Towards Intelligent Team Composition and Maneuvering in Real-Time Strategy Games

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

***Abstract*—Players of real-time strategy (RTS) games are often annoyed by the inability of the game AI to select and move teams of units in a natural way. Units travel and battle separately, resulting in huge losses and the AI looking unintelligent, as can the choice of units sent to counteract the opponents. Players are affected as well as computer commanded factions because they cannot micromanage all team related issues. We suggest improving AI behavior by combining well-known computational intelligence techniques applied in an original way. Team composition for battling spatially distributed opponent groups is supported by a learning self-organizing map (SOM) that relies on an evolutionary algorithm (EA) to adapt it to the game. Different abilities of unit types are thus employed in a near-optimal way, reminiscent of human *ad hoc* decisions. Team movement is greatly enhanced by flocking and influence map-based path finding, leading to a more natural behavior by preserving individual motion types. The team decision to either attack or avoid a group of enemy units is easily parametrizable, incorporating team characteristics from fearful to daredevil. We demonstrate that these two approaches work well separately, but also that they go together naturally, thereby leading to an improved and flexible group behavior.**

***Index Terms*—Evolutionary algorithms (EAs), flocking, influence maps, neural networks, path finding, real-time strategy games, tactical decision making.**

I. Introduction

THE real-time strategy (RTS) genre recently developed some of the most successful and also most complex computer games. It may also be regarded as one of the roots of the popular massive multiplayer online games (MMOG) such as *World of Warcraft*. However, while graphics and sound in RTS games have been improved dramatically over recent years, opponent strategies, or more generally, the whole game AI has not advanced that quickly. This may be partly due to the complexity of the games coupled with the real-time component: game AI has to cope with many different subproblems (e.g., resource 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, path finding in a dynamic game setting with moving objects cannot be fully precomputed, and "teleporting" foot unit looks strange to human players. However, game AI should perform well and still look believable [1] in order to provide fun for human players.

Bio-inspired techniques [computational intelligence (CI)] have mostly been integrated into simple computer games, e.g., arcade video games, for improving the game's artificial intelligence such as the intelligence of nonplayer characters (NPC) or the handling of the game [2]. They are especially helpful in complex situations that cannot be solved in reasonable time by more accurate but slower methods and can also deal well with incomplete knowledge or noisy conditions [3]. Hence, they are increasingly employed also in RTS games, one of the most complicated but popular game types around. There have been several attempts to produce game AI that is capable of adaptive behavior and employ CI techniques in one or another way, e.g., [4], [5], [6], and [7].

Detecting the opponent's grouping behavior is a subproblem of opponent modeling, which has been addressed by Schadd *et al.* [8] as well as knowledge acquisition in RTS games in general by Ponsen *et al.* [9]. Sailor *et al.* [10] attempt to take the possible actions of an opponent into account to adapt their own maneuvering strategy where game tree methods are not applicable due to vast action spaces and very limited time.

Formation and movement of groups in RTS games is not trivial and has recently gained more attention, e.g., by van der Heijden *et al.* [11], and Hagelbäck and Johansson [12]. In [13], Hagelbäck and Johansson establish a similar approach that copes with very limited information. Thompson and Levine [14] attempt to construct meaningful complex unit behaviors by means of a layered agent controller architecture. Lichocki *et al.* [15] strive to achieve cooperation of units in their foraging behavior.

While developing a complex game plan in a single attempt appears rather hard, using the methods for subsidiary in-game decisions—especially for team composition and maneuvering—seems manageable. We here combine and extend two recent approaches of ours, namely [16] and [17].

Manuscript received October 24, 2008; revised July 05, 2009; accepted March 09, 2010. Date of publication April 05, 2010; date of current version June 16, 2010. This work was supported by the Deutsche Forschungsgemeinschaft (DFG) and the Federal Ministry of Economics and Technology (BMWi).

M. Preuss, N. Beume, H. Danielsiek, T. Hein, R. Stüer, A. Thom, and S. Wessing are with the Computational Intelligence Group, Department of Computer Science, Technische Universität Dortmund, Dortmund 44227, Germany (e-mail: mike.preuss@tu-dortmund.de; nicola.beume@tu-dortmund.de; holger.danielsiek@tu-dortmund.de; tobias.hein@tu-dortmund.de; raphael.stuer@web.de; andreas.thom@tu-dortmund.de; simon.wessing@tu-dortmund.de).

B. Naujoks was with the Computational Intelligence Group, Department of Computer Science, Technische Universität Dortmund, Dortmund 44227, Germany. He is now with Log!n GmbH, Schwelm 58332, Germany (e-mail: boris.naujoks@tu-dortmund.de).

N. Piatkowski is with the Computational Intelligence Group, Department of Computer Science and the Artificial Intelligence Unit, Technische Universität Dortmund, Dortmund 44227, Germany (e-mail: nico.piatkowski@tu-dortmund.de).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCIAIG.2010.2047645

Maneuvering of unit teams greatly benefits from flocking, a technique introduced by Reynolds [18], and influence maps (IMs) [19] to improve the path finding of units. The latter may be left to the A* algorithm [20], but from a strategic perspective, the results can be poor, e.g., if calculated paths are leading through enemy territory or the teams are split up on their way. Such problems can be overcome by flocking which is introduced in Section II-A. Although already performing well, we show that flocking alone is not able to cope with all eventualities. However, flocking teams up well with influence maps, area-based aggregations of the game AI's available data which are introduced in Section II-B. Note that the use of flocking+IM is not restricted to groups run by the game AI: it may also be applied to human controlled groups.

Another recent approach to find walkable paths for many bots is described in [21] and [22]. Contrary to our approach, Geraerts *et al.* construct a separate path for each steered unit and not one for the whole group. In robotics, there are several related approaches for controlling team maneuvers of which we only refer to one very recent one by Takács and Demiris [23]. An overview for existing crowd simulation solutions can be found in [24]. Van der Sterren [25] describes an approach to extend the standard A* algorithm to calculate tactical paths quite similar to ours. The costs of the A* algorithm for movement through locations subjected to enemy observation or fire have been increased with the aim that single units avoid enemy observation by using safe paths. However, his approach does not consider group movement or the balance of power between the different factions.

Another problem for game AI is the composition of teams for attacking a certain target or defending against a specific enemy group. We treat it by parallel learning (instead of training) of counterteams for the incoming (team composition) requests by means of a variant of self-organizing maps (SOMs) as introduced in Section II-D. SOMs can be prepared offline by using simple substitute objective functions (instead of game simulations) and are able to perform in-game unit selection almost instantly.

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 Section III). We end with the most interesting experiment from our point of view, where all proposed techniques act in unison. Teams behaving according to the proposed techniques—if beheld by a human—appear to act much more "authentically" or believably, although we are aware that such properties are very hard to measure.

II. Techniques

Flocking, influence maps, and SOMs 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., flocks of birds, schools of fish, military units, or crowds. A flock is able to reach a certain target as a group avoiding 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 [26] in *Unreal Tournament* as a testbed. Davison [27] indicates that the commercial releases of *Half-Life* and *Unreal Tournament* also employ flocking.

Reynolds [18] 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

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

where v_{ref} is the reference vector giving the direction of the calculated flock path (from the center of the flock to the next position in the computed A* 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.

Our implementation is different from Reynolds' original idea to have boids moving around randomly and eventually building a flock [18]. 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 [20] already implemented in *Glest*. Flocking is used for collision avoidance while all units follow the same path.

In our scenario, the most significant parameters of flocking are the angle of sight α (used for separation and alignment, not for cohesion), the separation weight w_{sep}, and the distance of separation determined by the length of the separation vector $|v_{\mathrm{sep}}|$. 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 loosely. The separation, i.e., the ambition of every boid to move away from other boids, depends on w_{sep} and $|v_{\mathrm{sep}}|$. Details may also be looked up in [28].

