Summary. In computer games, dynamic difficulty adjustment (DDA) tries to ensure that the challenge level offered by the game matches the skill of the human player. In this paper simple and fast methods for adjusting a difficulty level of a computer opponent are presented. An empirical investigation of the methods when playing FPS (First Person Shooter) game is conducted. Performance of the methods is analyzed for different values of method's parameters, different game's scenes and players with various difficulty levels.

Keywords: dynamic difficulty adjustment, computer games, game AI

1. Introduction

The aim of the game DDA is to provide challenges of the "right" difficulty i.e. such that players are simulated but not overloaded. When the game is too easy the player can become bored, when it is too hard - frustrated. A game in which the challenge level matches the skill
of the human player has the greatest entertainment value. Traditionally, the singleplayer games allow adjusting a difficulty level only at the beginning of the game by the player's choice of a game mode (e.g. easy, medium, hard). Such a strategy has many problems. When the number of levels is small, it is rather easy to choose the right one, but it is difficult to set it in a very satisfying way. With more levels the chance of satisfying settings is more possible, but finding it becomes more difficult. Additionally, an overall level of challenge cannot be adjusted according to the player's actual input during the game. In this situation, it is desirable to automatically adjust the difficulty level as the game proceeds.

To date, several methods have been proposed to dynamically adjust difficulty level in games: multi-layered perceptrons (MLPs) [1], dynamic scripting [2], Hamlet System [3], reinforcement learning [4], Upper Confidence bound for Trees (UCT) and neural networks [5], exponential update algorithm POSM predicting "just right" difficulty settings [6], self-organizing system [7], etc. The game industry, however, is suspicious of the new ideas proposed by academic researchers in a game AI. Many of AI techniques (e.g. neural networks) can't guarantee a failure-free performance for every experiment, which is usually essential in commercial games. Another problem is that most of the AI techniques are complex and time consuming, so their application slows down games, sometimes even to the extent that they can't perform in real time. Usually players prefer less "intelligent" opponents to ones who think too long.

In this paper the authors evaluate two simple and fast methods, proposed by authors, for adjusting a computer player's behaviour so its difficulty level is similar to a human player's level. In theirs general forms the methods are very adaptable. They do not dictate how the player should be defined, beyond the need for providing the measure of the player's skills.

The outline of the remainder of the paper is as follows. First, section 2 describes DDA methods. Then, in section 3 experiments and simulation results are presented and discussed. Finally, section 4 concludes the paper.

2. Adjusting a computer opponent

In the paper two methods adjusting a computer opponent, so its challenge level is compatible with the game player's skills, are studied: the Full Dynamic Difficulty Adjustment for a Computer Player (FDDACP) and the Single Feature Dynamic Difficulty Adjustment for a Computer Player (SFDDACP). In the SFDDACP method a computer opponent changes its behaviour, its characteristics so they are comparable with the player's. In the FDDACP a computer opponent changes its characteristics so its overall results match that of the human player.
The FDDACP method uses a quantitative evaluation function $ef$ to calculate a human player's and a computer opponent's level of skills. The evaluation function is game specific. If the absolute difference between player's ($ef_P$) and computer's ($ef_O$) level of skills $\text{diff}ef = |ef_P - ef_O|$ is less or equal to $p_{lim}*ef_P$, then it is believed that opponent's and player's skills are similar. A percentage value $p_{lim}$ is a method's parameter and defines a region without adjustment. An opponent is changed more often for smaller $p_{lim}$ values. If $\text{diff}ef$ is greater than $p_{lim}*ef_P$, then $\text{diff}ef$ value is used to adjust an opponent behaviour. The greater the $\text{diff}ef$ the more significant the change in an opponent's behaviour.

In the SFDDACP method each aspect of an opponent's behaviour is evaluated and adjusted separately. A quantitative value $efF_{O,i}$ for every evaluated enemy feature $i$ is obtained and compared with its player counterpart $efF_{P,i}$. If significantly different the opponent feature is changed to the player's value. Afterwards, to create diverse behaviours of computer players minor random changes are added to the analyzed feature.

