TECHNISCHE UNIVERSITÄT DORTMUND

REIHE COMPUTATIONAL INTELLIGENCE

COLLABORATIVE RESEARCH CENTER 531

Design and Management of Complex Technical Processes and Systems by means of Computational Intelligence Methods

Intelligent Group Movement and Selection in Realtime Strategy Games

Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael Stüer, Andreas Thom and Simon Wessing

No. CI-255/08

Technical Report ISSN 1433-3325 December 2008

Secretary of the SFB 531 · Technische Universität Dortmund · Dept. of Computer Science/LS 2 · 44221 Dortmund · Germany

This work is a product of the Collaborative Research Center 531, "Computational Intelligence," at the Technische Universität Dortmund and was printed with financial support of the Deutsche Forschungsgemeinschaft.

Intelligent Group Movement and Selection in Realtime Strategy Games

Mike Preuss, Nicola Beume, Holger Danielsiek, Tobias Hein, Boris Naujoks, Nico Piatkowski, Raphael Stüer, Andreas Thom, Simon Wessing

***Abstract*—Movement of groups in realtime strategy games is often a nuisance: Units travel and battle separately, resulting in huge losses and the AI looking dumb. This applies to computer as well as human commanded factions. We suggest to tackle that by using flocking improved by influence-map based pathfinding which leads to a much more natural and intelligent looking behavior. A similar problem occurs if the computer AI has to select groups to combat a specific target: Assignment of units to groups, especially for multiple enemy groups, is often suboptimal when units have very different attack skills. This can be cured by using offline prepared self-organizing feature maps that use all available information for looking up good matches. We demonstrate that these two approaches work well separately, but also that they go together very naturally, thereby leading to an improved and—via parametrization—very flexible group behavior. Opponent AI may be strenghtened that way as well as player-supportive AI. A thorough experimental analysis supports our claims.**

***Index Terms*—Realtime strategy games, path finding, neural networks, evolutionary algorithms, tactical decision making**

I. Introduction

The Realtime Strategy (RTS) genre recently featured some of the most successful and also most complex computer games. It may also be regarded as one of the roots of nowadays popular *massive multiplayer online games* (MMOG) like ‘World of Warcraft’. However, while graphics and sounds RTS games have been improved dramatically over the last years, opponent strategies, or more general, the whole game AI has not made such big steps. One reason for this nuisance may be the complexity of the games which is even increased by the realtime component: game AI has to cope with many different subproblems (e.g. ressource distribution, path-finding and strategic decisions like where and when to attack), without much time to search for suitable solutions. Several of these problems cannot be resolved by cheating because optimal solutions are not known beforehand. For example, pathfinding in a dynamic game setting with moving objects cannot be fully pre-computed, and ‘teleporting’ foot units may look strange to human opponents. However, game AI should perform well and still look believable [1] in order to provide fun to human players.

Bio-inspired techniques (*computational intelligence*, CI) have mostly been integrated in simple computer games, e.g. arcade video games, for improving the game’s artificial intelligence such as the intelligence of non-player-characters (NPC) or the handling of the game [2]. However, they are increasingly employed also in more complex RTS games. First attempts produce game AI that is capable of adaptive behavior, e.g. Louis et al. [3], [4] combined EAs with memories to produce strategies online in the game. Detecting the opponent’s grouping behavior is a subproblem of opponent modeling, which has been addressed by Spronck et al. [5] as well as knowledge acquisition in RTS games in general [6].

While developing a complex game plan by means of CI methods appears rather hard, using the methods for subsidiary in-game decisions—especially for group movement and selection—seems manageable. We here combine and extend two recent approaches of ours, namely [7] and [8].

The first approach employs flocking, a technique introduced by Reynolds [9], and *Influence Maps* (IMs) [10] to improve the path-finding of units. The latter may be left to the A* algorithm [11], but from a strategic perspective, the results can be poor, e.g. if calculated paths are directly leading through enemy territory or the moved groups split up on their way. Such problems can be overcome by flocking which is introduced in sec. III-A. Although performing already well, we show that flocking alone is not able to cope with all ocurring situations. However, flocking teams up well with influence maps, areal based aggregations of the game AI’s available data which are introduced in sec. III-B. Note that the use of flocking+IM is not restricted to groups run by the game AI: it may also be applied to humanly controlled groups.

Another problem for game AI is the selection of units for attacking a certain target or defending against a specific group. Here we suggest an approach based on *Self-Organizing Maps* (SOMs) as introduced in section III-D. SOMs can be trained offline by using simple substitute objective functions (instead of game simulations) and virtually need no time to give an answer to the question which units to select in-game.

The major part (section IV) of this work consists of a thorough experimental analysis, testing our approaches in different scenarios within the RTS game `Glest` (see sec. II). We end with the most interesting experiment from our point of view, where all proposed techniques act in common. Groups behaving according to the proposed techniques—if beheld by a human—appear to be ‘intelligent’ or human-like, although we are aware that such properties might never be measurable.

II. Glest

Strategy games are usually placed in war or battle scenarios, where players have to build bases and armies and destroy the other players’ units. `Glest` is an open source RTS game

Chair of Algorithm Engineering, Computational Intelligence Group, Faculty of Computer Science, Technische Universität Dortmund, Germany

TABLE I
DAMAGE MULTIPLIERS, ATTACK AGAINST ARMOR TYPES (TECHS).

	Leather	Plate	Stone	Wood
Arrow	1.5	0.5	0.3	1.0
Sword	1.0	1.0	0.5	1.0
Fire	1.0	0.8	0.5	1.0

published by Figueroa et al. [12] under GNU Public License (GPL, [13]). While this work is based on version 2.0.1, `Glest` version 3 has been released with improved gameplay and balancing. Therefore, some of the observations shown here might not be relevant to the new version. The game is settled in a fantasy medieval world and players can either control a *magic* or a *tech* faction. Both possess unique unit types, buildings and enhancements. While the game also has other aspects like resource harvesting and building, this work only deals with battle situations.

The flocking studies consider the magic faction only and group selection with SOM is done for the tech faction, only. However, his partitioning has no particular reason; both approaches could be applied to either faction. The magic faction uses wizards as basic units, which can be trained to become powerful warlocks and may conjure dragons and daemons as fighting units. The tech faction uses laborer as basic units, who may be trained to become knights, archers or engineers who are able to construct (flying) battle machines. Basic tech unit types are *archers* and *swordmen*, both armored with leather. *Armored swordmen* are slower than the two mentioned, but possess a plate armor. *Horsemen* are equally well armored and much faster. The *technician*, a weak unit, is prerequisite for all following units. The *catapult* is slow, but accomplishes a high damage. The *battle-machine* is very versatile, being the only unit that has a short- and a long-range weapon. At last, there are two flying units: The *airship* and the *ornithopter*. Archers and battle-machines are the only ground units that can attack flying units. The units take up different resources and usually become available in the mentioned order during the game.

Each unit has a number of *hit-points* (HP) representing the unit's health. When it is attacked, the HPs are decreased by the damage value of the attacking weapon. A unit dies when its HPs reaches zero. Moreover, the game provides a configurable intransitive superiority relation of the different unit types by means of a damage multiplier table. The actual damage inflicted by an attack of a specific unit type onto another is modified according to the armor type of the defending unit. Default damage multipliers are given in Tab. I.

III. TECHNIQUES

Flocking, influence maps, and self-organizing maps are introduced, accompanied by describing how they are modified to satisfy our needs and coupled within the game.

A. Flocking

Flocking is a suitable method to simulate natural group movement in computer games of e.g. swarms of birds, schools of fishes, military units, or crowds. A flock is able to reach a certain target as a group and to avoid obstacles on the way. Recently, it has been applied to control the movement and formation of units in ego-shooters e.g. by Shen and Zhou [14] in *Unreal Tournament* and by Davison [15] in *Half-Life*.

Reynold [9] gives three simple rules to adjust the behavior and motion of units (called *boids*).

- **Separation:** A boid tries to scatter from others. This rule can be seen as the opposite of the cohesion rule and represents the collision avoidance within the flock.
- **Cohesion:** A boid sticks together with the others and tries to advance to the centroid of the group.
- **Alignment:** A boid steers into the same direction as the neighboring boids calculated as the average of the summed up direction vectors of the boids in its sight.

