Adaptive Agent Generation Using Machine Learning For Dynamic Difficulty Adjustment

Joy James Prabhu Arulraj
Department of Computer Science and Engineering,
College of Engineering, Anna University,
Chennai, India - 600025
e-mail: prabhu.link@gmail.com

***Abstract*—Player experience is a significant parameter in evaluating the overall success of a game, both technically and commercially. It is necessary to provide the player a game that provides: (1) satisfaction and (2) challenge. To enhance player experience, the game difficulty needs to be dynamically adjusted with respect to the player. Dynamic scripting is an importantlearning technique used for dynamic difficulty adjustment (DDA), already implemented successfully in commercial games of different genres. However the DDA systems are not sufficiently consistent in creating equally competent agents and do not provide equal opportunity to human players of different capabilities. This paper focuses on solving these issues by introducing these concepts: (a) dynamic weight clipping, (b) differential learning and (c) adrenalin rush. Experimental results indicate that dynamic scripting, in combination with these features, can implement an ideal DDA system for creating a equally competent computer agent who can engage the human player in absorbing games.**

Keywords- Dynamic difficulty adjustment, Differential learning, Adaptive algorithm, Adrenalin rush

I. INTRODUCTION

Playing with an equally competent computer agent gives the player a sense of satisfaction and challenge, which is directly related to the core player experience **[1]**. Traditionally game difficulty is relatively static and the responsibility of choosing the appropriate level lies with the human player. As the players generally find it difficult to estimate their abilities, they end up playing a game which is either too difficult or too simple for them. The main issues faced here are:

- Limited difficulty variations :

In general, a few static levels are provided in commercial games for the player to choose from, thus restricting the player to limited difficulty variations.

- Difficulty gaps between levels :

There exist large differences in the difficulty settings of these levels. This restricts the player to be content with an uneven competitor, if the player's difficulty level falls inside these gaps.

- Unresponsiveness to player learning :

As the game progresses, the player's abilities dynamically grow and a static game is *unresponsive* to player learning.

- Implementation of difficulty variation :

The difficulty variation is generally achieved by means of quantitative changes rather than qualitative changes.

Hence, the design of the opponent AI needs to be greatly improved in addition to other game characteristics associated with high entertainment value. Adapting the game to respond to a player's abilities over the course of a game session can be achieved using a DDA system. It is important to note that these adjustments should ideally go *unnoticed* and must definitely not disrupt the player experience.

II. DYNAMIC SCRIPTING

Dynamic scripting is a learning technique for creating agent AI, which is characterized by stochastic optimization **[7]**. A brief description of the technique follows. A database consisting of rules of different difficulty levels, generated by varying their qualitative aspects is maintained. Whenever an agent is created, these rules are used to create an agent script that controls the agent behavior. The rules that comprise the script are selected from the rule-database based on the agent's difficulty level. For each rule in the database, a value termed as *weight* is assigned dynamically.

Probability of selection of a rule for a script is directly proportional to its weight. Dynamic scripting is the technique of adjusting these weights in such a manner that the fitness / performance of the agent varies dynamically in response to player learning and player sophistication. The rules can be assigned *priority* values to allow certain rules to be more favored than other rules. Rules can also be grouped together into *rule-sets* so that either all the rules in the set are executed in a specified order or none of them are executed.

A. Team Fitness Function

We use a role-playing game **MiniGate**, available online **[2]**, to implement the algorithms and techniques described here. The game provides the basis for several DDA modules currently available in commercial games. A screenshot of the RPG is provided in figure 1. In this game, the fitness functions are used for measuring the performance of a computer agent and the entire team. Based on these

978-1-4244-9034-/10/$26.00©2010 IEEE

measurements the weight adjustment mechanism is performed. In a similar way, the game developers can determine the fitness functions for games of *any genre*, based on the agent performance indicators.

