

Emergent Behavior in Modeling

H. Van Dyke Parunak, Ted Belding, Sven Brueckner, John Sauter¹

1 The Questioner's Dilemma

“Ah, but a man's reach should exceed his grasp,
Or what's a heaven for?”
Robert Browning, “Andrea del Sarto,” 1855.

In our quest for knowledge, we face a frustrating dilemma. We want to engage with reality—to anticipate its future behavior, and either adjust it, or adjust our behavior to accommodate it. Yet we are fundamentally limited in our freedom to manipulate reality. Our limitation may stem from our finite resources: we can seed a cloud, but we can't change the whole weather system. Or it may result from the inexorable flow of time: we can't change the past, or know the future.

One tool that we repeatedly use in attempting to break these shackles is modeling. Though we can't change reality, we can build and modify artificial structures that correspond with reality. By experimenting with these structures (or “models”), we can gain insight into how the real world behaves. The final test of our ideas is to engage them with the real world, but a model can let us try out a wide range of ideas that we could never afford to implement physically, so that when we do invest in the costly task of manipulating reality, we have a high chance of success.

Extensive use of modeling became common with the spread and reduction in cost of digital computers. As people began to develop models in various fields, they repeatedly encountered unexpected results: the whole is often more than the sum of the parts. When they composed a model of a system out of models of its individual components, the system as a whole exhibited behavior that they had not programmed into the components. We call such behavior “emergent behavior.” Clearly, if we are to use computer models responsibly, we need to understand this phenomenon.

Some researchers take a defensive posture, and seek to constrain the system to avoid such dynamics [34]. One such article is even entitled, “Emergent Behaviours Considered Harmful” [16]! This approach might make sense in an engineered system (though even there, we suggest it is misguided). But an increasing body of data suggests that emergent behavior is a characteristic of the real world. To the extent that we choke it out by constraining our models, we may in fact reduce their fidelity and delude ourselves. Our posture is more offensive. We seek to understand emergent dynamics in computer models and exploit them to improve the usefulness of our models.

In Section 2 of this chapter, we introduce the basic concepts of emergent behavior and modeling, and suggest some benefits of understanding and exploiting emergent behavior in modeling. In Section 3, we review models in five different areas, to illustrate the value of our emergent approach. In Section 4, we pull together lessons from this broad experience and offer some general conclusions and recommendations.

¹ The work reported in this paper was made possible by the Agent-Based and Complex systems (ABC) group, notably Steve Brophy and Bob Matthews.

2 The Concepts

2.1 Emergent Behavior

A classic example of emergence is Conway's Game of Life [1], which operates on a 2-dimensional lattice like a chess board. Each location, or *cell*, in the lattice can be either on or off (1 or 0). Each cell has 8 neighboring cells (Figure 1), which can be thought of as the cells to the north, south, east, west, northeast, southeast, northwest, and southwest of it, respectively. Each cell in the lattice is initially set to be either on or off. At each timestep after the initial one, the cells are each updated based on three very simple rules, which can be simulated either by hand or on a computer:

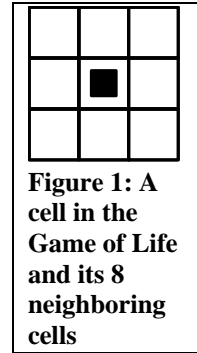


Figure 1: A cell in the Game of Life and its 8 neighboring cells

- 1) A cell that is on in the current timestep remains on in the next timestep if exactly two or three of its neighbors are on in the current timestep.
- 2) A cell that is off in the current timestep turns on in the next timestep if exactly three of its neighbors are on in the current timestep.
- 3) In all other cases, a cell is off in the next timestep.

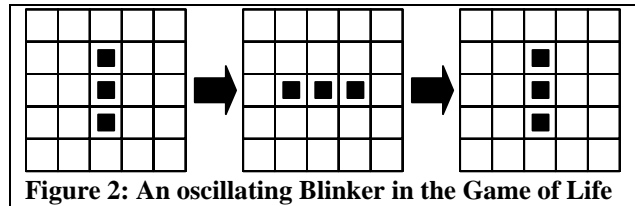


Figure 2: An oscillating Blinker in the Game of Life

Despite the extreme simplicity of these micro-level rules, complex behavior emerges at the macro-level. For example, the configuration in Figure 2, known as a "blinker", oscillates between two states, while the configuration in Figure 3, known as a "block", remains fixed in one state (a 2x2 square).

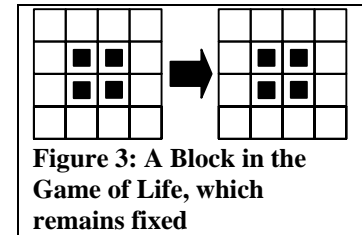


Figure 3: A Block in the Game of Life, which remains fixed

A more complicated configuration, known as a "glider", is shown in Figure 4. This emergent object will travel forever diagonally down and to the right, as long as it doesn't run into any cells that are on. These gliders can be produced continuously by objects known as "glider guns"; they can be combined with other objects to produce even more complex structures. In fact, it has been shown that gliders can be manipulated by other objects that implement the Boolean logic functions AND, OR, and NOT, where the gliders represent "true" or 1, and an absence of gliders represents "false" or 0 [1]. This implies that the Game of Life is Turing-complete and could be programmed to do anything that a standard PC can do, including running Microsoft Windows (albeit *much* more slowly).

As with other emergent phenomena, the emergence in the Game of Life can be viewed as the result of non-linear interactions between the micro-level rules. One cell alone cannot determine the future state of its neighbors; this depends on the interactions among a cell's entire neighborhood. Furthermore, there is no simple linear rule that determines whether a cell is on or

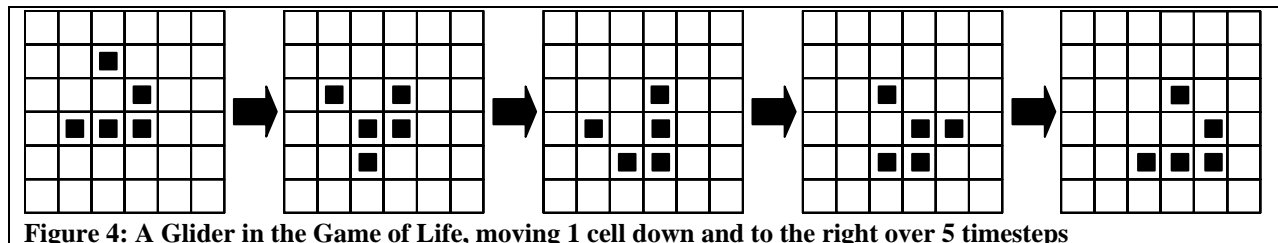


Figure 4: A Glider in the Game of Life, moving 1 cell down and to the right over 5 timesteps

off, as would be the case if the rule were simply “The probability that a cell turns on increases with the proportion of its neighbors that are on.” A cell’s behavior changes nonlinearly as the number of neighbors varies from 0, to two or three, to more than three.

Even simpler systems, such as one-dimensional cellular automata, can also exhibit emergent behavior, and achieve computational completeness [33]. Since emergence occurs in such simple systems, it is likely to be the rule rather than the exception in real-world systems.