The final direction of a boid's movement v is calculated as the weighted sum of the results of the three rules as

$$v = w_{ref} \cdot v_{ref} + w_{coh} \cdot v_{coh} + w_{sep} \cdot v_{sep},$$

where v_{ref} is the reference vector reflecting the direction of the calculated flock path, v_{coh} is the vector from the unit's position to the average position of the flock, and v_{sep} is the separation vector. The corresponding weights are w_{ref}, w_{coh}, and w_{sep}. The calculation of the direction requires information on the boid's neighborhood, namely the location of and distances to its neighbors, and the directions the neighbors are heading. A boid's orientation is defined by a line of sight. Its ability to recognize its environment is specified by a range and an angle of sight α.

The implementation at hand is different from Reynolds original idea to have boids moving around randomly and eventually building a flock [9]. Instead, the player selects some units he wants to group together. This group of units is called flock and a target is assigned to the flock by the player. A path to the given target is obtained using the A*-algorithm [11] already implemented in `Glest`. To follow this path, a local path-finding incorporating the flocking method is started.

In our scenario, the most significant parameters of flocking are the angle of sight α, the separation weight w_{sep} and the distance of separation determined by the length of the separation vector $|v_{sep}|$. A reduced angle of sight makes the flock behave like a snake, each unit following its predecessor in-line, whereas a broader value makes the flock move more scatteredly. The separation, i.e. the ambition of every boid to move away from other boids, depends on w_{sep} and $|v_{sep}|$.

B. Influence Maps

Influence Maps (IM) are an areal based representation of knowledge. In games they are usually employed to store information on the current game situation for tactical decisions. They exploit the topographic information of the actual map to represent how players influence different areas of it. Via IMs, players are able to find out where opponent units are concentrated or explore weakly exposed areas. We invoke two-dimensional environments only, a broader approach regarding different topologies can be found in [16].

Usually, a map is divided into tiles. The IM is set to the same size as the game's map with typically less tiles. As influence

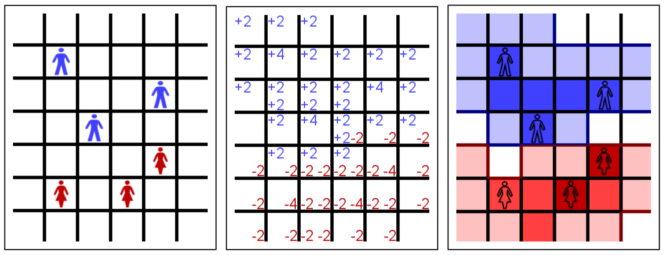

Fig. 1. Example of influence map generation. The first figure shows the units of two different factions (male, blue and female, red) placed on the map. The second figure yields the local influence of the units ($+4$ and -4, respectively) and their propagation of the influence values through the map. Finally, the right figure depicts the resulting influence map with the influence of the factions, whereas darker colors mean higher influence values. In addition, the positions of the units are shown as well as the borders of the faction's influence.

values a certain property of a unit is used, e.g. hit-points or attack strength. The tiles are initialized with a value of 0. To calculate the IM for a game situation, an influence value of every unit on the map is derived and propagated through the map. Usually, the influence of a unit decreases linearly or quadratically with increasing distance to the tile where the unit is located.

The example in Fig.1 shows the calculation of a simple IM with two hostile factions (blue/male and red/female). Each unit has an influence value of 4, which decreases by a factor of 2^{-d} in relation to the distance d (measured in tiles) to the given unit, and the propagation is aborted after values ≤ 1. The propagation is shown in the central image of figure 1 and the resulting influence map retrieved by summing up all influences in the right figure. Note that the influence on a tile may exceed the influence of a single unit if several units of one faction are close to each other. Vice versa, the influences of two adversary units may also cancel each other out. The right image of this figure shows a graphical representation of the calculated influence with the boundaries of the controlled area outlined for each faction. The degree of influence is shown by the intensity of the faction's color.

To use an IM for path-finding it has to be kept up-to-date at all times. For this purpose, every unit caches its influence. We combine the unit's hit-points HP and the damage D a unit is able to cause to its combat strength c_u as

$$c_u = \frac{D_u \cdot HP_u}{100}.$$

as the influence of a unit. It is propagated over the unit's range of sight like explained above. When a unit is moved its influence is subtracted from the map at the old position, added at the new one, and propagated through its area of influence afterwards.

To calculate a path for a unit, its influence has to be subtracted from the current IM so that it is not affected by itself. The resulting accumulated influence at each position (x, y) of the IM is called $\zeta_{x,y}$ in the following. This influence has to be integrated into the cost function used by the A*-algorithm of `Glest` to calculate an appropriate path for the unit. Note that the resulting path will not reflect the shortest Euclidean distance directly. To integrate the influence values received from the IM in the cost considered by the A*-algorithm, this cost $\tau(x, y)$ of tile (x, y) is defined as the product of the Euclidean distance δ and a factor based on the influence:

$$i(x,y) = \frac{\zeta_{x,y} - c_u}{180}; \quad \tau(x,y) = \begin{cases} \delta e^{-1} & \text{if } i(x,y) < -1, \\ \delta e & \text{if } i(x,y) > 1, \\ \delta e^{i(x,y)} & \text{otherwise.} \end{cases}$$

The unit's combat strength c_u is subtracted to allow the unit to pass tiles with low hostile influence; 180 seemed to be a good scaling value. The distinction according to $i(x, y)$ avoids inappropriate τ values. With τ close to zero, the A* algorithm would not be able to calculate an appropriate path with all cells sharing costs close to zero. Expensive costs may cause dead ends even in open-range areas when every surrounding cell is much more expensive than the one the unit is located on. Both problems impede the unit from reaching its target. The cost function effectuates that for a unit tiles with strong influence of its faction become extremely cheap, whereas tiles with enemy influence become much more expensive. Thus, the unit will prefer to move around enemy territory.

C. Combination of Flocking and IM

For path-finding, a flock is considered as a single unit. Each unit has to update the influence map with its own cached influence as described above. To determine the costs of a tile a new combat strength c_F for the complete flock F is calculated:

$$c_F = \frac{1}{SEW} \cdot \sum_{u \in F} HP_u \cdot \sum_{u \in F} D_u$$

The idea is that the combat strength of a group is higher than simply the sum of strengths of its units. This holds because the flock itself causes more damage to single opponent units, while damage caused to the flock is scattered over several units in the flock. To quantify the synergy effects, we introduced the *synergy effect weight* (SEW). This parameter enables the user to scale the aggressiveness of the flocked units. If SEW is very high, the flock will not attack enemies on its way although it would clearly win the combat. A very low SEW value makes the flock attack even overwhelming enemy forces.

An appropriate choice of SEW leads to units taking the secure path around stronger enemies whenever a path around is possible. If the safe path is too long, it is more expensive to take this path compared to traveling through the enemy territory. In this case, our unit will take the direct path. If our unit is stronger than the enemy, the unit/flock will take the direct path and attack the enemy in any case. This behavior is implemented having in mind that the territory is virtually already under control of the own faction.

The general idea to couple the techniques of flocking and IM is not new but to our knowledge, the combined techniques have not been used to support group movement and path-finding in RTS games. Flensbak [17] uses three two-dimensional influence maps for each of the desired behaviors alignment, cohesion, and separation. The influence maps are particularly used to provide information on the neighboring units and to control the shape of the flock. Miles and Louis use influence maps as nodes in decision trees to determine different game situations and as decision support system [18], [19]. The players use information from the A*-algorithm working on an IM to determine the costs of their objectives but they do not use it for moving their units. This approach is close to the original idea of using IM to analyze game situations for strategic decision making.

D. Self-Organizing Maps for Unit Selection

Group selection or rather unit selection for the creation of specific groups is of tactical as well as of strategic use.

- **Tactical attack:** If an enemy base or unit group shall be attacked by the AI, and some information about the defenders is available, it is favorable to bring enemy units that perform well against it.
- **Tactical defense:** In case the AI is attacked, possibly by multiple enemy groups, it makes sense to send the available units against the enemy units they can fight best.
- **Strategic build:** The unit build process—and thus also the exploration of the tech tree, including buildings—shall take into account information about the favored unit types of the adversary.

