Birlinghoven, D-53754 Sankt Augustin, Germany
firstname.lastname@iais.fraunhofer.de

Abstract. In this paper we aim at automatically adjusting the difficulty of computer games by clustering players into different types and supervised prediction of the type from short traces of gameplay. An important ingredient of video games is to challenge players by providing them with tasks of appropriate and increasing difficulty. How this difficulty should be chosen and increase over time strongly depends on the ability, experience, perception and learning curve of each individual player. It is a subjective parameter that is very difficult to set. Wrong choices can easily lead to players stopping to play the game as they get bored (if underburdened) or frustrated (if overburdened). An ideal game should be able to adjust its difficulty dynamically governed by the player's performance. Modern video games utilise a game-testing process to investigate among other factors the perceived difficulty for a multitude of players. In this paper, we investigate how local models can be combined to a global model for automatic difficulty adjustment. In particular, we combine the simple local models by means of a support vector machine. Experiments confirm the potential of our combination strategy in this application.

1 Introduction

We aim at developing games that provide challenges of the "right" difficulty, i.e., such that players are stimulated but not overburdened. Naturally, what is the right difficulty depends on many factors and can not be fixed once and for all players. For that, we investigate how general machine learning techniques can be employed to automatically adjust the difficulty of games. A general technique for this problem has natural applications in the huge markets of computer and video games but can also be used to improve the learning rates when applied to serious games.

The traditional way in which games are adjusted to different users is by providing them with a way of controlling the difficulty level of the game. To this end, typical levels would be 'beginner', 'medium', and 'hard'. Such a strategy has many problems. On the one hand, if the number of levels is small, it may be

* This is a version of "Olana Missura and Thomas Gärtner, *Player Modeling for Intelligent Difficulty Adjustments*. In: Proceedings of the 12th International Conference on Discovery Science (DS) (2009)"

easy to choose the right level but it is unlikely that the difficulty is then set in a very satisfying way. On the other hand, if the number of levels is large, it is more likely that a satisfying setting is available but finding it becomes more difficult. Furthermore, choosing the game setting for each of these levels is a difficult and time-consuming task.

In this paper we investigate the use of supervised learning for dynamical difficulty adjustment. Our aim is to devise a difficulty adjustment algorithm that does not bother the actual players. For that, we assume there is a phase of the game development in which the game is played and the difficulty is manually adjusted to be just right. From the data collected in this way, we induce a difficulty model and build it into the game. The actual players do not notice any of this and are always challenged at the difficulty that is estimated to be just right for them.

Our approach to building a difficulty model consists of three steps: (*i*) cluster the recorded game traces, (*ii*) average the supervision over each cluster, and (*iii*) learn to predict the right cluster from a short period of gameplay. In order to validate this approach, we use a leave-one-player-out strategy on data collected from a simple game and compare our approach to less sophisticated, yet realistic, baselines. All approaches are chosen such that the players are not bothered. In particular, we want to compare the performance of dynamic difficulty versus constant difficulty as well as the performance of cluster prediction versus no-cluster. Our experimental results confirm that dynamic adjustment and cluster prediction together outperform the alternatives significantly.

2 Motivation and Context

A game and its player are two interacting entities. A typical player plays to have fun, while a typical game wants its players to have fun. What constitutes the *fun* when playing a game?

One theory is that our brains are physiologically driven by a desire to learn something new: new skills, new patterns, new ideas [1]. We have an instinct to play because during our evolution as a species playing generally provided a safe way of learning new things that were potentially beneficial for our life. Daniel Cook [3] created a psychological model of a player as an entity that is driven to learn new skills that are high in perceived value. This drive works because we are rewarded for each new mastered skill or gained knowledge: The moment of mastery provides us with the feeling of joy. The games create additional rewards for their players such as new items available, new areas to explore. At the same time there are new challenges to overcome, new goals to achieve, and new skills to learn, which creates a loop of learning-mastery-reward and keeps the player involved and engaged.

Thus, an important ingredient of the games that are fun to play is providing the players with the challenges corresponding to their skills. It appears that an inherent property of any challenge (and of the learning required to master it) is its difficulty level. Here the difficulty is a subjective factor that stems from the

interaction between the player and the challenge. The perceived difficulty is also not a static property: It changes with the time that the player spends learning a skill.