In many games players are characterized with the use of various attributes (e.g. health, ammunition, lives, strength) that describe player's features. Players with different level of skills have different values of attributes. Therefore, if we change a value of an attribute we can change player's behaviour. For such games calculating $ef$ and $efF_i$ can be extremely fast and straightforward. To test the concept, in this work, the evaluation functions measuring, with the use of attributes values, separate player skills (SFDDACP) and total player level (FDDACP) were analyzed. In the tested SFDDACP method an attribute is equated with an individual skill. Two types of attributes were identified: BVBP attributes for which the bigger the attribute value the better the player's behaviour (e.g. strength) and SVBP attributes for which the smaller the value the better the player (e.g. reaction time). Depending on the attribute's type alternative estimation functions can be defined.

Throughout this paper, we will use subsequent symbols:

- $ef$ - an estimate of a player's level of skills,
- $efF$ - an estimate of a player's individual skill,
- $\text{diff}ef$ - an absolute difference between player's and computer's level of skills,
- $\text{diff}efF$ - an absolute difference between player's and computer's individual skill measure,
- $Fval$ - a value of an attribute,
- $Fmin$ - the minimal value of an attribute,
- $Fmax$ - the highest value of an attribute,
- DDA method's parameters:
  - $\text{weight}$ - a weight of a player's feature, informing how critical is that feature to the overall player's behaviour.
○ $p_{lim}$ - a percentage value defining an adjustment free region,
○ $p_r$ - a SFDDACP parameter adding slight irregularity into computer's behaviour, defining a maximum difference between the opponent's new attribute's value and attribute's value of a player.

Above symbols, can have subscripts. The subscript $i$ is an index of an attribute ($i=1...n$, where $n$ is a number of evaluated player's attributes), $O$ means that it is opponent's measure (e.g. $Fval_{O,i}$). Subscript $P$ means that it is a measure of a human player (e.g. $efF_{P,i}$).

$F_{max}$ and $F_{min}$ can be set on the basis of common knowledge, could be defined intuitively or obtained from practical experiments. A value $Fval_P$ is determined during or after each player's encounter with an opponent. For example, in games like FPS (First person shooter) and TPS (Third-person shooter) we can measure reaction time of human player (i.e. interval from an enemy becoming visible to a player starting shooting) or his weapon spread. In car racing games we can calculate player's speed in different circumstances or evaluate player's understeering, oversteering or counter-steering. There are many possibilities. A value $Fval_O$ is set at the beginning of the game at random or with predefined initial value and then changed according to DDA methods.

For player's features a following evaluation function was proposed:

$$
eff_i = \begin{cases} 
\frac{Fval_i - Fmin_i}{Fmax_i - Fmin_i} & \text{for BVBP attributes} \\
\frac{Fmax_i - Fval_i}{Fmax_i - Fmin_i} & \text{for SVBP attributes}
\end{cases}$$

(1)

The value $eff_i$ ranges from 0 to 1. The greater the $eff_i$ value the more skilled in analyzed aspect the player. If the $eff_{P,i}$ is equal to 1 and the $eff_{O,i}$ is equal to 0 then we have the easiest situation for human player in regard to the feature $i$.

In the FDDACP we can use a weighted sum of the $eff_i$ values to compute a human player's and a computer opponent's level of skills, if more than one attribute contributes to the final value (else we use equation 1):

$$ef = \frac{\sum_{i=1}^{n} eff_i \ast weight_i}{\sum_{i=1}^{n} weight_i}$$

(2)

The value $ef$ ranges from 0 to 1. The greater the $ef$ value the more skilled the player. In order to simplify obtaining $ef$ value we can often try to use values, which logically define how player performs in a game. For example, in racing games we can use player's time of race (SVBP attribute).
The SFDDACP method compares $efF_i$ values for a human player and a computer opponent for each player feature. The FDDACP method compares $ef$ values.

For the SFDDACP method, if $\text{diff}efF_i > (p_{lim}*efF_{P,i})$ a value of opponent's attribute $i$ is adjusted:

$$Fval_{O,i} = Fval_{P,i} \pm (Fmax_i - Fmin_i) p_r$$  \hspace{1cm} (3)

In the FDDACP method, if $\text{diff}ef > (p_{lim}*efF)$, opponent's $Fval_O$ for each attribute is adjusted by the means of increasing or decreasing $Fval_O$ value. Which operation will be chosen is decided based on two factors: firstly we check which player has bigger $ef$ value, secondly there are alternative functions for attributes BVBP and SVBP.

