Difficulty Adjustment in Tetris with Time Series*

Diana Lora, Antonio A. Sánchez-Ruiz, Pedro A. González-Calero

Dep. Ingeniería del Software e Inteligencia Artificial
Universidad Complutense de Madrid (Spain)
dlora@ucm.es, antsanch@fdi.ucm.es, pedro@fdi.ucm.es

Abstract. Keeping a player within the flow in a game is a central goal for game designers, making the game neither too easy nor too hard. Dynamic Difficulty adjustment seeks to fulfill this goal by dynamically tuning difficulty according to player actions in the game.
In this paper we demonstrate that case-based reasoning with time series can serve to automatically identify the ability of a player in a game, and thus serve as the input for difficulty adjustment based on player ability. We try different configurations for the generation of the case base from game logs, and compare them in terms of how well they classify player ability.

Keywords: Dynamic Difficulty Adjustment, Time Series, Tetris, Case-based Reasoning

1 Introduction

Player satisfaction is the most important goal for computer games, mainly because if a player does not have fun, then he will stop playing the game [16]. So, maintaining "flow" or the engagement of the player is crucial. According to Nakamura and Csikszentmihalyi [13] in order to maintain "flow" they should perceive challenges that enhance their skills, clear goals and immediate feedback. The challenges proposed should be at the "right" level, in the way that the players are challenged but not overwhelmed.

Even in the most simple games, there are several factors that distinguish one player from another, hence there is not a static configuration in level of difficulty for everyone. If we are able to differentiate players by their interaction within the game, then we had improved their satisfaction.

The dynamic difficulty adjustment (DDA) can be applied in many games, despite its genre [8]. The difficulty is considered a subjective factor which is derived from the interaction between the player and the proposed challenge. This is not a static property, because it changes depending on the time spent by the player mastering a skill [11,7]. According to Daniel Cook in the chemistry of

* Supported by Spanish Ministry of Economy and Competitiveness under grant TIN2014-55006-R

game design [3], players are driven by the desire of mastering new skills. Once the player overcomes a challenge fun is achieved. After that, the game gives rewards for the hard work and creates new challenges to conquer. The game creates a loop of learning-mastery-reward which needs to be balanced in order to keep players interested.

To improve players satisfaction, we can apply an implicit approach in order to use machine learning techniques to adjust a game's difficulty [21]. Dynamic Difficulty Adjustment helps to maintain the right balance depending on the player behavior. However, DDA usually involves a great amount of work for developers and designers that cannot be reused in other video games. That is why research is important to decrease the costs related to development of adaptive video games using different techniques, such us, automatically extracting information from players interaction.

It is possible to know players skill level by evaluating their behavior during gameplay. Their expertise can not be determined from a single action or move. However, it can be resolved with the behavior displayed in a period of time. A time series data is a series of observations made sequentially over time [6,18,20,19]. It distinguishes from static data, because of its numerical and continuous nature which is always considered as a whole instead of a single numerical field.

Our main goal, is to create a general approach for DDA that receive as input the players behavior and ways to modify difficulty of a game, and produces an output with the right level of difficulty. We had started our journey with a simple game like Tetris to determine the best way to achieve this. We extract traces from previous games and build a case base in which each case describes how the user places pieces in the game board. Case-based reasoning is a popular approach for learning by experience [1,4]. The main advantage of employing CBR is that each observation is stored as a concrete problem-solution pair. This way, we have a training set full with past player behaviors, which let us determine with an heuristic which behaviors are "better" than others, tag them and compare them with the behavior of a player which game have not finished. For the purpose of this paper, a game trace describes the evolution of different variables during the game and it may contain data of the interaction between the user and the game as well as data about the virtual environment where user interacts.

The rest of the paper is organized as follows. First, we discus the game and the features extracted from the game traces to characterize the level of skills. Then, we explain our approach to dynamically adjust the difficulty of the game. Next, we present the experimental setup and the results obtained. Finally, the paper closes with related work, conclusions and directions for future research.

