Phase-dependent Evaluation in RTS Games

Sander Bakkes Pieter Spronck Jaap van den Herik

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

Abstract

Modern video games present an environment in which game AI is expected to behave realistically (i.e., 'human-like'). One feature of human-like behaviour of game AI, is the ability to assess accurately the current situation. This requires an appropriate evaluation function. The high complexity of modern video games makes the task to generate such an evaluation function for game AI a difficult one. However, we assume that many difficulties can be solved automatically. Therefore, our aim is to generate fully automatically an evaluation function for game AI. This paper describes our approach, and discusses the experiments performed in the real-time strategy (RTS) game SPRING. Two evaluation terms are established, one to perform a unit-based evaluation, the other to evaluate positional safety. In addition, a mechanism is incorporated to perform the evaluation dependent on the phase of the game. From our results we may conclude that the generated evaluation function effectively predicts the outcome of a SPRING game.

1 Introduction

When implementing AI for modern video games, arguably the most important factor is the evaluation function that rates the quality of newly generated game AI. Due to the complex nature of modern video games, generating a suitable evaluation function is often a difficult task. This paper discusses our work on fully automatically generating a suitable evaluation function for real-time strategy (RTS) games. We aim at utilising the evaluation function for adequately adapting game AI to changing circumstances, which is called 'adaptive game AI'. Adaptive game AI has been explored with some success in previous research [6, 7, 13]. Our approach to generating a suitable evaluation function for RTS games is to use a central data store of samples of gameplay experiences. This approach can be particularly successful in games that have access to the Internet and that store and retrieve gameplay samples [12].

The outline of the paper is as follows. Section 2 discusses how we collect domain knowledge of an RTS game in a data store. In Section 3, we show how we generate an evaluation function for RTS games automatically, based on the collected data. How information on the phase of the game is incorporated into the evaluation function is described 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 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. Gameplay experiences are gathered in an RTS game environment, i.e., a simulated war game. 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 [5]. In the present research we use SPRING [15], which is a typical and open-source RTS game. A SPRING game is won by the player who first destroys the opponent's 'Commander' unit. 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 2000 radius of the Commander,
3. Number of enemy units within a 1000 radius of the Commander,
4. Number of enemy units within a 500 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 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.

3 Evaluation Function for RTS Games

This section discusses the automatic generation of an evaluation function for SPRING. The five defined features are selected as the basis of our evaluation function. To allow the evaluation function to deal with imperfect information inherent in the SPRING environment, we assume that is it possible to map reliably the imperfect feature-data to a prediction of the perfect feature-data. Our straightforward implementation of this mapping, is to scale linearly the number of observed opponent's units to the non-observed region of the environment. If opponent's units are homogenously distributed over the environment, the evaluation function applied to an imperfect-information environment will produce results close to those of the evaluation function in a perfect-information environment.

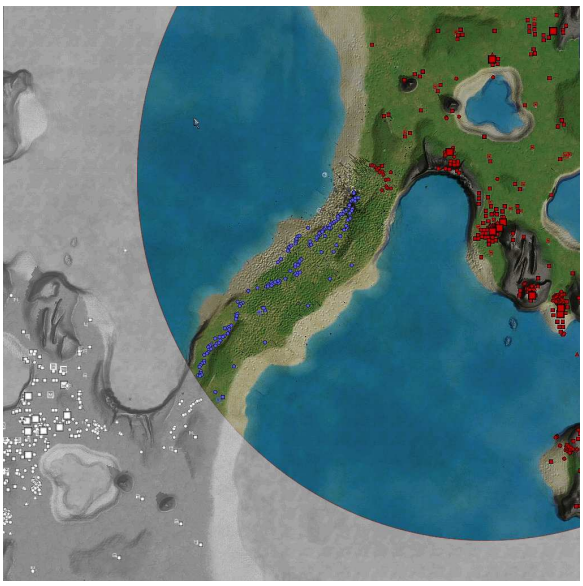

Figure 1: Observation of information in the SPRING game environment. Units controlled by the game AI are currently residing in the highlighted region. In an imperfect-information environment, only information in the highlighted region is available. In a perfect-information environment, information for both the highlighted and the light-grey region is available.

A general form of the evaluation function for the game's status is denoted by

$$v(p) = w_p v_1 + (1 - w_p) v_2 \tag{1}$$

where $w \in [-1 \dots 1]$ is a free parameter to determine the weight of each term v_n of the evaluation function, and $p \in \mathbb{N}$ is a parameter that represents the current *phase of the game*. In our experiments we use two evaluative terms, the term v_1 that represents the material strength and the term v_2 that represents the positional safety.