First of all we compute $adjF_i$ that will be used to adjust opponent's attributes values:

$$adjF_i = \begin{cases} 
  \text{diff}ef * (Fmax_i - Fmin_i) & \text{for BVBP attributes} \\
  -\text{diff}ef * (Fmax_i - Fmin_i) & \text{for SVBP attributes}
\end{cases}$$  \hspace{1cm} (4)

In the next step in FDDACP, we change all opponent's attributes as follows:

$$Fval_{O,i} = \begin{cases} 
  Fval_{Old_{O,i}} + adjF_i & \text{for } ef_F > ef_O \\
  Fval_{Old_{O,i}} - adjF_i & \text{for } ef_F < ef_O
\end{cases}$$  \hspace{1cm} (5)

where $Fval_{Old_{O,i}}$ is a value of an opponent's attribute $i$ used during previous confrontation.

After that we check, in both methods, if new $Fval_{O,i}$ ranges from $Fmin_i$ to $Fmax_i$. If it is not the case we clamp $Fval_{O,i}$ to valid range:

$$Fval_{O,i} = \begin{cases} 
  Fmax_i - (Fmax_i - Fmin_i) p_r & \text{for } Fval_{O,i} > Fmax_i \\
  Fmin_i + (Fmax_i - Fmin_i) p_r & \text{for } Fval_{O,i} < Fmin_i
\end{cases}$$  \hspace{1cm} (6)

Using proposed methods we should obtain a computer opponent with a skill level similar to a skill level of a human player. In the next section a performance of the presented methods is tested for different conditions with a simple FPS game.

3. Experimental evaluation

Proposed methods should behave similarly for every player, regardless of its skills. The performance of the methods shouldn't differ also for different games environments. In the ideal circumstances a player's win rate should be 50%. Therefore, to evaluate the proposed methods series of experiments were conducted. As the test bed the FPS game "Seek&Shoot" was employed. The game was developed in the Unity 3D game engine, with the aid of several leading systems like RAIN AI and Ultimate FPS. The game's characters mechanics and characters behaviours (AI) have been created with the RAIN system. The UFPS was utilized
for the implementation of shooting and the injury simulation. For the characters motion the Navigation Mesh and the Waypoint Network were used.

The aim of a human and a computer player in this game is to find an enemy in a three-dimensional scene and shoot him. In order to simplify a study of methods' performance and find clear relationship between method's parameters and opponent's actions, a simple game's environment and a player's behaviour were proposed. The game scenes are enclosed spaces with different number of randomly placed walls (fig. 2a). Walls block players' vision of each other. A virtual player is equipped with a visual sensor that allows him to spot the enemy. An opponent's behaviour is described by means of behaviour tree (fig. 1) with actions: look around, move, face player (when you see him or when he hits you), shoot player.

![Fig. 1. Computer player's behaviour algorithm](image)

During and after each confrontation data about player is collected: health, number of player hits, angle of weapon spread, ammunition used, reaction time (i.e. an interval from an enemy becoming visible to a player starting shooting), view range and view horizontal angle. These values are next used in DDA methods to define computer player attributes: BVBP attributes: a1-endurance (how much health decrease results in a player's death), a2-weapon damage force, a3-ammunition, a4-view range, a5-view horizontal angle and SVBP attributes: a6-weapon spread and a7-gun reload time (i.e. an interval, when a character can't shoot).

For every game there are 100 encounters (rounds). The round ends when one of the players dies (i.e. its health rate drops past an endurance level). If both players die during a fight the round ends in a draw. At first an opponent gets random attributes values ranging from $F_{\text{min}}$ to $1/3(F_{\text{max}} - F_{\text{min}})$. Then after each encounter its values are adjusted according to equations (3) - (6). For a calculation of $ef$ value a player health rate registered at the end of the round is used.
To test a mechanism of the evaluated methods three computer agents with different levels of skills, that can play through the rounds unaided, were used: beginner player (BP), average player (AP) and expert player (EP). Furthermore, the experiments for two scenes were conducted: simple scene (SS) and complex scene (CS). The complex scene has more walls than SS.

In addition, authors investigate the influence of the methods parameters on the obtained results. In particular, $p_{lim}$ value was examined (values: 0%, 5%, 10% and 15% were applied). Because we use one attribute to calculate $ef$ value the weight of a player's feature (weight) was omitted in this study. For the $pr$: 0% and 5% values were tested.

Computer opponents try to obtain the same difficulty level as players. Their attributes are adjusted by means of SFDDACP and FDDACP methods.