2 Tetris Analytics and previous work

Tetris (Figure 1) is a very popular video game in which the player has to place different tetromino pieces that fall from the top of the screen in a rectangular game board. When a row of the game board is complete, i.e. it has no holes,

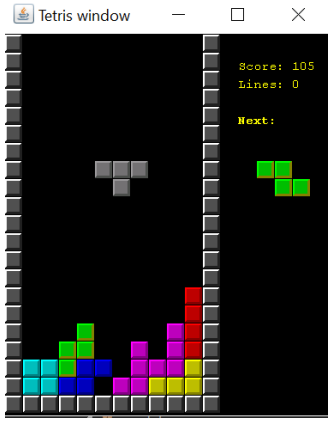

Fig. 1: A screen of the Tetris game.

the row disappears, all pieces above drop one row and the player is rewarded with some points. As the game progresses the new pieces fall faster and faster, gradually increasing the difficulty, until one of the new pieces does not fit in the board and the game ends. The goal of the game, therefore, is to complete as many lines as possible to increase the score and make room for the next pieces. Although the game is quite simple to play, it is also very difficult to master and hard to solve from a computational point of view [2].

In our experiments we use Tetris Analytics, a version of the game that looks like an ordinary Tetris from the point of view of the player but provides extra functionality to extract, store and reproduce game traces. From these traces, we can select the most significant features to characterize the playing style of each player, determine his skill level and dynamically adjust the difficulty of the game. This way, we can personalize the game for each player in order to improve his user experience.

Each time a new piece appears on the top of the game board, the player has to make two different decisions. The first one, that we call *tactical*, is to decide the final location and rotation of the piece. The second decision involves all the *moves* required to lead the piece to its final location. For now we have considered only the tactical decisions in order to define the skill level of the player, but we are aware that we could also extract valuable information from the concrete moves (looking at parameters like speed, cadence, moves undone, ...) and we plan to extend our work to consider them in the future.

In our previous work [10] we have been able to detect 3 different playing styles using clustering techniques on the game traces. We identified each of the clusters with a different set of players depending on their skill level: *newbie*, *average* and *expert*. For example, newbie players tend to place the pieces more often on the left and right sides of the board than in the middle, and they rotate the pieces less than more experienced players. Average and expert players, on the other hand, play most of the time placing pieces in the lower half of the game board

and only when the game is close to the end and the speed of the falling pieces is very high, they are forced to place the pieces in the upper area.

In Dynamic Difficulty Adjustment in Tetris [10], we performed a small experiment in which we used the first tactical decisions in new games to dynamically classify the skill level of the player (looking for the most similar cluster) and adjust the difficulty of the game accordingly. In Tetris there are 2 obvious ways to adjust the difficulty of the game: to change the speed of the falling pieces or to select a different next piece. In the experiments we used the second approach because it is more difficult to detect (player usually do not want to know that we are making the game easier for them). The difficulty of the game was adjusted only once at the beginning of the game depending on the first tactical decisions of the player. The results, although very preliminary, showed that DDA was perceived as something good and improved the game experience for most of the players.

In this work, we focus on the problem of deciding the skill level of the player more dynamically, not only at the beginning but throughout the entire game. This way, we expect to better adjust the difficulty to the progression of the player. In order to do it we have selected a few number of game features that we describe in the next section, and we study how they evolve over time.

3 Data collection and feature selection

In order to collect data we asked 10 different players with diverse skill levels to play a total of 60 games of Tetris Analytics. Next we analyzed the game traces and extracted the following features for every tactical decision the players made in the game, that is, every time they placed a new piece in its final location:

- *Number of piece*: the number of the current piece from the beginning of the game. Since we only consider tactical decisions, the number of piece corresponds to the moment of the game in which this sample is extracted.
- *Current score*: the accumulative game score obtain by the player after placing the current piece. We expect to see a clear correlation between the score during the game and the skill level of the player.
- *Number of holes*: if the pieces are not placed carefully in the board, it is common to leave empty *holes* under other pieces in the same column. Good players tend to compact very well the pieces in the board without leaving holes so that will be easier to complete lines with the next pieces.
- *Height of the board*: measured as the highest row with some piece. Good players tend to play most of the game in the lowest half of the board because each time they complete a line the height of the board decreases in one unit.