2.2 Modeling

The proliferation of inexpensive digital computers has enabled the development of a wide range of models [7]. Some focus on the *states* that the system can take on, and the transitions that convert one state into another. Others focus on discrete *functions* within the system and how these functions transform one variable into another in a cascade of data flows. Still others concentrate on the *entities* that make up a system and how they interact with one another and with the environment in which they are embedded. Our research focuses on entity-based models, but all of these different perspectives can yield emergent behavior, as long as they share two basic (and common) characteristics.

First, we have defined emergent behavior as system-level behavior that is not explicitly encoded in the components. This definition presumes that the model is *compositional*, assembled from smaller parts. Human cognitive limitations ensure that most models are compositional. For a system of any size, we simply can’t get our heads around the entire structure all at once, so we naturally try to identify its parts, model them, and then define the relations among them.

Second, not all compositional systems exhibit emergent behavior. The mass of a basket of peaches is equal to the sum of the individual masses of the basket and the peaches. We can understand cases like this without needing to build a model. When we resort to modeling, it is usually because *nonlinearities* in the interactions among the components keep us from simply adding up the parts. As the previous section suggests, these very nonlinearities can generate emergent behavior. They may be as simple as feedback loops (in which one of the inputs to an entity is a function of its output), or thresholds (such as an upper limit to some variable), or a mathematically nonlinear function (an entity that outputs the square of the sum of its inputs). We run the model because we can’t simply add up the parts, and emergent behavior is simply the difference between what we see and what we would have gotten if the model *were* additive.

2.3 The Promise

The previous section suggests that emergent behavior in models, far from being an isolated phenomenon, is unavoidable. We only bother to model systems that we can’t understand additively, and that non-additivity is the essence of emergence. But we shouldn’t be discouraged. Emergence need not be a threat to our endeavors. It has several desirable features.

First, repeated experience shows that it is a characteristic of the real world, not just of our models. If the purpose of our models is to help us understand the real world, those models need to support the same kind of emergence that we see in nature. Emergence-free models will inevitably be inaccurate models.

Second, the same cognitive limitations that lead us to construct composable models out of smaller components can also be helped by emergence. Emergence lets us achieve system-level behavior that is more complex than what we build into the individual components. An ant-hill

has behaviors that are much more complex than those of any individual ant, but emergence means that we can build a faithful model of the ant-hill just by building models of ants and letting them interact.

That last claim almost sounds too good to be true. We'll only get realistic system-level behavior if we get the individual behaviors and their interactions right. The third desirable feature of emergence is that it follows general principles that transcend any individual problem area, and we can use those principles to help us construct parts with the desired collective behavior. One important tool in this program is the branch of physics known as statistical mechanics. For over a hundred years, physicists have been studying how the properties of matter at the level of our daily experience (such as the temperature of the room, or the stiffness of a beam, or the melting point of ice) result from the nature and interactions of individual atoms and molecules. Many of the concepts and principles that they have discovered continue to be applicable when we move from atoms and molecules to robots and people and sensors. We are beginning to understand and exploit these common principles to enable us to engineer emergence.

3 Examples

Let's consider several applications of modeling in which emergence has proven useful. In each of these examples, we *explain* the real-world system being modeled and the reason for modeling it, *identify* the emergence involved, and *note* what we learned in the course of the activity about identifying and managing emergence.

3.1 Manufacturing Scheduling

3.1.1 Modeling Motives and Mechanisms

In summer 1998, there arose the following task in the ESPRIT LTR project MASCADA: Given a segment of a transport system of arbitrary layout in discrete high-volume production composed of unidirectional line-buffers (e.g., conveyors) and multi-input multi-output sequential routing devices (e.g., rotation tables, lifts), and assuming that the workpieces sent through the segment are all of one product but may be differentiated on the basis of the value of one product parameter (e.g., color); how may the segment be controlled in a decentralized manner so that the outflow of workpieces occurs in batches of workpieces of the same product parameter value with the average batch size of the outflow being significantly higher than that of the inflow?

Taking the inspiration from insect coordination mechanisms, the following simple but effective solution to the batching problem was found. Each sequential router is assigned a Router-agent. A Router-agent acts completely autonomous without even directly communicating with other agents. The simple task fulfilled by each Router-agent is to take workpieces sequentially from the entries of its router to the exits. Therefore, an agent needs to perceive the workpieces waiting at the entries and the current state of the exits (blocked or free). To achieve the required batching property at the level of the control system made up of these agents, a Router-agent has a simple memory. There it stores for each exit the value of the product parameter of the last workpiece that has passed the exit.

With multiple entries and multiple exits a Router-agent has to decide when to take which of the available workpieces to what exit. The decision is taken in a sequential execution of the following three simple rules, starting at rule one:

Sorting Rule (1).—IF at entry X there is a workpiece with a product parameter value of p and there exists a free exit Y whose related product parameter in agent memory has a value of p too, THEN take the workpiece from X to Y immediately and restart at rule one.

Extension.—IF there is currently more than one such action possible, THEN select one of them randomly.

Blocking Rule (2).—The ratio of free entries to the overall number of entries determines a probability PB to pause the routing operation for a fixed time TB. The higher the ratio, the higher is the probability to pause. After pausing restart at rule one.

Random Rule (3).—Select one occupied entry X and one free exit Y randomly, route the workpiece from X to Y, and then restart at rule one.

These rules are based on the assumption that every workpiece may be routed from any entry to any exit at each router. As a consequence, the following two requirements for the layout of the transport system are set:

“Open” Layout.—In the segment of the transport system under consideration, every entering workpiece must be permitted to leave through any exit.

“Directed” Layout.—There must be no cycles in any path from an entry to an exit of the segment.

There exist extensions to the Router-agent behavior that provide an explicit global routing in addition to the sorting of workpieces. But these extensions are outside of the scope of this discussion.

3.1.2 Emergence in Manufacturing

The inspiration for the design of the Router-agents came from the coordinated nest construction of ants, termites, or bees. The basic principle that governs the coordination is sematectonic stigmergy. In stigmergy in general, there are mechanisms that trigger individual work (Greek: „ergon“) through signs (Greek: „stigmata“) in the environment. If these signs are aspect of the task itself (e.g., form of an arc in a termites nest) then it is the sematectonic form of stigmergy. In sign-based stigmergy on the other hand, the task fulfillment is coordinated through additional markers (e.g., pheromones). Stigmergy requires agents to act upon changes in their environment that are caused by the agents themselves. These repetitions in space and time of small-scale activities in the environment result in a stable and self-organized system-level behavior.

A Router-agent takes decisions on the current configuration of its local environment. Through its actions it changes not only its own environment, but also the environment of other downstream and upstream Router-agents is changed too. The emergence of coordinated system-level behavior (here batching) requires the individual activities to be repeated as often as possible. As a consequence the quality of the task fulfillment by the agent system depends on the individual behavior as well as on the structure of the stigmergetic interactions as it is given by the layout of the transport system. Good batching behavior emerges if the following requirements for the layout are fulfilled:

“Alternative” Layout.—

There are many possible paths from an entry to an exit of the segment and these paths should intersect as often as possible to provide a large number of local routing points.

“Homogeneous”

Layout.—Most sequential routers have the same number of entries and exits to permit a homogeneous setting of agent parameters.

The Router-agents have been implemented and they have proven themselves in several different layouts.

