Predicting Success in an Imperfect-Information Game

Sander Bakkes, Pieter Spronck, Jaap van den Herik, and Philip Kerbusch

Universiteit Maastricht / MICC-IKAT
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
{s.bakkes,p.spronck,herik}@micc.unimaas.nl, p.kerbusch@student.unimaas.nl

Abstract. One of the most challenging tasks when creating an adaptation mechanism is to transform domain knowledge into an evaluation function that adequately measures the quality of the generated solutions. The high complexity of modern video games makes the task to generate a suitable evaluation function for adaptive game AI even more difficult. Still, our aim is to fully automatically generate an evaluation function for adaptive game AI. This paper describes our approach, and discusses the experiments performed in the RTS game Spring. TD-learning is applied for establishing a unit-based evaluation term. In addition, we define a term that evaluates tactical positions. From our results we may conclude that an evaluation function based on the defined terms is able to predict the outcome of a Spring game reasonably well. That is, for a unit-based evaluation the evaluation function is correct in about 76% of all games played, and when evaluating tactical positions it is correct in about 97% of all games played. A straightforward combination of the two terms did not produce improved results.

1 Introduction

Modern video games present a complex and realistic environment in which game AI is expected to behave realistically (i.e., 'human-like'). One feature of human-like behaviour of game AI, namely the ability to adapt adequately to changing circumstances, has been explored with some success in previous research [3, 4, 6]. This is called 'adaptive game AI'. When implementing adaptive game AI, arguably the most important factor is the evaluation function that rates the quality of newly generated game AI. An erroneous or suboptimal evaluation function will slow down the learning process, and may even result in weak game AI. Due to the complex nature of modern video games, the determination of a suitable evaluation function is often a difficult task. This paper discusses our work on fully automatically generating a good evaluation function for a real-time strategy (RTS) game.

The outline of the paper is as follows. In section 2, we discuss two approaches of creating adaptive game AI for RTS games. Section 3 describes how we collect domain knowledge of an RTS game in a data store. How we establish an evaluation function for RTS games automatically, based on the collected data, is discussed in section 4. In section 5, we test the performance of the generated evaluation function and provide the experimental results. We discuss the results in section 6, and in section 7 provide conclusions and describe future work.

2 Approaches

A first approach to adaptive game AI is by incrementally changing game AI to make it more effective. The speed of learning depends on the learning rate. If the learning rate is low, learning will take a long time. If it is high, results will be unreliable. Therefore, an incremental approach is not very suitable for rapidly adapting to observations in a complex video game environment.

A second approach is by allowing computer-controlled players to imitate human players. This approach can be particularly successful in games that have access to the Internet and that store and retrieve samples of gameplay experiences [5]. For this approach to be feasible, a central data store of gameplay samples must be created. Game AI can utilise this data store for two purposes: (1) to establish an evaluation function for games, and (2) to be used as a model by an adaptation mechanism. We follow the second approach in our research.

Our experimental focus is on RTS games, i.e., simulated war games. Here, a player needs to gather resources for the construction of units and buildings. The goal of the game is to defeat an enemy army in a real-time battle. We use RTS games for their highly challenging nature, which stems from three factors: (1) their high complexity, (2) the large amount of inherent uncertainty, and (3) the need for rapid decision making [2]. In the present research, we use the open-source RTS game Spring. A Spring game is won by the player who first destroys the opponent's 'Commander' unit.

3 Data Store of Gameplay Experiences

We define a gameplay experience as a set of observable features of the environment at a certain point in time. To create a data store of gameplay experiences for the Spring environment, we start by defining a basic set of features that will play an essential role in the game. For our first experiments, we decided to use the following five features.

1. Number of units observed (maximum 5000) of each type (over 200),
2. Number of enemy units within a 2000u radius of the Commander,
3. Number of enemy units within a 1000u radius of the Commander,
4. Number of enemy units within a 500u radius of the Commander.
5. Percentage of the environment visible to the friendly player.

