

# Learning Unit Values in Wargus Using Temporal Differences

P.J.M. Kerbusch

16th June 2005

## Abstract

In order to use a learning method in a computer game to improve the performance of computer controlled entities, a fitness function is required. In real-time strategy (RTS) games it seems obvious to found the fitness function on the game score, which usually is derived from the number and type of units a player has killed. In practice the value of a unit type is set manually. This paper proposes to use Temporal Difference learning (TD-learning) to determine the value of a unit type. An experiment was performed to determine good unit values for use by the learning mechanism ‘dynamic scripting’ in Wargus. The results of this experiment demonstrate significantly improved learning performance using the newly-determined unit values, compared to using the original unit values which were manually determined by the Wargus game developers.

**Keywords:** Temporal Difference learning, adaptive game AI, commercial games, dynamic scripting, Wargus, unit values, fitness function, prediction probabilities

## 1 Introduction

Nowadays realism is becoming one of the most important aspects in commercial gaming. Beside the increasing realism of the graphics and sound in commercial games, game developers strive for more realism in game play. One aspect of realistic game play is the behaviour of the computer controlled entities in the gaming environment. This is where developers can benefit from research in Artificial Intelligence (AI)<sup>1</sup>.

In order to achieve more realistic behaviour of computer opponents in games Adaptive Game AI can be used. Ponsen *et al.* [4] implemented an Adaptive Game AI in a game called Wargus [9], which is an open-source clone of the RTS game Warcraft II. Ponsen *et al.* [4] in

<sup>1</sup>Here AI is defined as intelligent software rather than scripted opponents in games (the latter definition of AI is used by commercial-game developers). In this paper the scripted opponents will be referred to as ‘Game AI’.

their learning mechanism use a fitness function that evaluates how well each of the two opponents in the game perform. The performance of the Adaptive Game AI is dependent on the quality of the fitness function.

The fitness function as explained by Ponsen *et al.* awards points for killing enemy units. The number of points awarded seems to be correlated to the number of hit points (maximum health) the killed unit has, but there is no justification for these points being a correct representation of the fitness of a player. This follows directly from the origin and purpose these points have: they were used by the developers to give the player a score, which has nothing to do with the game itself.

In this paper it is proposed to use TD-learning [7] to determine new unit values, to be used in the fitness function, in order to achieve a fitness function which improves the performance of the dynamic-scripting Game AI. The problem statement derived from the above is the following:

*To what extent can the learning performance of the dynamic-scripting Game AI for Wargus be improved by learning unit values using the method of TD-learning?*

The remainder of this article is as follows. Section 2 describes the approach used in this research. Section 3 outlines the experimental setup. Section 4 presents the evaluation of the results of the experiments. Section 5 discusses the results. Finally, in Section 6, conclusions and future work are presented.

## 2 Approach

In this section the approach of this research will be presented. In Section 2.1 the game Wargus will be explained. In Section 2.2 will be explained how dynamic-scripting Game AI was implemented for Wargus by Ponsen *et al.* [4]. In Section 2.3 the fitness function as used by Ponsen *et al.* [4] will be explained. Finally, in Section 2.4, the method of TD-learning will be explained.

### 2.1 Wargus

Wargus [9] is an open-source clone of the RTS game Warcraft II. In RTS games usually a player has the objective to destroy all opposing players. The player has to gather

and manage resources in order to construct buildings and units and to research upgrades. The key to defeating enemies commonly lies in strategically controlling these resources.

Usually the Game AI in RTS games is defined by a script, which is a collection of game actions (e.g. ‘attack with an army’ or ‘build a headquarters’) executed in sequential order.

In Wargus a game itself is played between two races, the Human race and the Orc race. Both races have their own units and buildings, of which some can be observed in Table 1. However, there is no difference between both races. Every unit of the Human race has a unit of the Orc race which has exactly the same statistics and vice versa. In Table 1 comparable units are placed in the same row.

	Human	Orc
1	Peasant	Peon
2	Footman	Grunt
3	Archer	Axe Thrower
4	Ballista	Catapult
5	Knight	Ogre
6	Ranger	Berserker

Table 1: Units in Wargus. Only those units used in this research are presented. In total both races have sixteen unit types available.

## 2.2 Dynamic Scripting

The dynamic-scripting technique is an online adaptation technique for Game AI, that can be applied to any game that meets three requirements [5]: (1) the Game AI of the game can be scripted, (2) domain knowledge on the characteristics of a successful script can be collected, and (3) an evaluation function can be designed to assess how successful the function was executed.