3.1 Material Strength

Below we discuss the evaluative term v_1, that represents the material strength. The term utilises data of feature number 1 and 5 that is collected in the data store. Term v_1 is denoted by

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

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, o_{u_1} is the *observed* number of opponent's units of type u, and $R \in [0, 1]$ is the fraction of the environment that is visible to the game AI.

3.2 Positional Safety

Below we discuss the evaluative term v_2, that represents the safety of the current tactical position. The term utilises data of feature number 2, 3, 4 and 5 that is collected in the data store. Term v_2 is denoted by

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

where w_r is the experimentally determined weight of the radius r, 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', $R_{r_0} \in [0, 1]$ is the fraction of the environment that is visible to the game AI within the radius r, and d is the set of experimentally determined radii $\{500, 1000, 2000\}$.

4 Phase of the Game

The phase of a strategy game can be straightforwardly derived from the observed traversal in the game's 'tech tree'. In strategy games, a tech tree is a directed acyclic graph that represents the possible paths a player can take. The player starts in the root of the tech tree, and by traversing the tech tree will be provided with advanced levels of technology. Traversing the tech tree typically is advantageous, yet there is a cost of time and game resources. The tech tree of the SPRING game is provided in Figure 2. In the SPRING game, three levels of technology are available. At the start of the game a player can only construct so-called level 1 structures and level 1 units. Traversing the tech tree, advanced structures and units of level 2 and level 3 will become available to the player.

Previous research performed in the SPRING environment showed that the accuracy of outcome predictions is closely related to the phase of the game [1, 2]. To distinguish game phases in the SPRING game, we analogously map existing tech levels to game phases. Additionally, we also regard the transition from one tech level to another as a game phase. For the SPRING game, this leads us to define the following five game phases:

Phase 1 Level 1 units observed that have a buildtime < 2500 or Level 1 structures observed[1]

Phase 2 Level 1 units observed that have a buildtime ≥ 2500.

Phase 3 Level 2 units observed that have a buildtime < 15000, or Level 2 structures observed

Phase 4 Level 2 units observed that have a buildtime ≥ 15000.

Phase 5 Level 3 units or Level 3 structures observed.

[1] By exception, we consider so-called 'Construction Units' to have a relatively short buildtime. In reality, Construction Units have an unusually long buildtime, which would incorrectly imply that these units are not built early in the game.

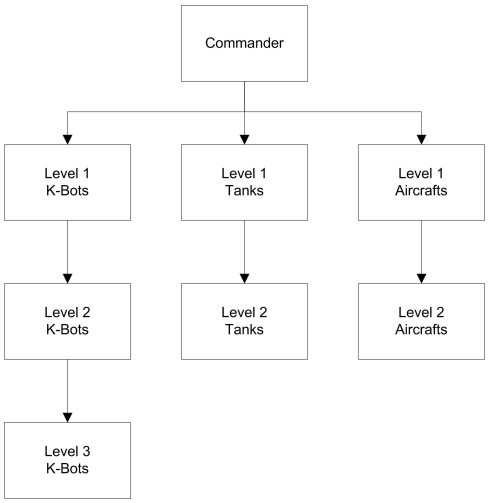

Figure 2: Tech tree of the SPRING game.

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' [10]. We enhanced this game AI with the ability to collect feature data in a data store, and the ability to disregard radar visibility 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' [3], 'CSAI' [8], and 'RAI' [9]. 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.

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.

We used a MatLab implementation of TD-learning [14] to learn the unit-type weights w_u for the evaluation function of Equation 2. Our application of TD-learning to generate an evaluation function for SPRING is analogous to its application by Beal and Smith [4] for determining piece values in chess. 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 learning rate α was set to 0.1, and the recency parameter λ was set to 0.95. Both parameter values were chosen in accordance with the research of Beal and Smith [4]. 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 for a radius of 2000, and 0.20 and 0.75 for a radius of 1000 and 500, respectively.

Term-weights for each phase of the game were learned with the gradient descent optimisation algorithm [11]. A step value of 0.01 was used initially to allow the algorithm to explore the state space using random jumps. Slowly, the step value was decreased to 0.001, to encourage the algorithm to perform a local optimisation at the end. Term weights for each phase were initialised to 0.5 before learning started.

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 'final prediction accuracy' as the percentage of games of which the outcome is correctly predicted just before the end of the game. A high final prediction accuracy 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 turning point' as the time at which at least 50% of the outcomes of the test games is predicted correctly. We defined the measure 'normal turning point' and 'strong turning point' as the time at which at least 60% and 70% of the outcomes of the test games is predicted correctly, 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 turning point 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 evaluation function in a perfect-information environment. Subsequently, we tested the evaluation function in an imperfect-information environment.