B. Influence Maps

IMs are an area-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

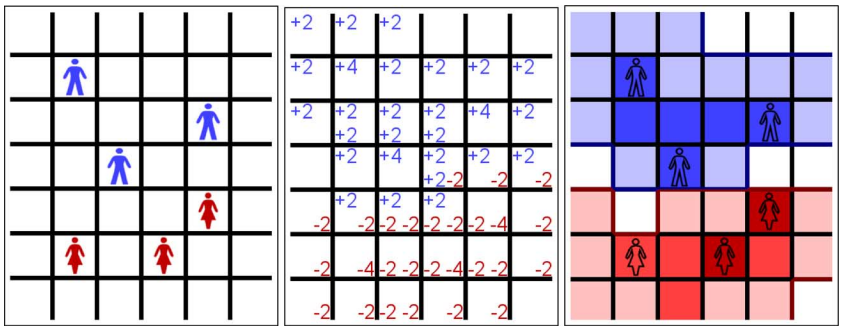


Fig. 1. Example of influence map generation. The first figure shows the units of two different factions (male, blue and female, red) placed on the map. The second figure yields the local influence of the units ($+4$ and -4, respectively) and their propagation of the influence values through the map. 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.

out where opponent units are concentrated or explore weak, exposed areas. We invoke 2-D environments only; a broader approach regarding different topologies can be found in [29].

Usually, a map is divided into tiles. The IM is set to the same size as the game's map with typically fewer tiles. A certain property of a unit is used as influence values, e.g., hit points (HPs) 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 for absolute values ≤ 1. The propagation is shown in the central image of Fig. 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 HPs and the damage D a unit is able to cause to its combat strength c_u as

$$c_u = \frac{D_u \cdot \mathrm{HP}_u}{100}$$

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

$$i(x,y) = \frac{\zeta_{x,y} - c_u}{180}$$

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

The unit's combat strength c_u is subtracted to allow the unit to pass tiles with low hostile influence; 180 was experimentally found as good scaling value. However, this is a subjective result; game designers with different preferences concerning movement may have to try out different values. 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

$$c_F = \frac{1}{\mathrm{SEW}} \cdot \sum_{u \in F} \mathrm{HP}_u \cdot \sum_{u \in F} D_u.$$

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 account for 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 when 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 [30] uses three 2-D 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 [31], [32]. 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. SOMs for Team Composition

Team composition or rather unit selection for the creation of teams with specific tasks is of tactical as well as of strategic use.

- Tactical attack: If an enemy base or unit group is to 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—will 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 thoroughly balanced unit types with different abilities, we often find a situation of *asymmetric superiority* (e.g., A beats B, B beats C, and C beats A as in the *Rock, Paper, Scissors* children's game; this has also been termed *intransitive superiority* in [33]). Here, no simple optimal strategy exists; we need a possibility to respond to any opponent group composition, a context-sensitive memory. When considering heterogeneous teams, i.e., consisting of many different types of units, instead of homogeneous unit groups, the situation easily becomes confusing, so that setting up a simple system of rules is no more appropriate.

SOMs, conceived by Kohonen [34], are special artificial neural networks that can deliver such a context-sensitive memory. We apply them in an innovative way so that they perform a highly effective classification for complex group settings. Having learned relations of groups of units in an unsupervised offline mode, our trained SOM is ready to be invoked within a game to assign combat groups to one another *ad hoc*. This technique is of general interest for RTS games.

The basic idea of a SOM is to assign high-dimensional data points to points in a (usually) 2-D map in such a way that similar input vectors are located close to each other. Each node of the map is visualized by a certain color reflecting the properties of the associated data vector. For example, 3-D vectors can be displayed by their represented RGB color, higher dimensional data points by a color encoding their highest value. This way, SOMs produce a meaningful clustering and visualization of a complex data set by dimension reduction. SOMs may also learn a certain function as suggested by Kohonen [35] instead of a distance function and this is how we apply them. So, groups given as input vectors are not to be mapped to preferably similar vectors but to groups which perform well against them and those do not have to be composed alike. This way, we perform an optimization rather than a classification, whereas clustering still plays its role.

A SOM consists of spatially embedded nodes. We employ a toroid-shaped lattice with connected opposite edges. Each node a_i of the lattice is associated with a randomly initialized reference or weight vector $\mathbf{w}_i$ of length n. The input vector $\mathbf{x}$ (also length n) specifies the distribution of n unit types over an opponent's army. Likewise, the weight vectors w_i represent a composition of the defending player's unit types. The size of the map equals the number of responses that can be learned, e.g., a 4×4 SOM stores 16 response team compositions. A trained SOM outputs that group composition—of those stored in its nodes—which would perform best against the attacking group given as input. For easier readability, the following description sticks to the perspective that the input groups are the attacking one and the SOM groups the defending ones. However, our method does not distinguish between attackers or defenders and thus can be applied to the opposite scenario as well.

Basically, the training or learning algorithm conforms to the original algorithm devised by Kohonen [34]. An overview is given in Algorithm 1 where the basic steps are written in normal font and our special details in parenthesized italics. First, an appropriate training set has to be generated which includes typical and important input vectors (Algorithm 1, step 1). Afterwards the weight vector of each node has to be initialized which we do uniformly at random (Algorithm 1, step 2). The training loop is then performed for a chosen number of $t_{\max}$ iterations. In each iteration t, an input vector $\mathbf{x}$ is randomly drawn from the training set using a desired probability distribution over the input set, where we use the uniform distribution. While traversing the SOM's nodes, the suitability measure between the input vector $\mathbf{x}$ and each node's weight vector $\mathbf{w}_i$ is calculated (Algorithm 1, step 3). The applied measure can be an arbitrary function that maps a pair of vectors to a scalar value, often a distance function. In our case, it is the success of the response team of the node while battling the input team. The node with the highest suitability concerning the input vector, thus the node that matches the input vector best, is called the

best-matching unit (BMU) (Algorithm 1, step 5). The BMU's weight vector is then adapted to match the input vector even better (Algorithm 1, step 6). Nodes within a defined lattice radius r around the BMU node in the toroid are adjusted as well. The intensity ℓ of the adaption is defined by a Gaussian curve centered at the BMU and declining with increasing lattice distance to it. The neighborhood size shrinks with increasing iterations (Algorithm 1, step 8). The nodes' colors are updated so that each node is displayed in the color that is associated with the highest values of its vector which here corresponds to its major unit type. To query the SOM works similar to the training process but the SOM remains unchanged. For the query vector, the BMU is determined and given as the SOM's response.

In the following, we detail the specialties of our work: the suitability measure and the adaptation method. The standard suitability measure is the (weighted) Euclidean distance to quantify similarity of the vectors. Our measure is designed to rate combat suitability instead. Consider the following situation: during the game, the AI gets aware that the opponent is about to attack with a certain group of units A. A part of the problem to find the optimal defending group is the question: How successful would a certain group D be when facing the attackers?

The answer is the suitability and this is used as the distance measure inside the SOM. The canonic way to generate an answer would be to execute the game with the certain setup of groups and to record which group wins the battle. These game simulations would make the SOM training very time intensive and would of course make the application of the SOM for *ad hoc* decisions during a game impossible. As a fast alternative, we developed a substitute function $f(A, D)$ based on the properties of the game's units. The function returns a score of an attacking group A (the input for the SOM) against a defending group D (stored in one of the SOM's nodes). It calculates how much damage would be done to the groups when each unit performs one attack.

A unit can have several attack skills. Every skill s is evaluated with a score c_s against the set of attackable units A_s in the opponent's group A. $dm(t_s, r_a)$ is the damage multiplier of attack type t_s against armor type r_a of hostile unit a

$$c_s = \sum_{a \in A_s} dm(t_s, r_a). \tag{1}$$

Every unit d in group D is awarded a score c_d for its most useful attack skill only. c_d is calculated as

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

A group's score $f(A, D)$ then is the sum of all its units' scores c_d. To account for the mutual strengths and weaknesses, the final suitability function is $(f(D, A) - f(A, D)) \rightarrow \min!$.

