Dynamic Difficulty Adjustment in a Whac-A-Mole like Game

Bruno E. R. Garcia and Marcio K. Crocomo
Department of Informatics
IFSP - PRC
Piracicaba, SP, Brazil
brunoely.gc@gmail.com, marciokc@ifsp.edu.br

Kleber O. Andrade
Department of Informatics
FATEC - AM
Americana, SP, Brazil
kleber.andrade@fatec.sp.gov.br

***Abstract*—Based on the Evolutionary Algorithm (EA) proposed by [1], this paper presents a new EA version with dynamic difficulty adjustment for a Whac-a-Mole like game used in motor rehabilitation of hand. This new version considers user performance as input and generates the position and time that the target will stay on screen. In order to simulate different users needs, a variety of player profiles were created with diverse horizontal movement speed (ulnar deviation/radial deviation of the hand) and vertical movement speed (supination/pronation of hand), as well as distinctive player attention deficits (response time to the apparition of the targets). The results show that the developed EA can adjust the difficulty factors of the game according to the skill set of each profile.**

***Keywords*-Human-Computer Interactive; Serious Games for Rehabilitation; Artificial Intelligence; Evolutionary Algorithm; Dynamic Difficulty Adjustment.**

I. Introduction

In the past years, the game industry has been profiting more than the movie industry [2], [3]. To keep this position, they are reinventing their ideas and techniques [4]. Nowadays, researchers are investing efforts in improving the performance, movements, and strategies of the games [5]. The game industry is using a lot of AI techniques like: Fuzzy [6], Decision Trees [7], Artificial Neural Networks [8], [9], Q-Learning [10] and others. All of these techniques have been used to improve the quality of the games and the results of these applications are positives in general [5].

One of the challenges of the game industry is to keep players interested while playing [1], [11]. For this purpose, the difficulty adaptation in a game is very important. In order to keep the player interested, we can't let him get neither frustrated or bored during gameplay. So, it is important to keep the player in balance between the difficulty of the game and his skill. One way of representing this balance is through the *Flow Channel* [12], [13]. Flow channel is defined as the state of the mind that keeps a person focused on an activity [12]. The adaptation of the game challenge based on the player skill level can keep the player in the flow channel.

The importance of the Dynamic Difficulty Adjustment (DDA) increases in rehabilitation games because we don't just have to adjust the difficulty to keep the player interested in the game while playing, we also have to adjust it to keep the player in his limits since exploring the limits of the player is the best way to his rehabilitation [14], [15]. So, if the game is attractive and explores the player's skill, the player rehabilitation tends to be very positive.

It is important to note that a number of factors may interfere with a game's difficulty, such as the game's response time, or the player's required motor skill. In games aimed at rehabilitation, it becomes important to consider these factors separately. As an example, a player who has a motor limitation may be frustrated if the game requires a high motor coordination capacity but allows a long response time. On the other hand, if the game requires a quick response time, but low motor coordination, the same player may become frustrated. In this paper, we represent these difficulty factors of a game as chromosomes parameters, to be dynamically adjusted by an Evolutionary Algorithm (EA). In doing so, we are able to adapt the game difficulty for different kind of simulated players. The objective of this research is not only to verify if our EA can adapt the difficulty level of the game according to simulated players skill level, but, more specifically, to verify if the difficulty factors of the adapted game correctly corresponds to the set of skills of the simulated player profiles used in our experiments.

The rest of the paper is structured as follows. In Section II we discuss related work in games with dynamic difficulty adjustment. In section III we present the selected game and the created virtual players used in the experiments. In Section IV we explain the EA approach used to dynamically adjust the game difficulty. Section V presents the results. Finally, Section VI presents the conclusion and ideas for future work.

II. Related Works

Although players might enjoy unpredictability or novelty during gameplay experiences, the DDA will only be effective if it does not disrupt or degrade the player experience [16]. For this purpose, the algorithm responsible for the DDA should be able to adapt the game difficulty based on the player experience. In [17], the researchers developed Evolutionary Fuzzy Cognitive Maps to adjust gameplay parameters in real-time according to the current player skill level. In a different research [18], the game difficulty was dynamically adjusted considering the emotional state of the player, using as input for the game balance the overt behavior and physiological responses of the player.

Games with Evolutionary Algorithm (EA) are the subject of many studies [1], [11], [19]. It is common using EA to Procedural Generation of Content (PGC) [20], Dynamic Difficulty Adjustment (DDA) [1] and others. In the case of PGC, the content generation tends to be complex and costly, however, EA has been used to dynamically generate contents like maps, visual arts and narratives based on player experience [20]. In other study, EA was used to game strategy learning. The author created a theory-inspired game and implemented an EA to generate strategies based on play-styles [19].

In [1] the authors propose the DDA based on an Evolutionary Algorithm (EA). The game proposed by them is called "The Catcher" (Figure 1), and has been used for rehabilitation of patients with motor deficits. In this game, the player must control the horizontal position of a squirrel (the main character of the game) and, in doing so, reach the targets (nuts) that fall vertically on the screen. So, the player has to worry about moving the controller in only one dimension.

Figure 1. Screenshot of the game "The Catcher" [1]

The game difficulty is based on the initial distance to the target and the reaction speed, so, the chromosome defined by them has 2 genes (distance to the target d and speed of the target v). The fitness of a chromosome is calculated by the following equation:

$$F = K_d \cdot d + K_v \cdot v - K_\epsilon \cdot \epsilon. \tag{1}$$

where, K_d, K_v and K_ϵ are coefficients that set the impact of all the terms of the equation. d and v reflect the game distance and velocity, ϵ represents how far from the target horizontal position the player is when the round ends.