Ultimately, there were 60 tests performed. Each test was repeated 50 times. Each test has been given an identifier created from a name of the used method, the scene, the player, the value of $p_{lim}$ and $pr$ (e.g. test FDDACP-SS-AP-lim15 for FDDACP method, simple scene, average player and and $p_{lim}=15%$; test SFDDACP-CS-BP-lim10-r5 for a SFDDACP method, a complex scene, a beginner player, $p_{lim}=10%$ and $p_r=5%$).

For every test six rates are calculated: $WR_O$ (opponent's win rate), $WR_P$ (player's win rate), $DR$ (draw rate), $HR_O$ (opponent's health rate), $HR_P$ (player's health rate) and $AR$ (adjustment rate). The sum of $WR_O$, $WR_P$ and $DR$ for each experiment is equal to 100%. These values tell us (in percentages) how many times one of three situations occurs: player wins ($WR_P$), opponent wins ($WR_O$), there is a draw ($DR$). The bigger $DR$ the better matched a player and an opponent. The $WR_P$ value, on the other hand, should be similar to $WR_O$. The health rate informs us about the percentage of health a winner of the round is able to retain averagely during the encounters. The last rate $AR$ shows how frequently a computer
opponent is changed for every game. In order to combine above values in one describing similarity between players: similarity rate (SR), was proposed:

$$SR = 100 - (WR_p \times HR_p - WR_o \times HR_o) \times (100 - DR) / 10000$$

The smaller the SR, the lesser the similarity between players. For example, in the worst scenario: DR is 0 and the difference between WRp and WRo is 100 with HRp equal to 100 then SR is 0. In the best scenario $WR_p \times HR_p = WR_o \times HR_o$, for which SR is 100.

<table>
<thead>
<tr>
<th>TEST</th>
<th>WRp</th>
<th>HRp</th>
<th>WRo</th>
<th>HRo</th>
<th>DR</th>
<th>AR</th>
</tr>
</thead>
<tbody>
<tr>
<td>FDDACP-SS-EP-lim10</td>
<td>1.84</td>
<td>2.86</td>
<td>7.84</td>
<td>3.80</td>
<td>90.32</td>
<td>0.04</td>
</tr>
<tr>
<td>FDDACP-SS-AP-lim10</td>
<td>13.04</td>
<td>5.54</td>
<td>19.81</td>
<td>5.41</td>
<td>67.15</td>
<td>0.00</td>
</tr>
<tr>
<td>FDDACP-SS-BP-lim10</td>
<td>24.31</td>
<td>7.37</td>
<td>19.54</td>
<td>9.38</td>
<td>56.15</td>
<td>0.18</td>
</tr>
<tr>
<td>SFDDACP-SS-EP-lim10-r0</td>
<td>34.07</td>
<td>10.13</td>
<td>6.83</td>
<td>6.47</td>
<td>59.10</td>
<td>0.01</td>
</tr>
<tr>
<td>SFDDACP-SS-AP-lim10-r0</td>
<td>1.69</td>
<td>4.86</td>
<td>29.93</td>
<td>5.43</td>
<td>68.38</td>
<td>0.01</td>
</tr>
<tr>
<td>SFDDACP-SS-BP-lim10-r0</td>
<td>58.97</td>
<td>8.81</td>
<td>8.07</td>
<td>7.54</td>
<td>32.97</td>
<td>0.01</td>
</tr>
<tr>
<td>FDDACP-CS-EP-lim10</td>
<td>15.26</td>
<td>7.96</td>
<td>17.70</td>
<td>8.03</td>
<td>67.04</td>
<td>0.14</td>
</tr>
<tr>
<td>FDDACP-CS-AP-lim10</td>
<td>19.68</td>
<td>12.34</td>
<td>34.56</td>
<td>8.80</td>
<td>45.76</td>
<td>0.25</td>
</tr>
<tr>
<td>SFDDACP-CS-EP-lim10-r0</td>
<td>28.13</td>
<td>8.18</td>
<td>30.57</td>
<td>8.22</td>
<td>41.30</td>
<td>0.22</td>
</tr>
<tr>
<td>SFDDACP-CS-AP-lim10-r0</td>
<td>3.17</td>
<td>6.32</td>
<td>28.13</td>
<td>8.46</td>
<td>68.71</td>
<td>0.00</td>
</tr>
<tr>
<td>SFDDACP-CS-BP-lim10-r0</td>
<td>12.58</td>
<td>6.89</td>
<td>26.77</td>
<td>7.98</td>
<td>60.65</td>
<td>0.00</td>
</tr>
<tr>
<td>SFDDACP-CS-BP-lim10-r0</td>
<td>22.94</td>
<td>8.31</td>
<td>32.55</td>
<td>9.22</td>
<td>44.52</td>
<td>0.00</td>
</tr>
<tr>
<td>SFDDACP-SS-EP-lim10-e5</td>
<td>16.77</td>
<td>10.56</td>
<td>30.06</td>
<td>10.46</td>
<td>53.16</td>
<td>0.03</td>
</tr>
<tr>
<td>SFDDACP-SS-AP-lim10-e5</td>
<td>4.52</td>
<td>5.01</td>
<td>34.45</td>
<td>8.03</td>
<td>61.03</td>
<td>0.09</td>
</tr>
<tr>
<td>SFDDACP-SS-BP-lim10-e5</td>
<td>22.03</td>
<td>7.97</td>
<td>50.10</td>
<td>12.71</td>
<td>27.87</td>
<td>1.00</td>
</tr>
<tr>
<td>SFDDACP-CS-EP-lim10-e5</td>
<td>11.77</td>
<td>8.67</td>
<td>41.20</td>
<td>13.30</td>
<td>47.03</td>
<td>0.93</td>
</tr>
<tr>
<td>SFDDACP-CS-AP-lim10-e5</td>
<td>21.95</td>
<td>11.48</td>
<td>29.49</td>
<td>10.05</td>
<td>48.57</td>
<td>0.98</td>
</tr>
<tr>
<td>SFDDACP-CS-BP-lim10-e5</td>
<td>37.60</td>
<td>10.34</td>
<td>20.19</td>
<td>7.78</td>
<td>42.21</td>
<td>1.00</td>
</tr>
</tbody>
</table>