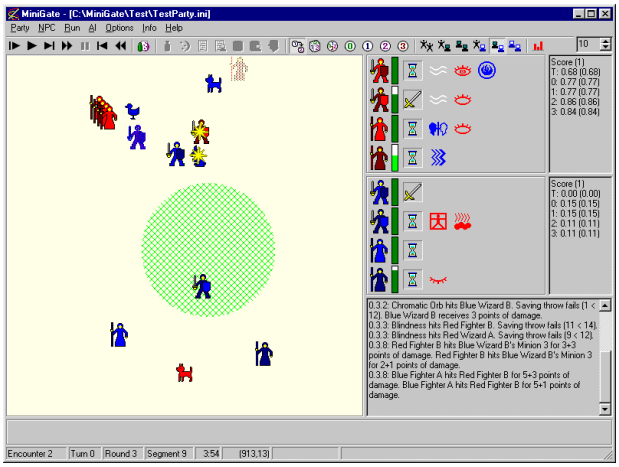

Figure 1. MiniGate - RPG used for the experiment

The team fitness function is given by Equation (1) :

$$F(g) = \sum_{c \in g} \left(\frac{1}{2N_g}\right)\left(1 + \frac{h_t(c)}{h_t(0)}\right) \qquad (1)$$

In Equation (1), g denotes the team, c denotes an agent, N_g is the total number of agents in team g, and $h_t(c)$ is the health of agent c at time t. By definition, a lost team is assigned a fitness of zero, while a *winning* team has a fitness > 0.5. An elaborate description of dynamic scripting is given in **[7]**.

III. COMPLEXITY SWITCHING

The weight adjustment mechanism performed in second phase of agent learning process is implemented in such a manner that the total weight remains constant. To implement this, the increment/decrement of specific rule-weights is evened out by decreasing/increasing all remaining rule-weights in a global manner. The *conservation of weight* realized by means of such a redistribution of weights is a key aspect of dynamic scripting.

A. Rule Classes

The human player's difficulty level may increase due to player learning as the game progresses. Then the agent's difficulty level is appropriately changed by increasing the weights assigned to complex rules and accordingly decreasing the weights assigned to simple rules. This mechanism of redistribution of weights to adjust the agent complexity is defined as **complexity switching**.

Rule-classes are collections of rules of similar complexity. It must be noted that all the rules in a rule-class need not have equal complexity. But, rules from different rule-classes differ considerably in their complexity. Such classification helps in creating player-specific game environment. Variation in complexity must be implemented qualitatively and not quantitatively in the game. The process of weight adjustment is *reversible*. It is possible to either scale-up or scale-down the agent-fitness by using complexity-switching.

TABLE I. DESCRIPTION OF RULE-CLASSES

Level	Class (i)	Class-Weight (μ_i)	Fitness
Novice	1	0.15	<25
Amateur	2	0.30	25 - 40
Intermediate	3	0.50	40 – 60
Professional	4	0.70	60 – 75
Expert	5	0.85	>75

B. Learning Limits

In the MiniGate simulator **[2]** used for our experiment, five rule-classes are defined. Each class is assigned a *class-weight* in the range [0,1], which are denoted by μ_i. The rule-classes in the experiment are listed in Table 1. The fitness values of different levels are given in the range of [0,100]. The class-weights are indicative of the general complexity of the rules in the rule-class. The extent of difficulty scaling desired is constrained by these class-weights. Consider the extreme learning situations in which all the weight is assigned only to the expert class or the novice class (due to principle of conservation of weight). These situations determine the limits of learning by the agent. Beyond this, the agent AI is unable to further scale-up or scale-down its complexity in response to player learning. In such cases, the agent AI is defined to have reached its *learning limit*.

In general, for any game, higher variation among class-weights leads to higher learning capacity of agent. However, high variations deteriorate the agent's capacity to remain unobtrusive. Thus, depending on the desired extent of agent-learning, the class-weights are chosen for a game. The fitness functions must be formulated from game-specific constraints to control the trade-off between *learning capacity* and *the ability to remain unobtrusive* **[3]**.