Obviously, the function takes neither costs like time and resources nor features like speed or HPs into account. Finding an optimal group under these additional constraints would be a multicriteria problem that is not covered by our approach. Our objective function only considers aspects concerning the *relative* strength of units. Still, we assume that the function shows sufficient correlation to the in-game simulation. This is tested in Section IV.

If the SOM approach is utilized in another game, the game designer just has to set up such a substitute function that generates approximate results for the battle of two unit groups, and the SOM takes care of selecting proper groups. Otherwise, it would be necessary to design a complicated rule set solving the unit assignment problem. Note that this also has the advantage of scalability: if a new unit is added to the game, it suffices to capture its combat strength in the substitute function (which will usually be the case without any changes). No other specific treatment as adding new rules is needed.

In the training process, the defending groups in the SOM's nodes have to be adapted to the input attacking group (Algorithm 1, step 6). While this is a trivial task with the standard Euclidean distance measure, we need to optimize the suitability through the substitute function. Here, an evolutionary algorithm (EA) carries out the optimization of the defender's composition of units. An introduction to EA is given by Beyer and Schwefel [36]. Incorporating an EA into a SOM has already been proposed by Kohonen [34], [35]. Each time a node has been identified as BMU during the training, an $(1+1)$ EA is executed. It is initialized with the node's weight vector and uses the suitability measure (substitute function) as objective function. By slight changes of this vector (meaning slightly different output groups), it heuristically searches for an improved performance against the input group. 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 an analogy to the learning rate often used with a Euclidean distance. For the BMU, the EA is run for ten function evaluations. For the other nodes to be adapted, the EA is run for $\lfloor \ell \cdot 10 \rfloor$ function evaluations, whereas $\ell < 1$ declines with increasing lattice distance to the BMU, which reflects the decreasing influence of the adaptation.

Note that the SOMs as used here may be adapted to the game by means of game runs (simulation) or by calls to the substitute function. However, for the query at runtime within a game, one has to rely on the substitute as one cannot run short games within a game to find the BMU. For the query, the possible responses (one for each node of the SOM) the teams set up during training are tested for matching the current query best. Finding the BMU thus resembles answering the question: Which of the preset teams in the SOM should be used? One could however apply some similarity measure to compare the in-game queries to the ones that were received for a specific node during training. Nevertheless, we feel that such complicated "workaround" is not necessary and one can stay with the simple approach as long as the substitute function is reasonably correlated to the result obtained by game runs, which is investigated in Experiment 4.

Also note that it is not a problem for the approach if the BMU cannot be realized due to lack of specific units. A query obtains an ordered list of suitable unit groups and one can work down the list and match it with the available units to find the one that is the best possible and can be realized.

```
generate representative input set for training
(input set: vectors defining composition of attacking opponent group)
initialize nodes' weight vectors randomly
(weight vectors: composition of player's defending groups)
repeat
    choose vector x probabilistically from input set
    (choose x uniformly at random)
    for all nodes a_i do
        compare x with node's weight vector w_i by suitability measure
        (usually: distance, here: relative strength of defending group of w_i against attacking group x)
        save node with minimal value as BMU for x
    adapt BMU and neighboring nodes inside radius r acc. to better fit to x
    (adapt defending groups by optimizing their composition for attacking group x by an EA)
    update colors of changed nodes
    (color node with color associated with its highest value)
    decrease r; increase t
until t ≥ t_max
```

Algorithm 1. SOM training.

Fig. 2. *Glest* screenshot; a battle between two "magic" forces.

TABLE I
OVERVIEW OF TECH UNIT TYPES, ORDERED BY THEIR AVAILABILITY DURING THE GAME; ARMOR. SM MEANS ARMORED SWORDMAN

Name	Speed	HP	Power	Range	Armor	Flies
Archer	medium	low	high	ranged	Leather	no
Swordsmen	medium	medium	medium	melee	Leather	no
Armor.SM	slow	high	medium	melee	Plate	no
Horseman	fast	high	medium	melee	Plate	no
Technician	slow	low	weak	melee	Leather	no
Catapult	slow	high	high	ranged	Wooden	no
Bat.Mach.	medium	high	high	both	Wooden	no
Airship	slow	high	high	ranged	Wooden	yes
Ornithoper	medium	medium	medium	ranged	Wooden	yes

TABLE II
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

III. 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 published by Figueroa *et al.* [37] under GNU Public License (GPL, [38]). While this work is based on version 2.0.1, *Glest* version 3 has been released with improved game play and balancing. Therefore, some of the observations shown here might deviate quantitatively from the new version. However, both versions have near-commercial qualities and playing them is fun for RTS gamers. 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; see Fig. 2 for an impression.

The flocking studies consider the magic faction only and group selection with SOM is done for the tech faction only. However, this 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 laborers as basic units, who may be trained to become knights, archers, or engineers who are able to construct battle machines and flying units. For a short overview, see Table I.

Each unit has a number of HPs 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. As many RTS games, *Glest* employs an asymmetric 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 Table II.

Glest's AI is mainly rule based but we refrain from getting into details as we intend to use it as a black box. For our techniques, interactions with the movement and pathfinding algorithms are important which are described in the respective context. The AI uses slight random influences, e.g., in group fights, but is deterministic otherwise. It is challenging for unexperienced players as it heavily builds on repeated rushes towards

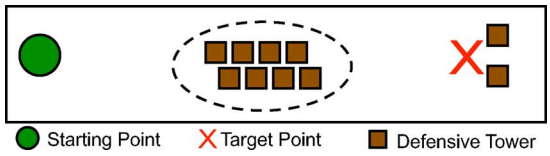


Fig. 3. Map sketch for the first and second scenario tackled. The encircled defensive tower position in the middle only exists in scenario 2. It is possible to pass by the defensive towers out of firing range.

the enemy basis. Consequently, a frequently expressed opinion of players is that the game often ends too quickly with the first fight. However, experienced players are usually able to survive the first attacks with the help of a good defence (e.g., towers) and can then counterattack and win, as in many RTS games. The AI has no means to react to this kind of strategy.

IV. Experimental Analysis

The experimental analysis will 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 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 antigrouping problem in standard *Glest*, also studying proper parametrization and parameter robustness of the SOM training process. Experiment 6 investigates a sample scenario in which the combination of all techniques will lead to a clear advantage—and in fact it does.

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

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

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

Setup: A team of combat units is sent to two hostile defensive towers (high combat strength, 7000 HPs). The first map (cf., Fig. 3) is empty apart from these entities and features no obstacles like mountains or rivers, and is rather small (64×64 tiles). The team consists of six units with different abilities: two demons (average speed, weak combat strength, short-ranged, 700 HPs), two battlemages (average speed, weak combat strength, long-ranged, 700 HPs), and two drakeriders (fast speed, high combat strength, medium-ranged, 1300 HPs).

In the second map, (cf., Fig. 3), hostile turrets are added along the way of the combat units and the attacking team is enlarged by an initiate (slow speed, weak combat strength, medium-ranged, 450 HPs). Note that the initiate is the weakest unit in the game so that the team's strength does not increase significantly. The units have to pass along eight defense towers with sufficient range to attack the team. The flocking parameter settings are: angle of sight $\alpha = 90$, separation weight $|w_{\text{sep}}| = 3.0$, and separation distance $|v_{\text{sep}}| = 4.0$. When using IM, it is applied within the A* algorithm to calculate the movement 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. 4 presents the number of surviving units within the different scenarios. Note that zero survivors 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 team 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 arrive and thus defeat the team. Thus, only about 8% of the games are won and only a few units survive, if at all (Fig. 4).

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 the especially slow units get hurt by these towers, for example, initiates are killed in most games as they need too much time to cross the dangerous area (Fig. 4). 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 team member. So in the first map, the team units reach the two defensive towers simultaneously. As the team outnumbers the two towers, these are hardly able to defeat it. Consequently, in nearly all games, the flock wins with four surviving units (Fig. 4). On the second map, flocking causes a significant drawback. The flock sticks together but maneuvers 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. 4).

