Automated Adaptation and Assessment in Serious Games: a Portable Tool for Supporting Learning

Enkhbold Nyamsuren, Wim van der Vegt, Wim Westera

PenOW, Open University of the Netherlands
(enkhbold.nyamsuren, wim.vandervegt, wim.westera)@ou.nl

Abstract. We introduce the Adaptation and Assessment (TwoA) component, an open-source tool for serious games, capable of adjusting game difficulty to player skill level. Technically, TwoA is compliant with the RAGE (Horizon 2020) game component architecture, which offers seamless portability to a variety of popular game development platforms. Conceptually, TwoA uses a modified version of the Computer Adaptive Practice algorithm. Our version offers two improvements over the original algorithm. First, the TwoA improves balancing of player's motivation and game challenge. Second, TwoA reduces the selection bias that may arise for items of similar difficulty by adopting a fuzzy selection rule. These improvements are validated using multi-agent simulations.

1 Introduction

Serious games [1, 2] are becoming an effective tool for pedagogy and learning in general [3]. In this domain, one of the questions we are interested in is how to assess player's learning progress. Player assessment can provide teachers and students with formative and summative information about learning progress. Data from the player assessment can be used to dynamically adjust game mechanics which in turn improves the learning experience.

We introduce the Adaptation and Assessment (TwoA) component, an open-source library that offers automated game difficulty adaptation and player's learning assessment. TwoA is being developed within the RAGE project [4], an EU's initiative for supplying serious game developers with portable and reusable open-source software components providing pedagogical utility.

In TwoA, we implemented a modified version of the Computerized Adaptive Practice algorithm [5] for game difficulty and player skill assessments and a real-time adaptation of the game difficulty to the player skill. The CAP algorithm offers many benefits. First, it was extensively validated in many studies involving human players [6–8]. Second, it was specifically designed for serious games to assess and match game difficulty to player skill to promote learning. It is a major distinction from existing matchmaking algorithms, such as TrueSkill [9] or variations of Elo [10], that are aimed at competitive matching of two human players. Finally, the CAP algorithm is not proprietary.

TwoA's version of the algorithm provides two main benefits over the original CAP algorithm. First, we describe and validate improvements to CAP's real-time adaptation of game difficulty. Second, TwoA adopts an RAGE-client architecture [11] making the TwoA component easy to integrate and use with game development platforms.

2 Computerized Adaptive Practice System

In this section, we briefly introduce the original CAP algorithm. Not all equations are discussed here. For a more in-depth overview of the CAP system, the reader can refer to the original study [5].

The CAP system assumes that a player m should have a skill rating θ_m to solve a problem i with a difficulty rating β_i. Given above notions, the CAP system provides two main functions. First, it can assess the skill ratings θ_m and the difficulty rating β_i based on the player m's performance in the problem i. Second, given a player with a known skill rating θ, the CAP system can recommend a problem with the desired difficulty β where the player has a probability P of successfully solving the problem.

Assessment of ratings depends on the accuracy x_{im} and the response time t_{im}. If the player m is able to solve the problem i then x_{im} is one and zero otherwise. t_{im} is time the player m spent on the problem i. These two measures are translated into the observed score S_{im} in equation 1 using the High Speed High Stakes scoring rule [6] that accounts for the speed-accuracy trade-off. The expected score $E(S_{im})$ is calculated based on the skill rating θ_m and difficulty rating β_i as shown in equation 1. The term d_i is time limit for problem i. Following equation 2, the difference between $E(S_{im})$ and S_{im} is used to update the two ratings using a modified Elo function [10]. The terms K_m and K_i are factors reflecting uncertainties in measurements of skill and difficulty ratings [12]. Equations 1 and 2 are of little relevance to our study and described for the purpose of providing a basic understanding of the CAP system's dynamics.