Spring implements a so-called 'Line of Sight' visibility mechanism to each unit. This implies that game AI only has access to feature data of those parts of the environment that are visible to its own units (illustrated in Figure 1). When the game AI's information is restricted to what its own units can observe, we call this an *imperfect-information environment*. When we allow the

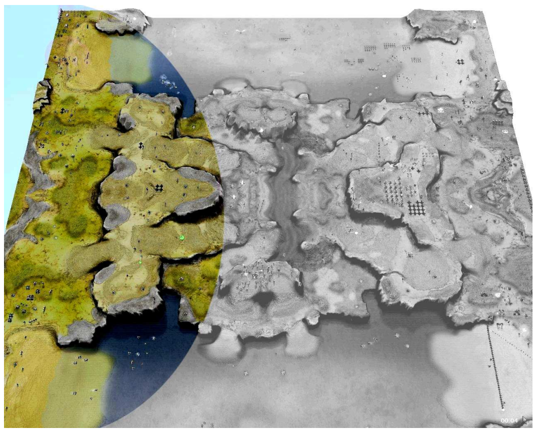


Fig. 1. Observation of information in a gameplay environment. Units controlled by the game AI are currently residing in the dark region. In an imperfect-information environment, only information in the dark region is available. In a perfect-information environment, information for both the dark and the light region is available.

game AI to access all information, regardless whether it is visible to its own units or not, we call this a *perfect-information environment*. We assume that the reliability of an evaluation function is highest when perfect information is used to generate it. For our experiments a data store was generated consisting of three different data sets: the first containing training data collected in a perfect-information environment, the second containing test data collected in a perfect-information environment, and the third containing test data collected in an imperfect-information environment.

4 Evaluation Function for Spring Games

This section discusses the automatic generation of an evaluation function for Spring. First, we discuss the design of an evaluation function based on weighted unit counts and tactical positions. Second, we discuss the application of temporal-difference (TD) learning [7] using a perfect-information data store to determine appropriate unit-type weights. Third, we discuss how the established evaluation function can be enhanced to enable it to deal with imperfect information.

4.1 Design of the Evaluation Function

The five defined features are selected as the basis of our evaluation function. Accordingly, the reference evaluation function for the game's status is denoted by

$$v = pv_1 + (1 - p)v_2 \tag{1}$$

where $p \in [0...1]$ is a free parameter to determine the weight of each term of the evaluation function, the term v_1 represents the 'number of units observed of each type', and the term v_2 represents the 'number of enemy units within a certain radius of the Commander'. The term v_1 is denoted by,

$$v_1 = \sum_u w_u (c_{u_0} - c_{u_1}) \tag{2}$$

where w_u is the experimentally determined weight of the unit u, c_{u_0} is the number of units of type u that the game AI has, c_{u_1} is the number of units of type u that its opponent has. The term v_2 is denoted by:

$$v_2 = \sum_{r \in d} w_r (c_{r_1} - c_{r_0}) \tag{3}$$

where w_r is the experimentally determined weight of the radius r, c_{r_1} is the number of units the opponent has within a radius r of the game AI's Commander, c_{r_0} is the number of units the game AI has within a radius r of the opponent's Commander, and d is the set of experimentally determined radii $[500, 1000, 2000]$.

The evaluation function v can only produce an adequate result in a perfect-information environment, because in an imperfect-information environment c_{u_1}, c_{r_0} and c_{r_1} are unknown.

4.2 Learning Unit-type Weights

We used TD-learning to establish appropriate values w_u for all unit types u. TD-learning has been applied successfully to games, e.g., by Tesauro [8] for creating a strong AI for backgammon. Our application of TD-learning to generate an evaluation function for SPRING is similar to its application by Beal and Smith [1] for determining piece values in chess.

Given a set W of weights $w_i, i \in \mathbb{N}, i \leq n$, and successive predictions P_t, the weights are updated as follows [1]:

$$\Delta W_t = \alpha (P_{t+1} - P_t) \sum_{k=1}^{t} \lambda^{t-k} \nabla_W P_k \tag{4}$$

where α is the learning rate and λ 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 wP_k.