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 path finding does not change the flock behavior for map one, but still all games are won. On the second map the results change from 100% losses to about 80% wins (Fig. 4) by IM and more than 40% wins with four surviving units.

Discussion: Overall, the attained results are well in accordance with expectation. The bad results for the normal movement stem from the different speeds of the considered units.

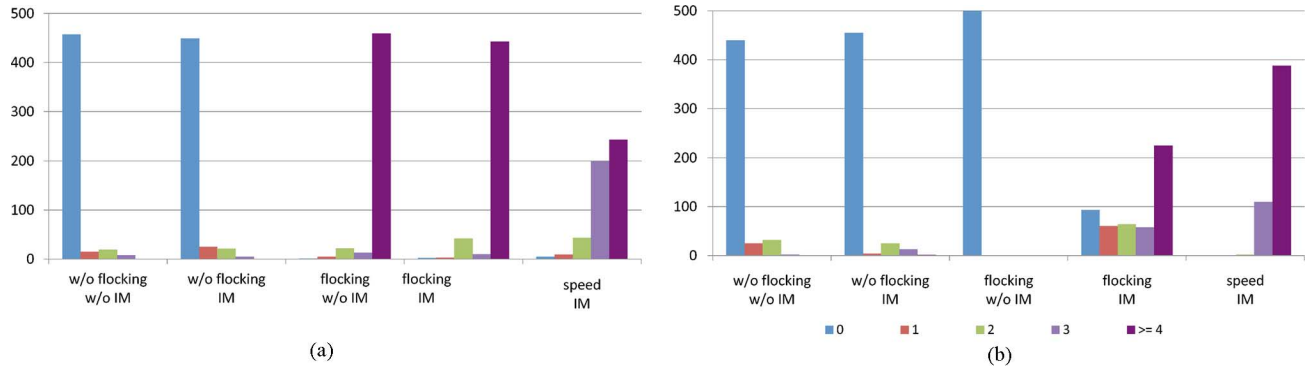


Fig. 4. Number of surviving units on (a) the first map and (b) the second map for five different strategies with 500 runs each. We show all combinations of normal and flocking movement with and without IM and one speed-adapting movement with IM.

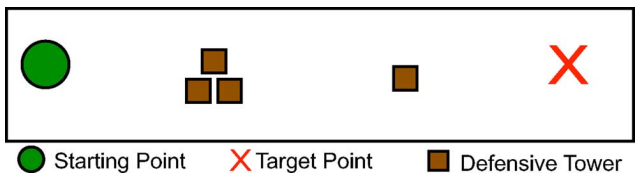


Fig. 5. Sketch of the map for the third scenario. It is possible to pass by the defensive towers at a distance.

The units reach their common target one after another, which enables the towers to kill them easily. Incorporating flocking, the team 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 approximately 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 realized by adding IM-supported path finding. 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, team behavior changes significantly with the IM path finding which leads the team around the invincible set of towers in the middle of the map. Thus, the attacking units do not lose any HPs on their way to their target. When flocking is not active, the team 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. 4).

Combining flocking and IM is the most successful alternative for the second map, without losing performance for the first one. Flocking lets the team stick together and the IM takes care of secure paths around opponent units, thereby successfully avoiding the eight towers in map 2.

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. We also compared this outcome to the situation with flocking disabled and enforced equal unit speeds instead: the outcome is very similar in numeric results, but the team maneuvering looks much less "natural." Visual inspection during our experiments revealed such a clear difference that no specific test with human observers was done regarding the degree of naturalness of both approaches.

However, enforced equal speeds may be a simpler alternative to flocking, also usually keeping the group together in most situations (as long as there are no huge obstacles on the path which could lead to separation into two or more groups), but it comes at a higher cost. Path calculation has to be done at least in part separately for every unit, instead of a "group path" that is computed only once. Summarizing, we obtain an authentic team maneuvering behavior as an alternative to the movement currently seen in many RTS games.

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

Preexperimental Planning: We performed first tests with varying SEW value in order to find a reasonable interval to investigate in the following.

Task: We demand that the team behavior invoked with changing SEW is 1) significantly different in the extreme values (aggressive/defensive) and 2) 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 team's target (cf., Fig. 5). The first enemy position consists of three defensive towers (with same values as in Experiments 1 and 2). The second one is a single but stronger defensive tower featuring a high combat strength with 13 000 HPs. The distance between the two positions is big enough so that their influences do not overlap each other. The team is the same as in scenario 1 plus an additional archmage, resulting in seven 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 III gives the percentage of games in which a certain number of units reached the target point. Table IV depicts equivalent values for the two teams

TABLE III
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 IV
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

of defensive towers. *Sdt team 1* stands for the three towers, whereas *sdt team 2* refers to the stronger standalone tower.

Observations: Table III shows that all attacking units survive with SEW set to 12. Notably, for SEW $= 8$, exactly two 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 standalone tower surviving every game again. Note that in about 60% of the games, the towers are completely destroyed by the attacking units.

Discussion: Depending on the three SEW values, three different situations occur. A high SEW causes a more defensive team 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 team power. However, the high SEW does not guarantee a complete defensive behavior. An extremely powerful team might still decide to attack.

When SEW is set to 8, the team 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 team with a SEW value of 4 attacks the first emplacement directly. Its synergy effect is strong enough to let the team act aggressively. The results show that at least one out of three towers is destroyed by the team (cf., Table IV). 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 (twelve) to normal (eight) to aggressive (four). The concrete values of course depend on the team compositions and have to be found out experimentally for other scenarios.

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

	Armor 1 (a)	Armor 2 (s)	Armor 3 (h)
Attack 1, archer (a)	0.5	5.0	0.5
Attack 2, swordsmen (s)	0.5	0.5	5.0
Attack 3 horseman (h)	5.0	0.5	0.5

TABLE VI
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

TABLE VII
BMU DETECTION FRACTION 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%

Experiment 3: Proof-of-principle—Does the SOM generate the "right" results in a simplified scenario?

Preexperimental Planning: According to the asymmetric superiority principle, we designed a simplified scenario (balanced factions) with only three unit types employing the damage multipliers given in Table V. 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 systematic variation of the SOM parameters, a good SOM configuration will 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 detection fraction of at least 90%,[1] 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 three in the homogeneous case), and the attacker team type, according to Table VI. 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 ten times and the resulting substitute function value is averaged over all (three) homogeneous and all (three) two-unit equally shared mixed attacking teams. The BMU test is performed over four configurations. The largest SOM (dimension 16) is repeatedly (five times) trained for 1000 iterations with 30 units and 100 mixed patterns and also homogeneous attacker

[1]The BMU (which is known here) is of a single unit type in this example, and the answer should contain this type with a fraction of at least 90%.

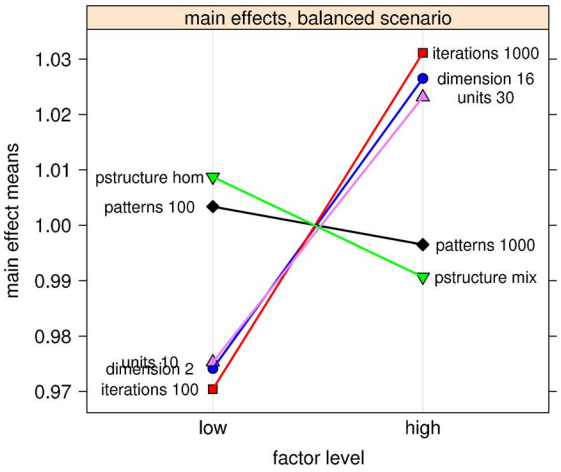


Fig. 6. Main effect estimations of parameters for the balanced scenario. All parameters except the number of patterns have a significant effect.

teams, respectively. The smallest SOM (dimension 2) is trained under the same conditions, but only for 100 iterations and ten units. The BMU fraction is computed from the averaged fraction of "right" answers (archers for swordsmen, swordsmen for horsemen, and horsemen for archers) over the three homogeneous test patterns. Note that this experiment is performed completely outside the game; we only rely on the substitute function that "emulates" a fight and compare the responses of the resulting SOMs to the optimal team which is known due to the construction of the scenario. Correlation of the substitute function with the real game is assumed and tested only in the next experiment.