To complicate things further, not only the perceived difficulty depends on the current state of the player's skills and her learning process, the dependency is actually bidirectional: The ability to learn the skill and the speed of the learning process are also controlled by how difficult the player perceives the task. If the bar is set too high and the task appears too difficult, the player will end up frustrated and will give up on the process in favour of something more rewarding. Then again if the challenge turns out to be too easy (meaning that the player already possesses the skill necessary to deal with it) then there is no learning involved, which makes the game appear boring.

It becomes obvious that the game should provide the challenges for the player of the "right" difficulty level: The one that stimulates the learning without pushing the players too far or not enough. Ideally then, the difficulty of any particular instance of the game should be determined by who is playing it at this moment.

Game development process usually includes multiple testing stages, where a multitude of players is requested to play the game to provide data and feedback. This data is analysed to tweak the games parameters in an attempt to provide a fair challenge for as many players as possible. The question we investigate in this work is how the data from the α/β tests can be used for the intelligent difficulty settings with the help of machine learning.

We proceed as follows: After reviewing related work in Section 3, we describe the algorithm for the dynamic difficulty adjustment in general terms in Section 4. In Sections 5 and 6 we present the experimental setup and the results of the evaluation before concluding in Section 7.

3 Related Work

In the games existing today we can see two general approaches to the question of difficulty adjustment. The traditional way is to provide a player with a way to set up the difficulty level for herself. Unfortunately, this method is rarely satisfactory. For game developers it is not an easy task to map a complex gameworld into a single parameter. When constructed, such a mapping requires additional extensive testing, creating time and money costs. Consider also the fact that generally games require several different skills to play them. The necessity of going back and forth between the gameplay and the settings when the tasks become too difficult or too easy disrupts the flow component of the game.

An alternative way is to implement a mechanism for dynamic difficult adjustment (DDA). One quite popular approach to DDA is a so called *Rubber Band AI*, which basically means that the player and her opponents are virtually held together by a rubber band: If the player is "pulling" in one direction (playing better or worse than her opponents), the rubber band makes sure that her opponents are "pulled" in the same direction (that is they play better or worse respectively). While the idea that the better you play the harder the game should

be is sound, the implementation of the Rubber Band AI often suffers from disbalance and exploitability.

There exist a few games with a well designed DDA mechanism, but all of them employ heuristics and as such suffer from the typical disadvantages (being not transferable easily to other games, requiring extensive testing, etc). What we would like to have instead of heuristics is a universal mechanism for DDA: An online algorithm that takes as an input (game-specific) ways to modify difficulty and the current player's in-game history (actions, performance, reactions, ...) and produces as an output an appropriate difficulty modification.

Both artificial intelligence researchers and the game developers community display an interest in the problem of automatic difficulty scaling. Different approaches can be seen in the work of R. Hunicke and V. Chapman [9], R. Herbich and T. Graepel [8], Danzi et al [5], and others. As can be seen from these examples the problem of dynamic difficulty adjustment in video games was attacked from different angles, but a unifying approach is still missing.

Let us reiterate that as the perceived difficulty and the preferred difficulty are subjective parameters, the DDA algorithm should be able to choose the "right" difficulty level in a comparatively short time for any particular player. It makes sense, therefore, to conduct the learning in the offline manner and to make use of the data created during the test phases to construct the player models. These models can be used afterwards to generalise to the unseen players.

Player modeling in computer games is a relatively new area of interest for the researchers. Nevertheless, existing work [12, 11, 2] demonstrates the power of utilising the player models to create the games or in-game situations of high interest and satisfaction for the players.

In the following section we present an algorithm that learns a mapping from different player types to the difficulty adjustments and predicts an appropriate one given a new player.

4 Algorithm

To simplify the problem we assume that there exists a finite number of types of players, where by type we mean a certain pattern in behaviour with regard to challenges. That is certainly true, since we have a finite amount of players altogether, possibly times a finite amount of challenges, or timesteps in a game. However, this realistic number is too large to be practical and certainly not fitting the purpose here. Therefore, we discretize the space of all possible players' behaviours to get something more manageable. The simplest such discretization would be into beginners, averagely skilled, and experts (corresponding to easy, average, and difficult settings).