$$S_{im} = (2x_{im} - 1)(1 - t_{im}/d_i); E(S_{im}) = \frac{e^{2(\theta_m - \beta_i)} + 1}{e^{2(\theta_m - \beta_i)} - 1} - \frac{1}{\theta_m - \beta_i} \tag{1}$$

$$\tilde{\theta}_m = \theta_m + K_m \left(S_{im} - E(S_{im})\right); \tilde{\beta}_i = \beta_i + K_i \left(E(S_{im}) - S_{im}\right) \tag{2}$$

$$\beta_t = \theta_m + \ln\left(P_t/(1 - P_t)\right) \tag{3}$$

The problem selection process involves three steps: (a) defining a target probability of success P_t, (b) estimating a target difficulty rating β_t, and (c) selecting a problem that closely matches the target difficulty rating. Equation 3 is used for estimating the β_t. P_t is drawn from a normal distribution $N(P = 0.75, SD = 0.1)$ and restricted such that $0.5 < P_t < 1$. Such value of P_t allows the player to maintain an average success rate of 75% [5, 13]. According to these studies, the success rate of $P = 0.75$ provides a reasonable balance between keeping a player motivated and maintaining measurement accuracies of ratings. The problem i is selected if it has the difficulty rating closest to the β_t: $min|\beta_i - \beta_t|$. We refer to it as a minimum distance rule. The original study [5] provides a full description of the CAP algorithm including the recommended values for the free parameters. In our study, we used the recommended values. In other cases, we explicitly mention values used by TwoA.

3 Improving the Problem Selection in the CAP system

We made two improvements to the problem selection algorithm of the CAP system. First, we changed equation 3 so that the target difficulty rating β_t reflects better the target probability of success P_t. Second, we changed the selection criteria for a problem i to minimize the chances of having a selection bias for a particular item.

3.1 Maintaining the Target Success Rate

The problem difficulty rating β and the player skill rating θ are measured along the same scale and directly comparable. For example, equation 3 shows that the skill rating and the difficulty rating are equal $\beta_t = \theta_m$ if $P_t = 0.5$. Logically, the following properties should hold: $\theta_m > \beta_t$ if $P_t > 0.5$, and $\theta_m < \beta_t$ if $P_t < 0.5$. If these properties do not hold then the player's actual success rate may not follow the normal distribution $N(P, SD)$ from which the P_t is drawn. These properties are not enforced by equation 3. For example, $\theta < \beta_t$ when $P_t = 0.75$. To address the issue, we changed the sign for the logarithmic component resulting in equation 4. The new logarithmic component $ln((1-P_t)/P_t)$ is always negative when $P_t > 0.5$ and always positive when $P_t < 0.5$. Thus, the above-mentioned properties always hold.

$$\beta_t = \theta_m + \ln((1 - P_t)/P_t) \tag{4}$$

3.2 Problem Selection Bias

Problems of the same difficulty may have small differences in difficulty ratings estimated by the CAP. Let us assume that problems i and j have the same difficulty but slightly different ratings β_i and β_j. This discrepancy in ratings can affect the problem selection and result in a bias. Let us assume that the target difficulty rating β_t is closer to β_i. Ideally, there should be a 50% chance of selecting either problems i or j. However, the problem i is preferred due to the minimum distance rule. If β_t is repeatedly estimated closer to the β_i then the problem i is repeatedly chosen over the problem j. The issue can become worse since the problem's rating is re-estimated after each administration resulting in an increasing discrepancy between β_i and β_j. This is an undesirable feature. Ideally, problems of similar difficulty should be administrated equally often so that ratings are updated at the same rates and stay close to each other.

One way to address the above issue is to select the least played problem among the N number of problems closest to the β_t. For example, problems i and j can be administered in turns if $N = 2$. The drawback with this approach is the difficulty of finding an appropriate value for N. If N is too big then it may include problems with ratings too distant from β_t. Administration of such problems will affect negatively the system's ability to maintain the desired success rate P.