One of the most effective layouts for batching is the matrix layout shown in Figure 5. The illustrated segment has only one entry (lower left) and one exit (upper right). The different shades indicate different product parameter values. The distributed control system self-organizes to sort a random inflow into a high-quality outflow.

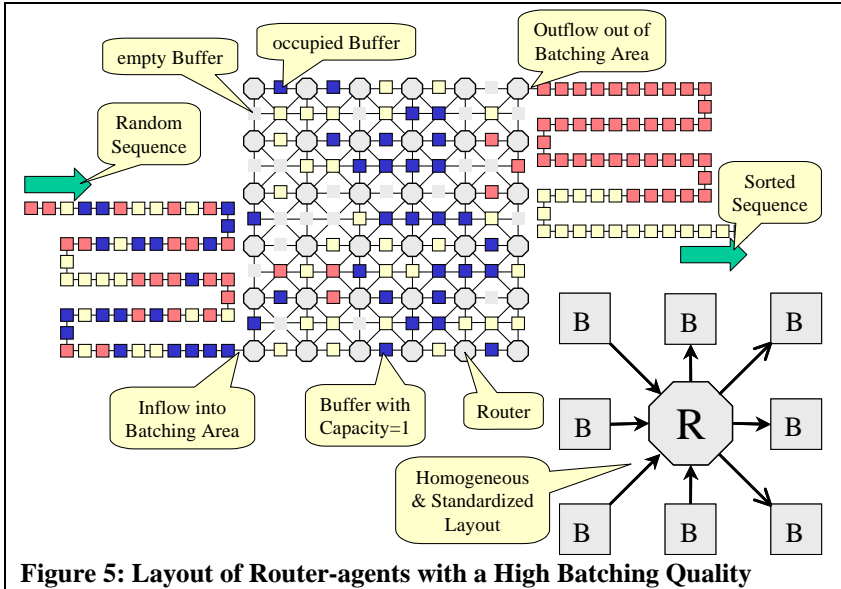


Figure 5: Layout of Router-agents with a High Batching Quality

3.1.3 Lessons Learned

The design of a self-organizing control system that creates batches in an initially random material flow was at that time spontaneous and intuitive. But its success raised the question whether there is a systematic approach to the design. What underlying (bio-)logic has to be employed to successfully engineer an agent system of industrial strength that yields global properties comparable to those of complex natural systems? What specific support may be given to an engineer who eventually implements such an agent society? Finally, in addition to the engineering aspect, the analytic aspect also comes into play. Is there a way to support the tuning and the evaluation of coordination mechanisms that give rise to emerging global properties?

Insect societies and the coordination mechanisms they employ have evolved to yield an optimal colony behavior. It is not the single insect behavior that is evaluated for its fitness because the individual cannot survive without the colony. The “goal” of the optimization process is to achieve the best system-level properties in the given environment by tuning the individual behavior only. The ordering force of self-organization supports the process [11].

The engineering of agent systems follows the same goal – the resulting system is evaluated by its global properties. Hence, engineering could also try to design all required properties into a homogeneous set of agents, employing, for instance, artificial evolution. But more intuitive, and hence easier to realize and to change, is it to split the system into clusters of different agents, each cluster providing different aspects in the overall system-level behavior.

When considering the general requirements for modern manufacturing, two different kinds of system-level properties are identified. Primarily, the operation of the manufacturing system must be robust, agile, and flexible in the face of changes and disturbances. But, when these primary goals are reached, the system is also required to fulfill the production goals as good as possible.

*G*Target pheromone when they were returning back to the "nest" after having found a target. Targets and Threats, once they were discovered also deposited pheromones. The ghosts used an evaluation equation that determined the effect that these pheromones had on their movement decisions. Initially the parameters weighting the impact of each pheromone on the ghost's decision were tuned by hand. It took one skilled engineer over three months of experimentation to get the ghosts to plan paths consistently. Eventually a genetic algorithm was used to tune the parameters online during the execution. Within a few minutes the genetic algorithm was able to outperform the hand-tuned algorithm by an order of magnitude. While observing how rapidly the ghosts were improving their performance the engineer commented, "I felt I should turn off the computer before the ghosts got smarter than me!"

In the OSD studies, ghosts were not used. The applications focused on coordinating the movements of multiple UAVs to accomplish surveillance, sensor cueing, target identification, and target tracking. The avatars deposited and reacted directly to the digital pheromones in the spatial grid.

There were several characteristics of these applications that relied on modeling to evaluate the system effectively:

- *Dynamic* – In JFACC targets and threats could move in addition to the UAVs. Planning was dynamic since the algorithm continuously updated its plans, refining them as time went on and adapting them to new information that arrived about the battlespace. In the OSD studies targets and threats were mobile, areas of interest could change, and surveillance priorities might change. These factors were represented in a modeling environment so their effects on the behavior and performance of the system could be determined.
- *Diverse* – in both the JFACC and OSD programs many kinds of entities were represented. We routinely needed to model between a half dozen to several dozen different kinds of units in the battlefield. An agent-based modeling environment provided a convenient mechanism to describe the behavior and capabilities of each of those units so they could participate in the scenarios.
- *Partial, Uncertain Information* – In both JFACC and OSD, not all targets or threats were known ahead of time, nor were all target locations known. Once a target or threat was discovered, it might be lost again if it was mobile and was not continuously tracked. In OSD, multiple sensor acquisitions were sometimes required to confirm a target identity. The modeling environment maintained the ground truth while the swarming algorithms received partial information about the environment through a sensor model. The accuracy of the world models built by the algorithms from the partial information could be calculated using the model's ground truth. The impact of varying levels of sensor information on the performance of the algorithm could also be evaluated through the modeling environment.
- *Stochastic* – In all the applications the behaviors of the entities (targets, threats, and the UAVs) were modeled stochastically. Stochasticity in the targets and threats represented the inability to predict exactly what an enemy unit might do in the future. This included if and where they moved, when and how they decided to attack, as well as their ability to hide and deceive. Stochasticity in the sensor model represented the variability in the sensor's ability to detect a target. Stochasticity in the UAVs was

used to represent the normal variations in response to a control command found in real-world vehicles and error rates in communications channels.

In JFACC an open source agent based modeling environment called Swarm was used to model the entities and their behaviors and Java was used to model the swarming avatars and ghosts. In the OSD project both the Extended Air Defense Test Bed (EADTB) and System Effectiveness Analysis Simulation (SEAS) simulation platforms were used to model the battlefield simulations. A Java-based model of the UAV platforms and environment was developed for running controlled experiments on the swarming algorithm. This model was used to characterize its behavior under various conditions and tune the algorithm.

3.2.2 Emergence in UAV Control

There were several examples of emergent phenomena that were identified in these projects, but one example stood out from the rest. In the JFACC project the ghosts were never able to plan the shortest path to the target for the gauntlet problem (Figure 6). As can be seen in the figure, the path moves away from the line of air defense threats before entering the mouth of the gauntlet. At first we considered this to be a shortcoming in the algorithm until one Subject Matter Expert (SME) looked at the result and asked, "How did you program the ghosts to plan such a path?" He informed us that air defenses can be linked so that information about an incoming target missed by one air defense unit could be passed to the next unit down the line giving it a better chance of taking out the target. In other words, the threat effectively increases as you move further down the line of air defense units. The safest path is to move further out beyond the threat envelope of the air defense unit exactly as the ghosts had done. Upon closer examination, the ghosts were simply reacting to the accumulation of threat pheromones from the multiple air defense units in the area and choosing a path around that higher concentration of threat. This integration of the threat pheromone was effectively implementing the more advanced threat avoidance concepts articulated by the SME. This is an example of an unplanned emergent phenomenon that was beneficial.