However, in this paper we adapt the proposed EA in [1] for a Whac-A-Mole type game, which is also a type of game that has been used in rehabilitation [21]. This game is similar to the game proposed in [1] but the difference is that Whac-A-Mole has a two-dimensional gameplay, since the player can use the controller to move horizontally and vertically. So, adaptations are required in EA proposed in that work. Besides that, observing the results we aim to verify the game behavior for each different profile (Section V-C).

III. Methods and Materials

This section presents the methods and materials used to perform the game simulator (Section III-A) and the player simulator (Section III-B).

A. The Game Simulator

This section presents the proposed "Wack-a-Mole" simulator, which has been developed in C language. In this type of game, the player's goal is to control the image of a hammer on the screen so it can reach the targets that appears in random positions before they disappear, as illustrated by Figure 2.

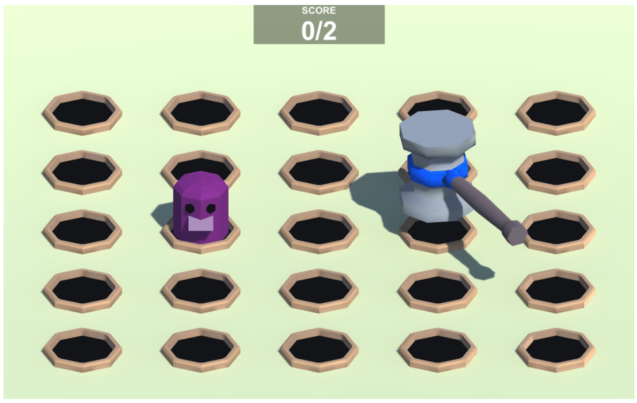

Figure 2. Screenshot of a Whac-A-Mole (5 × 5) type game created in Unity 3D, but simulated in C language.

Four constants are important to explain the game: W,H,C and R. W and H represents, respectively, the width and height of the screen in pixels. C represents the number of columns and R represents the number of rows on the screen, relative to the screen division that delimits the regions where the targets can appear. To better understand it, considering the example in Figure 2, the respective game would have 5 columns and 5 rows ($C = 5$ and $R = 5$).

The simulator also uses 2 variables: p_x and p_y, that represents the player (hammer) position on the screen (respectively the coordinates , in pixels, being p_x the hammer position horizontally and p_y the hammer position vertically. At the beginning of the game, the hammer is placed at the center of the screen $\left[\frac{W}{2}, \frac{H}{2}\right]$. Then, a target appears at a distance from the hammer given by d_x and d_y (respectively, the distance on the x-axis and on the y-axis) and stays on the screen for t seconds before it disappears. Note that d_x,

d_y and t are values that define how difficult it is for the player to reach the target and, for this reason, they are the ones that must be adapted according to the player skill level, as explained in Section IV.

We limit the maximum distance from the player that the target can appear on the screen by the half of screen width and height, because, supposing the player is at the center of the screen, the maximum distance that the target may appear from the player is half the screen. Any distance greater than that would exceed the screen limit and the value of the chromosome would not represent the real difficulty of the match. Because of that, to define the target position we created two variables g_x and g_y. First of all, to calculate the value of them we defined 2 variables to indicate the target direction (λ_x and λ_y) will appear from the player, both of them have only two values: 1 or -1. In the case of λ_x, -1 indicates that the target is on the left the player and 1 indicates that the target is on the right the player. In the case of λ_y, -1 indicates that the target is under the player and 1 indicates that the target is above the player, then, to define the target position we get the player position and depending on the values of λ_x and λ_y, we subtract or add the value of the respective distances that the target should appear from the player (d_x , in the case of g_x and d_y, in the case of g_y). Since the values of d_x and d_y are normalized in the chromosome, we multiply them by the half of screen size, as its demonstrated on the equations 3 and 4.

To get the real time in seconds that the target will stay on the screen, we defined two constants t_{MIN} and t_{MAX} that represent the minimum and maximum time, respectively, that the target can stay on the screen. After that, we created a variable g_t to represent the time unnormalized, this variable gets the time from the chromosome and transforms it to seconds based on t_{MIN} and t_{MAX}, given by the equation 2.

$$g_t = t_{MIN} + t \cdot (t_{MAX} - t_{MIN}) \tag{2}$$

$$g_x = \begin{cases} p_x - d_x \frac{W}{2} & \text{if } \lambda_x = -1 \\ p_x + d_x \frac{W}{2} & \text{otherwise} \end{cases} \tag{3}$$

$$g_y = \begin{cases} p_y - d_y \frac{H}{2} & \text{if } \lambda_y = -1 \\ p_y + d_y \frac{H}{2} & \text{otherwise} \end{cases} \tag{4}$$

Valid target positions are restricted by a screen division given by the rectangular areas shown in Figure 2. Having the hammer position and its distance to the target, we determine the target position by adding the distance values (d_x and d_y to the hammer coordinates (p_x and p_y) and verifying in which area of the screen the target is. Then, we adjust the target position, given by t_x and t_y, so the target is located at the center of its current area.

B. Player Simulators

The constants and variables defined in this section represent the necessary parameters to start one game match. When a target appears on the screen, the player must then use the game controller to move the hammer and try to reach the target. In this paper, we use a method to dynamically adjust the difficulty of the matches (Section IV) adapted from the one proposed in [1] and, for testing it, we use simulation of players.

