Adaptive Spatial Reasoning for Turn-based Strategy Games

Maurice Bergsma and **Pieter Spronck**
Tilburg centre for Creative Computing
Tilburg University, The Netherlands
p.spronck@uvt.nl

Abstract

The quality of AI opponents often leaves a lot to be desired, which poses many attractive challenges for AI researchers. In this respect, *Turn-based Strategy* (TBS) games are of particular interest. These games are focussed on high-level decision making, rather than low-level behavioural actions. For efficiently designing a TBS AI, in this paper we propose a game AI architecture named ADAPTA (Allocation and Decomposition Architecture for Performing Tactical AI). It is based on task decomposition using asset allocation, and promotes the use of machine learning techniques. In our research we concentrated on one of the subtasks for the ADAPTA architecture, namely the Extermination module, which is responsible for combat behaviour. Our experiments show that ADAPTA can successfully learn to outperform static opponents. It is also capable of generating AIs which defeat a variety of static tactics simultaneously.

Introduction

The present research is concerned with artificial intelligence (AI) for turn-based strategy (TBS) games, such as CIVILIZATION and HEROES OF MIGHT AND MAGIC. The AI for TBS games offers many challenges, such as resource management, forward planning, and decision making under uncertainty, which a computer player must be able to master in order to provide competitive play. Our goal is to create an effective turn-based strategy computer player. To accomplish this, we employ learning techniques to generate the computer player's behaviour automatically.

As TBS games are in many ways related to real-time strategy (RTS) games, the research challenges of these domains are rather similar. Buro and Furtak (2003) define seven research challenges for RTS games, six of which are also relevant to the TBS domain. These are (1) Adversarial planning, (2) Decision making under uncertainty, (3) Spatial reasoning, (4) Resource management, (5) Collaboration, and (6) Adaptivity. While these six challenges are similar for both genres, due to the increased 'thinking time' available, TBS games may offer more depth in some of them.

The outline of this paper is as follows. First, we discuss related work and provide a definition of the TBS game used for this research. We explain the AI architecture designed for this game. Next, we describe how spatial reasoning is used in our AI, followed by an overview of the learning algorithm used to generate the behaviour of our AI automatically. We discuss the experimental setup and the results achieved. Finally, we provide conclusions.

Related Work

A significant portion of TBS game research employs the technique of Case Based Reasoning (CBR). Two examples are the work of Sánchez-Pelegrín, Gómez-Martín, and Díaz-Agudo (2005), who used CBR to develop an AI module for an open source CIVILIZATION clone, and the work of Obradović (2006), who focussed on learning high-level decisions to model complex strategies used by human players accurately. In order to achieve visibly intelligent behaviour, Bryant and Miikulainen (2007) use human-generated examples to guide a Lamarckian neuroevolution algorithm, which trained an agent to play a simple TBS game. Their experiments showed that their technique was able to approximate the example behaviour better than a backpropagation algorithm, but that backpropagation agents achieved better game performance. Ulam, Goel, and Jones (2004) built a knowledge-based game agent for a TBS game using model-based reflection and self-adaptation, combined with reinforcement learning. The agent performed the task of defending a city in FREECIV. Their results show that their approach performs very well and converges fast. Hinrichs and Forbus (2007) combined a symbolic Hierarchical Task Network (HTN) planning system with analogical learning and qualitative modelling, and applied it to a resource management task in FREECIV. They conclude that transfer learning, i.e., training the system on a (separate) data set, improves results, especially at the start of a game.

Game Definition

For this research, we have developed our own game definition, based on Nintendo's ADVANCE WARS, which is a relatively simple TBS game that still supports most of the major features of the genre. Each of the 4 X's (Exploration, Expansion, Exploitation, and Extermination) are represented in some form in the environment. We named our version SIMPLE WARS.