3.2.3 Lessons Learned

Getting the desired system level behavior in JFACC was not easy. Though the basics on ant path planning algorithms had been published by others, we found that it took numerous attempts to get the ghost agents to behave like respectable ants. Sometimes they would get stuck on a pheromone hill that might build up between the nest and the target and then end up going back and forth making the hill all the larger, but never completing the path back to the nest. Evaporation rates of some pheromones made it difficult to establish paths to far away targets. With multiple parameters that could be tuned, it quickly became apparent that some kind of automatic tuning method was needed. For this particular application a simple genetic algorithm proved to be highly effective. Through the objective function we could evolve behaviors that met several high level command objectives. Did you want the safest or quickest route? Did you want to focus on high priority targets or service multiple targets in an area? By indicating the commander's preferences in these high level terms, one could adjust the objective function automatically to evolve ghosts tuned for those particular preferences. At the same time, such an approach is not a panacea. In other applications we have found that identifying suitable objective functions can be challenging and commander's intent is not always easy to capture as a simple preference that can be described in an objective function. Still this seems to be a very promising aspect of the use of evolutionary techniques in tuning swarming algorithms.

One of the experimental questions posed at the start of the J9 experiments was the amount of effort that would be required to adapt the algorithm to new scenarios and applications. How much would the algorithm have to change when it was given differing tasks? How much tuning would be required for changes in the parameters of the same scenario? The answers to these questions turned out to be quite promising. A wide range of applications was investigated: surveillance, reconnaissance, target tracking, target trailing, sensor cueing, plume detection, mapping, and monitoring, and ad hoc communications network support. Different unmanned systems needed to cooperate with humans and other sensors in the battlefield. Varying number of units, varying threats, varying environments (urban, mountainous, and mixed), and varying threats had to be accommodated. Yet despite the diversity of the scenarios and the UAV taskings, the same algorithm effectively managed the UAV swarm to accomplish the mission more effectively than traditional approaches. At most a new task required the addition of a new pheromone into the equation and some slight hand tuning. For all the scenarios a maximum of three pheromones was all that was required to accomplish all the functions. Tuning the system proved fairly straightforward since the performance of the system was robust against large changes in individual parameter settings.

These algorithms were able to manage a large and diverse population of entities, they proved to be remarkably adaptable to a wide range of applications, and they were robust against failure demonstrating gradual degradation in system performance as the size of the swarm decreased. All this was accomplished with very simple agents essentially driven by a single equation and a few pheromones.

3.3 Battle Prediction

The DARPA RAID program [12] constructed a system that provides military commanders involved in urban combat with near-term predictions of what the adversary will do and recommendations of how to proceed with the battle. The program employed several different predictive mechanisms, including Bayesian belief networks [24] and game theory [15, 29]. Here, we focus on the BEE (Behavioral Evolution and Extrapolation), a component that evolves a swarming model of the engagement and uses it to learn the adversary's decision rules and then extrapolate them into the future [22].

3.3.1 Modeling Motives and Mechanisms

The Prussian military strategist von Clausewitz characterized war with two metaphors from daily life, "fog" and "friction" [30]. For von Clausewitz, the "fog of war" referred not only to the incomplete information available to commanders, but also to the distortion and unnatural appearance of the information that is available. The "friction of war" describes the deviations of the evolution of the battle from idealized causal models that do not take into account the stresses that war places on decision-makers.

Because of fog and friction, abstract reasoning about how a conflict will evolve is of little use. To understand a battle, one must fight it. Yet, in keeping with the dilemma we discussed in Section 1, actually engaging in a battle is far too costly to undertake, just to see how it will turn out.

As a result, models have long been used by military commanders to understand the battlefield and explore options *before* entering combat. Even before the days of computers, commanders would reconstruct battlefields on a sand table, or even on the ground in the field (Figure 7), so

that they could visualize the relative locations of friendly and enemy forces and the consequences of possible actions on either side. In 1916, F.W. Lanchester published a set of differential equations that expressed how the change in strength of each side in a conflict varies with the current strength of the other side [13]. Game theory, originally developed for economic analysis [31, 32], after WWII became a central tool for military planning.



Figure 7: Manual battlefield model.—Marines view a sand table representation of the operating area of Exercise Desert Punch constructed on the desert floor near Yuma, Arizona [17].

More recently, agent-based models such as EINSTEIN [10] and MANA [14] allow commanders to study the interactions of individual units.

To predict the course of an urban battle, RAID’s BEE maintains an agent-based simulation of the battlefield similar to EINSTEIN and MANA. Each unit is represented by a software agent that acts out the unit’s likely behavior in interaction with the environment and with other agents.

We know the doctrine taught to friendly troops, so in principle we could program their agents accordingly. But the friction of war has the effect of warping that doctrine in practice, and we do not have even a theoretical knowledge of our adversary’s doctrine. So BEE *learns* a unit’s decision rules by observing its recent behavior. Figure 8 illustrates the basic mechanism. The software structure that represents each unit actually consists of several agents, a “polyagent” [19]. A single persistent *avatar* generates and modulates a stream of *ghosts*. It generates a “guess” (initially, a random guess) as to the decision rules of the ghosts, inserts them into the simulation of the battle (which runs faster than real-life) some distance into the past, and evolves their decision rules against their observed behavior to learn what their rules (their

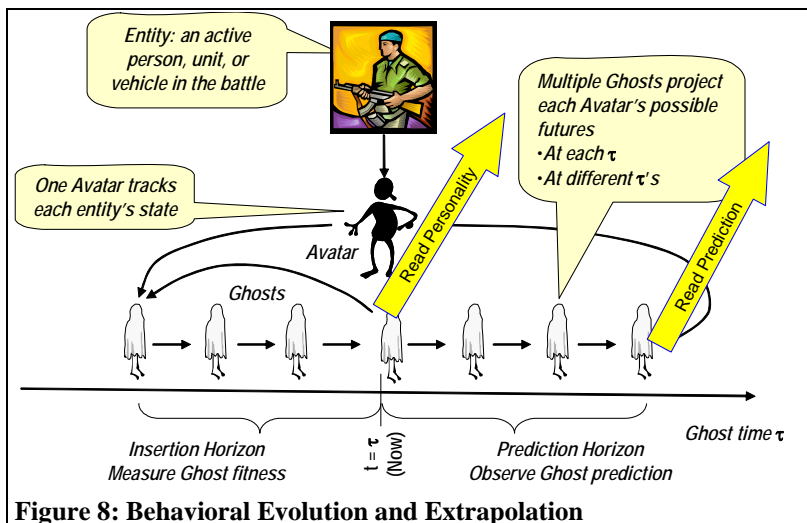


Figure 8: Behavioral Evolution and Extrapolation

“personality”) actually are (left-hand side of Figure 8). Then it allows the fittest ghosts to run into the future to generate predictions (right-hand side of Figure 8). In extensive wargames, these predictions have proven more accurate than those generated by game-theoretic mechanisms, and also than those generated by experienced military officers [18].