For example, the game state in Figure 1 correspond to the 8th piece of the game, the game score will be 105 plus the points obtained for placing the current piece, and the height of the board is 6 because that is the highest occupied row in the last column on the right side of the board. In this way, a game can be seen as a sequence of tuples or *samples* describing these values for each piece placed.

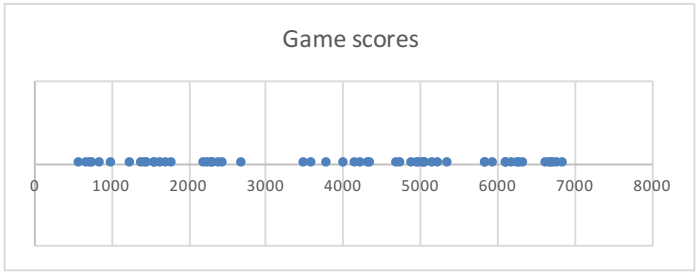

Fig. 2: Scores of the games in the training data.

Note that, in contrast with other approaches that use more complex state representations (like the state of each cell in the board), our features try to summarize the skill of the player from very little information. On the other hand, we will look at the progression of those features over time to classify the player instead of just looking at the current game state.

We can classify each game in a different skill class based on the final score obtained. Figure 2 shows the distribution of the game scores in the traces. We can see groups of points very close to each other in some intervals and some gaps between those intervals. Those gaps are good candidates to split the different skill classes and in fact we used them to range 3 different skill levels:

- Newbie: games with a total score between 0 and 2999.
- Average: games with a total score between 3000 and 5999.
- Expert: games with more than 6000 points.

After the analysis 21.6% of the games were classified in the newbie category, 46.2% in the average category and 32.2% in the expert category. Once we know the category of each game we can tag every sample in that particular game with the same category and use any supervised learning algorithm to build a classifier.

4 Temporal Series and Cases

Samples in the dataset summarize the state of the game (score, number of holes and board height) after the player places each new piece in the board. In general it is not easy to predict the skill level of the player from a static picture of the game, it is much easier if we consider the evolution of the game during some seconds. With this idea in mind, we decided to reorganize the data to create time series describing the evolution of each parameter.

For example, Figure 3 shows the progression of the different parameters during one particular game. The blue line represents the game score. It starts in 0 and always increases because the player gets points each time a new piece appears on the top of the board. The abrupt changes in the score correspond to complete lines removed from the board as the result of a good tactical move.

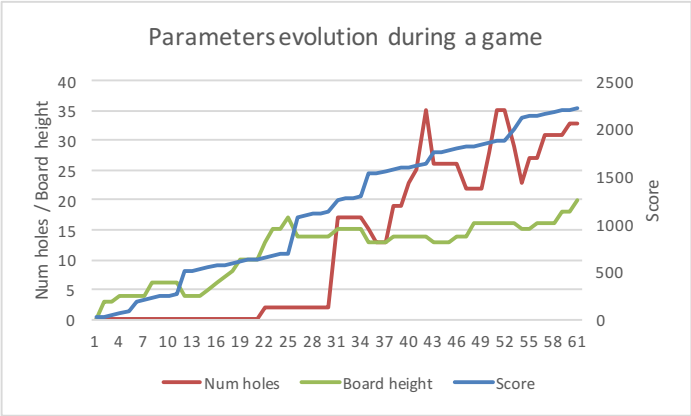

Fig. 3: Evolution of the parameters throughout a game.