SIMPLE WARS takes place on a two-dimensional, tile-based map, shown in Figure 1. A tile is of a predefined type,

such as *Road*, *Mountain*, or *River*. Each type has its own set of parameters, which define the characteristics of the tile, such as the number of moves that it takes for a unit to move over it, or the defense bonus that a unit receives while standing on it. By default, movement in the game can occur either horizontally or vertically, but not diagonally. A defense bonus lowers the amount of damage that the unit receives in combat. Some tiles contain a base, for example a *City*, which provides a certain amount of resources to the player that controls the base, or a *Factory*, which can produce one unit each turn. Each unit requires a specific amount of resources to be built. A newly created unit is placed on the same tile as the *Factory* that produced it.

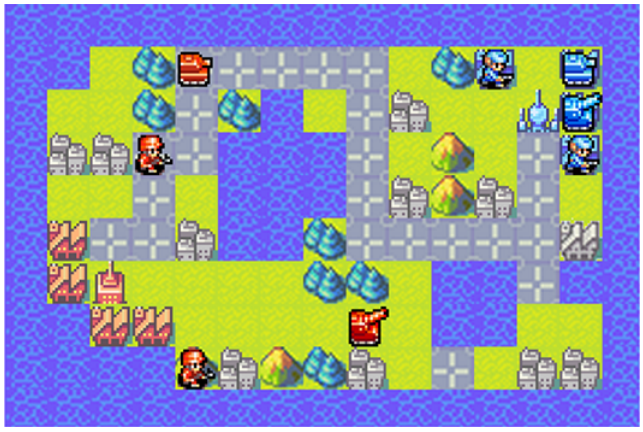

Figure 1: SIMPLE WARS.

As is the case with bases and tiles, units come in several types. Each type has different values for its parameters, which include (1) the starting amount of health points for the unit, (2) the number of moves it can make per turn, (3) the amount of damage it can do to each type of unit, and (4) the actions the unit can perform. Because a unit in the game actually represents a squad of units, and damaging the unit represents destroying a part of the squad, the effectiveness of actions such as *Attack* (attacking another unit) is tied to the number of health points that remains for the unit. Additionally, the effectiveness of an *Attack* depends on the types of units involved.

The three types of units in this game are *Infantry*, *Tank*, and *Anti-Tank* units. Of these, only the *Infantry* unit is capable of moving over *Mountain* tiles. None of these units is able to move over *Sea* tiles. The units follow a rock-paper-scissors approach, which means that each of the three types is, where doing damage to the other two unit types is concerned, stronger than one and weaker than the other (Infantry defeats Anti-Tank, Anti-Tank defeats Tank, and Tank defeats Infantry).

Every turn, a player can build a single unit at each *Factory* under control, and perform (1) a *Move* and (2) an *Attack* or other action for each unit under control. An *Attack* action can be performed without moving, but a unit cannot move after performing this action. A tile can only contain a single unit at any time. This implies that a *Factory* is unable to produce a new unit whenever another unit is located at the same tile as this *Factory*. Additionally, a moving unit is able to pass through a tile occupied by a friendly unit, but not through a tile occupied by an enemy unit.

The ADAPTA Architecture

In many commercial strategy games, the AI is implemented using scripts. In order to keep the size of the AI manageable, the complex task in the script can be decomposed into several subtasks, which operate independently of each other, and concentrate each on performing a specific part of the complex task, without taking the other subtasks into account. While the goals of the subtasks are independent, they all share the same environment, namely the game world. Moreover, they need to share the finite number of *assets* (resources and game objects) that are available. By nature, a subtask only sees a part of the big picture, and is not concerned with the overall path to victory. Therefore, separate modules are required to keep track of the AI's goals, to determine which goals (and therefore, which subtasks) have priority at a point in the game, and to allocate control over the available assets to the different subtasks. Because these modules keep track of the overall strategy, they are called the strategic modules. The subtasks are called tactical modules, as they are each responsible for one type of tactics. Combined, these modules make up the ADAPTA architecture, which is depicted in Figure 2. The ADAPTA approach is reminiscent of a goal-based RTS AI, which is used in some commercial games, such as KOHAN 2 (Dill 2006).