We propose a solution inspired by fuzzy intervals used in fuzzy systems. Instead of the single probability P_t, we use two core probabilities $P_{c,L}$ and $P_{c,U}$, and two support probabilities $P_{s,L}$, and $P_{s,U}$ such that $P_{s,L} < P_{c,L} < P_{c,U} < P_{s,U}$. The core probabilities $P_{c,L}$ and $P_{c,U}$ are randomly drawn from a normal distribution $N(P, SD)$ where P is a desired success rate. $P_{s,L}$ is randomly drawn from a normal distribution $N(P-w*SD, SD)$ provided that $P_{s,L} < (P-w*SD)$. Similarly, $P_{s,U}$ is randomly drawn from a normal distribution $N(P+w*SD, SD)$ so that $P_{s,U} > P - w * SD$. The term w is a weight parameter controlling the amount of shift in distributions' means. Its default value is one so that there is a distance of one standard deviation between distributions from which support and core probabilities are drawn. The term w is the only new free parameter added to TwoA. For the parameters inherited from the CAP algorithm, TwoA uses the default values recommended in the original study [5].

With four probability values, we calculate four difficulty ratings with equation 4: $\beta_{c,L}$, $\beta_{c,U}$, $\beta_{s,L}$, and $\beta_{s,U}$. Given these ratings, all problems can be divided into three categories: problems with difficulty ratings within the core range $[\beta_{c,L}, \beta_{c,U}]$; problems with ratings within one of two support ranges $[\beta_{s,L}, \beta_{c,L})$ and $(\beta_{c,U}, \beta_{s,U}]$; and problems with ratings outside of the range $[\beta_{s,L}, \beta_{s,U}]$. Any problem within the core range is preferred to the problems outside of the core range. Any problem within the range $[\beta_{s,L}, \beta_{s,U}]$ is preferred to the problems outside of it. Within the core and support ranges, the least played problem is preferred to others. If the range $[\beta_{s,L}, \beta_{s,U}]$ does not contain any problems, then the problem with the rating closest to the range is chosen.

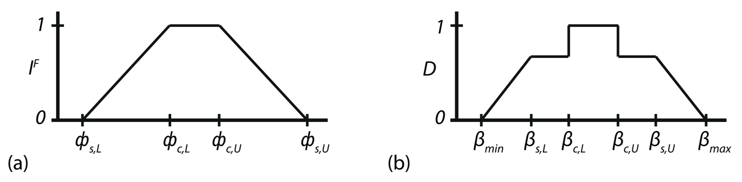

Fig. 1. (a) A fuzzy interval specified by four parameters ϕ. (b) A visualization of four support and core ratings forming a shape resembling a fuzzy interval.

This solution is inspired by fuzzy selection rules [14] used in fuzzy logic. A fuzzy rule consists of an antecedent with one or more selectors (predicates) and a consequent with a class assignment. In an ordinary rule, a selector is a binary condition verifying if some value k belongs to some interval $I = [u, v]$. In a fuzzy rule, the selector has a fuzzy interval defined with trapezoidal membership function (e.g., Fig. 1a) specified by four parameters ϕ. Given some value k, the degree k belongs to the interval I^F is defined by the position of k relative to the four parameters. The likelihood of the term k belonging to the interval I^F decreases as k's distance to the core interval $[\phi_{c,L}, \phi_{c,U}]$ increases.

Fig. 1b visualizes the four support and core ratings. β_{max} and β_{min} are the maximum and minimum ratings among all problems. The term D is a reverse distance to the desired success rate P calculated as $D = (P - w * SD)/P$. For the core range, the weight parameter w is equal to zero. For the support ranges, w can be set to a positive non-zero value. For the remaining two ranges, $D < (P - w * SD)/P$. As can be observed, the shape roughly replicates the trapezoidal membership function. It provides a fuzzy estimation of problem's closeness to the desired success rate P. The problems with the highest reverse distance D are preferred, but problems' frequencies are integrated as a nested selection criterion for problems inside the range $[\beta_{s,L}, \beta_{s,U}]$. Therefore, the fuzzification avoids bias toward either of selection criteria defined by success rate or problem frequency.