Dynamic scripting generates a new script at each encounter between players. The method of generating such a script consists of randomly selecting tactics from a knowledge base, the latter being constructed from domain-specific knowledge. The probability of a tactic being selected depends on its associated weight value. After completion of the encounter the dynamic-scripting Game AI’s learning is achieved by reinforcement learning techniques [6]: weights of tactics leading to success (the encounter was won) are increased and the weights of tactics leading to a loss are decreased.

In order to implement dynamic scripting in Wargus the game is divided in distinct states according to types of available buildings (see Figure 3). As presented schematically in Figure 1, each of these states has a unique knowledge base from which the dynamic-

scripting Game AI can select tactics until a final state is reached or a maximum number of tactics is used (100 was used as a maximum by Ponsen *et al.*). In the final state a maximum of 20 tactics is selected before the script moves into a repeating cycle of attacking the opposing player (‘attack loop’).

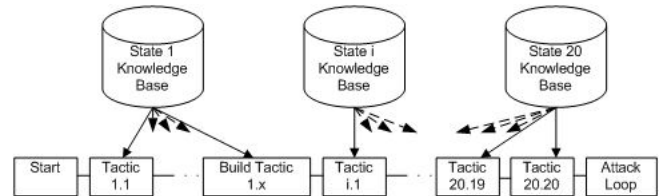


Figure 1: Schematic overview of dynamic script generation in Wargus [4].

## 2.3 Fitness Function

As explained in Section 2.2 the dynamic-scripting Game AI learns after each encounter. Learning entails that the weights in the knowledge bases are updated based on both the performance of the dynamic-scripting Game AI during the whole game and the performance between the state changes. Ponsen *et al.* call these the ‘overall fitness’ respectively ‘state fitness’. The overall fitness  $F$  is defined as follows:

$$F = \begin{cases} \min\left(\frac{S_d}{S_d+S_o}, b\right) & \{lost\} \\ \max\left(b, \frac{S_d}{S_d+S_o}\right) & \{won\} \end{cases} \quad (1)$$

where  $S_d$  represents the score of the dynamic-scripting Game AI and  $S_o$  the score of the opponent player.  $b \in [0, 1]$  is the break-even point, at which weights remain unchanged. The ‘state fitness’  $F_i$  at state  $i$  is defined as:

$$F_i = \begin{cases} \frac{S_{d,i}}{S_{d,i}+S_{o,i}} & \{i = 1\} \\ \frac{S_{d,i}}{S_{d,i}+S_{o,i}} - \frac{S_{d,i-1}}{S_{d,i-1}+S_{o,i-1}} & \{i > 1\} \end{cases} \quad (2)$$

where  $S_{d,i}$  represents the score of the dynamic-scripting Game AI after state  $i$  and  $S_{o,i}$  the score of the opponent player after state  $i$ .

In Equations 1 and 2, the score  $S_x$  of player  $x$  is defined as a combination of ‘Military’ points ( $M_x$ ) and ‘Building’ points ( $B_x$ ):

$$S_x = 0.7M_x + 0.3B_x \quad (3)$$

Ponsen *et al.* chose the weights 0.7 for the ‘Military’ points (awarded for destroying enemy units and buildings) and 0.3 for the ‘Building’ points (awarded for constructing buildings) arbitrary, expecting ‘Military’ points being a better indication of successful tactics than ‘Building’ points.

As a final step the weights of all rules employed are updated. Weight values are bounded by a range  $[W_{min}, W_{max}]$ . A new weight value is calculated as  $W + \Delta W$  where  $W$  is the original weight value and  $\Delta W$  is defined by the following formula.

$$\Delta W = \begin{cases} -P_{max} \left( C \frac{b-F}{b} + (1-C) \frac{b-F_i}{b} \right) & F < b \\ R_{max} \left( C \frac{F-b}{1-b} + (1-C) \frac{F_i-b}{1-b} \right) & F \geq b \end{cases} \quad (4)$$

In Equation 4,  $P_{max}$  and  $R_{max}$  are the maximum punishment and maximum reward respectively.  $C \in [0, 1]$  is the fraction of the weight adjustment that is determined by the overall fitness  $F$ . Ponsen *et al.* [4] took  $C = 0.3$  because it is expected that rulebases for different states become successful at different times. Moreover, when a game is lost, rules which were successful will not be punished too much.