In this example the player got 2260 points after 61 pieces so this game will be classified in the *Newbie* category. The green line represents the evolution of the board height and changes from 0 (an empty board) to 20 (last row and end of the game). We can see that the player had some problems around piece 25 when the board height reached the row 17, but he was able to control the situation and keep the board height stable for another 30 pieces. It is interesting to note the correlation between the score and the board height: completed lines decrease the height of the board and quickly increase the game score. Finally, the red line corresponds to the number of holes in the board. Usually, the more holes in the board the more difficult is to complete lines. This last parameter shows a higher variability compared to the other two during the game.

Next we divided each game in small fragments that we call *cases* and represent the evolution of the game during a certain time window. The size of the time window controls what part of the game is captured in each case and determines the length of the time series that we have to consider. Each case is made of 4 parameters: the number of the piece describing the moment of the game in which the case was created; and 3 time series describing the evolution of the score, number of holes and board height during the last n pieces or tactical decisions.

There are different approaches to divide a game into cases. Figure 4 shows the ones we have considered:

- *Single*: each case only describes the current game state and there is a case for each piece in the game. We do not use time series in this first approach (o equivalently the time window size is 1).
- *Consecutive*: we only create new cases every 10 pieces and each case contains 3 non-overlapping time series to describe the evolution of the parameters during the last 10 pieces. A disadvantage of this approach is that the player can only be classified again every 10 pieces.

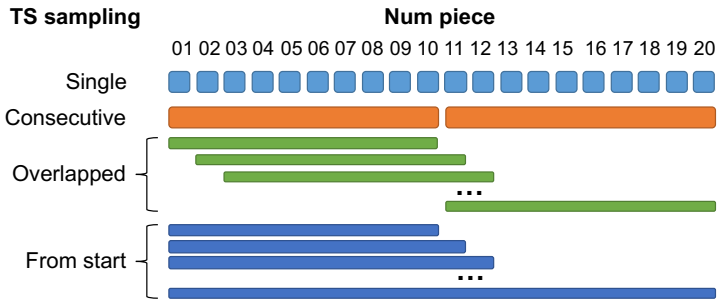

Fig. 4: Different time series sampling approaches.

– *Overlapped*: same idea but now the time series overlap in time, so we create a case describing the evolution of the parameters during the last 10 pieces.
– *From start*: we create a new case for every piece describing the evolution of the parameters from the beginning of the game to the current moment. In this approach time series have different length depending of the moment of the game when they are sampled.

These approaches result in 4 different case bases. In order to decide the skill level of a new player we use a k nearest neighbor classifier (k-NN), and the player skill class is decided by a majority vote among the retrieved cases. It is interesting to note that when we look for similar cases in the case base we only consider those that were created at the same moment in previous games. For example, in order to predict the skill level of the player at piece number 20 we only consider cases created at piece 20 in previous games. This filter reduces significantly the number of cases to consider and let us to compare time series with the same length (even in the *From start* approach).

The similarity between 2 cases is computed as a liner combination of the similarities between the normalized time series. Several different distances have been proposed for time series [18], in this work we use a simple similarity measure based on the euclidean distance:

$$\begin{aligned} sim_c(c_1, c_2) = \alpha_1 \; & sim_{ts}(c_1.score, c_2.score) + \\ \alpha_2 \; & sim_{ts}(c_1.holes, c_2.holes) + \\ \alpha_3 \; & sim_{ts}(c_1.height, c_2.height) \end{aligned}$$

$$sim_{ts}(r, s) = 1 - \sqrt{\sum_{i=1}^{n}(r_i - s_i)^2}$$

As part of the future work, we would like to investigate other distances between time series and study their effect in the case retrieval phase. In the following section we evaluate the performance of each case base.

Classifier	Precision	α_1 (score)	α_2 (num holes)	α_3 (board height)
Clusters	39.64	-	-	-
CBR Single	59.47	0.40	0.50	0.10
CBR Consecutive	64.09	0.50	0.45	0.05
CBR Overlapped	**65.42**	0.70	0.25	0.05
CBR From start	60.53	0.70	0.25	0.05

Table 1: Precision and best weights using 10-NN.