Obtained results (high DR and low HR) show that both methods allow a creation of an opponent with performance similar to a player (table 1). Close examination, however, reveals that FDDACP method behaves better then SFDDACP method (fig. 2). At first, this can seem surprising, as in SFDDACP tests with $p_s=0\%$, opponent's individual skills are identical as player's. In our simple game, we even went further and treated a skill and an attribute evenly so player's and opponent's attributes have identical values. This shows an important drawback of the SFDDACP method: it needs to recognize all player's skills influencing an overall player's behaviour. For complex behaviours it can be impossible. As we can see, even for simple cases, when we seem to control every aspect of player behaviour, we still are not able to identify all features deciding about an outcome of players encounters.

In the FDDACP we are adjusting player’s attributes based on the knowledge of a player’s overall behaviour, so if we make an error while identifying essential attributes others features of a computer player can be used to mitigate that. For example in shooting game we can improve an opponent by increasing its endurance or improving its hitting aim. In the SFDDACP we improve each feature separately.

Figure 2b shows results for the SFDDACP method with parameter $p_s$ equal to 0% or 5%. The aim of these tests is discovering if an introduction of randomness to values of opponent's attributes can improve the results. The answer is negative. For both values we acquire similar results. None of the methods produce behaviours that are better in most of the conducted
experiments. In this situation we analyze results dividing them according to a different criteria. First we take into account $p_{lim}$ parameter. This parameter influences how often opponent's attributes are adjusted during the game, which is measured by AR. For $p_r$ equal to 0, AR is close to 0, regardless of the $p_{lim}$ value (fig. 2b). This can be observed also in the figure 3a. SR value increases and decreases without a correlation with $p_{lim}$. For $p_r=5\%$ AR is closer to 1, but frequent attributes adjustments don't change opponent's behaviour significantly. This is because changes add in only values that fluctuates at around player's constant value. For this reason, in figure 3b we use average SR value, calculated for experiments differing only in the value of $p_{lim}$.

As it is seen in figure 3a tests SFDDACP-r0 and SFDDACP-r5 are comparable for different scenes. The tests SFDDACP-BP and SFDDACP-AP have better results for the CS scene. The tests SFDDACP-EP-r5 show better results for the SS scene, when tests SFDDACP-EP-r0 have the same results for both scenes.