In our experiments we do not predefine the types, but rather infer them using the clustering of the collected data. Instead of attempting to create a universal mechanism for a game to adapt its difficulty to a particular player, we focus on the question of how a game can adapt to a particular player type given two sources of information:

1. the data collected from the alpha/beta-testing stages (offline phase);
2. the data collected from the new player (online phase).

The idea is rather simple. By giving the testers control over the difficulty settings in the offline phase the game can learn a mapping from the set of types into the set of difficulty adjustments. In the online phase, given a new player, the game needs only to determine which type he belongs to and then apply the learned model. Therefore, the algorithm in general consists of the following steps:

1. Given data about the game instances in the form of time sequences
$$T_k = ((t_1, f_1(t_1), \ldots, f_L(t_1)), \ldots, (t_N, f_1(t_N), \ldots, f_L(t_N))),$$
where t_i are the time steps and $f_i(t_j)$ are the values of corresponding features, cluster it in such a way that instances exhibiting similar player types are in the same cluster.
2. Given a new player, decide on which cluster he belongs to and predict the difficulty adjustment using the corresponding model.

Note that it is desirable to adapt to the new player as quickly as possible. To this purpose we propose to split the time trace of each game instance into two parts:

- a prefix, the relatively short beginning that is used for the training of the predictor in the offline phase and the prediction itself in the online phase;
- a suffix, the rest of the trace that is used for the clustering.

In our experiments we used the K-means algorithm [7] for the clustering step and an SVM with a gaussian kernel function [4] for the prediction step of the algorithm outlined above.

We considered the following approaches to model the adjustment curves in the clusters:

1. The constant model. Given the cluster, this function averages over all instances in the cluster and additionally over the time, resulting in a static difficulty adjustment.
2. The regression model. Given the cluster, we train the regularised least squares regression [10] with the gaussian kernel on its instances.

The results stemming from using these models are described in Section 6.

5 Experimental Setup

To test our approach we implemented a rather simple game using the Microsoft XNA framework [1] and one of the tutorials from the XNA Creators Club community, namely "Beginner's Guide to 2D Games" [2]. The player controls a cannon

[1] http://msdn.microsoft.com/en-us/xna/default.aspx
[2] http://creators.xna.com/en-GB/

that can shoot cannonballs. The gameplay consists of shooting down the alien spaceships while they are shooting at the cannon (Figure 1). A total of five spaceships can be simultaneously on the screen. They appear on the right side of the game screen and move on a constant height from the right to the left. The spaceships are generated so that they have a random speed within a specific δ-interval from a given average speed. Whenever one of the spaceships is shot down or leaves the game screen, a new one is generated. At the beginning of the game the player's cannon has a certain amount of hitpoints, which is reduced by one every time the cannon is hit. At random timepoints a repair kit appears on the top of the screen, floats down, and disappears again after a few seconds. If the player manages to hit the repair kit, the cannon's hitpoints are increased by one. The game is over if the hitpoints are reduced to zero or a given time limit of 100 seconds is up.

Fig. 1. A screenshot showing the gameplay.

Additionally to the controls that allow the player to rotate the cannon and to shoot, there are also two buttons by pressing which the player can increase or decrease the difficulty at any point in the game. In the current implementation the difficulty is controlled by the average speed of the alien ships. For every destroyed spaceship the player receives a certain amount of score points, which increases quadratically with the difficulty level. During each game all the information concerning the game state (e.g. the amount of hitpoints, the positions of the aliens, the buttons pressed, etc) is logged together with a timestamp. At the

current state of our work we held one gaming session with 17 participants and collected the data on how the players behave in the game.

Out of all logged features we restricted our attention to the three: the difficulty level, the score, and the health, as they seem to represent the most important aspects of the player's state. The log of each game instance k is in fact a time trace

$$T_k = ((t_1, f_1(t_1), \ldots, f_L(t_1)), \ldots, (t_N, f_1(t_N), \ldots, f_L(t_N))),$$

where $t_1 = 0$, $t_N \leq 100$, and $f_i(t_j)$ is the value of a corresponding feature (Figure 2). Therefore, to model the players we cluster provided by the testers time sequences.