Although there are many parameters of this ‘weight updating’ that can be subjected to further research, in this article the only parameter investigated is the number of points awarded for killing enemy units. In the implementation of Ponsen *et al.* those points are not justified as being a correct representation of the fitness of a player since they are originally used for giving a player a score, which they found was only marginally indicative of which player would win the game.

## 2.4 Temporal Difference Learning

TD-learning is a method proposed by Sutton [7] and has successfully been applied to games, for instance by Beal and Smith [2] for determining piece values in Chess. In this research the methods used by Beal and Smith [2] are used as a guidance for determining the unit values in Wargus.

Unlike most other prediction learning methods, which are driven by the difference between the prediction and the actual outcome, TD-learning is an incremental prediction learning method that uses differences between temporally successive predictions. Sutton [7] has shown TD-learning converges faster and produces more accurate predictions than conventional methods, since TD-learning makes more efficient use of its experience. Also, since TD-learning is incremental, it can be computationally more efficient.

Given a set  $W$  of weights which are to be learned,  $w_i$  ( $i..n$ ), and successive predictions  $P_t$ , the values of the weights are updated as follows [7]:

$$\Delta W_t = \alpha (P_{t+1}, P_t) \sum_{k=1}^t \lambda^{t-k} \nabla_W P_k, \quad (5)$$

where  $\alpha$  is the learning rate and  $\lambda$  is the recency parameter controlling the weighting of predictions occurring  $k$

steps in the past.  $\nabla_W P_k$  is the vector of partial derivatives of  $P_t$  with respect to  $W$ , also called the gradient of  $W P_k$ .

## 3 Experimental Setup

In this section the experiments will be described. In Section 3.1 the conversion from a game situation to a prediction probability will be presented. In Section 3.2 the environment in which the gamebase was created will be presented. In Section 3.3 will be described how the unit values were determined using TD-learning. Finally, in Section 3.4, will be explained how was tested whether the newly-learned unit values influenced the learning performance of the Adaptive Game AI.

### 3.1 Prediction Probabilities

In order to use the method of TD-learning [7] a series of successive prediction probabilities (in this case: the probability of a player winning the game) has to be available. Thus, at every possible moment in the game a prediction probability has to be able to be generated. One way of doing this, following the approach of Beal and Smith [2], is to denote the prediction probability given the game’s current status  $v$ ,  $P(v)$  by

$$P(v) = \frac{1}{1 + e^{-v}}, \quad (6)$$

where  $v$  is the evaluation function of the game’s current status, denoted by

$$v = \sum_{unit\_types} w_{type} c_{type}, \quad (7)$$

where  $w_{type}$  is the current weight of the unit  $type$  and  $c_{type}$  is the count of units of type  $type$  the Adaptive Game AI has minus the count of units of type  $type$  the opponent has.

The use of the squashing function  $P(v)$  has the advantage it has a simple derivative:

$$\frac{dP}{dv} = P(1 - P). \quad (8)$$

Thus for every unit type  $type$  the partial derivatives at step  $k$  in the past are defined as follows:

$$\nabla_w P_k = c_{type} P_k (1 - P_k). \quad (9)$$

In Figure 2 can be observed how an game state value  $v$  of 0.942 is converted to a prediction probability  $P(v)$  of 0.72.

### 3.2 Experimental Environment

A gamebase containing information from 299 Wargus games was created for this research. During each game

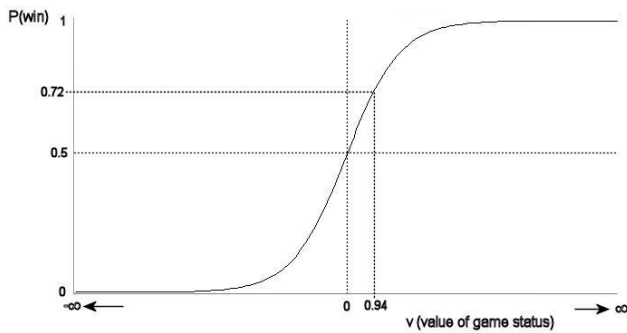


Figure 2: Conversion from game status to prediction probability [2].

information on the current number of units of both players was extracted every 1000 game cycles. With 30 game cycles per second this results in information being extracted every  $33\frac{1}{3}$  seconds. This resulted in a gamebase containing 14635 records which were used to derive prediction probabilities. The games themselves were played with restrictions on the opponent, maximum state and the map. These are discussed below.