4 Validation of TwoA's Algorithm

4.1 Simulation tool

We validate our improvements to the algorithm using multi-agent simulations. For this purpose, we used the game TileZero, a variation of the Qwirkle game (released by MindWare, `http://www.mindware.com`). Qwirkle is a turn-based board game with colored and shaped tiles. There are six colors and six shapes resulting in 36 unique tiles. Two or more players compete against each other. A player can put up to six tiles on the board per turn. The goal is to build sequences of tiles where each sequence has the same color and different shapes or vice versa. The player earns a score equal to the length of the sequences built in the turn. The player with the highest score wins. The game was chosen for its clear success criteria, short game duration, easy difficulty level generation, and easy AI coding.

In TileZero, a human player can play against one of six AI opponents of varying difficulties. We refer to the six AI opponents as *Very Easy*, *Easy*, *Medium Color*, *Medium Shape*, *Hard* and *Very Hard*. The *Very Easy* opponent puts only one tile per turn. The *Easy* opponent puts a random combination of tiles per turn. The *Medium Color* opponent puts a combination of tiles of the same color that gives the maximum score per turn. The *Medium Shape* opponent does the same but using tiles of the same shape. The *Hard* opponent always selects the combination with the second highest score among all available combinations of tiles. The *Very Hard* opponent always selects the combination resulting in the highest score per turn.

We can objectively evaluate difficulties of AI opponents by having them to play against each other. Each AI opponent played 4000 games against the *Very Hard* opponent. The win rates are 0.0, 0.02, 0.23, 0.23, 0.23, and 0.49 for *Very Easy*, *Easy*, *Medium Color*, *Medium Shape*, *Hard* and *Very Hard* respectively. These win rates can be correlated with difficulty ratings to verify ratings' validity.

4.2 Simulation 1: Demonstrating TwoA's Adaptive and Assessment Capabilities

In this subsection, we demonstrate that TwoA's adaptive and assessment capabilities are not negatively affected by the modifications to the original CAP algorithm.

Simulation Setup The simulations consisted of ten independent blocks where parameters were reset to initial values after each block. The block design was used to compensate for random factors present in the game and TwoA. The human player was simulated by a "learning" AI. We refer to it as the player. Each block consisted of 2400 games played in sequence by the player. The player adopted a new strategy after every 400 games. The strategies changed in the following order: *Very Easy*, *Easy*, *Medium Color*, *Medium Shape*, *Hard* and *Very Hard*. Thus, the player started the first game with the same strategy as the *Very Easy* opponent and played the last 400 games using the strategy from the *Very Hard* opponent. These changes in strategies simulated gradual learning in human players.

In each block, the player and all opponents started with the rating of one. The opponent to the player was selected by TwoA at the beginning of each game. TwoA re-estimated the player's skill rating and the AI opponent's difficulty rating after each game. The target probability P_t was drawn from a normal distribution $N(P = 0.75, SD = 0.1)$. For all other free parameters, TwoA used values recommended by the original study of the CAP algorithm [5].

First, TwoA is expected to estimate the difficulty ratings of the opponents so that there is a high and significant correlation between ratings and the win rates. Second, TwoA is expected to capture the learning in the player. The player's skill rating should gradually increase after every 400 games. The exceptions are when the player transitions from *Medium Color* to *Medium Shape* and to *Hard* given their similar objective difficulties. In these cases, there should be a plateau in player's ratings since no learning is happening.

Simulation Results First, we explore changes in player's skill rating and opponents' difficulty ratings. Fig. 2a shows how these ratings changed over the course of 2400 games. The ratings were averaged over all ten blocks. Standards errors are too small to be visually identifiable. The horizontal dashed line indicates the starting rating of one. The vertical dashed lines indicate the points of strategy transitions.