Whereas for the latter use data must be gathered and accumulated on a longer time scale, tactical decisions have to be taken ad hoc, or on a very narrow time scale.

Since most RTS games feature thouroughly balanced units with different abilities, we often find a situation of *intransitive superiority* (e.g. A beats B, B beats C, and C beats A as in the “Rock, Paper, Scissors” children’s game). Here, no globally optimal strategy exists; we need a possibility to respond to any opponent group composition, a context-sensitive memory. Self-Organizing Maps (SOMs) as conceived by Kohonen [20] can deliver exactly that. A SOM maps a high-dimensional data space to a low-dimensional, usually 2-dimensional map space combining the goals of visualization and dimension reduction of complex data sets. SOMs are topology-preserving maps, i.e. while mapping the data from the input space to the map space they consider the underlying structure of the data and attempt to obtain the topological relationships. Neighboring elements of the input space are also close to each other in the projected map space. Consequently, a SOM creates a map of regions of similarity by grouping similar data.

The SOM consists of a set of nodes arranged in a discrete lattice, defining the topology of the map. We employ a toroid-shaped lattice by connecting its opposite edges. Each node i of the lattice is associated with a randomly initialized reference or weight vector $w_i = (\omega_{i1}, \omega_{i2}, \ldots, \omega_{in})^T \in R^n$. The dimension n of the weight vectors correspond to the dimension of the input vector. The input vector $x = (\epsilon_1, \epsilon_2, \ldots, \epsilon_n)^T$ specifies the distribution of unit types over an opponent’s army. Likewise, the weight vectors w_i represent a mix of the defending player’s unit types.

Basically, the learning algorithm conforms to the original algorithm devised by Kohonen [20]. In each iteration t an input vector x is randomly drawn from the data set using a desired probability distribution over the input space. While traversing the SOM’s nodes, the distance between the input vector x and each node’s weight vector w_i is calculated. The applied distance measure can be an arbitrary function that takes two objects as input and returns a scalar value like the Euclidean distance. The node with the shortest distance to the input vector, thus the node that matches the input vector best is called the best-matching unit (BMU). The BMU’s weight vector is manipulated to approximate the input vector even further. Nodes within a defined radius r around the BMU are being adapted as well. The influence ℓ of the adaption is defined by a Gaussian curve centered at the BMU and declining with increasing distance to the BMU. The neighborhood size shrinks with increasing iterations. The learning process will terminate after a predefined number of iterations is reached.

Operation of the established SOM works by looking up the BMU in the same way as during training, but without manipulating the nodes. The found BMU (and its neighborhood) deliver information that may then be processed further.

Usually, SOMs are used in conjunction with a (weighted) Euclidean distance. A specialty of our work is the development of a non-standard distance measure for this purpose. It is designed to relate to suitability instead of similarity. Consider i.e. the following situation: During the game, the AI gets aware that the opponent is about to attack with a certain group of units E. A part of the problem to find the optimal defending group is the question: How successful would a certain group U be when facing the attackers? Of course, this could be answered by a simulation of the upcoming battle in the game, thereby obtaining a suitability measurement. The simulation ends when one party is defeated or after 5,000 game cycles elapsed. While p_E and p_U denote the fractions of surviving units from the attacking and defending groups, the outcome is measured as $(p_E - p_U) \rightarrow min!$ at the end of the game. The optimum of this function is -1.0, which means that all attacking units have been eliminated, while all defending units survived. However, simulating is far to computationally intensive during the game. Instead, we developed a substitute objective function $f(E, U)$ based on our knowledge of the properties of the game’s units. The function returns a score of a group U against a hostile group E. In our example situation, E is the attacking group and U is the AI’s defending group. In our setup, U is retrieved from the AI’s memory, the SOM.

A unit can have several attack skills. Every skill s is evaluated with a score c_s against the set of attackable units E_s in the opponent’s group E. $d(t_s, r_e)$ is the damage multiplier of attack-type t_s against armor-type r_e of hostile unit e.

$$c_s = \sum_{e \in E_s} d(t_s, r_e) \tag{1}$$

Every unit u in group U is awarded a score c_u for its most useful attack skill only. c_u is calculated as

$$c_u = \max\{c_{s,1}, \ldots, c_{s,n}\}. \tag{2}$$

A group’s score $f(E, U)$ then is the sum of all its units’ scores c_u. To account for the mutual strengths and weaknesses, the final objective is $(f(U, E) - f(E, U)) \rightarrow min!$.

Obviously, the function does neither take costs like time and resources, nor features like speed, hitpoints and attack-strength into account. Finding an optimal group under these additional constraints would be a multi-criteria problem that is not covered by our approach. Our objective function only considers aspects concerning the *relative* strength of units. Preliminary experiments showed that otherwise only the most expensive units would benefit.

As already stated, it is necessary to adapt defending groups to the attacking ones in the learning process. While this is a trivial task with Euclidean distance, we need to optimize the substitute objective function. Here, a $(1+1)$-EA carries out the optimization of the defender's mix of units. An introduction to evolutionary algorithms (EA) is given by Beyer and Schwefel in [21]. Incorporating an EA into a SOM has already been proposed by Kohonen [20], [22].

The influence ℓ of adaptation is controlled by the number of objective function evaluations. Every time the BMU is found during the training, the EA is run for $\lfloor \ell \cdot 10 \rfloor$ function evaluations. ℓ is 1.0 for the BMU and declines with increasing distance. Mutation is carried out on the real number representation of the groups, by shifting a certain amount between two randomly drawn unit types. The shifted amount is determined by a normal distributed random variable with mean 0.0 and variance 0.05. The mutation-strength can be seen as analogy to the learning rate often used with a Euclidean distance.

IV. Experimental Analysis

The experimental analysis shall demonstrate advantages and drawbacks of the considered techniques in RTS games. Using artificial settings that clearly deviate from standard `Glest`, it is easy to test if the desired output, e.g. in terms of a recommended unit group, is indeed attained.

By performing initial proof-of-principle tests as done for flocking+IMs in experiment 1, and for SOMs in experiment 3, we demonstrate the general viability of the proposed approaches. Experiment 2 investigates the adaptability of flocking+IMs concerning aggressive/defensive behavior by changing *one* parameter. To support our assumption that SOM training by means of a simple substitute objective function instead of simulations is reasonable, we investigate the correlation between these two data sources in experiment 4. Within experiment 5, we apply our technique to the anti-grouping problem in standard `Glest`, also studying proper parametrization and parameter robustness of the SOM training process. Experiment 6 investigates a sample szenario in which the combination of all techniques shall lead to a clear advantage—and in fact it does.

The reports on the received results adhere to the suggestion by Preuss [23]. It demands the author to think about the tackled questions and the employed means *before* the experiment is carried out and to report the results in a structured, easily accessible way.

Experiment 1: *Proof-of-principle: When does pure flocking improve group behavior? Do IMs cure its weaknesses?*

Preexperimental planning: By testing different settings, we established a scenario in which the attacking group loses the final combat when not employing flocking and wins in the other case. Additionally, we found a counter-example where flocking leads to performance losses. In this case the combination with IM brings success.

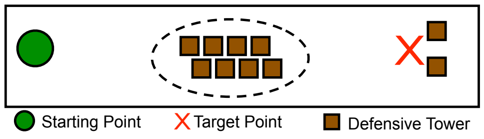

Fig. 2. Map sketch for the first and second scenario tackled. The encircled defensive tower position in the middle only exists in scenario 2.

Task: Our base hypothesis (we want to reject) is that neither flocking nor IM have any effect on the group behavior.

Setup: A group of combat units is sent to two hostile defensive towers (high combat strength, 7000 HP). The first map (cf. Fig. 2) is empty apart from these entities and features no obstacles like mountains or rivers, and is rather small (64x64 tiles). The group consists of six units with different abilities: two demons (average speed, weak combat strength, short-ranged, 700 HP), two battlemages (average speed, weak combat strength, long-ranged, 700 HP), and two drakeriders (fast speed, high combat strength, medium-ranged, 1300 HP).