Results/Visualization: Main effects and two-factor interactions are estimated via design of experiments (DoE) standard procedures by a simple full factorial design [39]. That is, over the parameter ranges regarded as suitable, the extremes are tested in all combinations, and the effects are accumulated from all configurations where the parameter in question has the high value versus all configurations where it has the low value. 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 Table VII.

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 three 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 seem 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 four.

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.

TABLE VIII
KENDALL'S RANK CORRELATIONS AND TEST p-VALUES BETWEEN SIMULATION AND SUBSTITUTE FUNCTION

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

Experiment 4: Does the SOM's substitute function properly represent simulation results on the real game?

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

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 teams, as this situation would almost never occur in a concrete game (at least not while playing against humans). Instead, we generate only homogeneous or two-type mixed attacking teams of size ten each. The defending team is drawn randomly from all unit types, also containing ten units (allowing randomness here makes sense as the composition of the defending team is the search space of the SOM). Much larger team sizes are infeasible for simulation as simulation time grows super linearly in unit numbers. We restrict our tests to tech level 3 (nine types; see Experiment 3) which provides the largest search space for the SOM. When succeeding here, we assume that tech levels 1 and 2 can also be modeled by the substitute function. The simulations are run within *Glest* on a small empty map where the two teams are placed directly next to each other and use the normal game AI for the fight itself. The simulation ends when one party is defeated or after 5000 game cycles elapsed. The outcome of the fight is evaluated based on the surviving units as $(p_A - p_D) \rightarrow \min!$, whereas p_A denotes the fraction of surviving units from the attacking group (SOM's input) and p_D

TABLE IX
SUGGESTIONS OF DIFFERENTLY TRAINED SOMS

	10 Iterations		30 Iterations		50 Iterations	
Input Team	Substitute function	Simulation	Substitute function	Simulation	Substitute function	Simulation
Airship	Battle Machine	Ornithopther	Ornithopther	Ornithopther	Ornithopther	Ornithopther
Archer	Battle Machine	Ornithopther	Ornithopther	Airship	Ornithopther	Ornithopther
Armored Swordman	Airship	Ornithopther	Airship	Airship	Ornithopther	Ornithopther
Battle Machine	Airship	Ornithopther	Ornithopther	Airship	Airship	Ornithopther
Catapult	Airship	Ornithopther	Airship	Airship	Ornithopther	Ornithopther
Horseman	Airship	Ornithopther	Airship	Airship	Ornithopther	Ornithopther
Ornithopter	Battle Machine	Ornithopther	Ornithopther	Ornithopther	Ornithopther	Ornithopther
Swordman	Airship	Ornithopther	Ornithopther	Ornithopther	Ornithopther	Ornithopther
Technician	Airship	Ornithopther	Ornithopther	Ornithopther	Ornithopther	Ornithopther

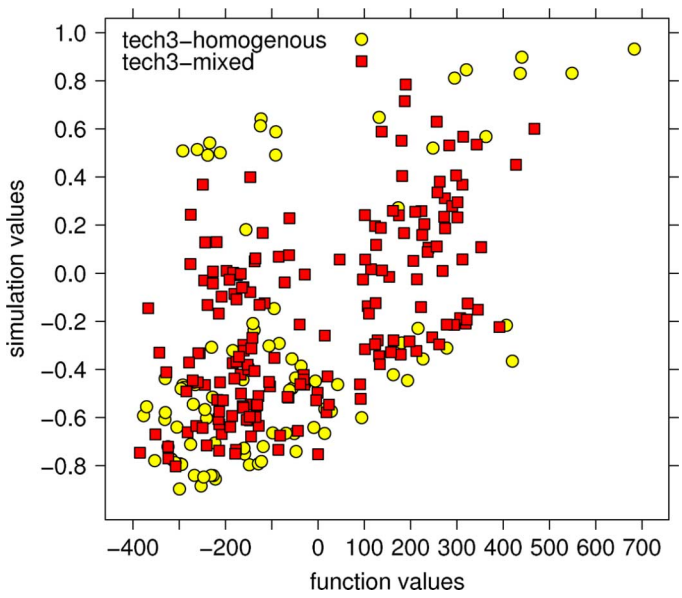


Fig. 7. Correlation plot: Simulation result over substitute function, 360 random samples of defenders against homogeneous, and mixed attacker teams, respectively. Theoretical minima are -1.0 for the simulation and -400 for the objective function.

that from the defending group (SOM's answer). The optimal value -1.0 means that all attacking units have been eliminated, while all defending units survived.

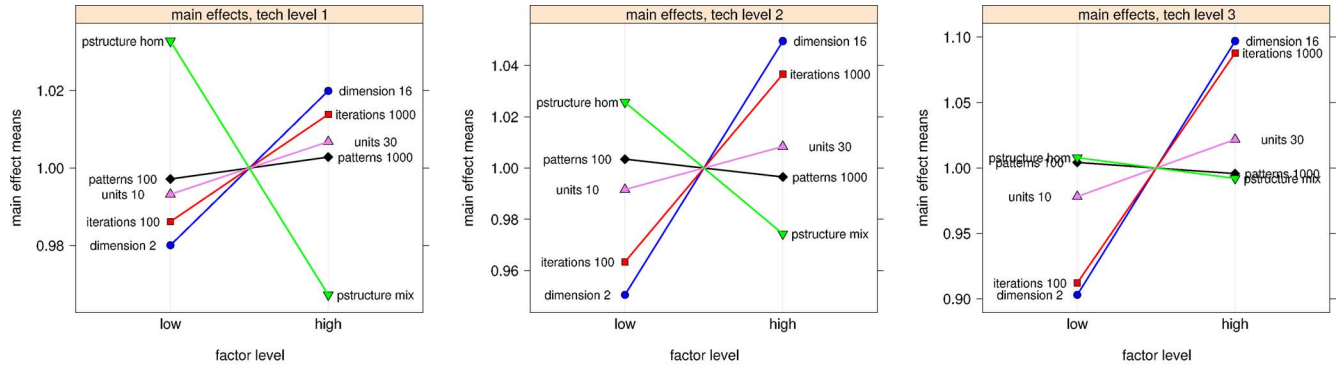
Results/Visualization: Table VIII provides correlation estimates and the associated p-values of the tests; Fig. 7 gives a visualization. Table IX shows a comparison of BMUs of 8×8 SOMs that were trained either on the game itself (simulation) or by means of the substitute function. We consider only a few iterations as training on the game needs runtimes from hours to days and thus cannot be done with a large number of iterations. We use homogeneous teams as test inputs here. The highest weighted units in the BMU for the actual input are shown in the table.

Observations: Table IX shows that, independent of raw function values, both functions lead to similar SOMs. Air units are especially favored. The rank correlation values (Table VIII) hardly show any difference between homogeneous and mixed samples, which is unexpected. Fig. 7 largely adheres to our expectations. Still, it reveals some interesting details. 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. Second, very good objective function values possess a very low variance in simulation values—this seems to support the previous observation. Third, the -/- quadrant is very dense, meaning that there are many more good than bad (randomly drawn) defending unit compositions.

Discussion: The overweighting of air units is unsatisfactory. Incorporating the units' cost is probably necessary for applying the technique to the whole game. 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 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 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 team. Obviously, it pays off to send different troops into combat than those used by the aggressor.