To apply TD-learning, a series of successive prediction probabilities (in this case: the probability of a player winning the game) must be available. The prediction probability of a game's status v_1, $P(v_1)$ is defined as

$$P(v_1) = \frac{1}{1 + e^{v_1}} \tag{5}$$

where v_1 is the evaluation function of the game's current status, denoted in Equation 2 (this entails that learning must be applied in a perfect-information environment). Figure 2 illustrates how a game state value v_1 of 0.942 is transformed into a prediction probability $P(v_1)$ of 0.72.

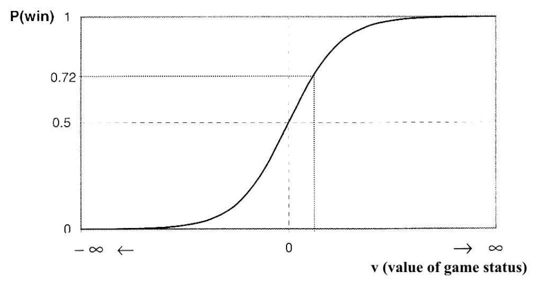


Fig. 2. Conversion from game status to prediction probability [1].

4.3 Dealing with Imperfect Information

In an imperfect-information environment, the evaluation value of the function denoted in Equation 1, which has been learned in a perfect-information environment, will typically be an overestimation. To deal with the imperfect information inherent in the SPRING environment, we assume that it is possible to map reliably the imperfect feature-data to a prediction of the perfect feature-data. A straightforward enhancement of the function to this effect is to scale linearly the number of observed units to the non-observed region of the environment. Accordingly, the approximating evaluation function v' for an imperfect-information environment is as in Equation 1, with the term v'_1 denoted by

$$v'_1 = \sum_u w_u(c_{u_0} - \frac{o_{u_1}}{R}) \tag{6}$$

where w_u, c_{u_0} are as in Equation 2, o_{u_1} is the *observed* number of opponent's units of type u, $R \in [0,1]$ is the fraction of the environment that is visible to the game AI. The term v'_2 is denoted by

$$v'_2 = \sum_{r \in d} w_r(\frac{o_{r_1}}{R_{r_1}} - \frac{o_{r_0}}{R_{r_0}}) \tag{7}$$

where w_r, r, d, and p are as in Equation 3, o_{r_1} is the *observed* number of units of the game AI within a radius r of the opponent's Commander, $R_{r_1} \in [0,1]$ is the fraction of the environment that is visible to the opponent within the radius r, o_{r_0} is the *observed* number of units of the opponent within a radius r of the game AI's Commander, and $R_{r_0} \in [0,1]$ is the fraction of the environment that is visible to the game AI within the radius r.

If enemy units are homogenously distributed over the environment, the approximating evaluation function applied to an imperfect-information environment will produce results close to those of the reference evaluation function in a perfect-information environment.

FRIENDLY TEAM	ENEMY TEAM	#GAMES IN TRAINING SET (Collected with perfect information)	#GAMES IN TEST SET (Collected with perfect information)	#GAMES IN TEST SET (Collected with imperfect information)
AAI	AAI (self-play)	500	200	200
AAI	TSI	100	200	200
AAI	CSAI	100	200	200
AAI	RAI	-	200	200

Table 1. The number of SPRING games collected in the data store.

5 Experiments

This section discusses experiments that test our approach. We first describe the experimental setup and the performance evaluation, and then the experimental results.

5.1 Experimental Setup

To test our approach we start collecting feature data in the data store. For each player, feature data was gathered during gameplay. In our experiments we gathered data of games where two game AIs are posed against each other. Multiple SPRING game AIs are available. We found one game AI which was open source, which we labelled 'AAI'. We enhanced this game AI with the ability to collect feature data in a data store, and the ability to disregard the line-of-sight visibility mechanism so that perfect information on the environment was available. As opposing game AIs, we used AAI itself, as well as three other AIs, namely 'TSI', 'CSAI', and 'RAI'. Table 1 lists the number of games from which we built the data store. The data collection process was as follows. During each game, feature data was collected every 127 game cycles, which corresponds to the update frequency of AAI. With 30 game cycles per second this resulted in feature data being collected every 4.233 seconds. The games were played on the map 'SmallDivide', which is a symmetrical map without water areas. All games were played under identical starting conditions.