In Section V-A, we describe 5 different player profiles that we create to simulate different kinds of skills. Each profile has 3 attributes (v_x, v_y and t_r), v_x defines the maximum speed the player can move horizontally, v_y defines the maximum speed the player can move vertically and t_r defines the value that will be subtracted from the time that the target will be visible on the screen, allowing the simulation of players with attention deficit that use some of the time searching for the target instead of moving the controller. Equation 5 shows how the final time (t_f) is obtained, which is used to calculate the player movement.

$$t_f = \begin{cases} g_t - t_r & \text{if } g_t > t_r \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

After that, we calculate the maximum distance that player can move the hammer based on its initial velocity capacity and the size of the screen, like the following equation.

Then, we use 2 variables Δ_x and Δ_y to represent the player movement capacity. To calculate it, we multiply v_x and v_y with the final time (Equation 5), as shown by Equations 6 and 7.

$$\Delta_x = (v_x \cdot W)t_f \tag{6}$$

$$\Delta_y = (v_y \cdot H)t_f \tag{7}$$

So, Δ_X is the maximum distance that the player can move the hammer horizontally and Δ_y is the maximum distance that the player can move the hammer vertically.

After we have the value of Δ_x and Δ_y, we are able to know if the player can reach the target, because, we get the target position and calculate in which rectangle on the screen the target is positioned. Then, we get the values of Δ_x and Δ_y and calculate if the player is able to hit that square, if yes, the algorithm positions the player at the center of the square.

If the player is not able to hit the target, we get the values of λ_x and λ_y, and based on his previous position ($p_{x_{n-1}}$ and $p_{x_{n-1}}$), we position the player as close to the target as he can, as shown in equations 8 and 9:

$$p_{x_n} = p_{x_{n-1}} + \Delta_x \lambda_x \tag{8}$$

$$p_{y_n} = p_{y_{n-1}} + \Delta_y \lambda_y \tag{9}$$

IV. The proposed Evolutionary Algorithm

We propose a new version of the EA used in [1] for a different type of game. In order to do so, we had to make some adaptations to the EA, based on a Whac-a-Mole like game.

Since the purpose of our EA is to adjust the difficulty level of the game, the chromosome structure we use is composed of 3 values, t, d_x and d_y, which are the parameters to be used in one match, directly defining its difficulty, as explained in Section III-A. If the EA can adjust these values accordingly to a specific player profile, it indicates that it can adjust the difficulty level for a specific type of player. The used population has a total of five chromosomes.

The chromosomes in the first population are initialized randomly with values varying from 0 to 1 for t, and one of the following values: 0, 0.4 or 0.8 for d_x and d_y. The 0.4 gap was chosen so these values represent, respectively, a distance of zero, one or two rectangles from the hammer position. As the target can appear at a maximum distance from the hammer of half the screen, the 0.4 gap was calculated as $\frac{2}{C}$ for d_x, and $\frac{2}{R}$ for d_y, which results in 0.4 for both, since our game presents $R = 5$ and $C = 5$. For variations of the proposed game, where the target can appear in different positions (different than the 5×5 scene shown in Figure 2), these values can be easily recalculated.

The fitness function we use is similar to Equation 1. It considers three different factors that reflects: *i)* the time that the target stays on the screen (T, Equation 10), *ii)* the distance that the target is from the player at the start of the match (D, Equation 11) and *iii)* the distance that the player is from the target at the end of the match (E, Equation 12). While T and I are values that the reflects the match difficulty, E measures the player success or failure in the match. When $E = 0$, the target was reached, otherwise, E represents how close to the target the player could get. Equation 13 shows the final fitness function used in our EA.

$$T = 1 - t \tag{10}$$

$$D = \left(\frac{\sqrt[2]{(g_x - p_{x_0})^2 + (g_y - p_{y_0})^2}}{\sqrt[2]{W^2 + H^2}} \right) \tag{11}$$

where p_{x_0} and p_{y_0} are the coordinates of the hammer at the start of the match.

$$E = \left(\frac{\sqrt[2]{(g_x - p_{x_1})^2 + (g_y - p_{y_1})^2}}{\sqrt[2]{W^2 + H^2}} \right) \tag{12}$$

where p_{x_1} and p_{y_1} are the coordinates of the hammer at the end of the match.

$$F = K_t \cdot T + K_d \cdot D - K_e \cdot E \tag{13}$$

.

K_t, K_d and K_e are coefficients that set the weight of the fitness elements, higher values for K_t and K_d set higher contribution to the fitness score of solutions that represent matches with higher difficulty. On the other hand, higher values for K_e results in higher values for solutions where the player presents a good performance. To find good values for these coefficients, we reproduced the ranking system used in [11], as reported in the experiments from Section V-B.

To evaluate all the individuals in our population, a match is executed for each chromosome (total of five matches), using its values as the match parameters (line 6 from Algorithm 1. After that, each chromosome is associated with a fitness value. The chromosome with a higher fitness score is selected, and a new population is formed maintaining the selected chromosome (elitism) and replacing the other five with mutated copies of the selected individual (asexual reproduction) as done in [11] and [1].

The Mutation operator used works as follows: For the d_x and d_y parameters, a random value between -0.4, 0 and 0.4 is added. Once again, the 0.4 gap was calculated based on our current 5x5 game scene, and should be recalculated for variations of the game as previously explained. After the randomly generated value is added, the parameter is adjusted to 0, if it is less than 0, or to 0.8, if it is higher than 0.8 (since 0.8 represents the maximum distance of two regions where the target can appear). For the time gene (t) we generate a random value between 0 and 1, after that, we define randomly if this value will be positive or negative, and add it to the original t value. If the resulting time value is bigger than 1, the algorithm sets the value to 1, and if the resulting value is less than 0, the algorithm sets it to 0.