### The Opponent

Wargus has several static Game AI scripts included, of which one is the ‘Improved Balanced Land Attack’ Game AI (IBLA). This Game AI gradually constructs buildings and attacks with a wide variety of the available units. To keep the game fair for both players, the IBLA Game AI has been adapted. It is restricted to build only those buildings the dynamic Game AI can. In order to keep the performance of the IBLA Game AI on a reasonable level it is using ‘rush’ tactics. This kind of tactic means focussing on offensive actions while defensive actions are less accounted for. Doing so the opponent is in many cases overwhelmed, unless specific counter actions are employed. As Ponsen *et al.* noted, ‘rush’ tactics are very strong tactics in Wargus, which are very hard to defeat. Ponsen did not succeed in using dynamic scripting to learn an answer to ‘rush’ tactics.

### Maximum state

In order to reduce the complexity of this research the focus was limited to state 12. State 12 is, as can be observed in Figure 3, a state which every game should pass if it does not end in a state lower than 12. Thus, information was only gathered when the game was in state 12.

Both players were modified in a way they did not want to evolve to state 13. At the end of state 12 the so called ‘attack loop’ commences. This is a process of attacking which loops until the game ends.

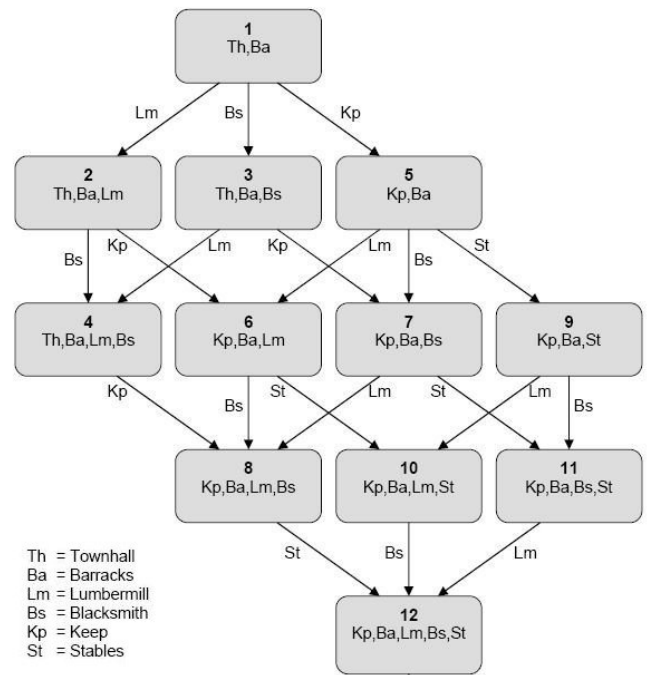


Figure 3: Overview of the possible state transitions in Wargus [4].

### The Map

Experience gathered by Ponsen *et al.* states most games played on ‘small’ maps end before state 12 is reached. In order to avoid this problem as much as possible during gamebase creation a ‘larger’ map is needed. In Wargus a ‘larger’ map is available, which can be seen on the left in Figure 4. However, by using this map the player starting at position 1 would be put at a disadvantage since it has two directions from which the opponent can attack, where the opponent can only be attacked from one direction. To overcome this problem the map has been modified in such a way that both sides have equal attack paths. The modified map can be seen in on the right side in Figure 4.

### 3.3 Learning Unit Values

Using all the data available from the gamebase, the unit values were learned using TD-learning, which is implemented in MATLAB [8] for this research. The parameter  $\alpha$  (learning rate) was set to 0.5 and the parameter  $\lambda$  was set to 0.95. The value of  $\lambda$  was chosen following the research of Beal and Smith [2]. The value of  $\alpha$  was chosen as a compromise between fast learning (high values) and low error sensivity (low values). The unit values were set to 1 before learning started.

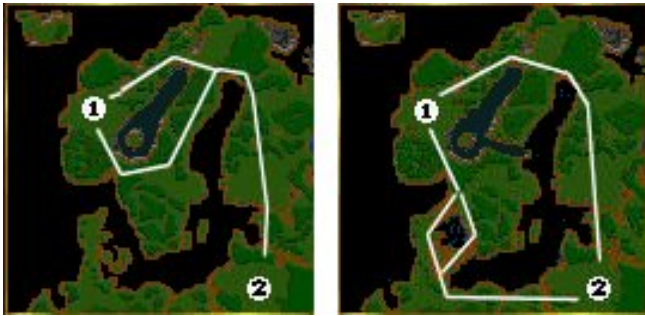


Figure 4: The original (left) and modified (right) maps. The numbers 1 and 2 are the starting positions of respectively the dynamic-scripting Game AI and its opponent. The white lines illustrate possible paths of attack.