We used a MatLab implementation of TD-learning to learn the unit-type weights w_u for the evaluation function of Equation 2. Unit-type weights were learned from feature data collected with perfect information from 700 games stored in the training set, namely all the games AAI played against itself (500), TSI (100), and CSAI (100). We did not include feature data collected in games where AAI was pitted against RAI, because we wanted to use RAI to test the generalisation ability of the learned evaluation function. The parameter α was set to 0.1, and the parameter λ was set to 0.95. Both parameter values were chosen in accordance with the research of Beal and Smith [1]. The unit-type weights were initialised to 1.0 before learning started.

Weights of the radii defined in the set d were chosen by the experimenter as 0.05, 0.20 and 0.75 for a radius of 2000, 1000 and 500, respectively. The parameter p was set to 1 (unit-based evaluation), 0 (evaluation of tactical positions) and 0.5 (a straightforward linear combination of the two terms).

5.2 Performance Evaluation

To evaluate the performance of the learned evaluation functions, we determined to what extent it is capable of predicting the actual outcome of a Spring game. For this purpose we defined the measure *absolute prediction* as the percentage of games of which the outcome is correctly predicted just before the end of the game. A high absolute prediction indicates that the evaluation function has the ability to evaluate correctly a game's status.

It is imperative that an evaluation function can evaluate a game's status correctly at the end of the game, and desirable throughout the play of a game. We defined the measure *weak relative prediction* as the game time at which the outcome of all tested games is predicted correctly at least 50% of the time. We defined the measure *normal relative prediction* and *strong relative prediction* as the game time at which the outcome of all tested games is predicted correctly at least 60% and 70% of the time, respectively. Since not all games last for an equal amount of time, we scaled the game time to 100% by averaging over the predictions made for each data point. A low relative prediction indicates that the evaluation function can correctly evaluate a game's status early in the game.

We determined the performance using two test sets. One test set contains feature data collected in a perfect-information environment, the other feature data collected in an imperfect-information environment. Feature data is collected of 800 games, where AAI is posed against itself (200), TSI (200), CSAI (200), and RAI (200).

We first tested the reference evaluation function in a perfect-information environment. Subsequently, we tested the approximating evaluation function in an imperfect-information environment.

5.3 Results of Learning Unit-type Weights

The Spring environment supports over 200 different unit types. During feature collection, we found that in the games played 89 different unit types were used. The TD-learning algorithm therefore learned weights for these 89 unit types. A summary of the results is listed in Table 2. Below, we give three observations on these results.

First, we observed that the highest weight has been assigned to the Advanced Metal Extractor. At first glance this seems surprising since it is not directly involved in combat situation. However, at the time the game AI destroys an Advanced Metal Extractor not only the opponent's ability to gather resources decreases, but it is also likely that the game AI has already penetrated its opponent's defences, since this unit typically is well protected and resides close to

UNIT-TYPE	WEIGHT
Advanced Metal Extractor *(Building)*	5.91
Thunder Bomber *(Aircraft)*	5.57
Metal Storage *(Building)*	4.28
Freedom Fighter *(Aircraft)*	4.23
Medium Assault Tank *(GroundUnit)*	4.18
...	...
Minelayer/Minesweeper with Anti-Mine Rocket *(GroundUnit)*	-1.10
Arm Advanced Solar Collector *(Building)*	-1.28
Light Amphibious Tank *(GroundUnit)*	-1.52
Energy Storage *(Building)*	-1.70
Defender Anti-air Tower *(Building)*	-2.82

Table 2. Learned unit-type weights (summary).

the Commander. This implies that destroying an Advanced Metal Extractor is a good indicator of success.

Second, we observed that some unit types obtained weights less than zero. This indicates that these unit types are of little use to the game AI and actually are a waste of resources. For instance, the Light Amphibious Tank is predictably useless, since our test-map contains no water.