The Algorithm 1 presents the execution of the game using the virtual players and the proposed EA.

V. Experiments and Results

This section presents the created experiments and results from testing the proposed EA difficulty adjustment in our proposed game simulator. Section V-A shows the profile settings used in the experiments. Section V-B presents experiments conducted for testing the proper implementation of the EA and adjusting the coefficients used in its fitness function. Section V-C shows experiments that verifies if the difficulty presented in our proposed game is consistent with the ability presented by each tested player profile, when using the EA with the proper adjusted coefficients. Section V-D presents experiments with dynamic player profiles, to see if the EA can properly adjust the game difficulty when the player skills are changed during playtime.

A. Player Profiles

Simulating different kinds of players is important to observe how the EA can adjust the difficulty of our game for players with different set of skills. To simulate different kinds of players we created 5 different players profiles to

Algorithm 1 "Whac-A-mole" simulation with EA

```
Input: $N_G \geq 1, N \geq 1, W > 0, H > 0, C \leq W, R \leq H$
Output: $h$
1: Load current player profile values: $v_x$, $v_y$ and $t_r$
2: Define initial hammer position as $p_x \leftarrow \frac{w}{2}$ and $p_y \leftarrow \frac{h}{2}$
3: Randomly initializes the population of the EA
4: Initialize hit counter $h \leftarrow 0$
5: for $i \leftarrow 0$ to $i < N_G$ step 1 do
6:   for $j \leftarrow 0$ to $j < N$ step 1 do
7:     Get the values $d_x$, $d_y$ and $t$ to be used in this match
8:     Calculate $g_t$, $g_x$ and $g_y$ (Equations 2, 3 and 4)
9:     Calculate $t_f$, $p_{x_n}$ and $p_{y_n}$ (Equations 5, 8 and 9)
10:    if player is able to reach the target then
11:      Increment hit counter $h \leftarrow h + 1$
12:    end if
13:    Use Equation 13 to evaluate chromosome
14:  end for
15:  Select chromosome with higher fitness
16:  Make N copies of the selected chromosome
17:  Apply mutation operator in $N - 1$ chromosomes
18:  Define as the new population the set of N chromosomes obtained from the two previous steps
19: end for
```

use with our player simulator, explained in Section III-B. Profile 1 simulates a player with good performance to move the controller horizontally and bad performance to move the mouse vertically, profile 2 simulates a player with good performance in moving the controller vertically and bad performance in moving the controller horizontally, profile 3 simulates a player which really does not have a good performance in moving the controller in any axis, profile 4 simulates a player which has a better performance than profile 3, however, this profile also does not have a good performance in moving the controller and he has attention deficit and profile 5 simulates a player with good performance in moving the controller in both ways, but has attention deficit. All of these profiles are represented in Table I.

TABLE I
PLAYER PROFILES USED IN THE EXPERIMENTS

Profile	v_x	v_y	t_r
1	0.60	0.01	0.00
2	0.01	0.60	0.00
3	0.03	0.03	0.00
4	0.25	0.25	3.00
5	0.60	0.60	3.00

B. EA adjustments

Our first experiment aimed to adjust the K_t, K_d and K_e coefficients of our fitness function so the EA can work properly. For this purpose, the experiments in [11] were repeated. First, we define that each coefficient may one of the 5 different values (1,2,4,8,16). Since we have 3 coefficients to adjust, we have a total of 125 possible combinations (5^3). For each combination, we run a sequence of 30 tests for each player profile. Each test consists in running 30 generations of the EA using the given coefficient combination. After that, we calculate the player profile skill score (Θ), which is the rate of matches in which the player successfully hits the target , calculated by Equation 14, and also the game overall difficulty (Ψ), calculated by Equation 15.

$$\Theta = \frac{h}{N_G N} \tag{14}$$

where h is the hit counter, calculated in line 11 of Algorithm 1.

$$\Psi = \frac{1}{N_G}\sum_{i=0}^{N_G}\left[\frac{1}{3}\left(1 - \frac{\sum_{j=0}^{N} t}{N} + \frac{\sum_{j=0}^{N} d_x}{XN} + \frac{\sum_{j=0}^{N} d_y}{YN}\right)\right] \tag{15}$$

$$X = \left\lfloor \frac{C}{2} \right\rfloor \frac{2}{C} \tag{16}$$

$$Y = \left\lfloor \frac{R}{2} \right\rfloor \frac{2}{R} \tag{17}$$

where, $\lfloor z \rfloor$ is the floor function, gives the largest integer less than or equal to z.

Using the obtained values, we can plot a dot like the ones shown in Figure 3. The closest a dot is to the diagonal line in the picture the better, because the most likely it is that the game kept the player in the flow channel area, explained in Section I) (greater game difficulty for players with good skill level, and lower game difficulty for players with lower skill level). However, a good set of coefficients should be able to keep all player profiles near the flow channel. So, to obtain a good combination of values for our coefficients, the same ranking system presented in [11] was used and, from this method, we detected that the set of coefficients $K_t = 2, K_d = 4$ and $K_e = 8$ was presented in each top 38, 46, 37, 30, 4, coefficients for every player profile, respectively. Then, we selected this set of coefficients to be used in our EA. Figure 3 shows the plot of a dot for each player profile obtained as previously explained using the given coefficients set.

Figure 4 represents a simple test using profile 5 and the EA with the adjusted coefficients, in which we can see that the EA is working as expected, since the tendency line indicates an improvement of the fitness values as the generation number increases.