### 3.4 Learning Performance Comparison

In order to answer the problem statement, a series of comparison experiments was performed. To determine the difference in learning performance twenty learning runs were performed. A learning run is defined as a sequence of 200 games during which the dynamic Game AI has to learn to defeat its opponent. Ten of the learning runs were performed using the original unit values. The other ten were performed using the unit values as learned by TD-learning as explained in Section 3.3. Furthermore, the same restrictions as explained in Section 3.2 applied.

## 4 Results

In this section the results of the experiments described in Sections 3.3 and 3.4 will be presented in 4.1 and 4.2, respectively.

### 4.1 Learning Unit Values Results

After feeding all available game data from the game base to the TD-learning algorithm, as explained in Section 3.3, new unit values for the used unit types were found. How the values of the different unit types changed during the process of TD-learning is illustrated in Figure 5. The original and newly-learned unit values are listed in Table 2. We point out three observations on these results.

Unit Type	Original	New
Peasant	30	239
Footman	50	24
Archer	60	-2
Ballista	100	79
Knight	100	94
Ranger	70	-13

Table 2: Unit value learning results. Original and newly-learned values of each unit type.

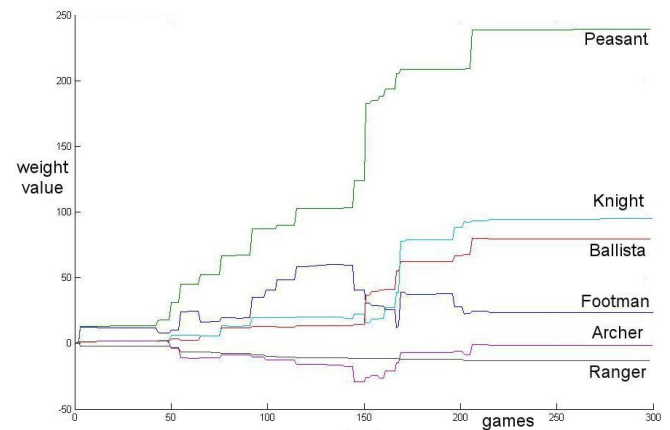


Figure 5: The weights of the unit types during the process of TD-learning.

First, it can be observed that the learned value of a ‘Peasant’ is significantly higher than the original value. Giving the ‘Peasant’ a low value may at first seem reasonable, since this type of unit is of no use in combat situations. But at the time the dynamic-scripting Game AI kills a ‘Peasant’ not only the static Game AI’s ability to gather resources will decrease, but it is also very likely the dynamic-scripting Game AI has already penetrated its opponents defences, since ‘Peasants’ usually reside inside or close to their base. This indicates killing ‘Peasants’ is a good indicator of success.

Second, it can be observed the ‘Archer’ and ‘Ranger’ units have weights less than zero. This indicates they are of little use to the army and actually a waste of resources.

Third, when looking at the weights themselves of the ‘Knight’ and ‘Ballista’, they seem to be the most valuable units in combat although they are also the most expensive.

### 4.2 Learning Performance Comparison Results

As explained in Section 3.4 twenty learning runs were performed of which ten were performed using the original unit values, and ten using the learned unit values from Section 4.1. In order to determine the difference in learning performance two measures were used, namely the absolute turning point and the relative turning point. These two measures were also used by Bakkes *et al.* [1] for measuring effectiveness of online Adaptive Game AI in action games.

**Absolute turning point:** The absolute turning point is defined as the point from which on the score of the dynamic-scripting Game AI exceeds the score of the static Game AI for the remainder of the learning run.

In Figure 6 two typical learning runs can be seen.

One is using the original unit values and one is using learned unit values. It can be observed, when using the learned unit values, that the dynamic-scripting Game AI's score exceeds the static-scripted Game AI's score after the 14<sup>th</sup> game is played. Thus, the absolute turning point in this case is 14. When using original unit values it can be observed the dynamic-scripting Game AI is unable to develop a winning strategy against the static-scripted Game AI. Therefore there is no absolute turning point defined in that case.