5 Experimental Evaluation

For every approach, we evaluate data from different moments throughout the game to predict players skill level. We do this to avoid two possible scenarios. First, the users cheating the classifier playing like newbie at the beginning and then playing with their real skill level, which will give them great advantage. Second, the user playing odd in brief period of time due to external factors.

Each one of the 4 approaches described before results in 4 different case-based reasoning (CBR) systems. Now we intent to know which approach is more accurate to determine the players skill level throughout the game. This way, we will be able to adjust the difficulty of the game in several moments during gameplay. In the comparison we also include our previous approach based on clustering of game traces [10]. Table 1 summarizes the precision results using 10-fold cross validation and the optimal weights in the similarity function for each classifier.

The *Cluster* based classifier used in our previous work does not perform very well when use it to classify users according to their skill level throughout the entire game. CBR approaches, on the contrary, obtain better results because they are based on a reduced number of selected features (the training data is simpler and less noisy) and because we can tune the similarity measure to provide different importance to each of the parameters. All CBR approaches in Table 1 were evaluated using 10-NN in which the player skill class was decided by a majority vote, and the optimal weights were selected using a exhaustive grid search with increments of 0.05.

It is interesting to note that all the approaches based on time series are better than the *Single* classifier. In particular, the approach based on overlapped time series obtains the highest precision (65.42%) and has the advantage that can be used in any moment of the game after piece number 10. Using time series from the start of the game to the current time seems to have no beneficial effect, probably because the euclidean distance considers equally important all the values in the time series. In games, recent events are usually more important that those that occurred long time ago and we think the precision could be improved using a similarity measure for time series that takes this into account. Regarding the weights, the score is the predominant parameter to decide the skill level as it was expected, followed by the number of holes. The height of the

	k = 1	k = 3	k = 5	k = 10
CBR Single	57.83	58.98	59.82	59.47
CBR Consecutive	60.68	63.64	63.86	64.09
CBR Overlapped	**63.51**	**64.71**	**65.07**	**65.42**
CBR From start	63.46	64.49	62.93	60.53

Table 2: Precision when we retrieve k cases to predict the skill class.

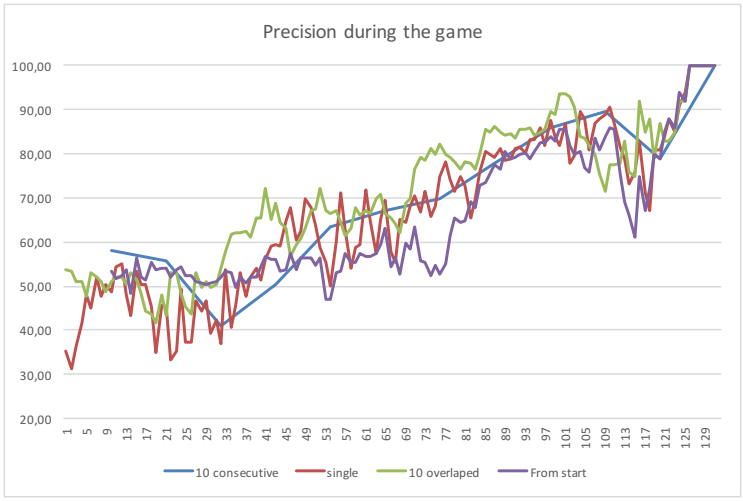

Fig. 5: Precision during the game.

board is negligible, probable because of the correlation between this parameter and the score that was explained in the previous section (see Figure 3).

Table 2 shows the precision of each system when we retrieve a different number of similar cases to make the prediction. Except in the *From start* approach, the more cases we consider to classify the player the higher the precision of the system.

Finally, Figure 5 shows the evolution of the precision during the game. In general, the more advanced is the game the easiest is to predict the level of the player. Note however that in order to dynamically adjust the difficulty of the game and improve the player experience it is important to classify the player soon and adapt the difficulty of the game accordingly.