5.1 Technical considerations

Several complications arise from the characteristics of the collected data:

1. Irregularity of the time steps. To reduce the computational load the data is logged only when the game's or the player's state changes (in the case of a simple game used by us it may seem a trivial concern, but this is important to consider for the complex games). As a result for two different game instances k and $\hat{k}$ the time lines will be different:

$$t_{ik} \neq t_{i\hat{k}}.$$

2. Irregularity of the traces' durations. Since there are two criteria for the end of the game (health dropped to zero or the time limit of a hundred seconds is up), the durations of two different game instances k and $\hat{k}$ can be different:

$$t_{Nk} - t_{\hat{N}\hat{k}} \neq 0.$$

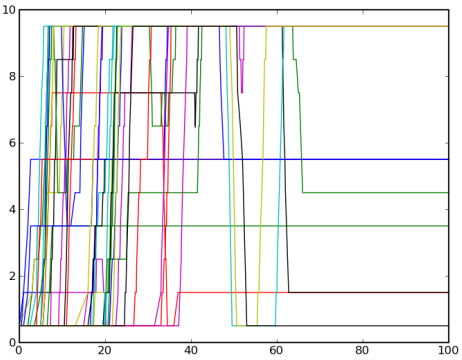
The second problem may appear irrelevant, but as described below it needs to be taken care of in order to create a nice, homogeneous set of data points to cluster.

To overcome the irregularity of the time steps we will construct a fit for each trace and then interpolate the data using the fit for every 0.1 of a second to produce the time sequences with identical time steps:

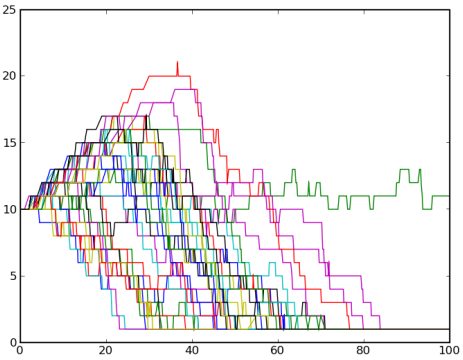
$$T_{kfitted} = ((t_1, f_1(t_1), \ldots, f_L(t_1)), \ldots, (t_N, f_1(t_N), \ldots, f_L(t_N))),$$

where $t_1 = 0$, $t_N \leq 100$, and for each $i \in [2, N]$ $t_i = t_{i-1} + 0.1$.

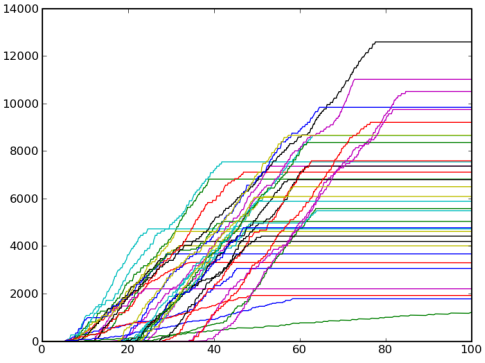
Now it becomes clear why we require the time traces to have equal durations. Since the longest game instances last for a hundred seconds, we need to be able to sample from all of the interpolated traces in the interval between zero and a hundred seconds to create a homogeneous data set. If the original trace was shorter than a hundred seconds, the resulting fitting function wouldn't necessarily provide us with the meaningful data outside of its duration region. Therefore, we augment original game traces in such a way that they all last for a hundred



(a) Difficulty level.



(b) Health.



(c) Score.