C. Aggregate complexity

Agent AI starts exploiting what it has learned as early as possible, to be *effective* in practice. But, agent AI must learn as much as possible for exploration, to create a dynamic environment and be *efficient* in practice. To ensure that the overall script controlling the agent behavior generates an even competitor, the script-complexity is measured using a

quantity defined as **aggregate complexity**. Aggregate complexity is denoted by ξ and is defined in equation (2):

$$\xi = \sum_{i=1}^{\eta} (\mu_i)(w_i) \qquad (2)$$

In equation (2), η denotes the number of rule-classes chosen, w_i denotes the weight assigned to the i^{th} rule-class and μ_i denotes the class-weight of the i^{th} rule-class. Here, η =5 as is indicated in Table 1. Let ω represent the total weight conserved determined from estimated player-fitness. Using the above notation, we obtain these relations giving the theoretical limits of agent-learning using complexity-switching model:

$$\omega = \sum_{i=1}^{\eta} (w_i) \qquad (3)$$

$$\frac{\xi}{\omega} \in [\mu_1, \mu_5] \qquad (4)$$

The maximum pair wise difference between class-weights constrains the ability of the game to remain unobtrusive. This ability is denoted by κ and is quantitatively defined in equation (5):

$$\kappa = \frac{|\mu_{\max} - \mu_{\min}|}{|\mu_{\min}|} \qquad (5)$$

In this experiment, κ = (0.85-0.15)/0.15 ≈4.67. Higher κ enhances agent-learning but makes the agent AI more obtrusive. Its value must be chosen as per desired level of agent learning and practical experimentation with human players.

IV. DYNAMIC WEIGHT CLIPPING

Complexity switching results in variation of weights assigned to different rule-classes as agent-learning progresses. This variation inherently leads to *biased-scripting*, wherein rules in a few rule-classes are always favored by virtue of the weight of their rule-classes, while the remaining rule-classes are relatively less favored. Biased scripting has been addressed using static weight clipping described in **[4]**. To control this phenomenon, W_{max} and W_{min} are defined. The weights of all the rule-classes must fall within this range $[W_{min}, W_{max}]$. This principle constrains biased-scripting by limiting complexity-switching. However, static weight clipping also constrains dynamic difficulty scaling to a large extent, thus unfortunately creating an *uneven* agent competitor as the game progresses. To eliminate this issue, the weight clipping mechanism must be dynamic by nature.

A. Window-Size of a script

In dynamic weight clipping, W_{max} and W_{min} are defined by using the current agent configuration as a reference and hence vary dynamically as the agent AI learns. It can be quantitatively described using a quantity termed as *window-size*. Window-size of the script is defined as the ratio of the difference between the maximum and minimum values of the set of class-weights CW= (w_1, w_2, w_3, w_4, w_5) to the minimum value of the same set. It is observed that static weight clipping mechanism uses a fixed window-size. In contrast, dynamic weight-clipping mechanism uses a **variable-sized window** dependent on current agent configuration. Window-size is denoted by , and is defined in (6) :

$$= \frac{(CW_{\max} - CW_{\min})}{CW_{\min}} \qquad (6)$$

B. Sliding-lines effect

Thus, W_{max} and W_{min} are defined based on current aggregate complexity ξ. The equation (7) quantitatively defines dynamic weight-clipping:

$$W \in [\lambda_{\min}\xi, \lambda_{\max}\xi] \qquad (7)$$

λ_{max} and λ_{min} are based on desired trade-off between exploration and exploitation by script. λ_{max} = 2.0 and λ_{min} = 0.2 are suitable for the CRPG game. Maximum window-size is denoted by $_{max}$, and is defined in equation (8) :

$$\phi_{\max} = (\frac{\lambda_{\max} - \lambda_{\min}}{\lambda_{\min}}) \qquad (8)$$

As the value of $_{max}$ is reduced, variance in class-weights is reduced. This effectively eliminates biased-scripting. However, this can constrain agent-learning capacity. Hence, values of λ_{max} and λ_{min} are chosen to generate an equally competent or *even* competitor. Thus, $_{max}$= (2.00-0.20)/ 0.20 = 9.0 in the experiment.

The value of is constrained by $_{max}$ and varies with player learning as game progresses. Hence the window-size is variable and the window itself is determined by *sliding-lines*. This effect is termed as the **sliding-lines effect** and is illustrated in figure (2).