3.3.2 Emergence in RAID

RAID’s BEE exhibits emergence both in the evolution of unit personalities and in the extrapolation of agent behavior to form predictions.

We do not code an agent’s personality by hand. That personality is the result of an evolutionary process that observes how many different ghosts representing the agent traverse the recent past, and how their behavior compares with the observed behavior of the real-world unit. Each ghost has a single personality. The personality finally adopted by the avatar is a combination of many ghost personalities. The combination is not simply a sum or average of all ghost personalities (a linear operation). It favors the personalities that are fittest in terms of best matching the unit’s observed behavior when the personality is exercised against the constraints imposed by the environment (including the agents representing other units) in the recent past. These constraints are highly nonlinear, reflecting the fog and friction of war. In fact, since the initial ghost decision rules are largely random, the successful rules that result really reflect the environment, not the initial state of any ghost. The personality discovered by the BEE is a nonlinear function of that environment.

Similarly, the prediction generated by an avatar emerges from the nonlinear interactions of many different units. As the fittest ghosts of an avatar run into the future, they interact with the ghosts of other avatars, representing other units. Depending on each ghost’s decision rules, these interactions may result in engagements and fatalities, or in evasive behavior leading to diversion of a unit from its objective.

3.3.3 Lessons Learned

BEE was designed and constructed on the hypothesis that warfare is essentially an emergent process, and is best predicted by emergent mechanisms rather than logical analysis of the most rational actions for adversaries to take at each step. BEE’s model of rationality (the decision rules used by the agents) are extremely simple, consisting of a vector of weights that indicate how strongly a unit is attracted to or repelled from various environmental stimuli. At the level of a single agent, such a mechanism is more appropriate to an ant or a termite than to a human. However, we hypothesized that the complex interactions of fighting units would so dominate the evolution of the battle that a simplistic model of rationality combined with a mechanism for exploring rich interactions would yield superior predictions to the more sophisticated rational models embedded in game-theoretic predictors.

Our hypothesis was vindicated in realistic wargames. Repeatedly, BEE yielded more accurate predictions than game-theoretic predictors, and even out-predicted human military experts. In RAID, emergence is not just an ancillary feature of a model. It is the basis of the model and the key to its predictive success.

3.4 Clutter Agents

3.4.1 Modeling Motives and Mechanisms

Military operations unfold within the constraints of a particular physical environment. Operational effectiveness is a product of the joint “system” of the military operations and their environment. These emergent dynamics often vary widely as the state of the environment varies, to the point where the dynamics and thus the outcome of particular military operations may be completely dominated by the environment in which they are embedded.

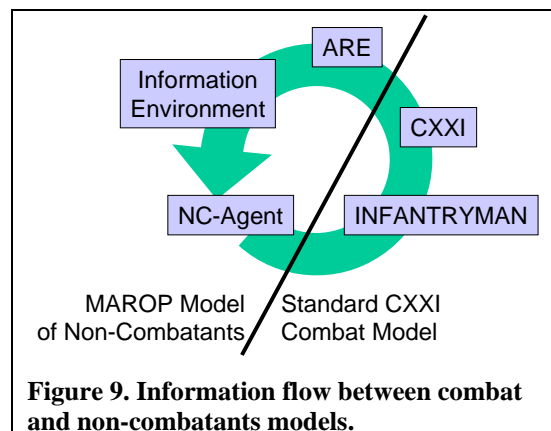
One aspect of the environment is the presence of non-combatants, which have to be included in realistic models of urban combat. But, the number of non-combatants is typically much larger than that of the combatants and one cannot assume detailed entity-tracking information to be available from representative real-world scenarios. Therefore, the approach to modeling this particular dynamical aspect of the environment of the combat operation must de-emphasize the individual and focus more on the resulting group-level behavior.

The emergent patterns of non-combatant movements in the area of combat operations are a function of the dynamics of the combat operation, the constraints of the physical environment, and the internal parameters (e.g., personality model) of the agents that comprise the population. Furthermore, the individual response to a particular situation (as presented by the graph and local pheromone concentrations on it) is non-deterministic and thus, the population is a system with non-deterministic dynamics too.

To analyze these emergent dynamics, we cannot rely on formal methods, because those typically break down in the face of the complexity of the system under scrutiny. Rather, we need to rely on visual inspection and systematic simulations that will provide us with the empirical data necessary to make analytic statements about the system-level behavior with relative (statistical) confidence.

We model non-combatants as simple software agents simultaneously embedded in “physical” environment simulated by the Army’s Combat-XXI (CXXI) platform as well as a shared dynamical information environment maintained by our extension to CXXI – the Agent Runtime Environment (ARE). The software agents guide the movement of their associated CXXI entity (here of type INFANTRYMAN) based on the local state of the information environment, which in turn is influenced by the unfolding combat operation.

Figure 9 illustrates the flow of information (i.e., state and control) between the CXXI simulation of the combat operation and our model of the non-combatants. The information environment represents the current state of the world in a spatial structure that captures the movement constraints of the non-combatants (road network in urban environment). Non-Combatant (NC) agents access their local world state and make movement decisions based on their Personality Model (see below). They translate these decisions into movements of their associated INFANTRYMAN entity in the CXXI simulation. In the simulation, these entities interact with the other entities modeled to represent the



combat scenario in the CXXI Playboard. The ARE regularly queries the Playboard for the state of all entities in the simulation as well as for particular simulation events (e.g., Munition Launched) and updates the information environment accordingly, which, in turn influences the decisions of the NC agents.

3.4.1.1 The Information Environment

The information environment provides an abstraction of the specific sensing and information sharing processes that occur among real-world non-combatants and it enables us to greatly reduce the complexity of the NC agents below that of the other entities in the CXXI simulation. Through the information environment, the NC agents learn directly about their movement options at a particular location, and they are provided with a pre-processed representation of the state of their local environment tailored to the specific needs of their decision process (Figure 10).

The information environment has a graph structure, comprising nodes and (bi-directional) edges among them. This graph structure compiles the constraints of the physical environment of the non-combatant entities into a set of discrete movement options (edges) available at a certain location (node). Thus, we reduce the reasoning process about where an agent could go next from complex (simulated) sensing operations to a table lookup.

In our current implementation, the graph structure is static – we do not represent dynamic terrain changes. At the initialization of the model, we read the graph structure (node locations and edge relationships) from a text file that has been generated by our GIS preprocessing tools.

Each node in the graph manages local state information accessible to the agents located at the node or at one of its immediate neighbors. We represent the local state of the simulation as concentrations (scalar numerical values) of a set of pheromone flavors. Pheromone flavors are data items that the non-combatant agents can identify and they form the “vocabulary” of these agents in their sensing of the state of the world. In our final demonstration scenario, we distinguish between seven different flavors. Six communicate the presence of alive or dead entities of the three sides (RED, BLUE, GREEN) and the seventh one indicates the nearby occurrence of a weapons fire event. The implementation of our CXXI extension allows us to define pheromone flavors and their specific characteristics easily in the setup of the model.