The used heuristic aims to find a set of coefficients that results in dots that are close to the diagonal line for all the

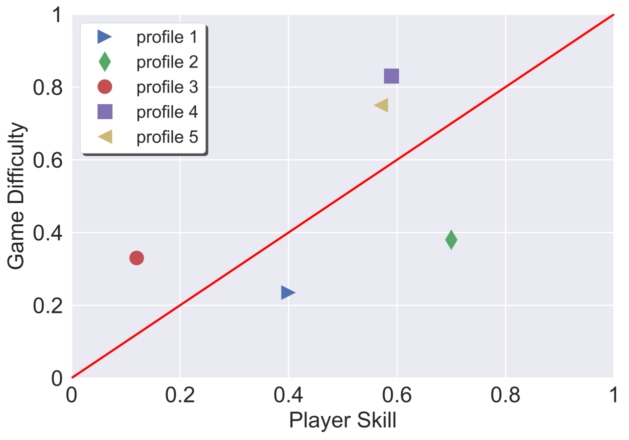

Figure 3. Dispersion diagram obtained using $K_t = 2, K_d = 4$ and $K_e = 8$ for each player profile.

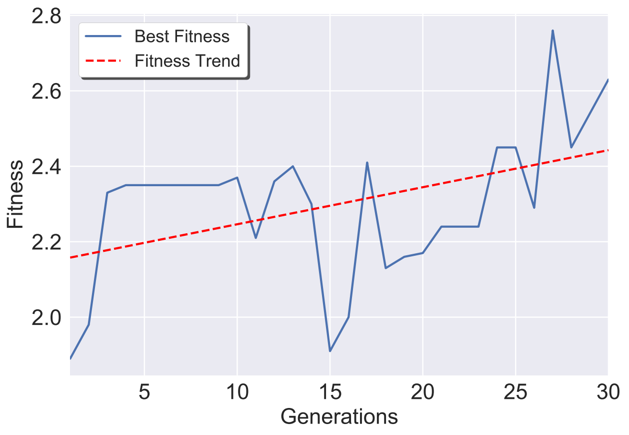

Figure 4. Fitness diagram with trend line from profile 5.

different player profiles (Figure 3). By doing that, we expect that players represented by these profiles will have a lower chance of feeling frustrated or bored when playing. We can't, however, be certain of that, since we are not working with real players, but simulated player profiles. Furthermore, we still can't determine if the type of difficulty presented by the game is consistent with the set of abilities presented by each tested player profile, which is what we hope to validate from the solutions found by our EA.

C. Comparing types of challenges found for each player profile

With the adjusted fitness function for our EA, we got the average values for each chromosome parameter, considering all generations and all 30 tests run in the previous experiment for each given profile. This reflect the average settings used by the game considering all matches with the simulated player. The results are shown in Figures 5, 6, 7, 8 and 9.

As described on section V-A, the first profile has good skills in moving the control horizontally and poor ability to move it vertically. As we can see in Figure 5, the average settings for the game matches make the target appears in a distant position in the horizontal axis (high d_x value), close position in the vertical axis (low d_y value). These settings for the game matches this type of player skills. The time (t) that the target stays visible is a low value, which is also coherent with the player skills, since profile 1 does not have attention deficit ($t_r = 0$).

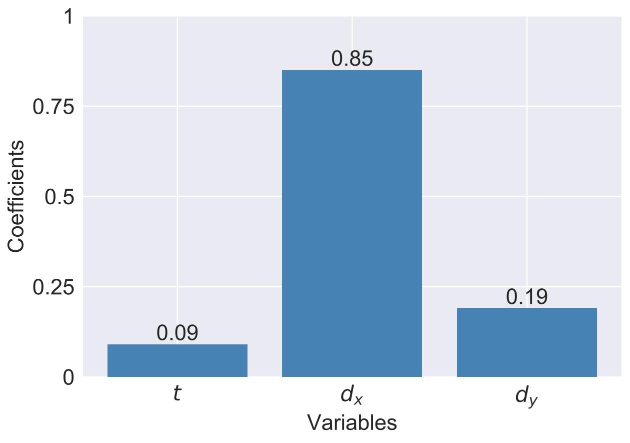

Figure 5. Average values for parameters t, d_x and d_y used in the matches for player profile 1.

Profile 2 is very similar to the first one, but with inverted skills: good skills in moving control vertically and poor ability to move horizontally. Figure 6 shows the results that, once again, match the player profile skills.

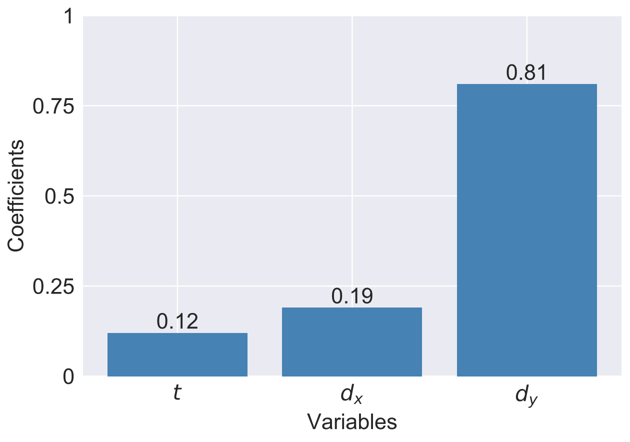

Figure 6. Average values for parameters t, d_x and d_y used in the matches for player profile 2.

The third player profile does not have good movement skills. Observing the Figure 7, we can see that, once again, the algorithm matches the player skill by using low values for the distances in both axis.