The initial skill rating of the player is an overestimation relative to the difficulty rating of the AI opponents. TwoA corrects it by lowering the skill rating within the first 100 games. After the transition to the *Easy* strategy, there is a rapid increase in the player's skill rating. The next transition to the *Medium Color* strategy also results in an increase of the skill ratings. It is followed by a plateau for the next 800 games. It is expected since *Medium Color*, *Medium Shape* and *Hard* are similar in difficulty. Finally, the transition to the *Very Hard* strategy invokes another increase in the player's skill ratings. Overall, TwoA was able to capture the learning process happening in the simulated player.

TwoA also adjusted its recommendations of the opponents based on the player's learning progress. Fig. 2b shows frequencies of the opponents in every 400 games. These are mean values averaged from all 10 blocks. Standard errors are too small to be visible. Note how the frequency of the *Very Easy* opponent drops almost to zero in the second half of the games. This opponent was too easy for the player using the *Medium Color*, *Medium Shape*, *Hard* or *Very Hard* strategy. TwoA reacted by administering more frequently the *Easy* opponent instead. In the last 400 games, the *Easy* opponent became less challenging resulting in the decreased frequency of its administrations.

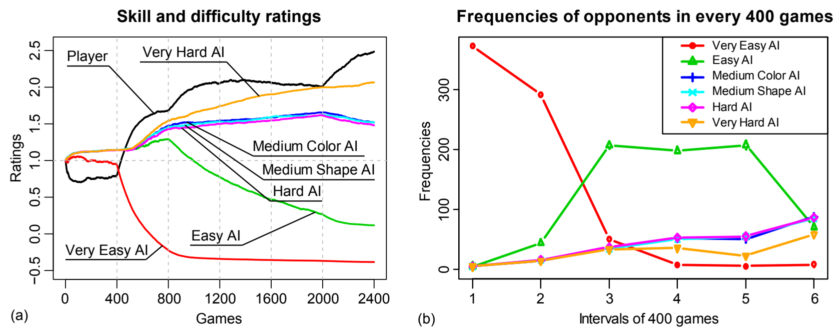

Fig. 2. Player skill rating and opponents' difficulty ratings over 2400 games.

Note how changes in the opponents' frequencies reflect on its difficulty ratings shown in Fig. 2a. As the frequencies of opponents increase, TwoA is able to gain more information about their difficulties and gradually correct the ratings. We can estimate the accuracy of difficulty ratings by correlating them with the win rates. The ratings after 2400 games are -0.384, 0.117, 1.520, 1.519, 1.48 and 2.066 for *Very Easy*, *Easy*, *Medium Color*, *Medium Shape*, *Hard* and *Very Hard* opponents, respectively. The Pearson's product-moment correlation between these ratings and the win rates is $r(4) = .92, p < .01$. This is a very high correlation indicating that TwoA was able to accurately capture relative difficulties of AI opponents.

4.3 Simulations 2 and 3: Original Versus Adjusted Log Probability Models

In this and the following subsection, we demonstrate how changes to the original CAP algorithm improved TwoA's performance. In this subsection, we describe two simulations.

Each simulation consisted of ten blocks with 1000 games per block. The human player was imitated by AI that adopted the same strategy as the *Very*

Hard opponent. We refer to it as the player. In each game, the player had to play against one of six AI opponents. The opponent was selected by TwoA at the beginning of the game. The selection algorithm used by TwoA differed in simulations. TwoA re-estimated the player's skill rating and the AI opponent's difficulty rating after each game. These estimates were reset at the start of a new block. The starting difficulty ratings for the AI opponents are -0.369, 0.268, 1.656, 1.624, 1.613, and 2.0 for *Very Easy*, *Easy*, *Medium Color*, *Medium Shape*, *Hard* and *Very Hard* respectively. The Pearson's product-moment correlation between these ratings and the win rates is $r(4) = .88$, $p = .02$. The starting skill rating for the player was 2.011. The starting difficulty ratings and player's starting skill rating were taken at the end the 2000-th game of simulation 1.