Fig. 2. Game traces from one player. Different colours represent different game instances.

seconds, but the features retain their last achieved values (from the "game over" state):

$$T_k = ((t_1, f_1(t_1), \ldots, f_L(t_1)), \ldots, (t_N, f_1(t_N), \ldots, f_L(t_N)), \\ (t_{N+1}, f_1(t_N), \ldots, f_L(t_N)), \ldots, (100, f_1(t_N), \ldots, f_L(t_N))).$$

As mentioned in Section 4, after the augmenting step each time trace is split into two parts:

$$T_{k_{pre}} = ((t_1, f_1(t_1), \ldots, f_L(t_1)), \ldots, (t_K, f_1(t_K), \ldots, f_L(t_K))),$$

$$T_{k_{post}} = ((t_{K+1}, f_1(t_{K+1}), \ldots, f_L(t_{K+1})), \ldots, (t_N, f_1(t_N), \ldots, f_L(t_N))),$$

where t_K is a predefined constant, in our experiments set to 30 seconds, that determines for how long the game observes the player before making a prediction. The *pre* parts of the traces are used for training and evaluating the predictor. The *post* parts of the traces are used for clustering.

6 Evaluation

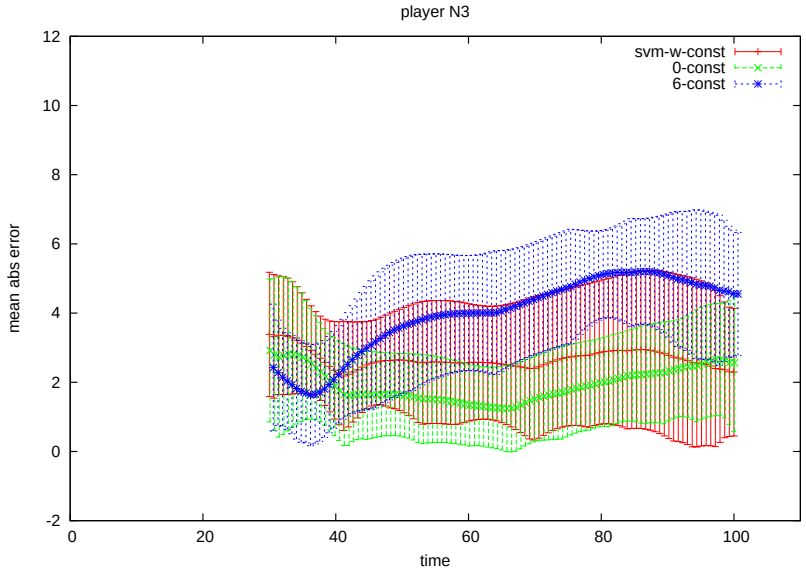
To evaluate the performance of the SVM predictor we conduct a kind of "leave one out" cross-validation on the data. For each player presented we construct a following train/test split:

- training set consists of the game instances played by all players except this one;
- test set consists of all the game instances played by this player.

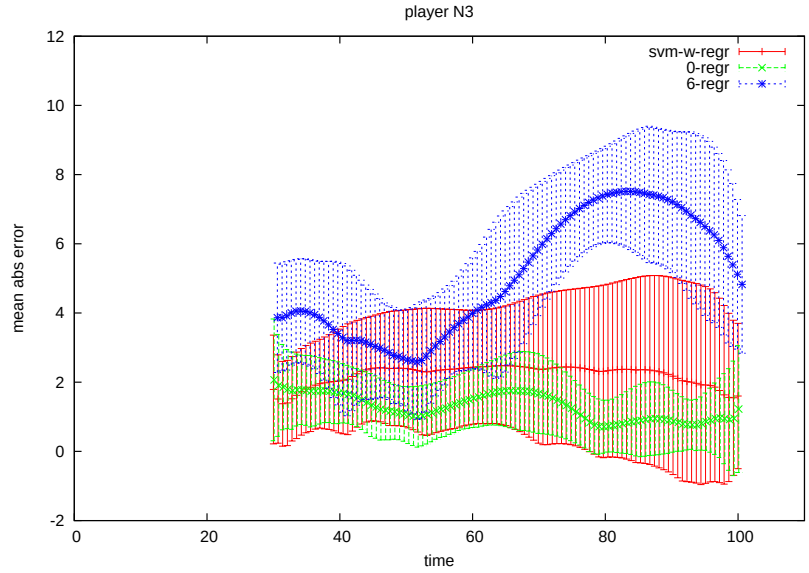
Constructing the train and test sets in this way models a real-life situation of adjusting the game to a previously unseen player. As a performance measure we use the mean absolute difference between the exhibited behaviour in the test instances and the behaviour described by the model of the predicted cluster. The mean is calculated over the test instances.