In the ADAPTA architecture, the strategic AI acts as an arbitrator between the different tactical modules. The Asset Allocation Module decides which tactical module gets control over which assets. This is achieved through auctioning. Tactical modules generate bids, which consist of one or more assets that a module wants to use, and a utility value that the module assigns to these assets (e.g., it assigns a high utility value to assets which it considers to be very useful for achieving its goals). The Asset Allocation Module uses these bids to find the allocation which maximises 'social welfare.' Bids are generated by the various tactical modules, which are not concerned with the bids of competing modules. Therefore, the strategic layer contains a Utility Management Module which weighs each bid's utility value according to the tactical module that generated it and the overall goal of the game. After the assets have been allocated, the Movement Order Module decides in which order the actions generated by the tactical modules are executed.

The three main tasks of a tactical module are (1) bid generation, (2) utility calculation, and (3) action generation.

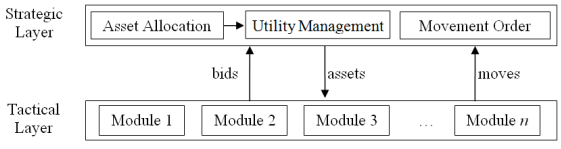

Figure 2: The ADAPTA Game AI architecture.

1. A bid generator is responsible for submitting a set of bids to the Asset Allocation module. These bids should represent the optimal (or near optimal) actions for a subset of the available assets, according to a certain tactic.
2. A utility function calculates a numerical utility value given a certain set of assets. This utility is used by the strategic layer to determine asset allocation. Utilities should provide an accurate measurement of the relative effectiveness of a certain tactic, compared to other tactics generated by the same module.
3. After the winning bids have been determined, game actions are generated by the module for the assets assigned. These actions are submitted (along with the utility of the associated bid) to the Movement Module in the strategic layer, which executes them in a certain order.

Spatial Reasoning

As defined in the previous section, a tactical layer can contain any number of tactical modules, and a module can be responsible for any task. In this research we focus on the creation of a single tactical module, which will serve as an example of how tactical modules fit into the ADAPTA architecture. This example module is named the Extermination module, as it is responsible for handling combat between units. In our environment, this amounts to assigning Move and Attack actions to each unit under the command of an AI.

We use influence maps (Tozour 2004) to determine the optimal tile to perform an action. Traditionally, an influence map assigns a value to each map tile to indicate the relative tactical influence that the player has on that tile. In our approach, we let the influence value indicate the desirability for a unit to move towards the tile. For TBS games, an influence map is typically calculated just before making a decision, so that it will, due to the turn-based nature, always reflect the current game state. The influence map is calculated separately for each tile, as follows:

$$I(x,y) = \sum_{o}^{O} p(w(o), \delta(o,x,y))$$

where O is the set of all objects used for this influence map, $p(W,d)$ is a *propagation function* of weight vector W and distance d, $w(o)$ converts object o into a vector of weights, and $\delta(o,x,y)$ is a distance function calculating the distance from object o to tile (x,y).

The behaviour of an influence map is defined by the two aforementioned functions, (1) the distance function δ and (2) the propagation function p. The distance function can be a general distance metric, such as Euclidean or Manhattan distance, or a domain-specific function, such as the number of steps that a unit needs to get to the target tile. The propagation function defines the influence that an object has on each tile. It is a function of the distance between the game object and current tile.

Because different types of game objects have different effects on the game, the calculation of influences should be performed differently for each type. Additionally, by summing both positive and negative influences for different types of objects in a single influence map, information may be lost. For these reasons, multiple influence maps should be maintained for different types of objects. These influence maps can then be analysed on their own as well as combined with others by a *layering algorithm*. A layering algorithm is a function that combines the contents of a number of influence maps and combines them into a single, new influence map. Figure 3 illustrates the process of layering two different influence maps.

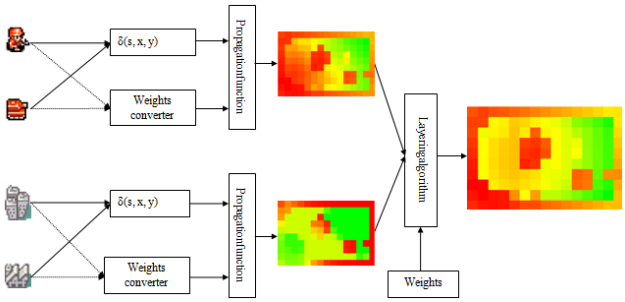

Figure 3: Layering two influence maps.