In the second map, (cf. Fig. 2), hostile turrets are added along the way of the combat units and the attacking group is enlarged by an initiate (slow speed, weak combat strength, medium-ranged, 450 HP). Remark that the initiate is the weakest unit in the game so that the group's strength does not increase significantly. The units have to pass along eight defense towers with sufficient range to attack the group. The flocking parameter settings are: angle of sight $\alpha = 90$, separation weight $|w_{sep}| = 3.0$, and separation distance $|v_{sep}| = 4.0$. When using IM, it is applied within the A*-algorithm to calculate the moving path. For each map and all four movement types (movement with/without flocking, pathfinding with/without IM), we run 500 games with different random seeds. Note that inflicted damages are subject to random variation.

Results/Visualization: Fig. 3 presents the number of surviving units within the different scenarios and Fig. 4 the HP values of the teams after the combat. Note that zero survivers

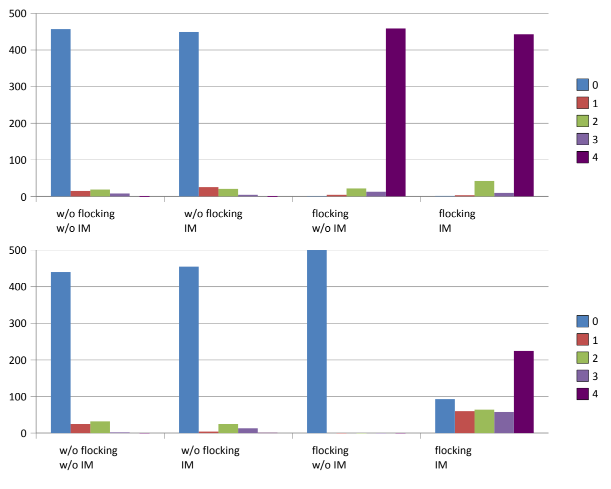

Fig. 3. Number of surviving units on the first map (top) and the second map (bottom) for four different strategies with 500 runs each. We show all combinations of normal and flocking movement with and without IM.

indicates that the attacker lost the game.

Observations: In the following, the different settings and their influence on the resulting movement behavior are described.

Without flocking. Fast units break away from the slower ones and the group splits up. On the open-ranged map, the units reach the two hostile turrets with long delays, so that the turrets have to fight only a few units at the same time. In most cases, the turrets kill each unit before the next ones arrives and thus defeat the group. Thus, only about 8% of the games are won thereby with only few units surviving (Fig. 3).

On the second map, the path calculated for each unit passes the range of the eight defense towers located in the middle of the map. So especially the slow units get hurt by these towers, e.g. initiates are killed in most games as they need too much time to cross the dangerous area (Fig. 3). Nevertheless, the winning ratio is roughly the same as on map 1, since faster units are hardly afflicted by the towers.

Using flocking. The units stick together by adapting their movement to the speed of the slowest group member. So in the first map, the units of the group reach the two defensive towers at the same time. As the group outnumbers the two towers, these are hardly able to defeat the group. Consequently, in nearly all games the flock wins with four surviving units (Fig.3). On the second map, flocking causes a significant drawback. The flock sticks together but moves rather slowly due to the slow units. The path of the flock leads through the range of the eight towers in the middle of the map. Compared to the results without flocking, the flock is under attack for a longer time and many units are lost. None of the games is won (Fig. 3).

IM without flocking. No significant differences for wins, surviving units, or HP values can be recognized for the normal movement supported by IM-based paths on both maps.

IM with flocking. IM-supported pathfinding does not change the flock behavior for map one, still all games are won. On the second map the results change from 100% losses to about 80% wins (Fig. 3) by IM and more than 40% wins with four surviving units. The mean HP values for the attacking team exceeds the value for the rival team, while it is 0 without IM (Fig. 4).

Discussion: The bad results for the normal movement stem from the different speeds of the considered units. Here, these units reach their common target one after another, which enables the towers to kill them easily. Incorporating flocking, the group arrives at the target as a whole. In this situation, the flock is strong enough to destroy the towers.

However, flocking decreases the speed of faster units to the speed of the slowest flock member. This may cause fatal drawbacks in specific situations as demonstrated on map 2, which is almost never won with flocking, although it is nearly always won without. Thus, we can clearly state that flocking often *makes* a difference and reject the initial hypothesis. However, the results also underline the need to improve the flocking method.

This is demonstrated to be done by the IM supported pathfinding. In the first map, no improvements occur since all games are already won and there is no dangerous area to circumvent intelligently. On the second map, group behavior

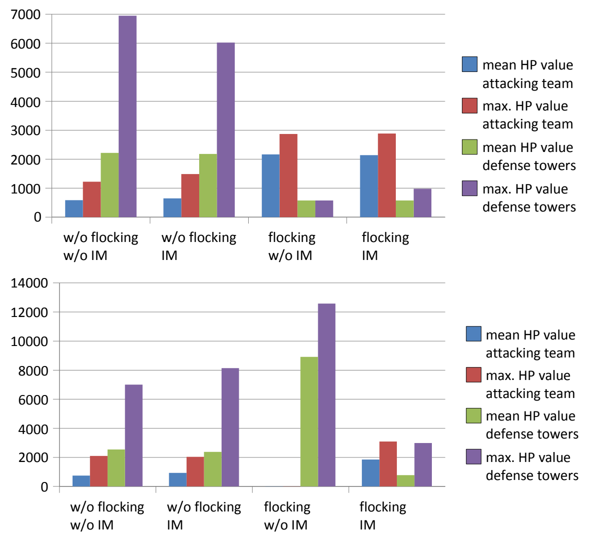

Fig. 4. Mean and maximal HP values of the attacking team and the two defense towers on the first map (top) and the second map (bottom) for four different strategies (movement with/without flocking, pathfinding with/without IM). The starting HP value of the attacking team is 5400 on map 1 and 5950 on map 2. The two defense towers own 14000 HP together (7000 each).

changes significantly with the IM path-finding which leads the group around the invincible set of towers in the middle of the map. Thus, the attacking units do not lose any hit-points on their way to their target.

When flocking is not active, the group separates even further than without IM as the path is longer. Consequently, the attacked towers have more time to fight arriving units and the attacking units are defeated (Fig. 3).

Combining flocking and IM is the most successful alternative for the second map, without losing performance for the first one. Flocking lets the group stick together and the IM takes care for secure paths around opponent units, thereby successfully avoiding the eight towers in map two.

The combination of the flocking method and the path calculation with the modified A*-algorithm is a great improvement in the shown situation. The drawbacks of flocking could be invalidated with the help of the IM path-finding. Summarizing, this possibility of an intelligent moving behavior represents a real alternative to the normal movement in RTS games.

***Experiment 2:** Can we reliably control defensive/aggresive behavior under flocking and IM with the SEW parameter?*

Preexperimental planning: We perform first test with varying SEW value in order to find a reasonable interval to investigate in the following.

Task: We demand that the group behavior invoked with changing SEW is a) significantly different in the extreme values (aggressive/devensive) and b) follows an unambiguous pattern (similar to a linear model) that makes it controllable.

Setup: The SEW should control the aggressiveness that can be observed in either attacking stronger enemy units or passing around these with possibly attacking weaker ones. Thus, a third scenario is defined that differs from the prior ones in two enemy positions on the way to the group's target (cf. Fig. 5). The first enemy position consists of three defensive

TABLE II
PERCENTAGE OF GAMES WON WITH DIFFERENT NUMBERS OF SURVIVING UNITS FOR THREE VALUES OF SEW.

SEW value	surviving units							
	7	6	5	4	3	2	1	0
4	0	0	0	5.8	27.8	17.4	10.8	38.2
8	0	0	100	0	0	0	0	0
12	100	0	0	0	0	0	0	0

TABLE III
PERCENTAGE OF GAMES WITH SURVIVING DEFENSE TOWERS (SDT).

SEW value	sdt team 1				sdt team 2	
	3	2	1	0	1	0
4	0	16.8	21.4	61.8	100	0
8	100	0	0	0	0	100
12	100	0	0	0	100	0

towers (same values as in experiment 1 and 2). The second one is a single but stronger defensive tower featuring a high combat strength with 13000 HP. The distance between the two positions is big enough so that their influences do not overlap each other. The group is the same as in scenario 1 plus an additional archmage, resulting in 7 units. Both parties win this combat with roughly the same probability. SEW is set to any of $\{4, 8, 12\}$ and each configuration is repeated 500 times.