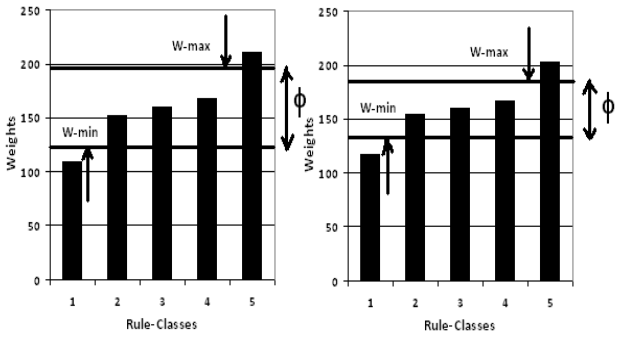

Figure 2. Sliding-lines effect in Dynamic weight-clipping

It is observed that varies in size and exhibits sliding lines effect as the game progresses. The window constrains the exploration aspect of the game AI and hence the parameters must be chosen to achieve the *perfect blend* of exploration and exploitation. To analyze the impact of dynamic weight-clipping, we simulate the game between the improved agent AI and human players of different difficulty-levels, using the MiniGate-based simulator. The results indicate that dynamic weight clipping succeeds in creating not only an **even** competitor, but does so much more *consistently* than its static counterpart.

V. DIFFERENTIAL LEARNING ALGORITHM

Learning curves indicate that the agent has to learn faster initially and then fluctuate as per the player at a slower pace. The agent learning rates must be chosen to ensure that the agent learns quickly initially using **giant-steps** to come closer to the player difficulty level and then fine-tune its difficulty using **baby-steps** to adapt to the player difficulty fluctuations. The concept described here is defined as **differential learning**. Non-smooth scaling can be introduced using appropriate learning rates if desired.

A. Learning Rate

The maximum difficulty variation possible by weight adjustment mechanism in a single round is restricted by a quantity defined as the *learning rate* of the agent. Learning rate is an important parameter, as the player experience is directly influenced by the difficulty adjustment performed. It is denoted by ξ_{rate} as it controls the variation in agent AI complexity. Differential learning must ensure that the player does not feel *cheated* due to difficulty adjustment.

An ideal implementation must keep the player ignorant of difficulty adjustment, which leads to increased player satisfaction **[5]**. The learning process is formalized here. The initial player fitness is denoted by Ψ_0. The difference between current and initial player fitness values is termed as *fitness change* and is denoted by $\Delta\Psi$. Learning coefficient is a measure of current player learning. It is defined as the ratio of the fitness change to the initial player fitness and is denoted by Ψ_c. Thus, the relationship is given in equation (9):

$$\psi_c = \frac{\Delta\psi}{\psi_0} \qquad (9)$$

Learning coefficient is an useful indicator for creating an absorbing game. This will be discussed in the section (6).

B. Giant -step Baby-step Algorithm

It is observed that the agent exhibits giant steps initially and then settles into taking baby steps. This is implemented using the *differential learning algorithm*. In the algorithm, the difference between the fitness values of the agent and the player is used to determine the learning rate of the agent. The constants in Table 2 are the different agent learning rates in the experiment.

TABLE II. DIFFERENTIAL LEARNING RATES USED FOR AGENT LEARNING

ξ_{rate}	α_1	α_2	α_3	α_4
Range	< 8	8 - 20	20 – 50	➢ 50
Value	2	4	18	25

VI. ADRENALIN RUSH

For a given Ψ_0, a corresponding value of $\Delta\Psi$ is defined as the *adrenalin limit* or *learning threshold*. The curve providing the fitness change needed to cross the adrenalin limit as a function of Ψ_0 is defined as the **adrenalin curve**. When the player's complexity Ψ crosses the adrenalin limit, the player must have learnt enough in the given game constraints, say in a finite time-interval. It must be noted that when the player starts playing the game again, the adrenalin limit is recomputed based on the initial player fitness Ψ_0.

A. Producing an Adrenalin Rush

The player learning slows down into a stable part after the steep initial learning component as indicated in the player complexity graph given in figure (4). When the player's complexity Ψ crosses the adrenalin limit defined by adrenalin curve, the learning rate must be reduced. If the agent still keeps learning at normal pace, the player will be disappointed when the agent gradually *outgrows* the player, even after the player having learnt much in the game. It is a serious issue to be taken care of while performing dynamic difficulty adjustment. The adrenalin curve is presented in figure (3). It must be noted that the curve is plotted based on experimental parameters.