In our research, the layering algorithm is a neural network with two outputs. This neural network generates two influence maps simultaneously, one of which indicates the preferability of moving to each tile, and the other doing the same for attacking each tile. In this way, the task of the Extermination module is reduced to selecting the tiles with the highest influence from all possible targets for the Move and Attack actions.

Adaptivity

The behaviour of an AI using the spatial reasoning approach described in the previous section depends on the choices for the propagation functions and layering algorithm, as well as their weights. In previous work concerning influence maps, weights were often chosen arbitrarily (Tozour 2004). A different approach is to generate the weights automatically (Sweetser 2006; Miles et al. 2007). This is what we do. In order to keep the size of the search space manageable, we limit ourselves to generating the weights, while the choices of all functions remain fixed.

The learning algorithm we used is an evolutionary algorithm (EA) (Yao 1999). In our approach, candidate solutions, or *individuals*, represent a neural network as a string of weights. Each generation, the effectiveness (or *fitness*) of each of the individuals is determined according to a certain *fitness measure*. Because the individuals define the behaviour of a game AI, their fitness can be determined by letting these AIs play the game that they are intended for. In order to attain consistent fitness values, all individuals are made to play against the same opponent(s). An individual's fitness is defined as

$$F_i = \frac{\sum_j R_{ij} + \sum_j R_{ji}}{2 \cdot |R_i|}$$

where R_{ij} is the numerical value resulting from a game between players i and j, where player i has starting position

1 and player j has starting position 2. The genetic operators chosen for the EA are capable of operating on neural networks. A description of each of the operators, as well as a detailed overview of the EA used, is given by Bergsma (2008).

Experiment 1: Iterative Learning

For our experiments, we implemented a research environment in which the game SIMPLE WARS can be played between different AIs. It is impractical for the ADAPTA AI to learn against human players. We decided to determine the ability of the ADAPTA AI to defeat a certain tactic by letting it play against other AI implementations, such as scripts. We implemented a (rather simple) rush tactic to serve as an initial opponent. This tactic always makes its available units move towards the nearest enemy unit, and attacks whenever possible.

The goal of the game is to destroy 10 enemy units. This is also the total amount of units a player is allowed to build during the game. Because the game is played on a small map, both players are only allowed to control 4 units at any time. Both players start off without any units, but with 4 Factories, each of which is able to build all three unit types. Aside from the 10-unit limitation, both players are given sufficient resources to build 10 units at the start of the game.

Our goal is to improve iteratively the performance and behaviour of the ADAPTA AI, as well as assessing to which extent the results of the learning algorithm generalise over different opponents. Therefore, after the ADAPTA AI has learned to convincingly defeat the initial opponent, we started a new run with the learned AI as the new opponent. Again, when this opponent was convincingly defeated, we replaced it with the newly generated AI. This process was repeated until ten new AIs were generated. Such an 'arms race,' in which a learning opponent must defeat opponents generated in previous cycles, works well for many games and game-like tasks (see for instance the work by Pollack and Blair (1998) on BACKGAMMON and the work by Chellapilla and Fogel (1999) on CHECKERS).

Note that no Fog of War (FoW) was used for this experiment (nor for the second one). While experiments with FoW were performed, their results were similar to the results described below. A possible explanation for this is that, for this environment, FoW has little effect on the game. More details concerning the FoW experiments are given by Bergsma (2008).

Results and Discussion

Table 1 lists the final minimum, average, maximum, and opponent fitness values for each iteration. The Rush AI is used as the opponent in the first iteration, the AI generated against the Rush AI is used as the opponent in the second iteration, etc.

From Table 1 we conclude that, each iteration, the learning algorithm is successful in generating a solution which outperforms its opponent. This can be derived from the fact that the maximum fitness for each iteration is positive. To compare the individual results, all eleven AIs (the Rush AI and the ten generated AIs) were made to play against each other. For each of these AIs, their fitnesses against every opponent is averaged in Figure 4. Here, iteration number 0 represents the Rush AI.