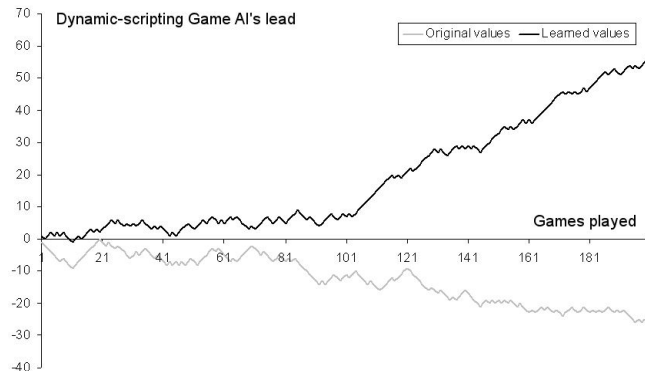


Figure 6: Two typical learning runs of which one is using original unit values and one is using learned unit values. The latter has an absolute turning point at 14.

**Relative turning point:** The relative turning point is the last game of the first sliding window of twenty points, in which the dynamic-scripting Game AI wins at least fifteen games. At this relative turning point the dynamic-scripting Game AI is more successful than the static-scripted Game AI with a reliability greater than 97% [3].

In Figure 7 the number of wins of the dynamic-scripting Game AI is displayed using a sliding window of size twenty. Two typical learning runs are displayed, one using the original unit values and one using the newly-learned unit values. It can be observed the dynamic-scripting Game AI using the newly-learned values reaches the point of winning fifteen out of twenty games for the first time after the 111<sup>th</sup> game. Thus, the relative turning point for this learning run is 111. When using the original weight values, it can be observed the dynamic-scripting Game AI does not reach the point at which it has won fifteen out of twenty games. Therefore, no relative turning point has been reached.

Table 3 lists the results of the experiment. In the table the (1) absolute turning point and (2) relative turning point are listed for the ten learning runs using the original unit values and for the ten learning runs using the

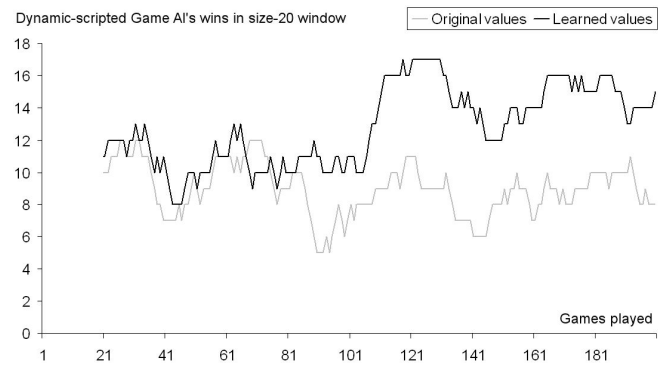


Figure 7: Two typical learning runs of which one is using the original unit values and one is using the newly-learned unit values. The latter has a relative turning point at 111.

newly-learned unit values.

	original unit values		learned unit values	
	ATP	RTP	ATP	RTP
1	> 200	> 200	114	124
2	36	108	198	> 200
3	> 200	> 200	> 200	> 200
4	> 200	> 200	14	111
5	> 200	> 200	158	> 200
6	> 200	> 200	116	107
7	> 200	> 200	68	39
8	> 200	> 200	> 200	> 200
9	> 200	> 200	20	37
10	20	37	> 200	> 200

Table 3: Results of the learning performance comparison experiment. Listed are the absolute turning point (ATP) and the relative turning point (RTP) for both the learning runs performed with the original unit values and the learning runs performed using the learned unit values.

As can be seen in Table 3 the dynamic-scripting Game AI using the unit values learned by TD-learning outperforms the dynamic-scripting Game AI using the original unit values. When using the original weights only two out of the ten learning runs were able reach the absolute turning point and relative turning point within 200 games. Using the learned unit values seven out of ten learning runs reached the absolute turning point and five out of ten reached the relative turning point, both within 200 games. The reason for the fact that not all learning runs using the learned unit values reach both turning points is the diversification of the dynamic-scripting Game AI which is necessary for learning.

It can be concluded that the learning of the dynamic-scripting Game AI in Wargus by using the learned unit values is significantly better than the learning by using

the original values.