To provide the baselines for the performance evaluation, we construct for each test instance a sequence of "cheating" predictors: The first (best) one chooses a cluster that delivers a minimum possible absolute error (that is the difference between the predicted adjustment curve and the actual difficulty curve exhibited by this instance); the second best chooses the the cluster with the minimum possible absolute error from the remaining clusters, and so on. We call these predictors "cheating" because they have access to the test instances' data before they make the prediction. For each "cheating" predictor the error is averaged over all test instances and the error of the SVM predictor is compared to these values. As the result we can make some conclusion on which place in the ranking of the "cheating" predictors the SVM one takes.

Figure 3 illustrates the performance of the SVM predictor and the best and the worst baselines for a single player and 7 clusters. We can see from the plots that for each model the SVM predictor displays the performance close to the best cluster. Figure 4 shows that the performance of the SVM predictor averaged over all train/test splits demonstrates similar behaviour.

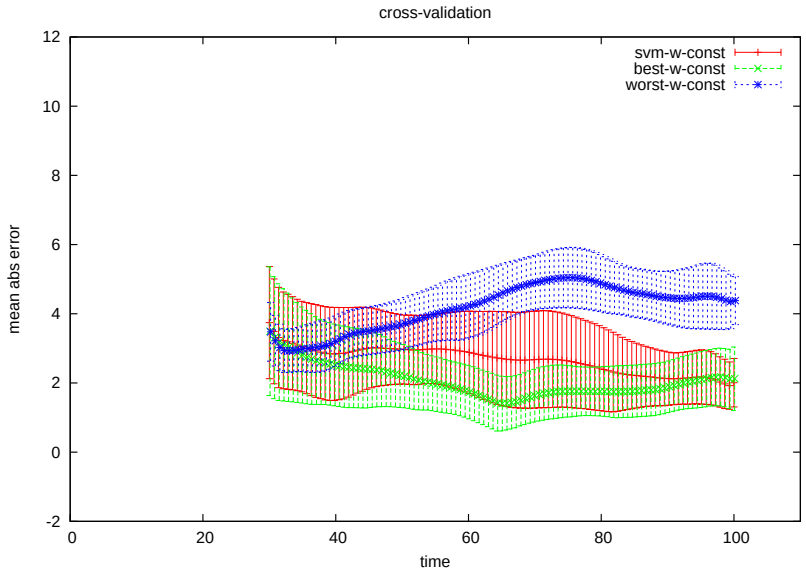


(a) Using the constant model.

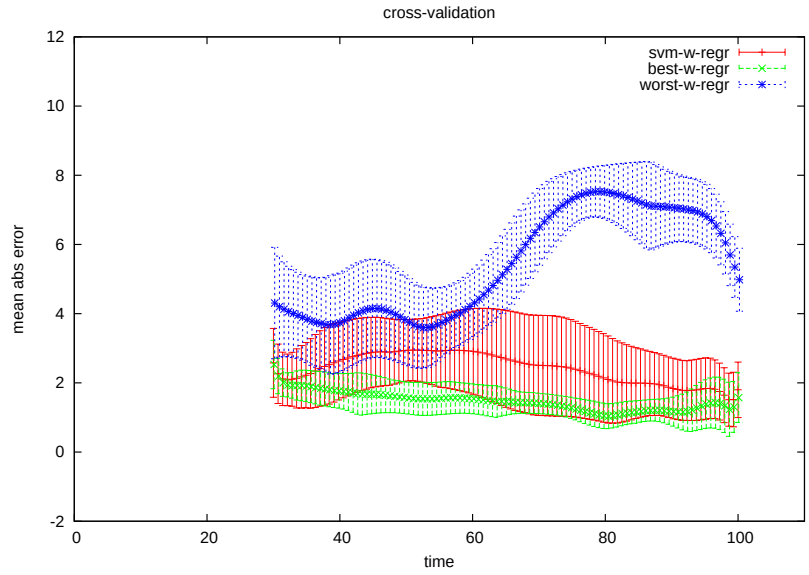


(b) Using the regression model.

Fig. 3. An example of the predictors' performances for one player.



(a) Using the constant model.