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 three sequential phases usually experienced in *Glest*. We thus not only get three levels of difficulty as test cases for the SOM, but also three different types of SOMs which



(a) (b) (c)

Fig. 8. Main effect estimations of parameters for (a) the tech level 1 scenario (four units) and (b) tech level 2 (seven units). All effects are significant, except pattern number and unit size. In (c) the tech level 3 scenario (nine units), all except pattern size and pattern structure are significant.

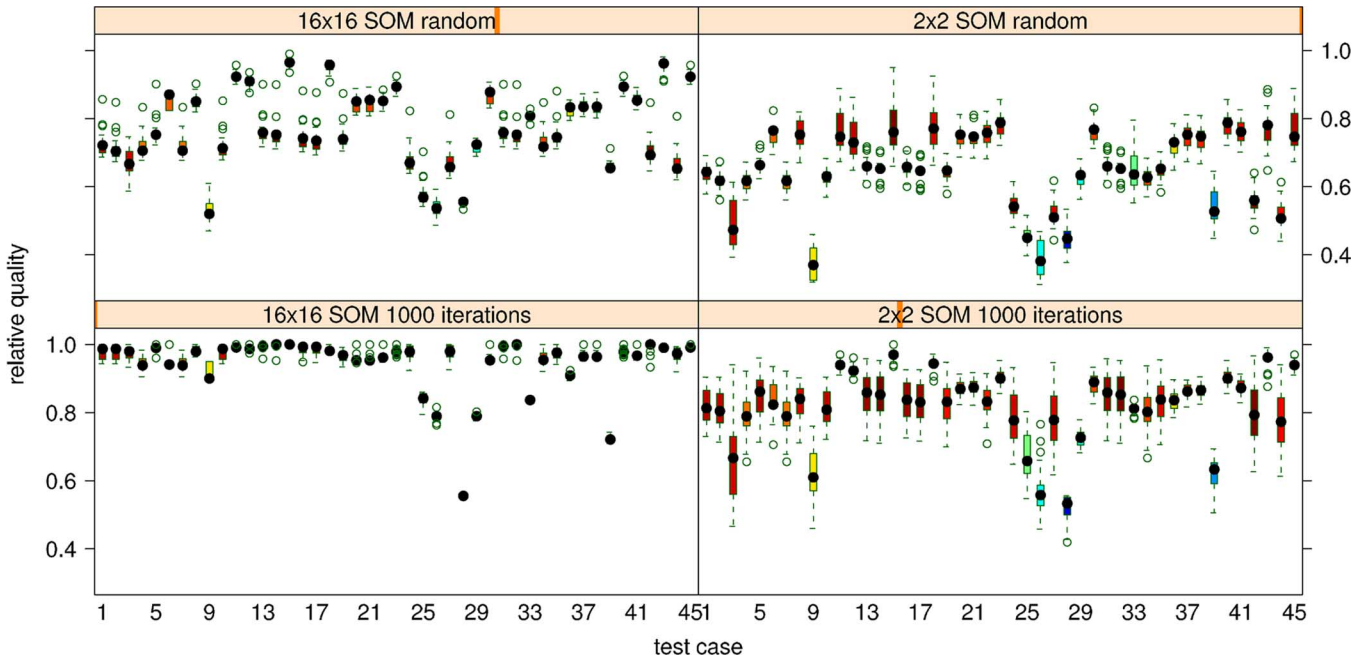


Fig. 9. Box plots of relative qualities of a large and a small SOM with and without training for the separate test cases (nine homogeneous, 36 mixed).

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 Experiment 3. We aim for obtaining information about the robustness of the parameter settings in Experiment 3. For being robust, we demand that the parameter recommendations from Experiment 3 are suitable for Experiment 5 also, or *vice versa*, so that at least one robust parameter setting exists. Additionally, we measure the SOMs performance by comparing its suggested defense team 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: For Experiment 3, the scenario had been constructed in a way that the optimal answer was obvious. Here we perform a complete enumeration for each tech level to obtain optimal defending teams for homogeneous as well as for two-type-mixed attacking teams of size ten. Knowing these optimal answers, we can compare the SOM performances on the substitute function. We compute the relative quality q_{rel} of a SOM answer after formula 3, comparing the SOM answer to the precomputed optimum. We perform 31 repeats for each SOM configuration to get an impression of the underlying distributions

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

The overall setup for the SOMs is the same full factorial design as in Experiment 1. As with Experiment 3, SOM training here only uses the substitute function, not the game itself. Apart from runtimes, this has the additional advantage that we can determine the correct solution (by total enumeration); this is not possible for the game itself.

Results/Visualization: Fig. 8 documents the measured main effects for tech levels 1 to 3. The obtained two-factor interaction effects are rather weak and thus omitted. Table X shows the average relative answer qualities of two different SOMs for tech level 3, before and after training. The SOMs are tested on all nine homogeneous and all 36 two-unit equally shared mixed attacker teams. This is visualized in more detail in Fig. 9.

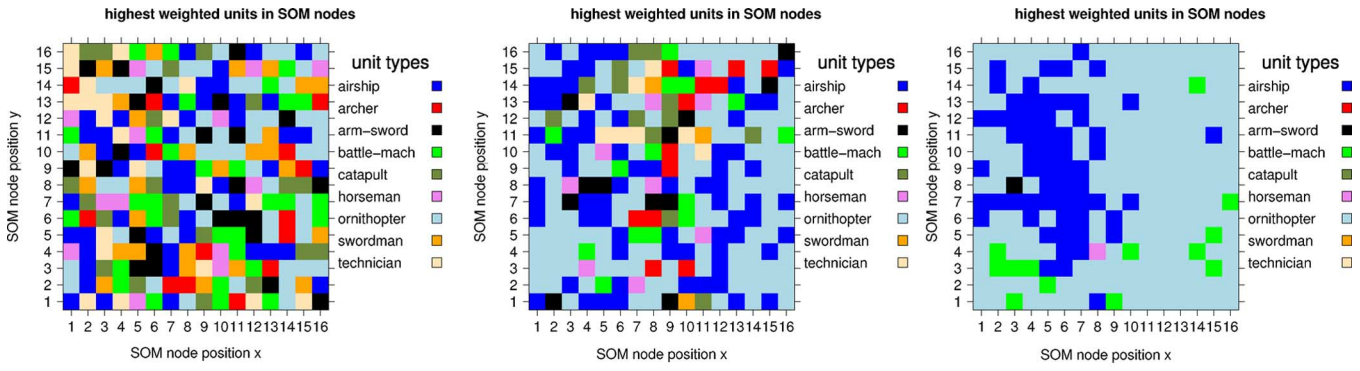


Fig. 10. Evolution of a *Glest* tech level 3 SOM (16 × 16, 30 units, nine homogeneous patterns) during the training process. After (a) 10, (b) 100, and (c) 1000 iterations, respectively. Learning ends with a clear domination of air units.

TABLE X
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%

Observations: In tech level 1, the pattern structure effect is remarkably strong, which is interesting as the level is not much different from 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 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: In regards to the resultant 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 as displayed in Fig. 10 for tech level 3 indicates 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 team 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 teams, 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. 10. Only few (highest rated) unit types remain after 1000 iterations, and most nodes favor air units (ornithopters and airships). The domination of these air units for tech level 3 could have several reasons. First, it may of course be intentional as the more advanced units will be stronger. Adding to this, we completely neglect cost and speed and many other unit attributes in our substitute function, which may compensate for unbalanced attack strengths in the game. Second, the majority of unit types have 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 team constellations from playing humans and to use these data as training input.

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 team composition). This scenario is depicted in Fig. 11 and is balanced so that if the best matching units are sent as flock along the right path to either of the two defending teams, the game is nearly always won by the attacker.

Task: We want to reject the hypothesis that with flocking+IM switched on, SOM team composition does entail a large change in results.