3.4.1.2 The NC Agent

Our model of non-combatants interacting with an unfolding combat operation distinguishes between the entity’s physical body (a CXXI simulation entity) and its agent “brain” that guides the movement of the entity. This distinction allows us to reduce the complexity of the agent further, since it does not have to concern itself with the actual “execution” of the motions. We have successfully applied this principle in other projects. For instance, it allowed us to develop an agent-based control system for a team of unmanned vehicles based on a simplified simulation

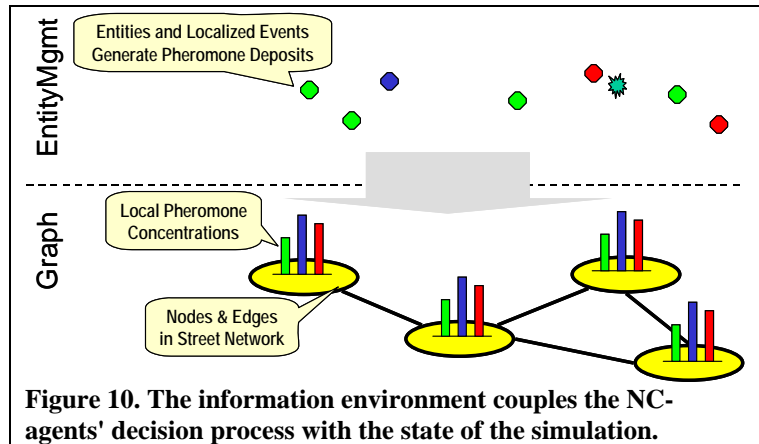


Figure 10. The information environment couples the NC-agents' decision process with the state of the simulation.

3.4.3 Lessons Learned

Our model of non-combatant behavior in combat operations reinforces two general insights that are important to the design of large-scale entity-based systems with emergent properties:

- 1) **Abstraction of Information Acquisition.**—While it is important to represent the detailed sensing capabilities of individual entities in small (3-5) agent systems, replicating these capabilities in large-scale systems is not only prohibitive (requires too much computation), but also unnecessary. MAROP demonstrated that we can achieve realistic population patterns even though the individual entities sense only numerically encoded abstract state information through synthetic pheromones.
- 2) **Probabilistic Personality Model.**—Just as in regards to modeling the details of the information acquisition process, modeling the detailed process of responding to this information is only necessary (and possible) in small-scale systems. Thus, traditional AI-based agent models are only useful in systems comprising only a few agents. MAROP demonstrated that a simple probabilistic response model to external stimuli is sufficient to produce the desired population behavior.

3.5 Resource Allocation

The DARPA ANT program explored innovative mechanisms for allocating resources to tasks (for example, assigning maintenance facilities and spare parts to military aircraft to maximize mission readiness, or assigning sensors to incoming targets to maximize coverage) [6]. The program's acronym refers to "Autonomous Negotiating Teams," and reflects a focus on using agent-based models to address this complex problem.

3.5.1 Modeling Motives and Mechanisms

Resource allocation is a textbook example of an NP-complete problem [8]. The number of possible combinations of resources and tasks, and the nature of their interactions, is so large that problems of realistic size cannot be solved exactly in any reasonable period of time.

The problem would not be nearly so difficult if resources could be allocated to tasks in a linear fashion. Allocating several billions of gasoline molecules to hundreds of aircraft is not a complex problem. One simply adds up all the fuel and divides by the number of aircraft. NP-complete problems are hard because they are nonlinear. One common type of nonlinearity is a threshold: an aircraft needs a fixed whole number of engines, so assigning three engines across four single-engine planes is not as simple as giving three-quarters of an engine to each plane. Another nonlinearity results from kitting. Each bolt needs a matching nut, and a solution that assigns all the bolts to one aircraft and all the nuts to another won't work. Because of these and other nonlinearities, successful patterns of allocation usually cannot be predicted analytically, but must be generated and tested, leading to emergent effects.

Our contribution to the ANT program focused on exploring these emergent effects, using tools and concepts inspired by statistical mechanics.

One of the models we explored is the "minority game" [4]. In the simplest form of the minority game (MG), N consumers (N odd) repeatedly seek access to $C = N - 1$ resources distributed evenly across $G = 2$ suppliers. In each turn of the game, each consumer selects one of the suppliers. Because $C < N$, one supplier will be overloaded. A consumer receives a point if the supplier it selects turns out not to be overloaded, and nothing if its supplier is overloaded. Each

consumer makes its choice with a look-up table, or “strategy,” mapping from vectors of m past system states to supplier choice. The larger m , the larger the space of strategies available to consumers, and the more of the past history each consumer can take into account in making its decision. Each consumer has two such tables. After each turn, it computes which strategy would have yielded the greatest payoff throughout the history of the game, and uses that strategy on the next turn. The objective of the game is to maximize the total points awarded to all consumers. The most points that can be awarded in any one turn is $C/2$, which happens when the population of the two suppliers differs by the least possible (one consumer). Thus a useful metric for the system’s overall behavior is the variance in the population of either of the suppliers, normalized by the number of consumers, σ^2/N . Because this metric varies inversely with total reward, it measures system inefficiency.

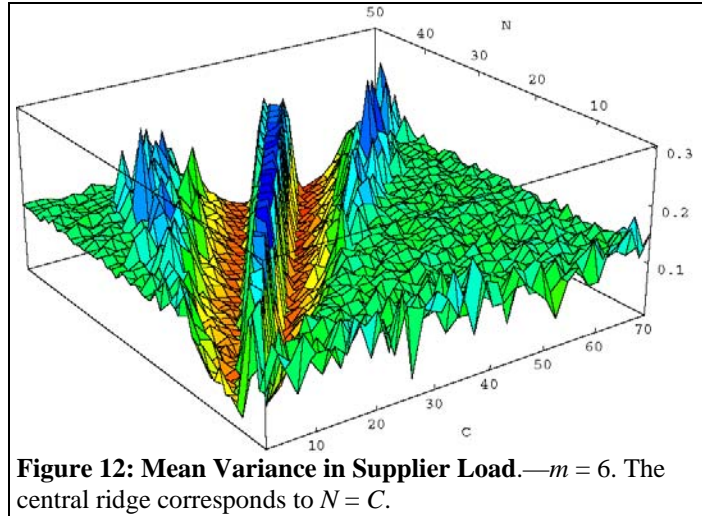


Figure 12: Mean Variance in Supplier Load.— $m = 6$. The central ridge corresponds to $N = C$.

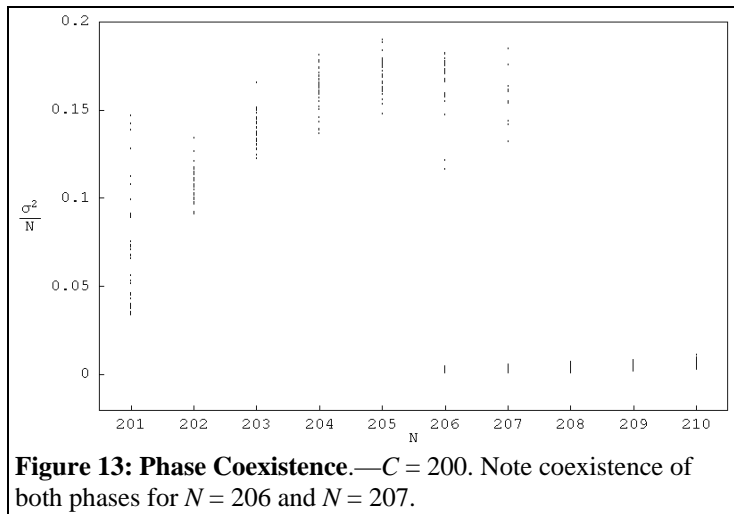


Figure 13: Phase Coexistence.— $C = 200$. Note coexistence of both phases for $N = 206$ and $N = 207$.