(b) Using the regression model.

Fig. 4. The predictors' performance averaged over all train/test splits for 7 clusters.

Statistical Tests

To verify our hypotheses, we performed proper statistical tests with the null hypothesis that the algorithms perform equally well. As suggested recently [6] we used the Wilcoxon signed ranks test.

The Wilcoxon signed ranks test is a nonparametric test to detect shifts in populations given a number of paired samples. The underlying idea is that under the null hypothesis the distribution of differences between the two populations is symmetric about 0. It proceeds as follows: (*i*) compute the differences between the pairs, (*ii*) determine the ranking of the absolute differences, and (*iii*) sum over all ranks with positive and negative difference to obtain W_+ and W_-, respectively. The null hypothesis can be rejected if W_+ (or $\min(W_+, W_-)$, respectively) is located in the tail of the null distribution which has sufficiently small probability.

For settings with a reasonably large number of measurements, the distribution of W_+ and W_- can be approximated sufficiently well by a normal distribution. Unless stated otherwise, we consider the 5% significance level ($t_0 = 1.78$).

Dynamic versus Static Difficulty

We first want to confirm the hypothesis that a dynamic difficulty function is more appropriate than a static one. To eliminate all other influences, we considered first and foremost only a single cluster. In this case, as expected, the dynamic adjustment significantly outperforms the static setting ($t = 2.67$).

We also wanted to compare the performance of dynamic and static difficulty adjustment for larger numbers of clusters. To again eliminate all other influences, we considered the best and the worst "cheating" predictor for either strategy. The t-values for these comparisons are displayed in Table 1.

While varying the amount of clusters from one to fifteen we found out that dynamic difficulty adjustment (the regression model) always significantly outperforms the static one (the constant model) for choosing the best cluster. The same effect we can observe for the worst predictor, but only until the amount of clusters used is greater than ten. For more clusters the static model starts to outperform the dynamic one, probably due to there being insufficient amount of instances in some clusters to train a good regression model. Based on these results in the following we consider only the regression model and vary the amount of clusters from one to ten.

Right versus Wrong Choice of Cluster

As a sanity check, we next compared the performance of the best choice of a cluster versus the worst choice of cluster. To this end we found—very much unsurprisingly—that for any non-trivial number of clusters, the best always significantly outperforms the worst.

This means there is indeed room for a learning algorithm to fill. The best we can hope for is that in some settings the performance of the predicted cluster is

Table 1. t-values for comparison of the constant model vs the regression model for the varying amount of clusters.

c	best-const vs best-regr	worst-const vs worst-regr
1	8.46	8.46
2	6.12	9.77
3	5.39	12.64
4	5.26	11.37
5	4.90	12.62
6	4.77	11.05
7	4.80	10.38
8	4.62	6.83
9	4.61	7.20
10	4.63	4.36
11	4.55	0.71
12	4.68	-0.77
13	4.60	-9.16
14	4.50	-5.54
15	4.57	-13.26

close to, i.e., not significantly worse than, the best predictor while always being much, i.e., significantly, better than the worst predictor.

One versus Many Types of Players

The last parameter that we need to check before coming to the main part of the evaluation is the number of clusters. It can easily be understood that the quality of the best static model improves with the number of clusters while the quality of the worst degrades even further. Indeed, on our data, having more clusters was always significantly better than having just a single cluster for the best predictor using the regression model.

Under the assumption that we do not want to burden the players with choosing their difficulty, this implies that we do need a clever way to automatically choose the type of the player. Adjusting the game just to a single type is not sufficient.

Quality of Predicted Clusters

We are now ready to consider the main evaluation of how well the type of the player can be chosen automatically. As mentioned above the best we can hope for is that in some settings the performance of the predicted cluster is close to the best cluster while always being much better than the worst cluster. Another outcome that could be expected is that performance of the predicted cluster is far from that of the best cluster as well as from the worst cluster.

To illustrate the quality of the SVM predictor we look at its place in the ranking of the "cheating" predictors while varying the amount of clusters. The

Table 2. Results of the significance tests for the comparison of performance of the SVM predictor and "cheating" predictors using the regression model.