Iter.	Minimum	Average	Maximum	Opponent
1	-0.293	0.129	0.473	-0.129
2	-0.275	0.059	0.326	-0.059
3	-0.060	0.155	0.486	-0.155
4	-0.416	-0.027	0.286	0.027
5	-0.021	0.318	0.591	-0.318
6	-0.252	0.141	0.500	-0.141
7	-0.533	0.057	0.357	-0.057
8	-0.022	0.025	0.302	-0.025
9	-0.425	0.053	0.300	-0.053
10	-0.212	0.111	0.457	-0.111

Table 1: Final fitness values for Experiment 1.

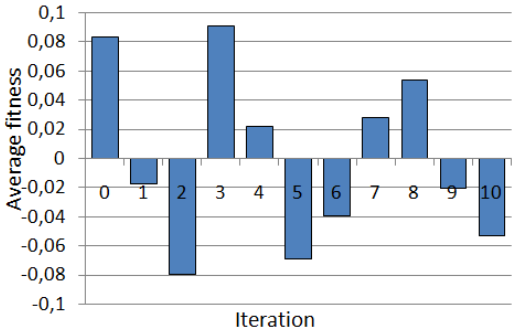

Figure 4: Average fitnesses of all AIs.

Figure 4 shows clearly that the overall performance of the AIs does not increase with each iteration. Even though each AI outperforms the AI from the previous iteration, the average fitnesses may go up or down. This fact suggests that the results do not generalise well over the opponents from previous iterations. For a better understanding of these results, we considered the types of tactics that the generated AIs use. We found that the AIs can be divided into three categories, each of which uses simple tactics.

1. The Defence tactic. This tactic keeps units close to each other at the base, and only attacks when provoked. It is used by AIs #1 and #5.
2. The Base Offence tactic. This tactic entails rushing the enemy bases with all units, and attacking enemy units whenever possible. It is used by AIs #2, #4, and #7.
3. The Unit Offence tactic. This tactic is similar to the Base Offence tactic, but it moves towards enemy units instead of bases. It is used by AIs #3, #6, #8, #9, and #10.

Figure 5 shows the performances for each of these types of tactics against each of the others, as well as against the Rush tactic. Each of the four graphs contains the average fitness values for the corresponding tactic against each of the four tactics. From this figure we can observe the following. The Rush AI outperforms the Base Offence and Unit

Offence tactics, but is outperformed by the Defence tactic. In turn, the Defence tactic is outperformed by the Base and Unit Defence tactics. This implies that none of the tactics dominates the others. It seems that, as is the case with the units, a rock-paper-scissors relationship exists at the level of strategies.

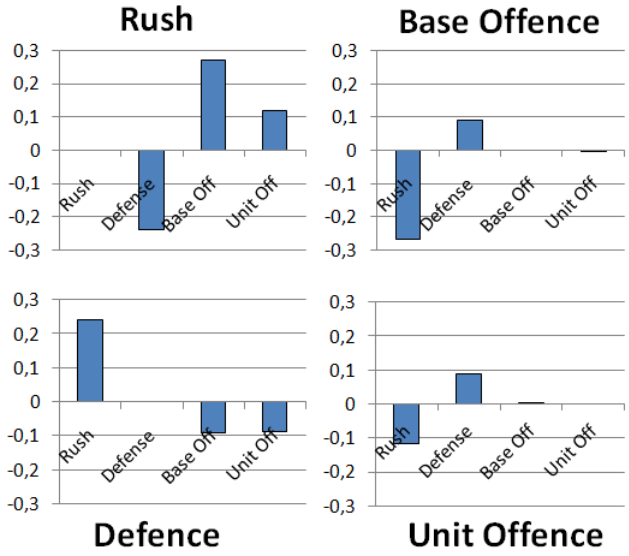

Figure 5: Fitnesses between each type of tactic.