Results/Visualization: Table II gives the percentage of games in which a certain number of units reached the target point. Table III depicts equivalent values for the two groups of defensive towers. *Sdt team 1* stands for the three towers, whereas *sdt team 2* refers to the stronger stand-alone tower.

Observations: Table II shows that all attacking units survive with SEW set to 12. Noteably, for $SEW = 8$, exactly 2 units are killed in each game. All towers survive if $SEW = 12$. When $SEW = 8$, only the second tower is destroyed in every game, while the three weaker towers survive. SEW value 4 results in the stand-alone tower surviving every game again. Note that in about 60% of the games, the group of towers is completely destroyed by the attacking units.

Discussion: Depending on the three SEW values, three different situations occur. A high SEW causes a more defensive group behavior, which is reflected in all attacking units as well as all towers surviving. The main aim here is to get to the target point and not to attack any enemy position. The high SEW provides significant compensation for the cumulated group power. However, the high SEW does not guarantee a complete defensive behavior. An extremely powerful group might still decide to attack.

When SEW is set to 8, the group does not attack the first strong enemy position, because its own cumulated power is still underestimated. Instead, it carefully passes around the strong position, and attacks the single turret, which has a higher HP but is not assisted by other towers. The observation of exactly two units being killed in all games can be explained with the preference of the towers for attacking the units with lesser HP, which refers to the two battlemages here.

The group with a SEW value of 4 attacks the first emplacement directly. Its synergy effect is strong enough to let the group act aggressively. The results show that at least one out of three towers is destroyed by the group (cf. Tab. III). Nevertheless, a big variance remains caused by the diversity of the damage. All possibilities—between two towers and four attacking units survive—occur.

Summarizing, it is clear that the intended behavior can be achieved by varying the SEW parameter, from defensive (12) to normal (8) to aggressive (4).

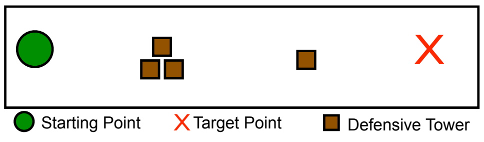

Fig. 5. Sketch of the map for the third scenario.

TABLE IV
DAMAGE MULTIPLIERS OF ATTACK TYPES AGAINST ARMOR TYPES IN THE BALANCED FACTIONS.

	Armor 1	Armor 2	Armor 3
Attack 1	0.5	5.0	0.5
Attack 2	0.5	0.5	5.0
Attack 3	5.0	0.5	0.5

Experiment 3: *Proof-of-principle; does the SOM generate the 'right' results in a simplified scenario?*

Preexperimental planning: According to the intransitive superiority principle, we designed a simplified scenario (balanced factions) with only three unit types employing the damage multipliers given in tab. IV. In-game experimentation confirmed that this leads to the desired behavior. For each of the three unit types, there is one favored counterpart performing best.

Task: By applying a full factorial design over the SOM parameters, a good SOM configuration shall be determined, accompanied by a parameter effect estimation. A parameter is only considered as important if the result set for one level significantly ($p < 0.05$) differs from the set for the other level in a Wilcoxon rank sum test (also U-test), likewise for the two-parameter interactions. For a well configured SOM, we require an average BMU fraction of at least 90%, compared to the optimal answer (homogeneous test cases with only one unit type).

Setup: We perform a full factorial design over extreme parameter values for the number of nodes on each axis of the SOM (dimension), the number of learning steps (iterations), the allowed unit number (unit), the number of learning patterns (always 3 in the homogeneous case), and the attacker group type, according to tab. V. Note that high unit numbers primarily increase the accuracy of the SOM, as it only uses fractions internally. For each configuration, the SOM is repeatedly trained 10 times and the resulting substitute objective function value is averaged over all (3) homogeneous and all (3) 2-unit equally shared mixed attacking groups. The BMU test is performed over four configurations. The largest SOM (dimension 16) is repeatedly (5 times) trained for 1000 iterations with 30 units and 100 mixed patterns and also homogeneous

TABLE V
PARAMETER VALUES FOR THE FULL FACTORIAL DESIGN

Parameter	dim.	iterations	units	patterns	attacker
Low value	2	100	10	100	homogeneous
High value	16	1000	30	1000	mixed

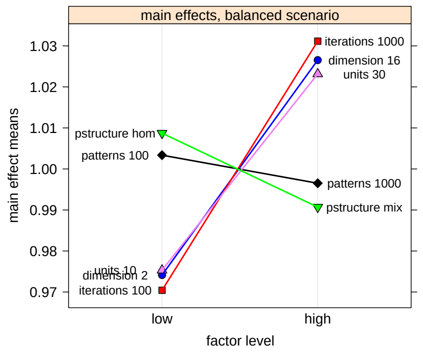

Fig. 6. Main effect estimations of parameters for the balanced scenario. All parameters except the number of patterns have a significant effect.

attacker groups, respectively. The smallest SOM (dimension 2) is trained under the same conditions, but only for 100 iterations and 10 units. The BMU fraction is computed from the averaged fraction of 'right' answers (archers for swordmen, swordmen for horsemen, and horsemen for archers) over the 3 homogeneous test patterns.

Results/Visualization: Main effects and 2-factor interactions are estimated via the Design of Experiments (DoE) standard procedure. We show all main effects in fig. 6 and omit the interactions as they do not provide additional insight. The BMU detection frequencies are given in tab. VI.

Observations: It is no surprise that increasing the SOM size and the numbers of iterations and units leads to better performance. However, the number of patterns seems to play no role at all (or 100 is already sufficient), and strangely enough, training with only homogeneous patterns is better than with mixed patterns. The interactions behave as expected, all combinations of the important factors also produce strong interactions. Overall, the effects are not very strong (about 6% maximum difference between best and worst). The BMU detection frequencies are all relatively high, even for the small SOMs with less training iterations. However, here the variance is quite large, being much smaller for large SOMs with more training iterations.

Discussion: The balanced scenario is easy to learn, and even the small SOMs cope with it acceptably (note that a SOM size of 2×2 is minimal as we have 3 unit types). We can only speculate why the performance with homogeneous test patterns is better than with mixed ones. A possible reason could be the reduced level of randomness: Homogeneous patterns may provide clear and stable guidelines, whether mixed patterns vary a lot. Deterministic training data seems to support learning here.

Concerning the BMU detection frequencies, the weaker results for small SOMs presumably stem from conflicts between neighboring nodes. The small SOMs often learn only two of the three optima. Additionally, large SOMs—initialized randomly—also have a head start because it is much more likely that already near optimal defense unit vectors can be found in 256 trials than in 4.

Overall, both results show that the SOM is reliable and easy to handle: It can be successfully trained with (few) homogeneous patterns, and even if the important parameters (size, iterations, and units) are set badly, performance decreases only gradually.

Experiment 4: *Does the substitute objective function properly represent simulation results on the real game?*

Preexperimental planning: Different variants of the objective function as presented in §III-D had been tried, and of the ones that intuitively gave the impression of performing well, we chose the simplest one.

Task: We apply Kendall's rank correlation test to paired samples of simulation and substitute function results (H_0 is that both data sources are indistinguishable) and expect a p-value of at most 5%. The computed correlation value should be positive and in the order of 0.5 or larger. Additionally, we expect that SOMs that have been trained with the substitute function give similar results as those trained with simulation.

Setup: We do not test with completely random attacking groups, as this situation would almost never occur in a concrete game (at least not while playing against humans). Instead, we generate only homogeneous or 2-type mixed attacking groups of size 10 each. The defending group is drawn randomly from all unit types, also containing 10 units (allowing randomness here makes sense as the composition of the defending group is the search space of the SOM). Much larger group sizes are infeasible for simulation as simulation time grows super linearly in unit numbers. We restrict our tests to tech level 3 (9 types; see exp. 3) which provides the largest search space for the SOM. When succeeding here, we assume that tech level 1 and 2 can also be modeled by the substitute objective function.