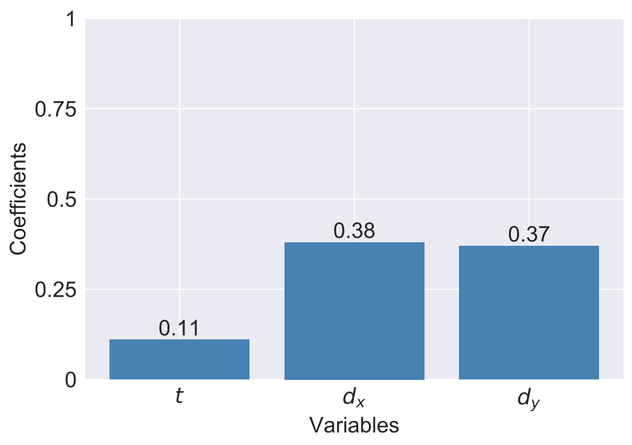

Figure 7. Average values for parameters t, d_x and d_y used in the matches for player profile 3.

The last two profiles, 4 and 5, represent cases of attention deficit. Figures 8 and 9 present the results of our tests, that shows that the EA was able to adapt the matches so that the average time that the target is visible is now, in both scenarios, higher than in the time found for the previous profiles (1,2 and 3). Also, the distance the target appears is, in average, bigger for profile 5 than for profile 4, which is expected since profile 5 has better movement skills than profile 4.

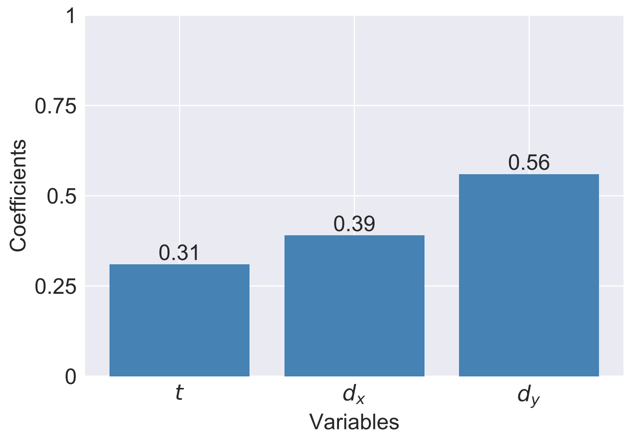

Figure 8. Average values for parameters t, d_x and d_y used in the matches for player profile 4.

The results presented in this section let us verify that our current EA is capable of adjusting the existing challenges in the game to properly adjust to the skill level of a static player profile (a player whose skill level remains the same during gameplay).

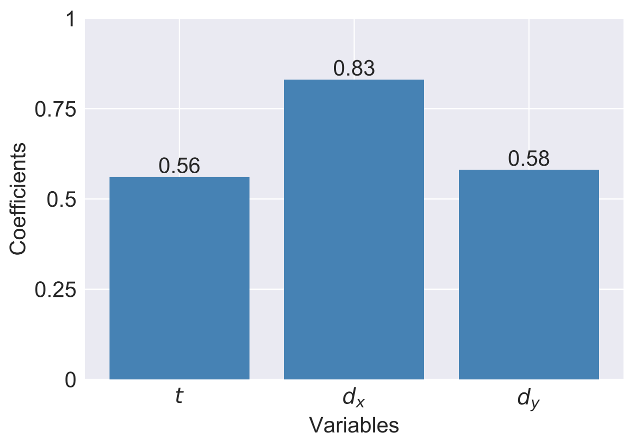

Figure 9. Average values for parameters t, d_x and d_y used in the matches for player profile 5.

D. Change in players skill level during gameplay

Players often change their skill level during gameplay, usually improving their skills as they learn from the game. To simulate this scenario, an experiment was conducted simulating a game with 750 matches, what corresponds to 150 EA generations. In order to simulate that the player improved his abilities, every 50 generations we changed the value of v_x and v_y as follows: First 50 generations: 0.03, generation 51 to 100: 0.12 and generation 101 to 150: 0.36. We kept the value of t_r as 0 through all generations. The results from this experiment are presented in Figures 10 and 11.

Figure 10 shows a moving average graph using a subset of size 10 for the average parameters (t, d_x and d_y) in each generation. We can see that there is a tendency to improve the target distance as the matches advance, as expected, since the player skill level is also improving. We can also notice that there is an increase in t on the second set of 50 generations (profile with v_x and v_y with value 0.12). To better visualize what happened, we created a second graph (Figure 11), showing the average of the used parameters in each set of 50 generations, each one corresponding to a different profile configuration. It is possible to visualize that each set of parameters corresponds to the player skill level. Although the values of d_x and d_y are slightly higher for the last 50 generations (101 to 150), we point out that the skill level of the player in these last 50 generations are significantly higher than the values used for the profile in generation 51 to 100, since value 0.36 to v_x and v_y represents a player who can move the controller 3 times faster than a player with value 0.12 of v_x and v_y. However, the results show that the difficulty level was improved in the last 50 generations by decreasing the time available to the player to reach the target, as he got faster.