Experiment 2: Multi-Objective Learning

Because each generated AI's performance was strongest against their direct opponent, it is expected that an AI which is learned against multiple opponents will be relatively strong against all of them. If each of these opponents uses a different tactic, the AI must learn to outperform them simultaneously, which potentially results in a generalised AI. In the second experiment we decided to generate an AI against three of the different AIs from the first experiment, namely #0 (the Rush AI), #1 (a Defence tactic) and #3 (A Unit Offense tactic). The approach we used for this experiment is based on the SPEA2 algorithm (Zitzler, Laumanns, and Thiele 2001). It uses the concept of Pareto dominance to establish an ordering between different individuals. An individual is Pareto dominated if another individual's fitness values are at least equal to the corresponding fitness values of the first individual, and at least one of them is greater. Using this concept, non-dominated individuals can objectively be considered better than their dominated counterparts. This approach is similar to a dominance tournament (Stanley and Miikkulainen 2002).

The SPEA2 approach differs from regular EAs in two important ways. Firstly, aside from the regular population, an archive containing non-dominated individuals is maintained. Each new population is now generated using the individuals from not just the population, but also from this archive. Secondly, two factors are used to determine the scalar fitness value of individuals, namely (1) the number of individuals which dominate this individual and (2) the location of this individual in the fitness space.

Results and Discussion

At the end of the second experiment (which we ran only once due to the time-intensive nature of the task), the learning algorithm had generated 28 non-dominated individuals. To determine the quality of these AIs, they were played against the test data, which consists of the previously generated AIs which were not used as training data (AIs #2 and #4 through #10). The resulting averages of the fitness values against both the training and the test sets are displayed in Figure 6. The test set averages are weighted according to the prevalence of their corresponding tactic within this set.

Unsurprisingly, the results against the test set are generally lower than those against the training set. The results show a clear favourite; NDI #32 outperforms the other AIs by a wide margin. To obtain a more accurate measure of the generated AIs' performances, they were also made to play each other in a tournament. Again, NDI #32 proved to be the best-performing AI. The average fitness it achieved was about 25% higher than that of the next best individual in this test, NDI #13. This is quite a big difference.

Analysing the behaviour of NDI #32 shows an interesting tactic. It does not simply rush toward enemy units or bases, but it is not strictly defensive either. For the first few turns, it does not move its units, until enemy units move into attack range. Then, the AI sends all of its units towards these enemy units. Because the enemy's units move at different rates and therefore do not arrive simultaneously, this results in a material advantage as well as the initiative of attack. This behaviour shows that the learning algorithm is able to generate an AI which not only outperforms multiple tactics, but does so using a new, somewhat more complicated tactic, instead of an improved version of the previously generated tactics.

Conclusions

The goal of this research was to generate automatically an effective TBS AI player. To accomplish this, our approach focussed on spatial reasoning, through the use of influence mapping. We extended the influence mapping approach by implementing an improved layering algorithm, and by creating a learning algorithm which generates the weights, and therefore the behaviour, of the influence maps automatically. We have also shown that influence maps can be used to determine directly the behaviour of an AI player. Moreover, in order to decompose the complex task of creating a TBS AI player, we proposed the ADAPTA architecture. This architecture makes it possible to concentrate AI design on a single subtask. This promotes the possibility to implement learning AIs.

In our experiments, we chose one subtask, namely an Extermination module which is aimed at combat. For this subtask we learned new AIs using an evolutionary algorithm. The results achieved showed that the ADAPTA AI is able to generate tactics that defeat all single opponents. Moreover, by learning against multiple opponents using different tactics simultaneously, an AI was created which was able to