Results/Visualization: Table VII provides correlation estimates and the associated p-values of the tests, fig. 7 gives a visualization.

TABLE VI
BMU DETECTION FREQUENCY FOR EXTREMELY CONFIGURED SOMS

Dim.	iter.	units	patterns	attacker	average	std.
2	100	10	(3)	homogeneous	84.09%	1.968%
2	100	10	100	mixed	84.68%	7.685%
16	1000	30	(3)	homogeneous	98.97%	0.080%
16	1000	30	1000	mixed	99.02%	0.251%

TABLE VII
KENDALL'S RANK CORRELATIONS AND TEST P-VALUES BETWEEN SIMULATION AND SUBSTITUTE OBJECTIVE FUNCTION

Attacker	samples	p-Value	rank correl.
Homogeneous groups	360	$< 10^{-15}$	0.4234
Mixed groups	360	$< 10^{-15}$	0.4238
Homogeneous+mixed groups	720	$< 10^{-15}$	0.4298

Table VIII shows a comparison of BMUs of SOMs that were differently trained for 10 iterations. We use homogeneous groups as test inputs here. The highest weighted units in the BMU for the actual input are shown in the table.

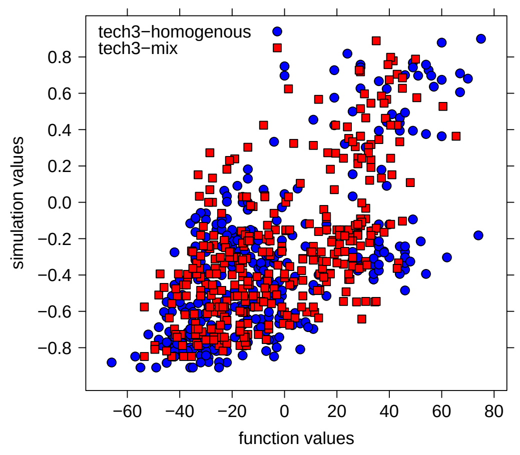

Fig. 7. Correlation plot: Simulation result over substitute objective function, 360 random samples of defenders against homogeneous and mixed attacker groups, respectively. Theoretical minima are -1.0 for the simulation and -100 for the objective function.

Observations: Both objective functions lead to SOMs that favor air units. While the imbalance is greater in the SOMs with substitute objective function, we cannot distinguish after 10 training iterations if training is only slower with simulation or if simulation is indeed less prone to favor air units.

The rank correlation values (tab. VII) hardly show any difference between homogeneous and mixed samples, which is unexpected. Figure 7 largely adheres to our expectations. Still, it reveals some interesting details. At first, the simulation minimum (-1.0) is nearly reached, but objective function values stay far from -100, its minimum. Obviously, the objective function is more *accurate* for small values, so that simulation results can already be nearly optimal even if the optimal unit composition—according to the objective function—is not yet reached. Secondly, very good objective function values possess a very low variance in simulation values—this seems to support the previous observation. Thirdly, the -/- quadrant is very dense, meaning that there are many more good than bad (randomly drawn) defending unit compositions.

Discussion: The missing difference in rank correlation values between mixed and homogeneous samples points to another source of variation with a stronger influence. Most likely, this is the inherent randomness of the simulation (damage values are partly random). We may interpret that as a quality indicator of the substitute objective function. Despite the noise in simulation data, the correlation is approximately on the required level. Applying other rank correlation metrics (e.g. Spearman) even leads to slightly higher values. The test values clearly state that the null hypothesis (no correlation) should be rejected. We deduce that the substitute objective function is sufficient for SOM training, and that due to its deterministic nature, it may be even better to use this function than to learn via simulations. Another interesting revealed fact is that there are very many good answers to a specific attacker group. Obviously, it pays off to send different troops into combat than used by the aggressor.

TABLE VIII
SUGGESTIONS OF DIFFERENTLY TRAINED SOMs

Input Group	Substitute function	Simulation
Airship	Ornithopter	Ornithopter
Archer	Ornithopter	Swordman
Armored Swordman	Airship	Ornithopter
Battle Machine	Airship	Archer
Catapult	Ornithopter	Ornithopter
Horseman	Airship	Catapult
Ornithopter	Ornithopter	Ornithopter
Swordman	Ornithopter	Armored Swordman
Technician	Ornithopter	Ornithopter

Experiment 5: *Do the SOMs perform well in a more complicated real game setting?*

Preexperimental planning: Our first experiments with the full unit set resulted in SOMs clearly favoring air units. Thus, we partitioned the units from standard `Glest` into three levels. Tech-Level 1 is similar to our balanced faction from experiment 1 plus the armored swordman (but with the original damage multipliers). It consists of archer, armored swordman, horseman and swordman. Tech-Level 2 adds battle machine, catapult and the technician to Level 1 whereas the air units airship and ornithopter are added in Level 3. This is consistent with the game flow, as these are the 3 sequential phases usually experienced in `Glest`. We thus not only get three levels of difficulty as test cases for the SOM, but also 3 different types of SOMs which may each assist a `Glest` AI in the sequential game phases and be switched as new units become available.

Task: Again, we strive for finding the significant main and interaction effects of the SOM configuration as in exp 1. We aim for obtaining information about the robustness of the parameter settings in exp. 1. For being robust, we demand that the parameter recommendations from exp. 1 are suitable for exp. 3 also, or vice versa, so that at least one robust parameter setting exists. Additionally, we measure the SOMs performance by comparing its suggested defense group against the precalculated, optimal solutions obtained from total enumeration. The SOM, if properly configured, should reach at least 90% of the optimal score on average.

Setup: In order to obtain reference values for comparing the SOM performances on the substitute objective function, we enumerated the search space for each tech-level to obtain optimal defending groups for homogeneous as well as for 2-type-mixed attacking groups of size ten. For these more complicated game situations, we have no means to otherwise obtain the optimal answer of the SOMs as by optimization. For exp. 1, the scenario had been constructed in a way that the optimal answer was obvious. As the worst substitute objective function values—0 stands for a draw—are around 100 (see fig. 7, smaller is better), we compute the relative quality of a SOM answer after formula 3. We perform 31 repeats for each SOM configuration to get an impression of the underlying distributions. The overall setup for the SOMs is the same full

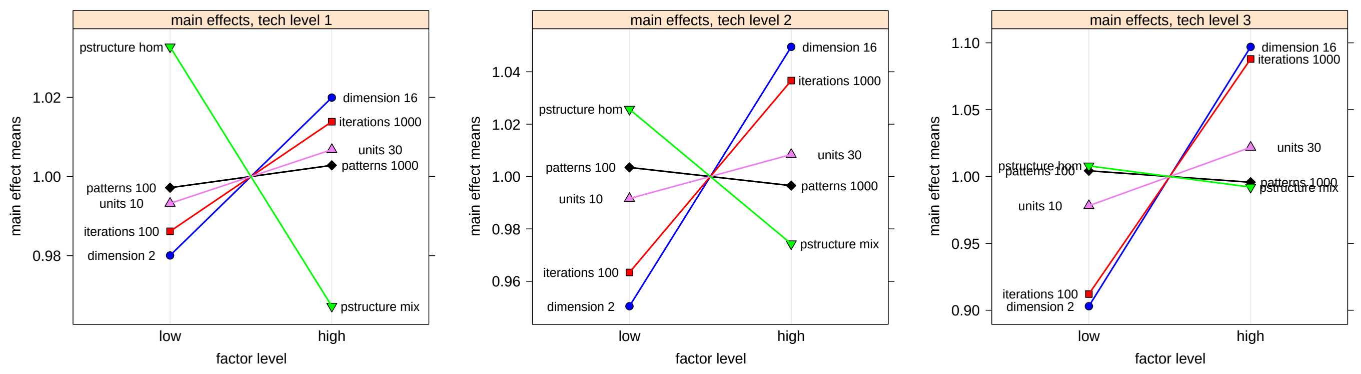

Fig. 8. Main effect estimations of parameters for the tech level 1 scenario (4 units, left) and tech level 2 (7 units, middle). All effects are significant, except pattern number and unit size. In the tech level 3 scenario (9 units, right), all except pattern size and pattern structure are significant.

factorial design as in exp. 1.

$$q_{rel} = \frac{f_{\text{subst}}(som) - 100}{\text{optimum} - 100} \tag{3}$$

Results/Visualization: Figure 8 documents the measured main effects for tech levels 1 to 3. The obtained 2-factor interaction effects are rather weak and thus omitted. Table IX shows the average relative answer qualities of 2 different SOMs for tech level 3, before and after training. The SOMs are tested on all 9 homogeneous and all 36 2-unit equally shared mixed attacker groups.