In simulation 2, TwoA used the original CAP equations [5] without any modifications. In simulation 3, equation 3 was substituted with equation 4. In both simulations, the target probability P_t was drawn from a normal distribution $N(P = 0.75, SD = 0.1)$. Therefore, the player was expected to achieve an average win rate of 75% in each block of games. For all other free parameters, TwoA used values recommended by the original study of the CAP algorithm [5]. We compared the results from the two simulations to identify the equation that is best able to maintain the expected win rate.

Fig. 3 shows how win rates changed every 200 games within a block of 1000 games. The values are averages of all ten blocks. Standard errors are too small to be shown on the graph. In simulation 2, the player achieved the average win rate of 49% ($SE < 1\%$) in each block. This is significantly lower than expected 75%. The low win rate is explained by the fact that the original algorithm selected *Very Hard* as the opponent in most games. The opponents with lower difficulties were mostly ignored due to overestimation of the target beta β_t. With $\theta = 2.011$ and $P_t = 0.75$, equation 3 results in $\beta_t = 3.11$. With this high target beta, the original algorithm is highly biased toward the *Very Hard* opponent.

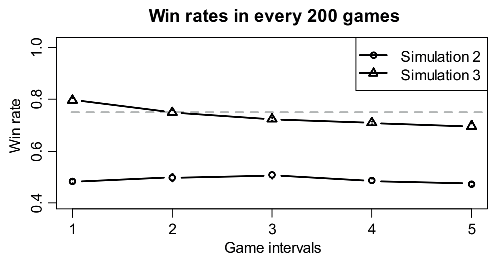

Fig. 3. Win rates in every 200 games. Values were averaged over all 10 blocks.

In simulation 3, the player achieved the average win rate of 74% ($SE < 1\%$) per block. This is in good agreement with the predefined expectation value. It is enabled by equation 4 which resulted in a more liberal target beta allowing selection of less difficult opponents. With $\theta = 2.011$ and $P_t = 0.75$, equation 4 results in $\beta_t = 0.91$. With such target beta, TwoA most often selected *Easy*, *Medium Color*, *Medium Shape*, and *Hard* opponents. Occasionally, *Very Easy* and *Very Hard* are selected due to the stochastic nature of the target probability P_t. The slight downward tendency in Fig. 3 is a stochastic walk following the normal distribution $N(P = 0.75, SD = 0.1)$.

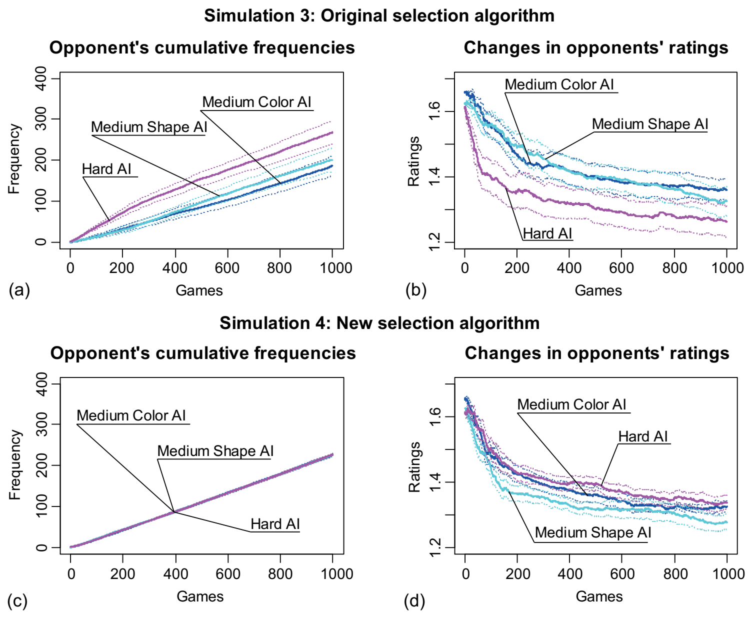

Fig. 4. (a) The original algorithm has a selection bias indicated by differences in cumulative frequencies of the opponents. (b) Large divergence in difficulty ratings due to the selection bias. (c) The new selection algorithm decreases the selection bias. (d) The difficulty ratings of three opponents remain close to each other.