Optimising Term Weights Results

Table 2 displays the resulting weights for each term of the evaluation function. The term weights have been learned dependent on the phase of the game. We observe that when evaluating by use of the learned weights, the evaluation function's ability to predict the outcome of the game increases. The most significant increase in prediction performance is observed in the phases the game resides in the most (i.e., phase one and phase three).

Phase (p)	Gameplay Samples	Initial Weights	Outcomes Correctly Predicted w. Initial Weights	Learned Weights	Outcomes Correctly Predicted w. Learned Weights	Performance Increase (%)
1	96,966	0.5	30,649	-0.370	38,451	25.5%
2	74,570	0.5	43,150	0.017	44,048	2.1%
3	97,892	0.5	43,441	-0.001	64,495	48.5%
4	46,535	0.5	35,464	0.079	36,139	1.9%
5	14,784	0.5	12,365	0.041	12,401	0.3%

Table 2: Learned term weights for each of the five defined phases.

Final Prediction Accuracy Results

Table 3 lists the final prediction accuracy for the trials where AAI was pitted against each of its four opponent AIs. When evaluating by use of the learned weights, instead of the initial weights, we observe a significant improvement in the final prediction accuracy. For the evaluation function in a perfect-information

environment the final prediction accuracy is 97% on average. For the evaluation function in an imperfect-information environment, the final prediction accuracy is 90% on average. From these results, we may conclude that the established evaluation function provides an effective basis for evaluating a game's status.

AAI-AAI	FINAL PREDICTION ACCURACY	AAI-CSAI	FINAL PREDICTION ACCURACY
learned weights	99% | 92%	learned weights	98% | 93%
initial weights	90% | 86%	initial weights	86% | 87%
AAI-TSI		**AAI-RAI**	
learned weights	96% | 88%	learned weights	95% | 87%
initial weights	56% | 50%	initial weights	52% | 56%

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

Turning Point Results

In Table 4 the obtained turning points are listed. We observe that when evaluating by use of the initial weights, the weak turning point is on average 54% in a perfect-information environment, and 51% in an imperfect-information environment. Results of the normal and strong turning points are more divergent. The normal turning points are on average 67% and 81% in a perfect-information and imperfect-information environment, respectively. The strong turning points are on average 78% and 96% in a perfect-information and imperfect-information environment, respectively. This indicates that, as expected, the evaluation function in a perfect-information environment manages to predict the outcome of a game considerably earlier than the evaluation function in an imperfect-information environment.

When evaluating in a perfect-information environment by use of the learned weights, we observe that for the AAI-AAI, AAI-TSI and AAI-RAI trials, the turning points improve significantly or remain comparable

AAI-AAI	WEAK TURNING POINT	NORMAL TURNING POINT	STRONG TURNING POINT
learned weights	39% | 18%	50% | 59%	64% | 64%
initial weights	32% | 50%	48% | 59%	64% | 73%
AAI-TSI			
learned weights	67% | 57%	74% | 87%	87% | 98%
initial weights	97% | 100%	100% | 100%	100% | 100%
AAI-CSAI			
learned weights	85% | 1%	88% | 1%	92% | 86%
initial weights	20% | 77%	26% | 83%	33% | 94%
AAI-RAI			
learned weights	1% | 48%	72% | 85%	89% | 94%
initial weights	99% | 95%	100% | 100%	100% | 100%

Table 4: Turning points. Each cell contains trial results for, in sequence, (1) the evaluation function applied in a perfect-information environment, and (2) the evaluation function applied in an imperfect-information environment.

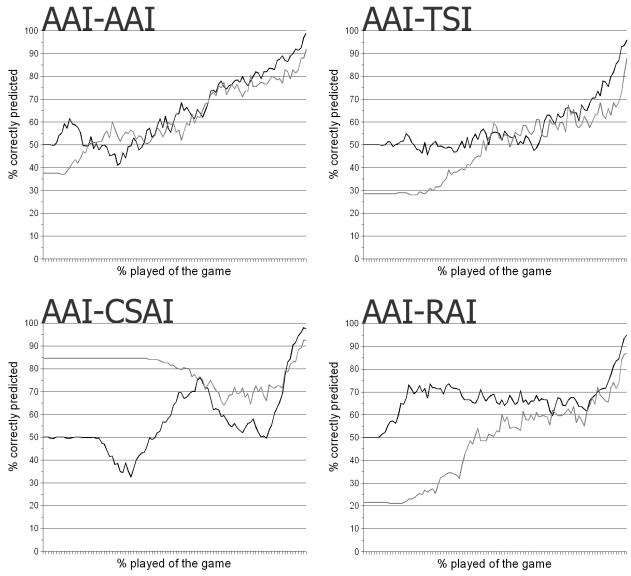

Figure 3: Comparison of outcomes correctly predicted as a function over time. The black line represents the prediction performance of the evaluation function in a perfect-information environment, the gray line that of the evaluation function in an imperfect-information environment.

to results obtained with initial weights. As an exception, turning points obtained in the AAI-CSAI trial show deteriorated results. The explanation for this phenomenon is rather straightforward: because games of AAI in competition against CSAI last for a very short time, relatively little training data of this particular trial is available for the process of optimising term weights. Due to this bias in the training set, optimised term weights will not be fit for use in the AAI-CSAI trial. From results obtained in the other three trials, we may conclude that the established evaluation function is capable of accurately evaluating the game's status relatively early in the game.