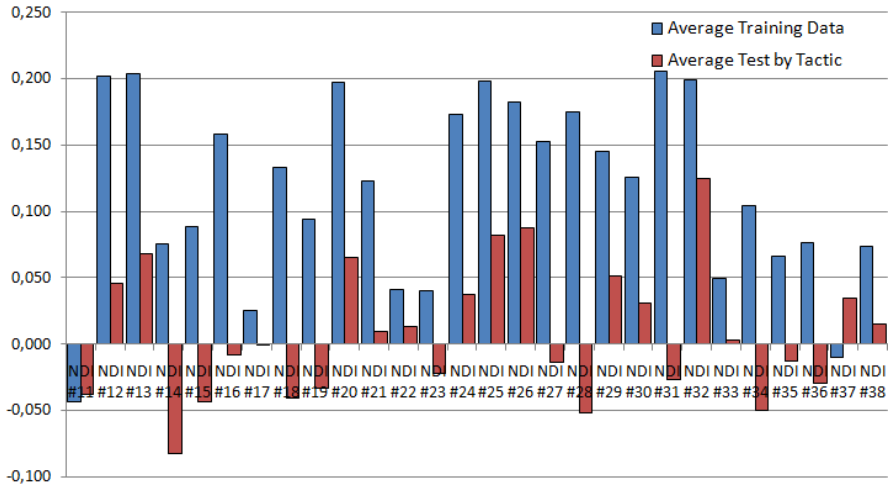

Figure 6: Average fitness of each non-dominated individual against the training set and the test set. The latter is weighed according to the prevalence of each tactic in the test set.

play at least equally well and outperform most of the previously generated AIs.

For future work, we intend to explore the capabilities and limitations of the auctioning mechanism in the strategic layer of the ADAPTA architecture, as this was not included in the present research. Furthermore, we intend to explore whether the successes achieved with learning of the Extermination module can be repeated for the other modules.

Acknowledgements

This research is supported by a grant from the Dutch Organisation for Scientific Research (NWO grant 612.066.406).

References

Bergsma, M. 2008. Adaptive spatial reasoning for turn-based strategy games. Master's thesis, Maastricht University.

Bryant, B. D., and Miikulainen, R. 2007. Acquiring visibly intelligent behavior with example-guided neuroevolution. In *Proceedings of the Twenty-Second National Conference on Artificial Intelligence*. Menlo Park, CA: AAAI Press.

Buro, M., and Furtak, T. 2003. RTS games as test-bed for real-time AI research. *Workshop on Game AI, JCIS 2003* 20.

Chellapilla, K., and Fogel, D. 1999. Evolution, neural networks, games, and intelligence. In *Proceedings of the IEEE*, volume 87:9, 1471–1496.

Dill, K. 2006. Prioritizing actions in a goal-based RTS AI. *AI Game Programming Wisdom* 3:331–340.

Hinrichs, T. R., and Forbus, K. D. 2007. Analogical learning in a turn-based strategy game. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence*, 853–858.

Miles, C.; Quiroz, J.; Leigh, R.; and Louis, S. J. 2007. Co-evolving influence map tree based strategy game players. In *Proceedings of the 2006 IEEE Symposium on Computational Intelligence in Games*, 88–95. New York: IEEE Press.

Obradović, D. 2006. Case-based decision making in complex strategy games. Master's thesis, Kaiserslautern, Germany.

Pollack, J., and Blair, A. 1998. Co-evolution in the successful learning of backgammon strategy. *Machine Learning* 32(3):225–240.

Sánchez-Pelegrín, R.; Gómez-Martín, M. A.; and Díaz-Agudo, B. 2005. A CBR module for a strategy videogame. In Aha, D. W., and Wilson, D., eds., *1st Workshop on Computer Gaming and Simulation Environments, at 6th International Conference on Case-Based Reasoning (ICCBR)*, 217–226.

Stanley, K., and Miikkulainen, R. 2002. The dominance tournament method of monitoring progress in coevolution. In Rabin, S., ed., *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2002) Workshop Program*. Hingham, MA: Charles River Media.

Sweetser, P. 2006. Strategic decision-making with neural networks and influence maps. *AI Game Programming Wisdom* 3:439–446.

Tozour, P. 2004. Using a spatial database for runtime spatial analysis. *AI Game Programming Wisdom* 2:381–390.

Ulam, P.; Goel, A.; and Jones, J. 2004. Reflection in action: Model-based self-adaptation in game playing agents. In *Challenges in Game Artificial Intelligence: Papers from the 2004 AAAI Workshop*, 86–90.

Yao, X. 1999. Evolving artificial neural networks. In *Proceedings of the IEEE*, volume 87, 1423–1447.

Zitzler, E.; Laumanns, M.; and Thiele, L. 2001. SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Swiss Federal Institute of Technology, Zurich, Switzerland.