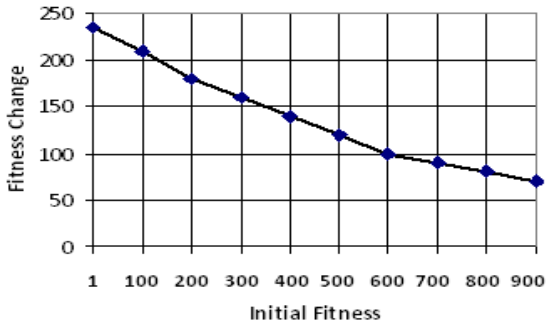

Figure 3. Adrenalin Curve

When the player's learning coefficient crosses the learning threshold, the learning rate must be reduced to enhance player's chances of winning. This ensures the *challenge* and *satisfaction* criteria initially proposed in the paper. This mechanism is termed as **adrenalin rush**, as the player gets to play an absorbing game consistently, after having crossed the adrenalin limit.

VII. EXPERIMENTAL SETUP

The experiment is performed using MiniGate **[2]**, a RPG simulator based on static weight clipping, along with the techniques (1) differential learning,(2) dynamic weight clipping, and (3) adrenalin rush. The game is the basis of recently implemented DDA modules in commercial games like Neverwinter Nights **[6]**.

The initial version is created by implementing dynamic scripting in the MiniGate - based simulator. The source data is obtained from 100 experiments, each comprising of 100 games performed using the simulator. In each game, the agent is pitched against human players from each of the 5 rule-classes/difficulty levels and also a *general class* comprising of equal proportion of players of all the 5 rule-classes. The techniques were integrated with the initial version in a step-by-step fashion, one technique at a time; the final version comprises of all the techniques implemented in the framework of dynamic scripting.

A. Agent Learning Curve

The learning curve of the final agent, generated by integrating all the techniques, is given in figure 4. For a few static levels, the configuration of weights for different rule-classes are predefined. The agent estimates the player fitness initially by measuring the player performance and then computes the static level nearest to the estimated value. For instance, in our experiment, if the player's estimated fitness is 556, the nearest predefined static level is 500. Using the differential learning technique, the agent learns quickly initially and then after crossing the adrenalin limit, adapts as per the player.

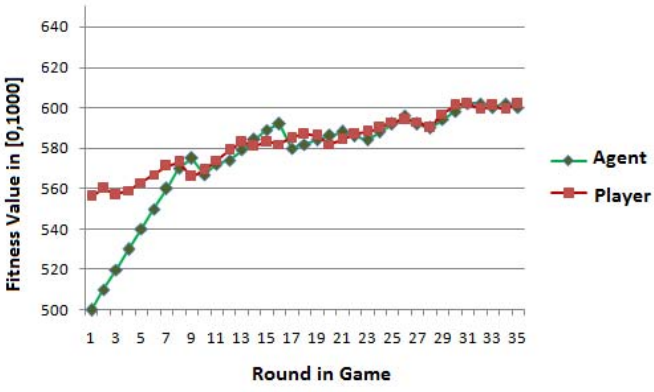

Figure 4. Minigate Simulator - Final Version

The Table 3 presents the experimental data obtained for the general class. The general class consists of players of all classes in equal proportion and is thus representative of the overall performance of the agent. As seen in the Table 3, the performance has consistently improved in successive versions of the CRPG simulator. An overview of the entire DDA module that is used in the final integrated version is provided in Figure 5.

TABLE III. COMPARATIVE ANALYSIS OF GAME VERSIONS

Id	Version Technique	Agent Win	Std. Deviation
0	Dynamic Scripting	65.41	300.22
1	Static Clipping	79.51	84.24
2	Differential learning	57.12	107.51
3	Dynamic Clipping	58.37	91.32
4	Adrenalin Rush	53.87	62.79

VIII. RESULTS

The results of the dynamic scripting integrated with these techniques are given in Table 4. It is observed that the agent consistency has improved, as the standard deviation values of agent win variable is significantly small. The agent performs consistently across different classes, providing equal opportunity to players ranging from novice to expert classes.