Third, when looking into the weights of the unit types directly involved in combat, the Medium Assault Tank, Thunder Bomber and Freedom Fighter seem to be the most valuable.

5.4 Absolute-Prediction Performance Results

Using the learned unit-type weights, we determined the absolute-prediction performance of the evaluation functions. Table 3 lists the results for the trials where AAI was pitted against each of its four opponent AIs.

For the reference evaluation function in a perfect-information environment the average absolute-prediction performance is 76% for a unit-based evaluation ($p = 1$), 97% for an evaluation of tactical positions ($p = 0$), and 71% for a combined evaluation ($p = 0.5$). As the obtained absolute-prediction performances consistently predict more than 50% correctly, we may conclude that the reference evaluation function provides an effective basis for evaluating a game's status. Additionally, we observe that in two of the four trials the absolute-prediction performance of the combined evaluation is higher than that of the unit-based evaluation. However, on average the absolute-prediction performance of the combined evaluation is lower than that of the unit-based evaluation. These results imply that a combined evaluation of the defined terms has potential, yet it is currently in need of fine-tuning.

For the approximating evaluation function in an imperfect-information environment, the average absolute-prediction performance is 73% for a unit-based evaluation ($p = 1$), 92% for an evaluation of tactical positions ($p = 0$), and

AAI-AAI	ABS. PREDICTION PERFORMANCE	AAI-CSAI	ABS. PREDICTION PERFORMANCE
p=1	82% | 79%	p=1	85% | 87%
p=0	99% | 92%	p=0	98% | 95%
p=0.5	90% | 86%	p=0.5	86% | 87%
AAI-TSI		**AAI-RAI**	
p=1	68% | 57%	p=1	70% | 70%
p=0	96% | 92%	p=0	96% | 89%
p=0.5	56% | 50%	p=0.5	52% | 56%
AVERAGE			
p=1	76% | 73%		
p=0	97% | 92%		
p=0.5	71% | 70%		

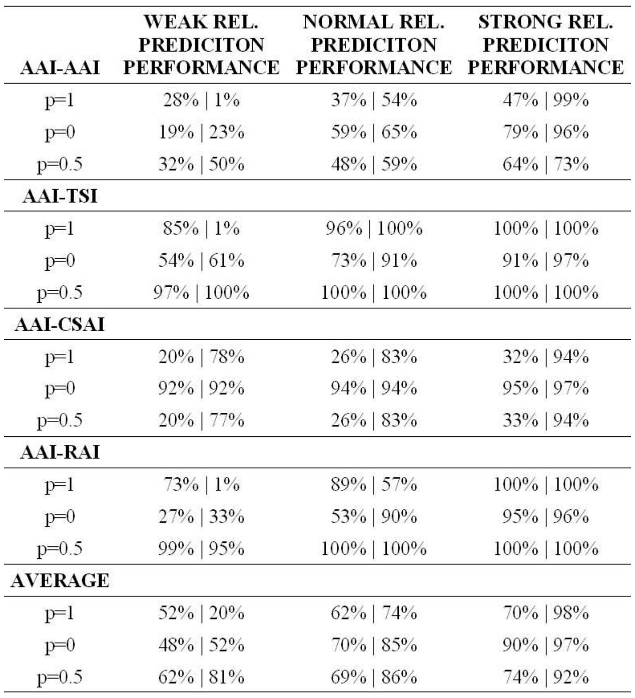
Table 3. Absolute-prediction performance results. Each cell contains trial results for, in sequence, (1) the reference evaluation function applied in a perfect-information environment, and (2) the approximating evaluation function applied in an imperfect-information environment.

70% for a combined evaluation ($p = 0.5$). From these results, we may conclude that the approximating evaluation function in an imperfect-information environment successfully obtained an absolute-prediction performance comparable to the performance of the reference evaluation function in a perfect-information environment.