## 5 Discussion

In this section several issues will be discussed. In Section 5.1 the fitness function will be discussed. Next, in Section 5.2 the validity of the learned unit values will be discussed.

### 5.1 Fitness Function

For adaptive Game AI the fitness function is important in that it provides the learning mechanism information on the quality of certain actions. Thus, having an inferior fitness function harms the performance of the Adaptive Game AI.

In this research only unit values have been investigated. Perhaps the effectiveness of the Adaptive Game AI could be further improved by researching the effects on the learning performance of including other parameters in the fitness function, e.g. the deployment of attack moves, ambushes, retreats, defense, etcetera.

### 5.2 Unit Value Validity

As stated in Section 4.2 the learning performance of dynamic-scripting Game AI in Wargus can be greatly improved by using unit values determined by TD-learning.

We wish to point out, that the unit values determined in this research might not be correct for all Wargus games. They might have been overfitted for the current static Game AI and current map. However, TD-learning can also be used to automatically determine correct unit values in other situations.

## 6 Conclusions and Future Research

In order to use a learning method in a computer game to improve the performance of computer controlled entities, a fitness function is required. The learning mechanism of Wargus as implemented by Ponsen *et al.* uses unit values to determine the fitness of a player. However, the original unit values were manually set by the developers of the game and no justification was available for these unit values being a correct representation of the fitness of a player. Therefore, in the introduction, the problem statement has been presented as to what extent the learning performance of the dynamic-scripting Game AI could be improved by learning new unit values using the method of TD-learning.

In order to learn new unit values a gamebase containing information on 299 games of Wargus was created. The information extracted included the number of units of every type both players had. This was done forcing restrictions on the static-scripted Game AI opponent, the map and the maximum state.

Using the gamebase which was created new unit values were determined by the method of TD-learning, which is an incremental prediction learning method that uses differences between temporally successive predictions. It was explained how the method of TD-learning works and how the status of a game could be turned into a prediction probability at every moment of the game. The most important observation of the learned unit values is the value of the ‘Peasant’, which is the worker unit. This unit proves to be a good indication of success.

Next was tested whether the newly-learned unit values would improve the learning performance of the dynamic-scripting Game AI. Twenty learning runs were performed of which ten using the original unit values and ten using the newly-learned unit values. The results of this experiment show that using newly-learned unit values does significantly improve the learning performance of the dynamic-scripting Game AI: where only two out of ten learning runs were able to reach both performance measures using the original unit values, five to seven out of ten learning runs were able to reach the performance measures using the newly-learned unit values.

As for future research several issues can be addressed: in Section 5.1 issues on the fitness function have been discussed. Future research in this area could be trying to extract higher level information on positions and the environment itself. This would perhaps lead to more intelligent behaviour. In Section 5.2 several issues on the validity of the learned unit values have been discussed. With respect to this topic future research could be to implement the TD-learning mechanism in the dynamic-scripting Game AI, for which TD-learning is exceedingly suitable since it is incremental. Doing so the Game AI will gradually learn correct unit values in every possible situation of Wargus.

## References

- [1] Bakkes, S., Postma, E., and Spronck, P. (2004). TEAM: The team-oriented evolutionary adaptability mechanism. *Entertainment Computing - ICEC 2004*, Vol. 3166 of Lecture Notes in Computer Science, pp. 273–282.
- [2] Beal, D.F. and Smith, M.C. (1997). Learning piece values using temporal differences. *International Computer Chess Association*, Vol. 20, No. 3, pp. 147–151.
- [3] Cohen, P.R. (1995). *Empirical Methods for Artificial Intelligence*. MIT Press, Cambridge, MA.
- [4] Ponsen, M., Spronck, P., Muñoz-Avilla, H., and Aha, D.W. (2005). Knowledge acquisition for adaptive game ai.

- [5] Spronck, P., Sprinkhuizen-Kuyper, I., and Postma, E. (2004). Online adaptation of game opponent ai with dynamic scripting. *International Journal of Intelligent Games and Simulation*, Vol. 3, No. 1, pp. 45–53.
- [6] Sutton, R.S. and Barto, A.G. (1998). *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, MA.
- [7] Sutton, R.S. (1988). Learning to predict by the methods of temporal differences. *Machine Learning*, Vol. 3, pp. 9–44.
- [8] The Mathworks (2005). MATLAB. <http://www.mathworks.com/products/matlab/>.
- [9] The Wargus Team (2005). Wargus. <http://wargus.sourceforge.net/>.