Comparing the results of the tests with different levels of player's skills we can notice that the SFDDACP method don't guarantee the same behaviour regardless of the player. Figure 3a
shows that the results of the tests SFDDACP-r0 are characterized by certain regularities: the best results are observed for average players, for the SS scene opponents adjust better to experts than to beginners, for CS scene vice versa. The same regularities can be detect for the tests SFDDACP-r5, except for the tests SFDDACP-CS-EP-r5, which have better SR than the tests SFDDACP-CS-AP-r5. The observed regularities are not due to the characteristics of the SFDDACP method, but to the characteristics of the players and scenes. This is because SFDDACP method takes into account only the previously selected aspects of the player's behaviour ignoring the specificities of the player and the surrounding environment.

Comparing the results for the tests with the method FDDACP we do not see especially significant differences between the results of the individual tests, regardless of the player skills. There are slightly better results for less complex scene, but the difference is not major. Obviously, it is easier to modify a player with less disruptions, as in the SS scene. As for the parameter $p_{lim}$ we can observe a slight decrease in the SR for the CS scene with the increase in the value $p_{lim}$. It seems that we need more frequent adjustments (fig.3) for beginner and average player. The worst SR value for FDDACP method we get for the test FDDACP-SS-BP-lim0. We can assume that it is because there are too many changes to an opponent resulting in its too strong fluctuations around the skill level of the player.

4. Conclusion

In this paper we presented two DDA methods, which match the difficulty level of an opponent to a player's skill level. The results shows that the FDDACP method behaves better than the SFDDACP method even for simple games with highly controlled players. The SFDDACP method could produce an opponent with a skill level identical to player's if only all player's characteristics could be defined and introduced to a computer player. Unfortunately, human behaviour is too complex for that. The perceived human difficulty depends not only on the game specified skills (e.g. speed, fighting), but also on the player personality traits (e.g. patience, strategic thinking), which are difficult or impossible to include into the model. For this reason, we suspect that for more complex players, the method SFDDACP will generate opponents even less adjusted than the ones obtained in our experiments. The FDDACP method, in contrast, is more universal, and observed results showed greater independence from the player's skills level and complexity of the environment.
BIBLIOGRAPHY


Omówienie

Głównym celem każdej gry jest zadowolenie gracza, na które składają się rozrywka oraz wyzwanie. Jeżeli gra jest zbyt łatwa gracz szybko się nudzi, jeżeli zbyt trudna - szybko frustruje. Zadaniem dynamicznego dostosowywania poziomów w grach komputerowych jest dostosowanie trudności gry do umiejętności gracza.

Bieżący artykuł przedstawia dwie metody, które pozwalają w prosty i szybki sposób dostroić komputerowego przeciwnika do poziomu gracza: metoda globalnego dostosowania komputerowego gracza (FDDACP) oraz metoda dostosowywania indywidualnych umiejętności gracza (SFDDACP). Metody porównują ilościowo zmierzone wyniki gracza komputerowego i człowieka i na podstawie otrzymannych wartości modyfikują ustawienia przeciwnika komputerowego.

Zaproponowane metody zostały zastosowane w prostej grze polegającej na strzelaniu do przeciowników, ukrywających się za ścianami rozstawionymi po scenie. W sumie
przeprowadzono 60 testów dla dwóch scen z różną liczbą ścian oraz trzech graczy z różnymi poziomami umiejętności (początkujących, przeciętnych, ekspertów). Testy różniły się też częstością poprawiania przeciwnika.

Otrzymane dla obu metod wynik (tabela 1) charakteryzują się dużą liczbą remisów, a w przypadku wygranej jednej ze stron jej bardzo niskim zachowanym poziomem życia. Świadczy to o silnym wyrównaniu poziomów obu graczy. Bardziej szczegółowa analiza wskazuje jednak na przewagę metody FDDACP nad metodą SFDDACP (rys. 3). Wynika to z faktu, że jakość SFDDACP zależy od umiejętności zidentyfikowania wszystkich warunków wpływających na zachowanie gracza. Nawet w przypadku prostej gry, z silnie kontrolowanym graczem jest to niewykonalne. Metoda FDDACP jest bardziej uniwersalna, pozwala na dostosowanie się przeciwnika komputerowego, niezależnie od definicji sceny i graczy.

**Address**

Ewa Lach: Silesian University of Technology, Institute of Informatics, Akademicka 16, 44-100 Gliwice, Poland, ewa.lach@polsl.pl.