Setup: The allowed units comprise the three types from the balanced scenario, in which swordsmen beat horsemen, these beat archers, and archers in turn beat swordsmen. We put ten towers in the middle of the scenario (Fig. 11) which have to be

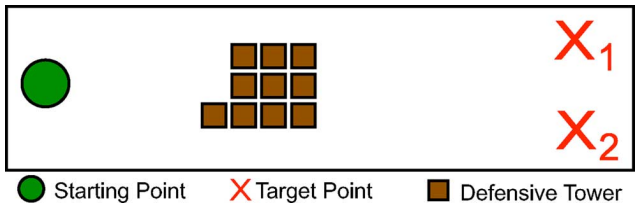


Fig. 11. Sketch of the underlying map for the combined scenario.

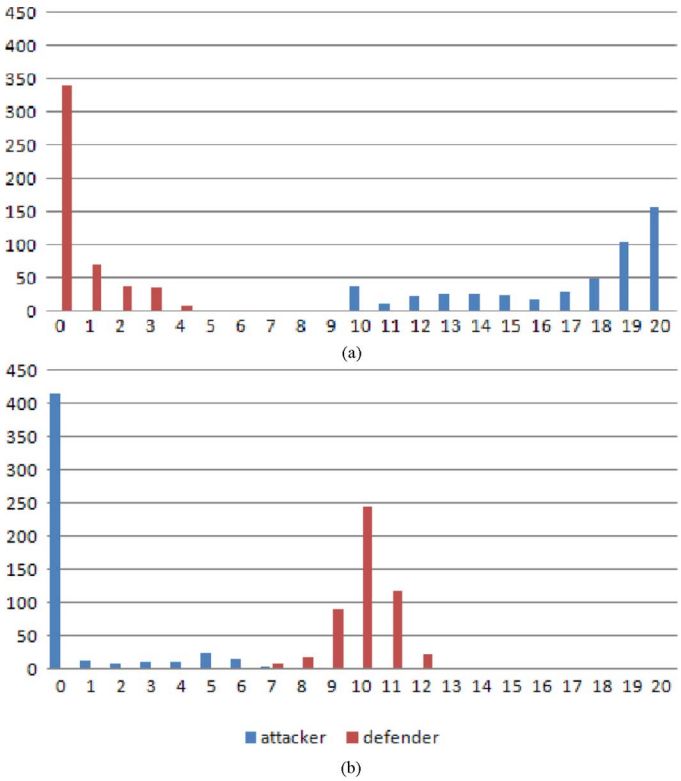


Fig. 12. Surviving attackers (blue) and defenders (red) of 20 units, respectively, in case of (a) SOM team composition and (b) with reversed team composition over 500 games each.

bypassed. At X1, seven horsemen and three swordsmen await the attackers, and at X2, seven archers and three horsemen are placed. The SOM gets free choice of ten units to send against X1, and again ten 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 teams, thereby sending them at the wrong targets, or simulating a faulty SOM team composition. We only control the maneuvering of the teams; for the fight itself, we entirely rely on the *Glest* normal game AI.

Results/Visualization: Fig. 12 shows the distribution of surviving units of attackers and defenders under SOM team composition and under reversed team composition.

Observations: The SOM selects ten swordsmen to capture X1, and ten horsemen to X2. All resulting games are won by the attacker, and the defenders of X2 are always annihilated. At X1, in 150 ($\approx 31\%$) 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 teams are reversed (ten horsemen to X1 and ten swordsmen to X2), the attacker always loses, and retains some (few) units in only 84 games. This seems to be due to the same effect of localizing the opponent team 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 team composition leads to a huge change in results and thereby rejects 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 team behavior may even be called "intelligent." We regard the reported problems (some defending units are not found) as minor and of a technical nature; they refer to implementation details depending on the used game.

V. Conclusion and Outlook

We have shown by thorough experimental analysis that the two proposed approaches, flocking with influence maps and SOM group selection, can straightforwardly be applied in an RTS game 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 team when combatting another can be found, training a SOM is a matter of a few minutes even for the largest investigated configurations. The experimental results clearly indicate that the number of nodes and the learning iterations are the most important parameters. Both should be enlarged if performance needs to be increased. Creating a substitute function was very easy, at least in our game context, and it may also be regarded as a possible way for the game designer to include subjective preferences (which is not explored here). Partly automated design of a substitute function, for example, by means of genetic programming would be an interesting area for further research. Designing the substitute function or setting it up via experiences with small game simulations is the only work the game designer has to do; the rest of the procedure is fully automatic and completely removes the need for designing rule sets for group selection. The approach is flexible as well as the SOM always returns a list of possible answers in "suitability order." If the best group cannot be realized due to lack of units, the list can be worked down and another may be chosen.

Training the SOM 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 unnecessary. 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 preproduced and game AI just has to switch between them. Note that according to our experience, exactly solving the problem treated by the SOM is not an option. Where training a SOM fully takes only seconds, it takes several minutes on a modern PC to enumerate all possibilities to set up a good team of ten against a single given team.

Flocking in combination with IM path finding improves team performances in each of the game situations investigated here and tackles the shortcomings detected for either technique alone. It seems to be quite robust, makes team maneuvering 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 team 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 team has to move where, and the flocking+IM accomplishes that. In addition to measurably good team performance, the resulting behavior 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 teams 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 teaming up 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 EA 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

The authors would like to 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, pp. 6–, 2006.

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

[3] S. M. Lucas and G. Kendall, "Evolutionary computation and games," *IEEE Comput. Intell. Mag.*, vol. 1, no. 1, pp. 10–18, Feb. 2006.

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

[5] S. J. Louis and C. Miles, "Case-injection improves response time for a real-time strategy game," in *Proc. IEEE Symp. Comput. Intell. Games*, G. Kendall and S. Lucas, Eds., Piscataway, NJ, 2005, pp. 149–156.

[6] S. Bakkes, P. Spronck, and J. van den Herik, "Rapid adaptation of video game AI," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Kingston, Eds., Piscataway, NJ, 2008, pp. 79–86.

[7] J. K. Olesen, G. Yannakakis, and J. Hallam, "Real-time challenge balance in an RTS game using rtNEAT," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Kingston, Eds., Piscataway, NJ, 2008, pp. 87–94.

[8] F. Schadd, S. Bakkes, and P. Spronck, "Opponent modeling in real-time strategy games," in *Proc. EUROSIS AI Simul. Games*, M. Roccetti, Ed., Ostend, Belgium, 2007, pp. 61–68.

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

[10] F. Sailer, M. Buro, and M. Lanctot, "Adversarial planning through strategy simulation," in *Proc. IEEE Symp. Comput. Intell. Games*, 2007, pp. 37–45.

[11] M. van der Heijden, S. Bakkes, and P. Spronck, "Dynamic formations in real-time strategy games," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Hingston, Eds., Piscataway, NJ, 2008, pp. 47–54.

[12] J. Hagelbäck and S. J. Johansson, "Using multi-Agent potential fields in real-time strategy games," in *Proc. 7th Int. Conf. Autonom. Agents Multi-Agent Syst.*, L. Padgham and D. Parkes, Eds., 2008, pp. 631–638.

[13] J. Hagelbäck and S. J. Johansson, "Dealing with fog of war in a real time strategy game environment," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Hingston, Eds., Piscataway, NJ, 2008, pp. 55–62.

[14] T. Thompson and J. Levine, "Scaling-up behaviours in evotanks: Applying subsumption principles to artificial neural networks," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Hingston, Eds., Piscataway, NJ, 2008, pp. 159–166.

[15] P. Lichocki, K. Krawiec, and W. Jaskowski, "Evolving teams of cooperating agents for real-time strategy game," in *Lecture Notes in Computer Science*. Berlin, Germany: Springer-Verlag, 2009, vol. 5484, pp. 333–342.

[16] H. Danielsiek, R. Stüer, A. Thorm, N. Beume, B. Naujoks, and M. Preuss, "Intelligent moving of groups in real-time strategy games," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Hingston, Eds., Piscataway, NJ, 2008, pp. 71–78.