5.5 Relative-Prediction Performance Results

In Table 4 the relative-prediction performance is listed. We observe that for all values of p the weak relative-prediction performance is on average 54% in a perfect-information environment, and 51% in an imperfect-information environment. Results of the normal and strong relative-prediction performance are more divergent. The normal relative-prediction performances are on average 67% and 81% in a perfect-information and imperfect-information environment, respectively. The strong relative-prediction performances are on average 78% and 96% in a perfect-information and imperfect-information environment, respectively. This indicates that the reference evaluation function in a perfect-information environment manages to predict the outcome of a game considerably earlier than the approximating evaluation function in an imperfect-information environment.

As an exception to the before-mentioned indication, we observe that for $p = 1$ the obtained weak relative-prediction performance is significantly better in an imperfect-information environment than in a perfect-information environment. This might seem strange at first glance, but the explanation is rather straightforward: early in the game, when no units of the opponent have been observed yet, the approximating evaluation function will always predict a victory for the friendly team. Even though this prediction is meaningless (since it is generated without any information on the opponent), against a weaker game AI it is very likely to be correct, while against a stronger AI, it is likely to be false.



AAI-AAI	WEAK REL. PREDICITON PERFORMANCE	NORMAL REL. PREDICITON PERFORMANCE	STRONG REL. PREDICITON PERFORMANCE
p=1	28% | 1%	37% | 54%	47% | 99%
p=0	19% | 23%	59% | 65%	79% | 96%
p=0.5	32% | 50%	48% | 59%	64% | 73%
AAI-TSI			
p=1	85% | 1%	96% | 100%	100% | 100%
p=0	54% | 61%	73% | 91%	91% | 97%
p=0.5	97% | 100%	100% | 100%	100% | 100%
AAI-CSAI			
p=1	20% | 78%	26% | 83%	32% | 94%
p=0	92% | 92%	94% | 94%	95% | 97%
p=0.5	20% | 77%	26% | 83%	33% | 94%
AAI-RAI			
p=1	73% | 1%	89% | 57%	100% | 100%
p=0	27% | 33%	53% | 90%	95% | 96%
p=0.5	99% | 95%	100% | 100%	100% | 100%
AVERAGE			
p=1	52% | 20%	62% | 74%	70% | 98%
p=0	48% | 52%	70% | 85%	90% | 97%
p=0.5	62% | 81%	69% | 86%	74% | 92%

Table 4. Relative-prediction performance results. Each cell contains trial results for, in sequence, (1) the reference evaluation function applied in a perfect-information environment, and (2) the approximating evaluation function applied in an imperfect-information environment.

In a perfect-information environment, the strong relative-prediction performance is 70% on average for ($p = 0$), 90% on average for ($p = 1$), and 74% on average for ($p = 0.5$). We observe that only when a game is nearly finished, ($p = 1$) can accurately predict the game's outcome correctly. As ($p = 1$) obtained an *absolute*-prediction performance of 97%, this result implies that the term that evaluates tactical positions is particularly effective in the final phase of playing the game, and less effective in earlier phases.

Figure 3 displays the percentage of outcomes correctly predicted as a function over time. The figure compares the predictions of the reference evaluation function in a perfect-information environment, with the predictions of the approximating evaluation function in an imperfect-information environment, for $p = 0.5$.

6 Discussion

When evaluating exclusively by means of the feature 'number of units observed of each type' ($p = 1$), the reference evaluation function obtained an average

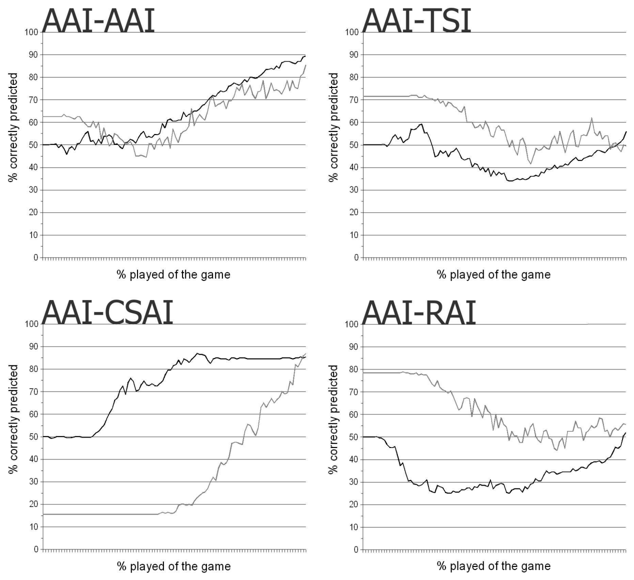


Fig. 3. Comparison of outcomes correctly predicted as a function over time. The black line represents the prediction performance of the reference evaluation function, the gray line that of the approximating evaluation function, for $p = 0.5$.