	1	2	3	4	5	6	7	8	9	10
1	s									
2	w	b								
3	w	s	b							
4	w	b	b	b						
5	w	s	b	b	b					
6	w	w	b	b	b	b				
7	w	s	b	b	b	b	b			
8	w	s	b	b	b	b	b	b		
9	w	w	s	b	b	b	b	b	b	
10	w	w	s	b	b	b	b	b	b	b

results of the comparison of the predictors' performance for the regression model are shown in Table 2. Each line in the table corresponds to the amount of clusters specified in the first column. The following columns contain values 'w', 's', and 'b', where 'w' means that the SVM predictor displayed the significantly worse performance than the corresponding "cheating" predictor, 'b' for the significantly better performance, and 's' for the the cases where there was no significant difference. The columns are ordered according to the ranking of the "cheating" predictors, i.e. 1 stands for the best possible predictor, 2 for the second best, and so on.

We can observe a steady trend in the SVM predictor's performance: Even though it is always (apart from the trivial case of one cluster) significantly worse than that of the best possible predictor, it is also always significantly better than that of the most other predictors. In other words, regardless of the amount of clusters, the SVM predictor always chooses a reasonably good one.

This last investigation confirms our hypothesis that predicting the difficulty-type for each player based on short periods of gameplay is a viable approach to taking the burden of choosing the difficulty from the players.

7 Conclusion and Future Work

In this paper we investigated the use of supervised learning for dynamical difficulty adjustment. Our aim was to devise a difficulty adjustment algorithm that does not bother the actual players. Our approach to builing a difficulty model consists of clustering different types of players, finding a local difficulty adjustment for each cluster, and combining the local models by predicting the cluster from short traces of gameplay. The experimental results confirm that dynamic adjustment and cluster prediction together outperform the alternatives significantly.

One parameter left out in our investigation is the length of the prefix that is used for the prediction. We will investigate its influence on the predictors'

performance in the future work. We also plan to collect and examine more players' data to see how transferable our algorithm is to the other games. Another direction for the future investigation is the comparison of our prediction model to the other algorithms employed for the time series predictions, such as neural networks or gaussian processes.

Acknowledgments

We would like to thank the anonymous reviewers for their helpful and insightful comments.

References

1. I. Biederman and E. Vessel. Perceptual pleasure and the brain. *American Scientist*, 94(3), 2006.
2. D. Charles and M. Black. Dynamic player modeling: A framework for player-centered digital games. In *Proc. of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, pages 29–35, 2004.
3. D. Cook. The chemistry of game design. Gamasutra, 07 2007.
4. C. Cortes and V. Vapnik. Support vector networks. *Machine Learning*, 20:273–297, 1995.
5. G. Danzi, A. H. P. Santana, A. W. B. Furtado, A. R. Gouveia, A. Leitão, and G. L. Ramalho. Online adaptation of computer games agents: A reinforcement learning approach. *II Workshop de Jogos e Entretenimento Digital*, pages 105–112, 2003.
6. J. Demšar. Statistical comparisons of classifiers over multiple data sets. *Journal of Machine Learning Research*, 7(1), 2006.
7. J. Hartigan and M. Wong. A k-means clustering algorithm. *JR Stat. Soc., Ser. C*, 28:100–108, 1979.
8. R. Herbrich, T. Minka, and T. Graepel. Trueskill$^{\text{tm}}$: A bayesian skill rating system. In *NIPS*, pages 569–576, 2006.
9. R. Hunicke and V. Chapman. AI for dynamic difficulty adjustment in games. *Proceedings of the Challenges in Game AI Workshop, Nineteenth National Conference on Artificial Intelligence*, 2004.
10. R. M. Rifkin. *Everything Old is new again: A fresh Look at Historical Approaches to Machine Learning*. PhD thesis, MIT, 2002.
11. J. Togelius, R. Nardi, and S. Lucas. Making racing fun through player modeling and track evolution. In *SAB06 Workshop on Adaptive Approaches for Optimizing Player Satisfaction in Computer and Physical Games*, pages 61–70, 2006.
12. G. Yannakakis and M. Maragoudakis. Player Modeling Impact on Player's Entertainment in Computer Games. *Lecture notes in computer science*, 3538:74, 2005.