4.4 Simulations 3 and 4: Original Versus Adjusted Item Selection Rules

In this section we reused simulation 3. The algorithm used in simulation 3 still suffers from the selection bias. Three AI opponents, *Medium Color*, *Medium Shape* and *Hard*, have the same objective difficulty. In an ideal situation, TwoA should be selecting these opponents equally often and their difficulty ratings should not diverge much. Fig. 4a shows cumulative frequencies of the three opponents. The frequencies were averaged over all 10 blocks. The dotted lines indicate standard errors. There is a clear selection bias toward the *Hard* opponent. During the first 200 games, the *Hard* opponent was clearly favored over the two other opponents resulting in steep increases in frequency discrepancies.

The bias is due to the *Hard* opponent having the starting rating ($\beta_{Hard} = 1.613$) closest to the starting target beta ($\beta_t = 0.91$). The selection bias diminished in later games due to the self-correcting nature of the algorithm. Hence, the lines in both graphs of Fig. 4 are becoming parallel. However, as Fig. 4b shows, the bias caused the *Hard* opponent's difficulty rating to diverge from the ratings of the other two opponents. The divergence is relatively small in this case. Yet, it is desirable to avoid it. In a real system with multiple players, the divergence may increase sharply for the ratings to become significantly different.

In simulation 4, TwoA used equation 4 and the fuzzy rule for selecting items described in section 3.2. As in previous simulations, the target probability P_t was drawn from a normal distribution $N(P = 0.75, SD = 0.1)$. The term w was set to one for calculating the support probabilities. Default values were used for other parameters inherited from the CAP algorithm. Results of the simulation are shown in Fig. 4. Fig. 4c depicts cumulative frequencies of three AI opponents: *Medium Color*, *Medium Shape* and *Hard*. Cumulative frequencies were averaged over the ten blocks. Standard errors are too small to be visually identifiable. Unlike in simulation 3, the frequencies of the opponents stay close to each other indicating that the opponents were chosen equally often by TwoA's new fuzzy selection rule. The absence of bias also has a positive effect on the ratings as shown in Fig. 4d. The ratings are much closer to each other compared to those in Fig. 4b. Overall, we can conclude that the fuzzy rule was able to better compensate for the small discrepancies in the ratings of the problems of similar difficulty.

5 Discussion

In the future, we are planning to test TwoA in a game environment with human players. To this end, we are collaborating with game development companies within the RAGE project to create and test practical serious games that make use of TwoA. While our simulations showed that the modified algorithm works well, unexpected issues may arise in real-time applications especially in those that involve large numbers of players and problems. We are especially interested in validating the modified algorithm with cases where multiple players can simultaneously access the same problem set. This will allow us to verify the robustness of the algorithm in selecting problems for multiple (simultaneous) users. Finally, we are looking for opportunities to collaborate with the authors of the CAP system that may give us access to large amount of empirical data on which we can test TwoA.

We are also planning to add other assessment, adaptation and matchmaking algorithms to TwoA so that game developers can choose the best one that suits their needs. Finally, it is possible to use TwoA for matching two human players. It will be interesting to compare TwoA with other human-to-human matchmaking algorithms. In the original work [5], the CAP system was already favorably compared with the Elo system. However, comparison with more state-of-the-art matchmaking systems remains problematic due to its proprietary nature where details of the algorithms are not revealed.

Its current version is fully functional and available to the public. TwoA was implemented as a software component that can be easily integrated with popular game development platforms. This portability is enabled by the RAGE architecture [11], an open-source library that was specifically created to simplify development and use of pedagogical components. The RAGE architecture implements a set of well-established design patterns from component-based software development [15, 16].