absolute-prediction performance of 76%. Thus, just before the game's end, it still wrongly predicts almost a quarter of the outcomes of the tested games. This is explained as follows. A SPRING game is not won by obtaining a territorial advantage, but by destroying the so-called Commander unit. Thus, even a player who has a material disadvantage may win if the enemy's Commander is taken out. Therefore, an evaluation function based purely on a comparison of material will not always be able to predict the outcome of a game correctly. Nevertheless, with a current average absolute-prediction performance of about 76%, and a strong relative-prediction performance of 70% on average, there is certainly room for improvement.

Evaluating tactical positions ($p = 0$) is likely to serve as such an improvement, as it obtained an average absolute-prediction performance of 97%. However, it also obtained a strong relative-prediction performance of 90% on average, which implies that it is not capable of correctly evaluating a game's status early in the game.

Straightforwardly combining of the two terms ($p = 0.5$) occasionally increased the absolute-prediction performance, but on average both the absolute- and relative-prediction performance deteriorated. We expect that including information on the phase of the game into the evaluation function will produce improved results. This enhancement will be the focus of future work.

7 Conclusions and Future Work

In this paper we discussed an approach to automatically generating an evaluation function for game AI in RTS games. From our experimental results, we may conclude that both the reference and approximating evaluation functions effectively predict the outcome of a Spring game expressed in terms of the absolute-prediction performance. The relative-prediction performance, which indicates how early in a game an outcome is predicted correctly, is lower (i.e., better) for the reference evaluation function than for the approximating evaluation function.

Two terms were defined to evaluate a game's status, namely a unit-based term and a term based on tactical positions. The term to evaluate tactical positions is capable of predicting the final outcome of the game almost perfectly. However, it only predicts accurately in the final phase of the game. The unit-based term, on the other hand, is only moderately accurate in predicting the final outcome of the game. However, it achieves a high prediction accuracy relatively early. This implies that the accuracy of outcome predictions is closely related to the phase of the game. Thus, to improve performance, the weights assigned to each term of the evaluation function should be made dependent on the phase of the game.

For future work, we will extend the evaluation functions with additional terms and will incorporate a mechanism to evaluate a game's status dependent on the phase of the game. Our findings will be incorporated in the design of an adaptation mechanism for RTS games.

References

1. Don F. Beal and Malcolm C. Smith. Learning piece values using temportal differences. *International Computer Chess Association (ICCA) Journal*, 20(3):147–151, 1997.
2. Michael Buro and Timothy M. Furtak. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*.
3. Pedro Demasi and Adriano J. de O. Cruz. Online coevolution for action games. *International Journal of Intelligent Games and Simulation*, 2(3):80–88, 2002.
4. Thore Graepel, Ralf Herbrich, and Julian Gold. Learning to fight. In Quasim Mehdi, Norman Gough, and David Al-Dabass, editors, *Proceedings of Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*.
5. Pieter H.M. Spronck. A model for reliable adaptive game intelligence. In David W. Aha, Hector Munoz-Avila, and Michael van Lent, editors, *Proceedings of the IJCAI-05 Workshop on Reasoning, Representation, and Learning in Computer Games*, pages 95–100, 2005.
6. Pieter H.M. Spronck, Ida G. Sprinkhuizen-Kuyper, and Eric O. Postma. Online adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):45–53, 2004.
7. Richard S. Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.
8. Gerald Tesauro. Practical issues in temporal difference learning. *Machine Learning*, 8:257–277, 1992.