Observations: In tech level 1, the pattern structure effect is remarkably strong, which is interesting as the level is not much different to the balanced scenario, where the effect is of minor importance. The other effects are slightly weaker than for the balanced case. In level 2, the SOM size and iterations effects take over again, and the overall effect sizes increase to a difference of about 10% between best and worst. For tech level 3, the SOM size and iterations effects are already very dominant and their absolute values are very high, leading to a 20% relative difference between best and worst configurations.

The well trained large SOMs approximate the optimum nearly perfectly, whereas the smaller SOMs barely reach 80% of the optimal substitute objective function value. Interestingly, the random (untrained) SOMs also reach relatively high qualities, although they stay well below the trained SOMs. Training—in our context equivalent to applying an optimization—undoubtedly has an effect.

Discussion: Regarding to the sought good and robust parametrization of the learning process, we can state that the parameter effects are nearly never contradictory, only their strength varies. For the more demanding scenarios with more unit types, a larger SOM clearly is an advantage, accompanied by a larger number of training iterations. Applying only homogeneous training patterns is better for the simpler scenarios, whereas it loses its importance for the more complex ones. Summarizing, one may always set the SOM size, iteration and unit numbers to high values, and train only with the homogeneous patterns. The low effect of the training patterns for tech level 3 could be interpreted as difficulty to produce *meaningful* mixtures with many unit types, too many of the randomly created patterns may be unreasonable and can thus not support the learning process. The comparison of the relative quality reached by different SOMs over the whole spectrum of 45 test cases for tech level 3 indicate that the large, trained SOM is clearly superior to its small and untrained counterparts, and shows little variance, making it very reliable. In some cases, the optimal defending group has not been attained. Here the EA based optimization may not be strong enough and may be developed further. However, the big picture is that the given answer is quite close to the optimal one.

To clarify what the SOM actually learns in terms of suggested defense unit groups, we additionally plot the highest rated unit type for each SOM node in three subsequent learning states for a well configured SOM on tech level 3 in fig. 9. Only few (highest rated) unit types remain after 1,000 iterations, and most nodes favor air units (ornithopters and airships). The domination of these air units for tech level 3 could have several reasons. Firstly, it may of course be intentional as the more advanced units shall be stronger. Adding to this, we completely neglect cost and speed and many other unit attributes in our substitute objective function, which may compensate for unbalanced attack strengths in the game. Secondly, the majority of unit types has only short-range attack skills, which reflects in the training patterns. Air units have the advantage that they cannot be attacked by short-range units. So, ensuring that skills and features are uniformly distributed in the training patterns might help. Another alternative would be to record group constellations from playing humans and to use this data as training input.

TABLE IX
RELATIVE ANSWER QUALITY OF WELL AND BADLY CONFIGURED TRAINED AND RANDOMLY ESTABLISHED SOMS (ITERATIONS=0) COMPARED TO THE OPTIMAL VALUES FOR TECH LEVEL 3

Dim.	iter.	units	patterns	attacker	average	std.
16	1000	10	(9)	homogeneous	94.32%	8.52%
16	0	10	(9)	homogeneous	77.15%	11.04%
2	100	10	(9)	homogeneous	81.68%	9.85%
2	0	10	(9)	homogeneous	65.32%	11.14%

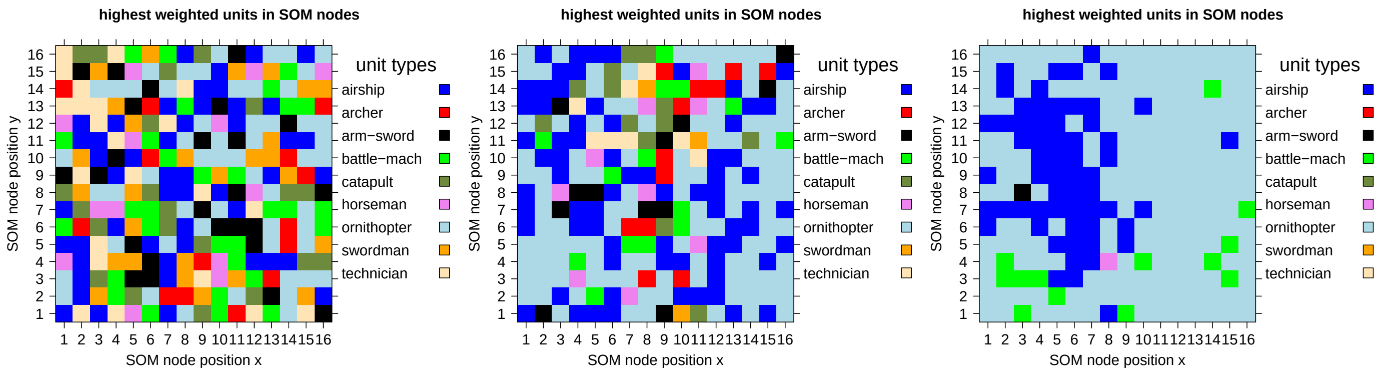

Fig. 9. Evolution of a `Glest` tech level 3 SOM (16 × 16, 30 units, 9 homogeneous patterns) during the training process. From left to right: After 10, 100, and 1000 iterations, respectively. Learning ends with a clear domination of air units.

Experiment 6: *Do all techniques work well together?*
Preexperimental planning: From the experiences of the previous experiments, we designed a scenario that enables the attacking team to win if all techniques are successfully used (Flocking+IM and SOM selection). This scenario is depicted in fig. 10 and is balanced so that if the best matching units are sent as flock along the right path to either of the two defending groups, the game is nearly always won by the attacker.

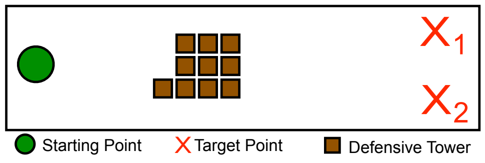

Fig. 10. Underlying map for the combined scenario.

Task: We want to reject the hypothesis that with Flocking+IM switched on, SOM selection does entail a large change in results.
Setup: The allowed units comprise the three types from the balanced scenario, in which swordmen beat horsemen, these beat archers, and archers in turn beat swordmen. We put 10 towers in the middle of the scenario (fig. 10) which have to be bypassed. At X1, 7 Horsemen und 3 Swordmen await the attackers, and at X2, 7 Archer und 3 Horsemen are placed. The SOM gets free choice of 10 units to send against X1, and again 10 units to send to X2 (the reasoning here is to strike at both targets at roughly the same time). We run 500 games with the chosen units and then reverse the attacking groups, thereby sending them at the wrong targets, or simulating a wrong SOM group selection.
Results/Visualization: Figure 11 shows the distribution of surviving units of attackers and defenders under SOM group selection and under reversed group selection.
Observations: The SOM selects 10 swordmen to capture X1, and 10 horsemen to X2. All resulting games are won by the attacker, and the defenders of X2 are always annihilated. At X1, in 150 (≈ 31%) of the games, some of the defenders survive, but these are always outnumbered by the surviving attackers. Visual observation of some games suggests that sometimes, the attackers stop fighting as they cannot locate all defending units, or the battle is just not finished when automatically terminated.

If the attacking groups are reversed, (10 horsemen to X1 and 10 swordmen to X2), the attacker always loses, and retains some (few) units only in 84 games. This seems to be due to the same effect of localizing the opponent group as above. Note that the defense towers in the middle are never attacked but circumvented by the attackers.

Discussion: We can clearly state that the SOM selection leads to a huge change in results and thereby reject the hypothesis. Indeed, we have shown that all three techniques (Flocking, IM, and SOM) can work well together. If observed by a human player, the obtained group behavior may even be called "intelligent". We regard the reported problems (some defending units are not found) as minor and of technical nature; they refer to implementation details depending on the used game.