[17] N. Beume, T. Hein, B. Naujoks, N. Piatkowski, M. Preuss, and S. Wessing, "Intelligent anti-grouping in real-time strategy games," in *Proc. IEEE Symp. Comput. Intell. Games*, L. Barone and P. Hingston, Eds., Piscataway, NJ, 2008, pp. 63–70.

[18] 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.

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

[20] P. E. Hart, N. J. Nilsson, and B. Raphael, "A formal basis for the heuristic determination of minimum cost paths," *IEEE Trans. Syst. Sci. Cybern.*, vol. 4, no. 2, pp. 100–107, Jul. 1968.

[21] R. Geraerts, A. Kamphuis, I. Karamouzas, and M. Overmars, "Using the corridor map method for path planning for a large number of characters," in *Proc. 1st Int. Workshop Motion in Games*, 2008, pp. 11–22.

[22] I. Karamouzas, R. Geraerts, and M. Overmars, "Indicative routes for path planning and crowd simulation," in *Proc. Int. Conf. Found. Digit. Games*, 2009, pp. 113–120.

[23] B. Takács and Y. Demiris, "Multi-robot plan adaptation by constrained minimal distortion feature mapping," in *Proc. IEEE Int. Conf. Robot. Autom.*, 2009, pp. 742–749.

[24] M. C. Lin, A. Sud, J. van den Berg, R. Gayle, S. Curtis, H. Yeh, S. Guy, E. Andersen, S. Patil, J. Sewall, and D. Manocha, "Real-time path planning and navigation for multi-agent and crowd simulations," in *Proc. 1st Int. Workshop Motion in Games*, 2008, pp. 23–32.

[25] W. van der Steeren, "Tactical path-finding with a*," *Game Programming Gems*, vol. 3, pp. 294–306, 2002.

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

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

[28] H. Danielsiek, C. Eichhorn, T. Hein, E. Kurtió, G. Neugebauer, N. Piatkowski, J. Quadflieg, S. Schnelker, R. Stüer, A. Thorm, and S. Wessing, "Technical report of the collaborative research centre 531 computational intelligence CI 252/08, SFB 531, 2008," Technische Universität Dortmund, Germany, Final Rep. PG 511 [Online]. Available: http://www.ciingames.de/publications/pg511finalreport.pdf

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

[30] J. Flensbak, "Flock behavior based on influence maps," B.S. thesis, Dept. Comput. Sci., Univ. Copenhagen (DIKU), Copenhagen, Denmark, 2007.

[31] C. Miles and S. J. Louis, "Towards the co-evolution of influence map tree based strategy game players," in *Proc. IEEE Symp. Comput. Intell. Games*, Piscataway, NJ, 2006, pp. 75–82.

[32] C. Miles and S. J. Louis, "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.

[33] R. Watson and J. Pollack, "Coevolutionary dynamics in a minimal substrate," in *Proc. Genetic Evol. Comput. Conf.*, L. Spector, E. Goodman, A. Wu, W. B. Langdon, H.-M. Voigt, M. Gen, S. Sen, M. Dorigo, S. Pezeshk, M. Garzon, and E. Burke, Eds., 2001.
[34] T. Kohonen, *Self-Organizing Maps*. Berlin, Germany: Springer-Verlag, 2001.
[35] T. Kohonen, "Fast evolutionary learning with batch-type self-organizing maps," *Neural Process. Lett.*, vol. 9, no. 2, pp. 153–162, 1999.
[36] H.-G. Beyer and H.-P. Schwefel, "Evolution strategies—A comprehensive introduction," *Natural Comput.*, vol. 1, no. 1, pp. 3–52, 2002.
[37] M. Figueroa, J. González, T. Fernández, F. Menendez, and M. Caruncho, "Glest, a free 3d real time strategy game," Dec. 18, 2007 [Online]. Available: http://www.glest.org
[38] GNU Software Foundation, Gnu General Public License, Version 2 June 2001, Nov. 16, 2007 [Online]. Available: http://www.gnu.org/licenses/gpl-2.0.html
[39] D. C. Montgomery, *Design and Analysis of Experiments*, 5th ed. New York: Wiley, 2005.

Mike Preuss (M'08) was born in Ahlen/Westfalen, Germany, on March 23, 1969. He received the Diploma degree from the Computer Science Department, Technische Universität Dortmund, Dortmund, Germany, in 1998

Since 2000, he has been a Research Associate at the Computer Science Department, Technische Universität Dortmund. His research interests focus on the field of evolutionary algorithms (EA) for real-valued problems, namely on multimodal and multiobjective niching. His mission is to further develop the experimental methodology for nondeterministic optimization algorithms by means of tuning, adaptability measures, and other experimental analysis techniques. These are essential for the design of specific EA-based algorithms for industrial optimization in various engineering domains, e.g., thermodynamics and ship propulsion technology. As member of the IEEE Computational Intelligence Society (CIS) games technical committee and chair of the CIS task force on real-time strategy games, he is active in designing authentic AI components for believable opponents and increased player satisfaction.

Nicola Beume received the Diploma degree in computer science from the University of Dortmund, Dortmund, Germany, in 2006.

Since then, she has been a Research Associate with the Faculty of Computer Science, Technische Universität Dortmund, Dortmund, Germany. Her research focuses on the design of multiobjective evolutionary algorithms as well as their theoretical and empirical analysis, and the application of computational intelligence methods in games.

Holger Danielsiek was born in Bochum, Germany, in 1981. He received the Diploma degree from the Technische Universität Dortmund, Dortmund, Germany, in 2009. His diploma thesis dealt with evolutionary multiobjective optimization and cluster-based niching. Currently, he is working towards the M.S. degree in the dortMINT-Project at the Technische Universität Dortmund.

Currently, he shifted his interests towards computer science education.

Tobias Hein was born in Herne, Germany, in 1981. He received the Diploma degree in computer science from the Technische Universität Dortmund, Dortmund, Germany, in spring 2010. His diploma thesis dealt with the visualization of high-dimensional musical data and an approach on motif-search by means of self-organizing maps.

During the course of studies he became interested in the field of computational intelligence (CI) with focus on the application of CI techniques in video games and music information retrieval.

Boris Naujoks received the Diploma degree from the faculty of computer science of Technische Universität Dortmund, Dortmund, Germany.

There he focused on the application of evolutionary multiobjective optimization algorithms. In early 2009, he took responsibility for the management of industrial and scientific projects at the Log!n GmbH, Schwelm, Germany, an IT company in logistics.

Nico Piatkowski was born in Dortmund, Germany, in 1984. Currently, he is working towards the Diploma degree in computer science at the Technische Universität Dortmund, Dortmund, Germany.

He has been working as a tutor and student assistant at the Computer Science Department, Technische Universität Dortmund, from 2005 to 2009. Since 2010, he has been working as a Student Assistant at the Artificial Intelligence Unit, Technische Universität Dortmund, where he also writes his diploma thesis. His research interests include machine learning with focus on graphical models, nonlinear optimization, as well as parallel algorithms especially for graphical processing units (GPUs).

Raphael Stüer was born in Bochum, Germany, on June 20, 1981. He received the Diploma degree in computer science from the Technische Universität Dortmund, Dortmund, Germany, in 2009.

From 2001 to 2004, he did an apprenticeship with a major company focused on communications. His research interests are knowledge discovery, improving player satisfaction using methods of computational intelligence as well as (multiobjective) optimization in real-world applications (which was the topic of his diploma thesis).

Andreas Thom was born in Wuppertal, Germany, in 1980. He received the Diploma degree from the Technische Universität Dortmund, Dortmund, Germany, in 2009. His diploma thesis dealt with density-based clustering.

Currently, he is a Research Assistant at the Computer Science Department, Technische Universität Dortmund. His research interests include computational intelligence and machine learning.

Simon Wessing received the Diploma degree in computer science from the Technische Universität Dortmund, Dortmund, Germany, in 2009.

Currently, he is a Research Associate at the Technische Universität Dortmund. His research interests include evolutionary, multiobjective, and robust optimization, as well as surrogate models, especially Kriging.