For the game developers, the architecture offers simple and standardized interfaces for integrating TwoA into different game development platforms such as Xamarin or Unity3D game engines. Since the architecture imposes restrictions on having platform-specific code within the component, the game developers do not have to worry about potential conflicts between component code and game code. Moreover, the architecture provides pre-implemented interfaces for cases where access to platform-specific functionalities is required. For example, for loading and saving to local files the architecture provides interfaces that connect to a platform-specific input-output library.

Overall, compliance of TwoA with the RAGE architecture offers a highly portable pedagogical component that can be easily integrated with different game development platforms. The source code for simulations can be downloaded from `https://github.com/E-Nyamsuren/TwoA-TileZero-simulation`. The source code and binary for the TwoA component as a standalone library can be downloaded from `https://github.com/rageappliedgame/HatAsset`.

Acknowledgment This work has been partially funded by the EC H2020 project RAGE (Realising an Applied Gaming Eco-System); `http://www.rageproject.eu/`; Grant agreement No 644187.

References

1. Zemliansky, P., Wilcox, D.: Design and Implementation of Educational Games: Theoretical and Practical Perspectives. Information Science Reference. ISBN 978-1-61520-781-7 (2010)
2. Abt, C.: Serious games. New York: Viking Press (1970)
3. Connolly, T.M., Boyle, E.A, MacArthur, E., Hainey, T, Boyle, J.M.: A systematic literature review of empirical evidence on computer games and serious games. Computers & Education, vol. 59, no. (2), pp.661686, DOIdoi: 10.1016/j.compedu.2012.03.004 (2013)
4. RAGE: Realising an Applied Gaming Eco-system, Retrieved from `http://rageproject.eu` (2017, March 25)
5. Klinkenberg, S., Straatemeier, M., Van der Maas, H. L. J.: Computer adaptive practice of maths ability using a new item response model for on the fly ability and difficulty estimation. Computers & Education, 57 (2), 1813-1824 (2011)
6. Klinkenberg, S.: High Speed High Stakes Scoring Rule. In International Computer Assisted Assessment Conference (pp. 114-126). Springer International Publishing. (2014)

7. Jansen, B. R., Louwerse, J., Straatemeier, M., Van der Ven, S. H., Klinkenberg, S., Van der Maas, H. L.: The influence of experiencing success in math on math anxiety, perceived math competence, and math performance. Learning and Individual Differences, 24, 190-197. (2013)
8. Gierasimczuk, N., van der Maas, H. L., Raijmakers, M. E.: Logical and psychological analysis of deductive mastermind. In ESSLLI Logic & Cognition Workshop (pp. 1-13). (2012)
9. Herbrich, R., Minka, T., Graepel, T.: TrueSkill: a Bayesian skill rating system. In Proceedings of the 19th International Conference on Neural Information Processing Systems (pp. 569-576). MIT Press (2006)
10. Elo, A. E.: The rating of chess players, past and present. Arco Pub (1978)
11. Van der Vegt, W., Nyamsuren, E., Westera, W.: RAGE reusable game software components and their integration into serious game engines. In International Conference on Software Reuse (pp. 165-180). Springer International Publishing (2016)
12. Glickman, M. E.: A comprehensive guide to chess ratings. American Chess Journal, 3, 59-102 (1995)
13. Eggen, T. J., Verschoor, A.J.: Optimal testing with easy or difficult items in computerized adaptive testing. Applied Psychological Measurement, 30(5), 379-393 (2006)
14. Huhn, J. C., Hullermeier, E.: An analysis of the FURIA algorithm for fuzzy rule induction. In Advances in machine learning I (pp. 321-344). Springer Berlin Heidelberg. (2010)
15. Bachmann, F., Bass, L., Buhman, C., Comella-Dorda, S., Long, F., Robert, J., Sea-cord, R., Wallnau, K.: Technical concepts of component-based software engineering, Volume II. Carnegie Mellon University, Software Engineering Institute, Pittsburgh (2000)
16. Mahmood, S., Lai, R., Kim, Y.S.: Survey of component-based software development. IET software, 1 (2), 57-66.(2007)