6 Related Work

Nowdays there are two approaches for setting up the difficulty adjustment of a game. The first approach is to let the user to manually set the level of difficulty of the game (e.g. easy, medium, hard). But this implies that the game have to use a specific heuristic that employs the main features of the game for each

difficulty level. This is a disadvantage because makes it hard to transfer to other games, requires additional testing and programming, which means higher costs [12]. Also, it still continues to be a static adjustment where every main feature is set in a specific value depending on the level of difficulty chosen by the player and will not change during gameplay to adjust accordingly to players evolution. Although, when a player is consider an average user or an expert one, does not mean that all the master skills will be above a expected value. In general, users do not learn all the skills at the same pace. That's why, this approach is not as accurate as desire.

Alternatively, we have Dynamic Difficulty Adjustment which is used for automatically alter games difficulty in order to meet players expectations [9]. One popular approach of DDA is the *Rubber Band AI*, which means that the player and her opponents are virtually bound together by a some kind of "rubber band". This cause that the behavior of the player reflects on how their opponents acts, i.e. if the player displays higher level of skills, then her opponents will have a complex behavior or if we have a newbie, then her opponents will have a simpler behavior [21].

With Machine Learning techniques, we can learn and make predictions based on data resulting from player behavior. Romdhane and Lamontagne [15,14] used a CBR system with Tetris to assess the quality of the cases and to maintain control in the size of the case base. They discover that case usage is the best criteria for forgetting cases. Also, Floyd and Esfandiari [5] created an agent based on case-based reasoning framework which learn by observation. In one of the cases presented, the agent learn to play Tetris from an external source. With CBR systems, it is possible to determine the skill level of a player based on how others played previously. And therefore, been able to adjust the difficulty of the game according to each player level. Moreover, time series had been used to know the evolution of a player during gameplay. Menéndez et al. [17] are able to extract gamer profile evolution based on the application of time series clustering techniques from their interaction with an Action-RPG game called "Dream".

7 Conclusions

In this work, we present a game called "TetrisAnalytics", which is like a simple Tetris game, but allows you to save the players interaction and the state of the board. During gameplay the user have to avoid the pieces reach the top of the board by doing lines. The game extract the number of pieces, current score, number of holes and height of the board per tactical move, i.e. every time the player locate a piece on the board, we collect the mentioned features.

With the data collected, we are able to create a database of past experiences and each tactical move can be classified by newbie, average and expert depending on the total score of the game to it's belong. By creating new cases every 10 consecutive tactical moves, each case containing 3 overlapped time series, we are able to describe the evolution of the parameters during that period of time. This way, the player only have to play without DDA the first 10 tactical moves. This

approach give us a better accuracy for predicting the players skill level, because not only evaluates the individual moves, but the evolution of the player in a period of time.

As future work, we will implement this approach in "TetrisAnalytics" and verify with new players if their user experience is improved notably by applying dynamic difficulty adjustment during gameplay in comparision to our previous experiments where difficulty was adjusted just once in the game.