To simulate a scenario where the EA should adapt the

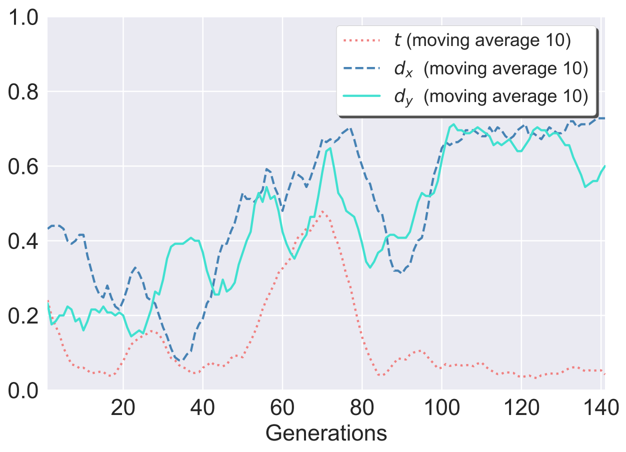

Figure 10. Genes evolution by moving average 10.

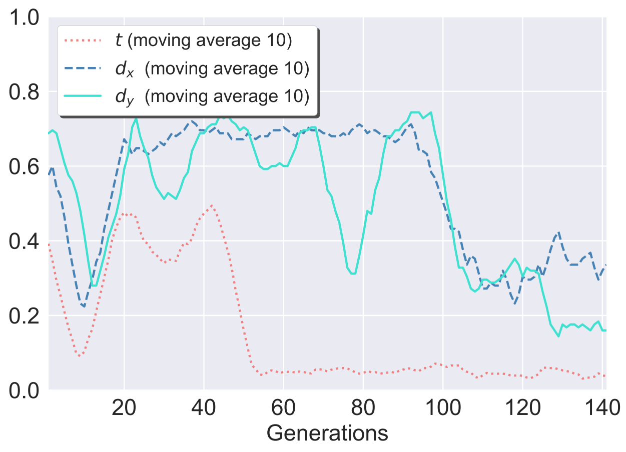

Figure 12. Genes evolution by moving average 10.

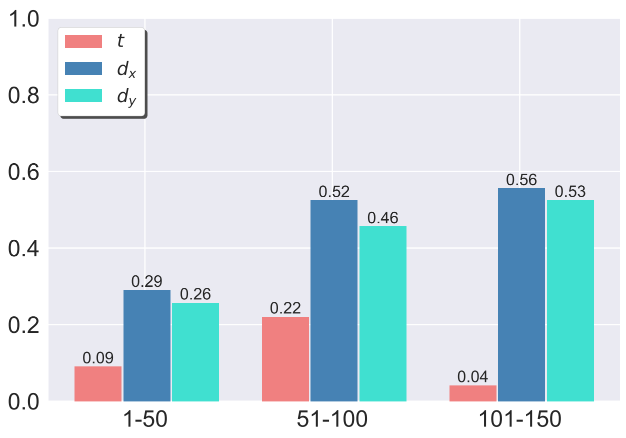

Figure 11. Average of the 50 generations used by each profile configuration.

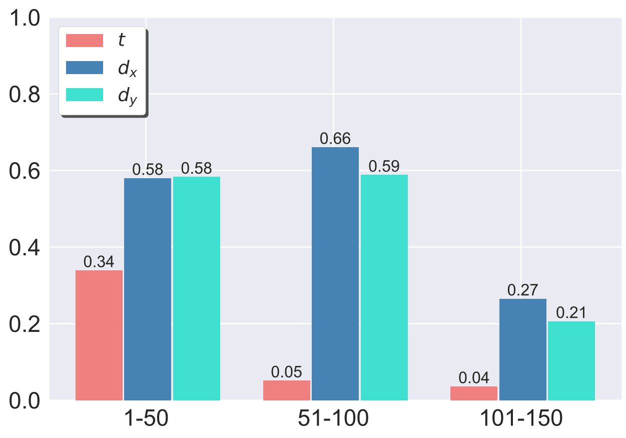

Figure 13. Average of the 50 generations used by each profile configuration.

difficulty level of the game when a player, for some reason, has a severe decrease in his motor abilities, we repeated the previous experiment using a different order in the changes made to the values of the profile every 50 generation, changing the values of v_x and v_y as follows: First 50 generations: 0.12, generation 51 to 100: 0.36 and generation 101 to 150: 0.03. Once again, we kept the value of t_r as 0 through all the experiment. The results are shown in Figures 12 and 13, and allow us to get to the same conclusions as before: that the EA can dynamically adjust the difficulty level based on the player abilities when they are changed during gameplay.

VI. Conclusion

The results from the performed experiments show that our initial proposal is valid. The developed EA can perform the DDA successfully for the simulated players profiles, adjusting different elements which influence the game difficulty according to the skill set of each player profile. In summary, the conclusions we draw from the results of this research are:

1) The technique used in [1] for obtaining a DDA can be adapted for different games;
2) The different game difficulty elements can be properly adjusted according to changes of the player profile skill set during gameplay;

Demonstrating that the research conducted in [11] and [1] can be adapted for different types of games is important to the research field of rehabilitation games. The results of the present research adds to [11] and [1], by showing that the DDA technique can be used in the creation of games that could help in the player rehabilitation process by adapting the game parameters to player specific difficulties during playtime.

We believe that the results are also promising for other applications of serious games, since we can argue that in educational games it is also important to adjust the game difficulty level according to the player knowledge, and furthermore, adjust different difficulty levels for different topics, according to the player knowledge level in each of these topics.

Continuations of this research can include: *i*) performing tests with real players, evaluating the impact of our proposal on clinical rehabilitation sessions; *ii*) testing the used technique in an educational game; *iii*) the creation of a larger set of profiles in order to simulate more types of players with more realistic behaviors; and *iv*) performing tests with different sizes of game scenarios.