Algorithm 1 Final Integrated DDA Module

Require: $\omega_i \in [W_{min}, W_{max}]$ for $0 \leq i \leq 5$
Ensure: $\xi/\omega \in [\mu_1, \mu_2]; \Phi \leq \Phi_{max}; \xi \in [0, 1000]$

```
δ ← 0
N ← 100 {Number of Rounds}
while N ≠ 0 do
    Measure agent fitness functions F(g)
    Estimate player fitness Ψ from game parameters
    Determine the closest static level ξ
    Evaluate δ = |Ψ − ξ|
    Compute learning rate ξ_rate
    Perform Weight adjustment ΔW_j using ξ_rate
    Measure the window-size Φ
    Perform sliding-line effect to adjust window-size
    N ← N − 1
end while
```

Figure 5. Algorithm used in Final Integrated DDA Module

We observe that the agent is not able to scale up beyond the learning limit in a few games. This is more frequent for novices and is almost non-existent for other classes. This can be attributed to the fact that the novice players learn a lot at a fast pace, leaving the agent behind when they cross the learning threshold. This occurs in about 7% of the games played in the novice class.

TABLE IV. ANALYSIS OF DYNAMIC DIFFICULTY SCALING

Class	Agent Win	Std. Deviation	Limit
Novice	53.44	59.91	7
Amateur	54.14	65.49	1
Intermediate	53.83	62.16	0
Professional	53.13	56.39	0
Expert	52.48	57.96	0
General	52.33	65.53	0

The expected window size in the general case is =1.96. The window size value is highest for the novice class as the agent has to scale up to a larger extent. Using equation (9), this implies that:

$$CW_{\max} \approx 3 * CW_{\min} \qquad (10)$$

The sliding line effect is successful in controlling the agent AI to create a consistent as well as even competitor. It ensures that the agent keeps *exploring* different strategies and uses successful techniques as a fall-back mechanism when needed. The agent performance in Final version is depicted using histogram in Figure 6. Histograms are based on agent win variable in 100 games performed against players of general class. The variance of the agent is low and the agent win variable is focused in the [40,60] range. Evidently, this agent can provide the player both challenge and satisfaction.

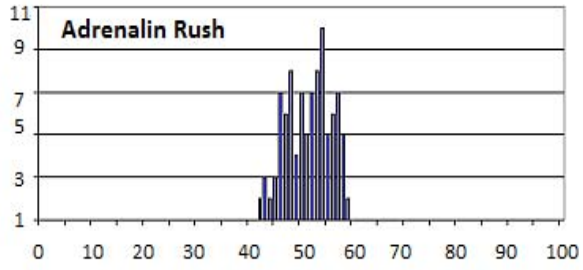

Figure 6. Final version's Histogram

IX. CONCLUSIONS

A significant observation from this work is that the dynamic difficulty adjustment system can be developed for *any* game wherein the human player plays against an agent. In such a case, the fitness functions and weight adjustment mechanisms should be defined by the game developers based on the game and desired difficulty scaling. In future work, introducing variation in script selection is to be analyzed.

REFERENCES

[1] P. Tozour, ``The evolution of game AI", AI Game Programming Wisdom, Charles River Media, 2002, pp.1-21.

[2] P. Spronck., ``MiniGate Role Playing Game Environment", url:http:/ticc.uvt.nl/~pspronck/minigate.html, 2007

[3] J. Manslow, ``Learning and adaptation", AI Game Programming Wisdom, Charles River Media, 2002, pp.557-566

[4] P. Spronck, I. Sprinkhuizen-Kuyper, and E. Postma., ``Online adaptation of game opponent AI with dynamic scripting," International Journal of Intelligent Games and Simulation 3 (2004), pp.45-53, 2004.

[5] R. Hunicke, ``The Case for dynamic difficulty adjustment in games," ACM International Conference on Advances in Computer entertainment technology, pp.429-433, 2005

[6] Bioware, Neverwinter Nights, url: http://nwn.bioware.com/, 2010

[7] P. Spronck, M. Ponsen, I. Sprinkhuizen-Kuyper, and E. Postma, ``Adaptive game AI with dynamic scripting," Machine Learning, 2006 (63), pp.217-248.