3.5.2 Emergence in ANT

As captured by these abstract models, resource allocation reflects emergence in several ways. Two of the most striking are phase shifts [20] and universality [21]. We discuss the first of these here.

A system exhibits a phase shift if some characteristic changes abruptly as a system variable increases or decreases. The notion originates in physics, where it describes phenomena such as the change of a substance from a liquid (with infinite symmetry) to a crystal (within limited symmetry) as the temperature drops through the freezing point. If the change can be demonstrated analytically as a mathematical discontinuity, we speak of a “phase transition.”

Figure 12 plots σ^2/N over the space of N (total demand) vs. C (total capacity). This surface shows seven distinct regions. At the extremities ($N \gg C$ or $C \gg N$) are two regions in which σ^2/N is fairly smooth as a function of N and C . As we move toward the diagonal, we pass into areas in which the dependence of σ^2/N on C and N is rougher. Moving further toward the diagonal from either direction, σ^2/N decreases and has a smoother dependence on C and N . Finally, very close to the diagonal σ^2/N increases, reaching its maximum value near $C=N$.

Figure 13 shows a slice at constant $C = 200$ through this peak, as it descends into the left-hand valley. In this region, the values of σ^2/N for different runs separate into two quite distinct groups.

For a given C and a range of N , there is a well-defined coexistence region between two very distinct phases, comparable to the coexistence of different phases in a physical system. The values of σ^2/N in Figure 12 at the top of the central peak and at the bottom of the neighboring valleys accurately reflect typical behavior of individual runs. But the average values along the flanks are atypical; any given individual run belongs either to the high or the low phase.

Understanding discontinuous behavior of this sort is critical to managing resources. It shows that a small change in demand can result in a huge change in the availability of whatever system the resources are supporting. As important as these discontinuities are, they are not explicit in the behavior of the individual agents. They emerge from the interactions of the agents, and modeling is an important tool to discover them.

3.5.3 Lessons Learned

Our research in emergent behavior (and phase shifts in particular) in resource allocation led to a number of important lessons, concerning both the nature of this behavior and tools for dealing with it.

Three principles are of broad applicability.

1. Phase shifts are ubiquitous. Using another model of resource allocation (graph coloring), we have found such shifts associated with almost every feature of the model [2].
2. The computational effort required for a system varies as one crosses a phase shift. Phase transitions have long been known in *decision* systems (yielding a “yes-no” answer) [5, 9], where they exhibit an easy-hard-easy effort profile: highly under- and over-constrained problems are easier to solve than those near the transition. The conventional wisdom is that the profile in *optimization* problems such as resource allocation is monotonic, becoming more difficult as constraints increase. On the contrary, the effort profile for optimization depends on the solution method, and we have demonstrated easy-hard-easy profiles for optimization problems [20].
3. Somewhat paradoxically, while some features of these transitions depend on the nature of the solution method, others do not, a phenomenon known as “universality.” The interactions of the agents can so constrain the system that their individual decision rules are of little importance [21].

To manage phase shifts, we have developed a number of tools.

- Once one recognizes the potential for phase shifts, it is natural to simulate a system extensively to identify configurations where they may occur. But searching a system’s parameter space exhaustively can require huge amounts of computation. By applying evolutionary methods to the search, we can explore the space much more efficiently [3].
- Real systems are not static. As they evolve, the locations of phase shifts may change, requiring the system to adapt dynamically to avoid catastrophic failures. Pheromone learning [23] enables individual agents to adjust their parameters incrementally based on locally available indicators of system state.
- Identifying such local indicators for global phenomena such as a phase shift is tricky. We have found that the option set entropy, the entropy over the set of choices

available to an agent at any moment, is a good candidate for such a local measure [2, 3].

4 Lessons

These experiences in multi-agent modeling, and others like them, yield several common insights. Emergence is real, it has the potential to help our work, and it can be managed to realize this potential.

The Reality of Emergence.—Emergence is not just a feature of the model. It is an integral part of the problem domains that we are modeling. Real entities (whether they be manufacturing stations, or UAV's, or soldiers, or civilians, or resources) interact nonlinearly, and those interactions produce emergent effects. A model of such domains with no emergence is an unfaithful model.

The Constructive Value of Emergence.—When emergent effects in multi-agent software were first noticed, some researchers suggested that they were harmful and should be avoided [16, 34]. Given our previous observation, these criticisms are self-defeating for modeling: if the world is emergent, our models must be as well. In addition, we have found that emergent behavior can yield important benefits. The schedules chosen in our manufacturing example, the routes discovered in our robotic example, and the predictions generated in our battle prediction example are all emergent outcomes of agent interactions that would not be possible without emergence. We could achieve realistic behaviors of clutter agents without detailed knowledge engineering because their interactions emergently yielded the same kinds of realistic behavior exhibited by real people under similar circumstances. Our experience suggests that recognition and exploitation of emergence can greatly reduce the complexity of models while enhancing the usefulness of their results.

Analyzing Emergence.—Realizing the promise of the previous paragraph requires developing an array of engineering tools and methods for dealing with emergence. Fortunately, many of these are already available in other domains. As our work in resource allocation showed, statistical physics is a rich source of techniques that can be applied to multi-agent models. Time-delay plots can reveal hidden regularity in apparently random time series. Awareness of universality can allow us to simplify models greatly, using simpler agents instead of more complex ones without losing the essential system-level outcomes that we wish to achieve. Pheromone learning, evolutionary detection of phase shifts, and the notion of option-set entropy are all useful tools to detect, monitor, and control emergence so that it enhances the promise of multi-agent modeling.

5 References

- [1] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways: For Your Mathematical Plays*. Academic Press, 1982.
- [2] S. Brueckner and H. V. D. Parunak. Information-Driven Phase Changes in Multi-Agent Coordination. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, Melbourne, Australia, pages 950-951, ACM, 2003.
<http://www.newvectors.net/staff/parunakv/AAMAS03InfoPhaseChange.pdf>.
- [3] S. Brueckner and H. V. D. Parunak. Resource-Aware Exploration of Emergent Dynamics of Simulated Systems. In *Proceedings of Autonomous Agents and Multi-Agent Systems*