References

1. A Case-Based Reasoning Framework for Developing Agents Using Learning by Observation. Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on pp. 531–538 (2011)
2. Breukelaar, R., Demaine, E.D., Hohenberger, S., Hoogeboom, H.J., Kosters, W.A., Liben-Nowell, D.: Tetris is hard, even to approximate. Int. J. Comput. Geometry Appl. 14(1-2), 41–68 (2004), http://dx.doi.org/10.1142/S0218195904001354
3. Cook, D.: The chemistry of game design. http://www.gamasutra.com/view/feature/129948/the_chemistry_of_game_design.php (2007), consultado: 2015-05-26
4. Fagan, M., Cunningham, P.: Case-Based Plan Recognition in Computer Games. In: Case-Based Reasoning Research and Development, pp. 161–170. Springer Berlin Heidelberg, Berlin, Heidelberg (2003), http://link.springer.com/10.1007/3-540-45006-8{_}15
5. Floyd, M.W., Esfandiari, B.: A case-based reasoning framework for developing agents using learning by observation. In: Tools with Artificial Intelligence (ICTAI), 2011 23rd IEEE International Conference on. pp. 531–538. IEEE (2011)
6. Fu, T.C.: A review on time series data mining. Engineering Applications of Artificial Intelligence 24(1), 164–181 (2011), http://dx.doi.org/10.1016/j.engappai.2010.09.007
7. Hunicke, R.: The case for dynamic difficulty adjustment in games. In: Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology. pp. 429–433. ACM (2005)
8. Hunicke, R., Chapman, V.: AI for dynamic difficulty adjustment in games. Challenges in Game Artificial Intelligence AAAI ... pp. 91–96 (2004), http://www.aaai.org/Papers/Workshops/2004/WS-04-04/WS04-04-019.pdf
9. Jennings-Teats, M., Smith, G., Wardrip-Fruin, N.: Polymorph: dynamic difficulty adjustment through level generation. In: Proceedings of the 2010 Workshop on Procedural Content Generation in Games. p. 11. ACM (2010)
10. Lora, D., Sánchez-Ruiz, A.A., González-Calero, P.A., Gómez-Martín, M.A.: Dynamic difficulty adjustment in tetris (2016)
11. Missura, O., Gärtner, T.: Player modeling for intelligent difficulty adjustment. In: Discovery Science, 12th International Conference, DS 2009, Porto, Portugal, October 3-5, 2009. pp. 197–211 (2009)
12. Missura, O., Gärtner, T.: Predicting dynamic difficulty. In: Shawe-Taylor, J., Zemel, R.S., Bartlett, P.L., Pereira, F., Weinberger, K.Q. (eds.) Advances in Neural Information Processing Systems 24, pp. 2007–2015. Curran Associates, Inc. (2011), http://papers.nips.cc/paper/4302-predicting-dynamic-difficulty.pdf

13. Nakamura, J., Csikszentmihalyi, M.: Flow and the Foundations of Positive Psychology: The Collected Works of Mihaly Csikszentmihalyi, chap. The Concept of Flow, pp. 239–263. Springer Netherlands, Dordrecht (2014), http://dx.doi.org/10.1007/978-94-017-9088-8_16
14. Romdhane, H., Lamontagne, L.: Forgetting reinforced cases. In: Advances in Case-Based Reasoning, 9th European Conference, ECCBR 2008, Trier, Germany, September 1-4, 2008. Proceedings. pp. 474–486 (2008), http://dx.doi.org/10.1007/978-3-540-85502-6_32
15. Romdhane, H., Lamontagne, L.: Reinforcement of local pattern cases for playing tetris. In: Proceedings of the Twenty-First International Florida Artificial Intelligence Research Society Conference, May 15-17, 2008, Coconut Grove, Florida, USA. pp. 263–268 (2008), http://www.aaai.org/Library/FLAIRS/2008/flairs08-066.php
16. Sweetser, P., Wyeth, P.: Gameflow: a model for evaluating player enjoyment in games. Computers in Entertainment (CIE) 3(3), 3–3 (2005)
17. Tom, C.F.: Combining Time Series and Clustering to Extract Gamer Profile Evolution pp. 262–271 (2014)
18. Warren Liao, T.: Clustering of time series data-a survey. Pattern Recogn. 38(11), 1857–1874 (Nov 2005), http://dx.doi.org/10.1016/j.patcog.2005.01.025
19. Yang, K., Shahabi, C.: A multilevel distance-based index structure for multivariate time series. In: Temporal Representation and Reasoning, 2005. TIME 2005. 12th International Symposium on. pp. 65–73. IEEE (2005)
20. Yang, K., Shahabi, C.: An efficient k nearest neighbor search for multivariate time series. Information and Computation 205(1), 65–98 (2007)
21. Yannakakis, G.N., Hallam, J.: Real-time game adaptation for optimizing player satisfaction. IEEE Transactions on Computational Intelligence and AI in Games 1(2), 121–133 (2009)