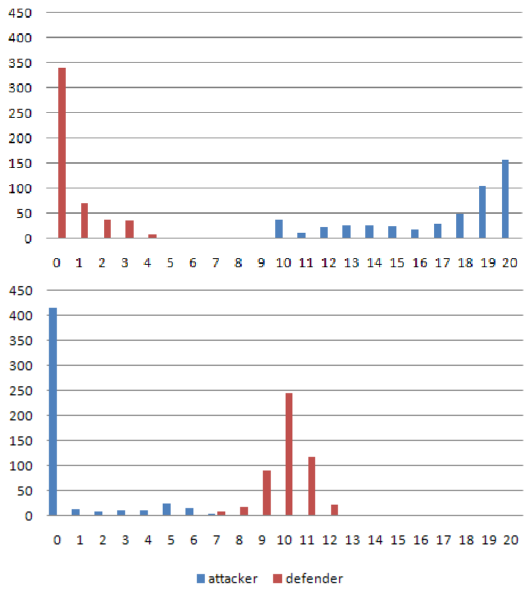

Fig. 11. Surviving attackers (blue) and defenders (red) of 20 units, respectively, in case of SOM group selection (top) and with reversed group selection (bottom) over 500 games each.

V. Conclusion and Outlook

We have shown by thorough experimental analysis that the two major proposed approaches, flocking with influence maps and SOM group selection, can straightforwardly be applied in an RTS and decidedly improve the previously experienced behavior of groups.

SOMs support a structured, content-addressable storage of information. They can be trained offline during the production of a game and remain static while playing. Given that a substitute objective function for the suitability of a group when combatting another can be found, training a SOM is a matter of a few minutes even for the largest investigated configurations. Creating such a function was very easy at least in our game context, and it may also be regarded as possible way for the game designer to include subjective preferences (which is not explored here). Training by means of the game itself as evaluation function is also possible but may be very computationally expensive, easily increasing training times from minutes to the order of days. However, the obtained results do not seem to differ that much, so that this approach may be unreasonable. The SOMs are also easily extendable: If different conditions like new unit types or upgrades occur within a played game, matching SOMs for these can be pre-produced and game AI just has to switch between them.

Flocking in combination with IM path-finding improves group performances in every of the game situations investigated here and tackles the shortcomings detected for either technique alone. It seems to be quite robust, makes group movement look more natural, and is flexible: By setting the SEW parameter, the AI or also a human player can change the level of aggressiveness of a group within a wide range.

As well as either approach (flocking+IM and SOM) works well on its own, they are also complementary and team up very well. The SOM decides which group has to move where, and the flocking+IM accomplishes that. In addition to measurably good group performance, the resulting behaviour also looks "intelligent" for human observers, thereby resolving a nuisance experienced by many players of RTS games. Of course, the two approaches are not restricted to AI use. Human players may also benefit from having their groups moved in a reasonable way once they order them to go somewhere. Given that a currently unattended human player base is attacked, it could be left up to a SOM to decide how to distribute available defenders, but even with the player overlooking the current battle, he could be provided help by the SOM when grouping defenders or even building more.

There are several possible directions for extending this work. For example, the influence calculation may be improved by employing more unit and game mechanic characteristics. The SOMs could be combined with rule-based systems, which still prevail in commercial games; and their inherent use of an evolutionary algorithm may also be investigated, possibly resulting in further training speedup. Last but not least, what we have shown is just a proof-of-concept, and it will be an important next step to replicate the observed behavior within another RTS game.

Acknowledgment

This work was kindly supported by the *Deutsche Forschungsgemeinschaft (DFG)* and the *Federal Ministry of Economics and Technology (BMWi)*. We thank the student project group 511 for developing the basics of this work.

References

[1] D. Livingstone, "Turing's test and believable ai in games," *Comput. Entertain.*, vol. 4, no. 1, p. 6, 2006.

[2] M. Deloura, *Game Programming Gems*. Boston, MA: Charles River Media, 2000.

[3] S. J. Louis and J. McDonnell, "Learning with case-injected genetic algorithms." *IEEE Trans. Evolutionary Computation*, vol. 8, no. 4, pp. 316–328, 2004.

[4] S. J. Louis and C. Miles, "Case-injection improves response time for a real-time strategy game." in *IEEE Symposium on Computational Intelligence and Games (CIG 2005)*, G. Kendall and S. Lucas, Eds. Piscataway, NJ: IEEE Press, 2005.

[5] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games," in *AI and Simulation in Games (GAME-ON 2007)*, M. Roccetti, Ed. Ostend, Belgium: EUROSIS, 2007, pp. 61–68.

[6] M. J. V. Ponsen, P. Spronck, H. Muñoz-Avila, and D. W. Aha, "Knowledge acquisition for adaptive game ai," *Sci. Comput. Program.*, vol. 67, no. 1, pp. 59–75, 2007.

[7] H. Danielsiek, R. Stüer, A. Thom, N. Beume, B. Naujoks, and M. Preuss, "Intelligent Moving of Groups in Real-Time Strategy Games," in *IEEE Symposium on Computational Intelligence and Games (CIG 2008)*. IEEE Press, Piscataway NJ, 2008, (in print).

[8] N. Beume, T. Hein, B. Naujoks, N. Piatkowski, M. Preuss, and S. Wessing, "Intelligent Anti-Grouping in Real-Time Strategy Games," in *IEEE Symposium on Computational Intelligence and Games (CIG 2008)*. Piscataway, NJ: IEEE Press, 2008, (in print).

[9] C. W. Reynolds, "Flocks, herds and schools: A distributed behavioral model," in *Computer graphics and interactive techniques (SIGGRAPH 1987)*. New York,: ACM Press, 1987, pp. 25–34.

[10] P. Tozour, "Influence mapping." in *Game Programming Gems 2*, M. DeLoura, Ed. Boston, MA: Charles River Media, 2001, pp. 287–297.

[11] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Transactions on Systems Science and Cybernetics SSC4*, pp. 100–107, 1968.

[12] M. Figueroa, J. González, T. Fernández, F. Menéndez, and M. Caruncho, "Glest, a free 3d real time strategy game," 2007, december 16, 2007. [Online]. Available: http://www.glest.org

[13] GNU Software Foundation, "Gnu general public license, version 2," June 2001, november 18, 2007. [Online]. Available: http://www.gnu.org/licenses/gpl-2.0.html

[14] Z. Shen and S. Zhou, "Behavior representation and simulation for military operations on urbanized terrain," *Simulation*, vol. 82, no. 9, pp. 593–607, 2006.

[15] A. Davison, *Killer Game Programming in Java*. Cambridge, MA: O'Reilly Media, 2005.

[16] A. Kirmse, Ed., *Game Programming Gems 4*. Boston, MA: Charles River Media, 2004.

[17] J. Flensbak, "Flock behavior based on influence maps," Department of Computer Science, University of Copenhagen (DIKU), Denmark, Bachelor thesis, 2007.

[18] C. Miles and S. J. Louis, "Towards the co-evolution of influence map tree based strategy game players," in *2006 IEEE Symposium on Computational Intelligence and Games (CIG 2006)*. Piscataway, NJ: IEEE Press, 2006, pp. 75–82.

[19] ——, "Co-evolving real-time strategy game playing influence map trees with genetic algorithms." in *Congress on Evolutionary Computation (CEC 2006)*. Piscataway, NJ: IEEE Press, 2006.

[20] T. Kohonen, *Self-Organizing Maps*. Berlin: Springer, 2001.

[21] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies - a comprehensive introduction," *Natural Computing*, vol. 1, no. 1, pp. 3–52, 2002.

[22] T. Kohonen, "Fast evolutionary learning with batch-type self-organizing maps," *Neural Processing Letters*, vol. 9, no. 2, pp. 153–162, 1999.

[23] M. Preuss, "Reporting on experiments in evolutionary computation," Dortmund University, Tech. Rep. Reihe CI 221/07, SFB 531, 2007. [Online]. Available: sfbci.uni-dortmund.de/Publications/Reference/Downloads/22107.pdf