Figure 3 displays the percentage of outcomes correctly predicted as a function over time. The figure compares the predictions of the evaluation function in a perfect-information environment, with the predictions of the evaluation function in an imperfect-information environment. Trial results reveal that when more environment information becomes available in an imperfect-information environment, the established evaluation function obtains a prediction performance comparable to that in a perfect-information environment.

6 Discussion

In the AAI-AAI trial we observe a relatively non-gradual prediction performance in the first phase of the game. A straightforward explanation of this phenomenon is that it results from optimising the term weights for each phase of the game. As for each phase of the game the optimisation process is focused on obtaining the highest overall prediction accuracy in that particular phase, an optimised, yet non-gradual prediction performance can be the result. Our recommendation for game developers who incorporate automatically generated score functions into their game environment, is to consider that in early phases of the game, outcome predictions will always be inaccurate. To obtain reliable predictions, optimisation of term weights should therefore be applied only to later phases of the game.

7 Conclusions and Future Work

In this paper we discussed an approach to automatically generate an evaluation function for game AI in RTS games. Two terms were defined to evaluate a game's status, namely a term based on material-strength, and a term based on positional safety. A mechanism was incorporated to evaluate the game status dependent on the phase of the game. Our experimental results show that just before the game's end, the established evaluation function is able to predict correctly the outcome of the game with an accuracy that approaches

100%. Additionally, the evaluation function predicts accurately before half of the game is played. From these results, we may conclude that the established evaluation function effectively predicts the outcome of a SPRING game.

For future work, we will incorporate our findings on evaluating a game's status into the design of an adaptation mechanism for RTS games. Our approach to automatically generate an evaluation function will be applied to other RTS games, and games in other genres.

Acknowledgements

The authors wish to thank Philip Kerbusch for his contribution to this research. Furthermore, we extend our gratitude to the anonymous reviewers for their insightful feedback.
This research was funded by a grant from the Netherlands Organization for Scientific Research (NWO grant No 612.066.406).

References

[1] Sander Bakkes, Philip Kerbusch, Pieter Spronck, and Jaap van den Herik. Automatically evaluating the status of an RTS game. In *Proceedings of the Belgian-Dutch Benelearn Conference 2007*, 2007.

[2] Sander Bakkes, Philip Kerbusch, Pieter Spronck, and Jaap van den Herik. Predicting success in an imperfect-information game. In *Proceedings of the Computer Games Workshop 2007*, 2007.

[3] Mateusz Baran and Michal Urbanczyk. AI:TSI. Creator of the game AI 'TSI', http://spring.clan-sy.com/wiki/AI:TSI, 2006.

[4] Don Beal and Malcolm Smith. Learning piece values using temportal differences. *International Computer Chess Association (ICCA) Journal*, 20(3):147–151, 1997.

[5] Michael Buro and Timothy Furtak. RTS games and real-time AI research. In *Proceedings of the Behavior Representation in Modeling and Simulation Conference (BRIMS)*, 2004.

[6] Pedro Demasi and Adriano Cruz. Online coevolution for action games. *International Journal of Intelligent Games and Simulation*, 2(3):80–88, 2002.

[7] 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)*, pages 193–200, 2004.

[8] Hugh Perkins. AI:CSAI. Creator of the game AI 'CSAI', http://spring.clan-sy.com/wiki/AI:CSAI, 2006.

[9] Reth. AI:RAI. Creator of the game AI 'RAI', http://spring.clan-sy.com/wiki/AI:RAI, 2007.

[10] Alexander Seizinger. AI:AAI. Creator of the game AI 'AAI', http://spring.clan-sy.com/wiki/AI:AAI, 2006.

[11] Jan Snyman. *Practical Mathematical Optimization: An Introduction to Basic Optimization Theory and Classical and New Gradient-Based Algorithms*. Springer Publishing, 2005.

[12] Pieter 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.

[13] Pieter Spronck, Ida Sprinkhuizen-Kuyper, and Eric Postma. Online adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games and Simulation*, 3(1):45–53, 2004.

[14] Richard Sutton. Learning to predict by the methods of temporal differences. *Machine Learning*, 3:9–44, 1988.

[15] SPRING. http://spring.clan-sy.com/.