Acknowledgment

The authors would like to thank the scholarship grant awarded to Bruno Ely Reis Garcia by the Institutional Scholarship Program of Scientific and Technological Initiation of the Federal Institute of Education Science and Technology of São Paulo (PIBIFSP) and Fernanda Goulart for the English review.

References

[1] K. O. Andrade, R. C. Joaquim, G. A. P. Caurin, and M. K. Crocomo, "Evolutionary algorithms for a better gaming experience in rehabilitation robotics," *Comput. Entertain.*, vol. 16, no. 2, pp. 4:1–4:15, Apr. 2018. [Online]. Available: http://doi.acm.org/10.1145/3180657

[2] MPAA, "2016 Theatrical Market Statistics Report," *Mpaa.Org*, 2017. [Online]. Available: http://www.mpaa.org/wp-content/uploads/2017/03/MPAA-Theatrical-Market-Statistics-2016_Final-1.pdf

[3] Newzoo, "2018 Global game Market Report: Trends, Insights, and Projections Towards 2021," 2018, pp. 1–25. [Online]. Available: https://resources.newzoo.com/hubfs/Reports/Newzoo_2018_Global_Games_Market_Report_Light.pdf

[4] G. B. David Wesley, *Innovation and Marketing in the Video Game Industry avoiding the performance trap*, 2016.

[5] M. Carrozo, "How Artificial Intelligence is changing the gaming industry." [Online]. Available: https://unbabel.com/blog/ai-changing-gaming-industry/

[6] T. Khalil, Y. S. Raghav, and N. Badra, "Optimal Solution of Multi-Choice Mathematical Programming Problem Using a New Technique," *American Journal of Operations Research*, vol. 06, pp. 167–172, 2016.

[7] M. W. Masato Konishi, Seiya Okubo, Tetsuro Nishino, "Decision Tree Analysis in Game Informatics," *Springer International Publishing*, 2018.

[8] S. Y. Chong, M. K. Tan, and J. D. White, "Observing the Evolution of Neural Networks Learning to Play the Game of Othello," *Trans. Evol. Comp*, vol. 9, no. 3, pp. 240–251, 2005. [Online]. Available: http://dx.doi.org/10.1109/TEVC.2005.843750

[9] G. A. P. Caurin, A. A. G. Siqueira, K. O. Andrade, R. C. Joaquim, and H. I. Krebs, "Adaptive strategy for multi-user robotic rehabilitation games," *Proceedings of the Annual International Conference of the IEEE Engineering in Medicine and Biology Society, EMBS*, no. 1, pp. 1395–1398, 2011.

[10] K. D. O. Andrade, G. Fernandes, G. A. Caurin, A. A. Siqueira, R. A. Romero, and R. D. L. Pereira, "Dynamic player modelling in serious games applied to rehabilitation robotics," in *Proceedings - 2nd SBR Brazilian Robotics Symposium, 11th LARS Latin American Robotics Symposium and 6th Robocontrol Workshop on Applied Robotics and Automation, SBR LARS Robocontrol 2014 - Part of the Joint Conference on Robotics and Intelligent Systems, JCRIS 2014*, oct 2015, pp. 211–216.

[11] K. O. Andrade, T. B. Pasqual, G. A. P. Caurin, and M. K. Crocomo, "Dynamic difficulty adjustment with Evolutionary Algorithm in games for rehabilitation robotics," in *2016 IEEE International Conference on Serious Games and Applications for Health, SeGAH 2016*, Orlando, FL USA, 2016, pp. 1–8.

[12] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. New York, NY: Harper Perennial, March 1991. [Online]. Available: https://www.researchgate.net/publication/224927532_Flow_The_Psychology_of_Optimal_Experience

[13] J. M. Thomas and R. M. Young, "Annie: Automated generation of adaptive learner guidance for fun serious games," *IEEE Transactions on Learning Technologies*, vol. 3, no. 4, pp. 329–343, 2010.

[14] P. Langhorne, F. Coupar, and A. Pollock, "Motor recovery after stroke: a systematic review," pp. 741–754, 2009.

[15] N. A. Borghese, M. Pirovano, R. Mainetti, and P. L. Lanzi, "An integrated low-cost system for at-home rehabilitation," in *Proceedings of the 2012 18th International Conference on Virtual Systems and Multimedia, VSMM 2012: Virtual Systems in the Information Society*, 2012, pp. 553–556.

[16] R. Hunicke, "The case for dynamic difficulty adjustment in games," in *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in Computer Entertainment Technology*, ser. ACE '05. New York, NY, USA: ACM, 2005, pp. 429–433. [Online]. Available: http://doi.acm.org/10.1145/1178477.1178573

[17] L. J. F. Prez, L. A. R. Calla, L. Valente, A. A. Montenegro, and E. W. G. Clua, "Dynamic game difficulty balancing in real time using evolutionary fuzzy cognitive maps," in *2015 14th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, Nov 2015, pp. 24–32.

[18] T. J. W. Tijs, D. Brokken, and W. A. IJsselsteijn, "Dynamic game balancing by recognizing affect," in *Fun and Games*, P. Markopoulos, B. de Ruyter, W. IJsselsteijn, and D. Rowland, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 88–93.

[19] Guanci Yang, "Game Theory-Inspired Evolutionary Algorithm for Global Optimization," *MDPI*, 2017.

[20] J. Forsblom and J. Johansson, "Genetic Improvements to procedural generation in games," *University of Boras*, 2017.

[21] Y. Tokuyama, R. P. C. J. Rajapakse, S. Miya, and K. Konno, "Development of a whack-a-mole game with haptic feedback for rehabilitation," in *2016 Nicograph International (NicoInt)*, July 2016, pp. 29–35.