- (AAMAS 2003), Melbourne, Australia, pages 781-788, ACM, 2003.
<http://www.newvectors.net/staff/parunakv/AAMAS03APSE.pdf>.
- [4] D. Challet, M. Marsili, and Y.-C. Zhang. *Minority Games: Interacting Agents in Financial Markets*. Oxford, UK, Oxford University Press, 2005.
- [5] P. Cheeseman, B. Kanefsky, and W. M. Taylor. Where the really hard problems are. In *Proceedings of IJCAI-91*, pages 331-337, Morgan Kaufmann, 1991. <http://ic-www.arc.nasa.gov/ic/projects/bayes-group/NP/ijcai91/paper/IJCAI91-paper.html>.
- [6] DARPA ITO. Autonomous Negotiating Teams (ANT). 1998. Web site, <http://www.dsic-web.net/ito/programs/ant/index.html>.
- [7] P. A. Fishwick. *Simulation Model Design and Execution: Building Digital Worlds*. Englewood Cliffs, NJ, Prentice Hall, 1995.
- [8] M. R. Garey and D. S. Johnson. *Computers and Intractability*. San Francisco, CA, W.H. Freeman, 1979.
- [9] T. Hogg, B. A. Huberman, and C. Williams. Phase Transitions and the Search Problem. *Artificial Intelligence*, 81:1-15, 1996.
<http://www.parc.xerox.com/istl/groups/iea/abstracts/PhaseTransitions/specialIssueIntro.html>.
- [10] A. Ilachinski. *Artificial War: Multiagent-based Simulation of Combat*. Singapore, World Scientific, 2004.
- [11] S. A. Kauffman. *The Origins of Order: Self Organization and Selection in Evolution*. Oxford University Press, 1995.
- [12] A. Kott. Real-Time Adversarial Intelligence & Decision Making (RAID). 2004.
<http://dtsn.darpa.mil/ixo/programdetail.asp?progid=57>.
- [13] F. W. Lanchester. *Aircraft in Warfare: The Dawn of the Fourth Arm*. London, Constable and Co, Ltd., 1916.
- [14] M. K. Lauren and R. T. Stephen. Map-Aware Non-uniform Automata (MANA)—A New Zealand Approach to Scenario Modelling. *Journal of Battlefield Technology*, 5(1 (March)):27ff, 2002. <http://www.argospress.com/jbt/Volume5/5-1-4.htm>.
- [15] W. M. McEneaney and R. Singh. Robustness against Deception. In A. Kott and W. McEneaney, Editors, *Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind*, pages 167-208. Chapman and Hall/CRC Press, Boca Raton, FL, 2006.
- [16] D. Moore and W. Wright. Emergent Behaviours Considered Harmful. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS 2003)*, Melbourne, Australia, pages 1070-1071, 2003.
- [17] National Wildfire Coordination Group. Sand Table Showroom. 2005.
http://www.fireleadership.gov/toolbox/documents/sand_table_showroom.html.
- [18] H. V. D. Parunak. Real-Time Agent Characterization and Prediction. In *Proceedings of International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'07), Industrial Track*, Honolulu, Hawaii, pages 1421-1428, ACM, 2007.
<http://www.newvectors.net/staff/parunakv/AAMAS07Fitting.pdf>
- [19] H. V. D. Parunak and S. Brueckner. Concurrent Modeling of Alternative Worlds with Polyagents. In *Proceedings of the Seventh International Workshop on Multi-Agent-Based Simulation (MABS06, at AAMAS06)*, Hakodate, Japan, Springer, 2006.
<http://www.newvectors.net/staff/parunakv/MABS06Polyagents.pdf>.

- [20] H. V. D. Parunak, S. Brueckner, J. Sauter, and R. Savit. Effort Profiles in Multi-Agent Resource Allocation. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS02)*, Bologna, Italy, pages 248-255, ACM, 2002.
www.newvectors.net/staff/parunakv/AAMAS02Effort.pdf.
- [21] H. V. D. Parunak, S. Brueckner, and R. Savit. Universality in Multi-Agent Systems. In *Proceedings of Third International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS 2004)*, New York, NY, pages 930-937, ACM, 2004.
<http://www.newvectors.net/staff/parunakv/AAMAS04Universality.pdf>.
- [22] H. V. D. Parunak and S. A. Brueckner. Extrapolation of the Opponent's Past Behaviors. In A. Kott and W. McEneaney, Editors, *Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind*, pages 49-76. Chapman and Hall/CRC Press, Boca Raton, FL, 2006.
- [23] H. V. D. Parunak, S. A. Brueckner, R. Matthews, and J. Sauter. Pheromone Learning for Self-Organizing Agents. *IEEE SMC*, 35(3 (May)):316-326, 2005.
<http://www.newvectors.net/staff/parunakv/ParunakIEEE.pdf>.
- [24] H. V. D. Parunak, P. E. Nielsen, S. Brueckner, and R. Alonso. Hybrid Multi-Agent Systems. In *Proceedings of Proceedings of the Fourth International Workshop on Engineering Self-Organizing Systems (ESOA'06)*, Hakodate, Japan, Springer, 2006.
<http://www.newvectors.net/staff/parunakv/ESOA06Hybrid.pdf>.
- [25] H. V. D. Parunak, M. Purcell, and R. O'Connell. Digital Pheromones for Autonomous Coordination of Swarming UAV's. In *Proceedings of First AIAA Unmanned Aerospace Vehicles, Systems, Technologies, and Operations Conference*, Norfolk, VA, AIAA, 2002.
www.newvectors.net/staff/parunakv/AIAA02.pdf.
- [26] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. Brueckner. Evolving Adaptive Pheromone Path Planning Mechanisms. In *Proceedings of Autonomous Agents and Multi-Agent Systems (AAMAS02)*, Bologna, Italy, pages 434-440, ACM, 2002.
www.newvectors.net/staff/parunakv/AAMAS02Evolution.pdf.
- [27] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. A. Brueckner. Demonstration of Digital Pheromone Swarming Control of Multiple Unmanned Air Vehicles. In *Proceedings of AAAI Infotech@Aerospace*, Arlington, VA, AIAA, 2005.
<http://www.newvectors.net/staff/parunakv/aiaa05.pdf>
- [28] J. A. Sauter, R. Matthews, H. V. D. Parunak, and S. A. Brueckner. Performance of Digital Pheromones for Swarming Vehicle Control. In *Proceedings of Fourth International Joint Conference on Autonomous Agents and Multi-Agent Systems*, Utrecht, Netherlands, pages 903-910, ACM, 2005.
<http://www.newvectors.net/staff/parunakv/AAMAS05SwarmingDemo.pdf>.
- [29] B. Stilman, V. Yakhnis, and O. Umanskiy. Strategies in Large-Scale Problems. In A. Kott and W. McEneaney, Editors, *Adversarial Reasoning: Computational Approaches to Reading the Opponent's Mind*, pages 251-286. Chapman and Hall/CRC Press, Boca Raton, FL, 2006.
- [30] C. von Clausewitz. *Vom Kriege*. Berlin, Dümmlers Verlag, 1832.
- [31] J. von Neumann. Zur Theorie der Gesellschaftsspiele. *Mathematische Annalen*, 100:295-320, 1928.
- [32] J. von Neumann and O. Morgenstern. *Theory of Games and Economic Behavior*. Princeton, Princeton University Press, 1944.
- [33] S. Wolfram. *A New Kind of Science*. Champaign, IL, Wolfram Media, 2002.

- [34] M. J. Wooldridge and N. R. Jennings. Pitfalls of Agent-Oriented Development. In *Proceedings of 2nd Int. Conf. on Autonomous Agents (Agents-98)*, Minneapolis, MN, pages 385-391, 1998. <http://citeseer.nj.nec.com/wooldridge98pitfalls.html>.