Chapter 9
EINSTein: A Multiagent-Based Model of Combat

Andrew Ilachinski

Artificial Life techniques – specifically, multiagent-based models and evolutionary learning algorithms – provide a powerful new approach to understanding some of the fundamental processes of war. This chapter introduces a simple artificial "toy model" of combat called EINSTein. EINSTein is designed to illustrate how certain aspects of land combat can be viewed as self-organized, emergent phenomena resulting from the dynamical web of interactions among notional combatants. EINSTein's bottom-up, synthesist approach to the modeling of combat stands in stark contrast to the more traditional top-down, or reductionist, approach taken by conventional military models, and it represents a step toward developing a complex systems theoretic toolbox for identifying, exploring, and possibly exploiting self-organized, emergent collective patterns of behavior on the real battlefield. A description of the model is provided, along with examples of emergent spatial patterns and behaviors.

9.1 Background

"War is... not the action of a living force upon lifeless mass... but always the collision of two living forces."

— Carl von Clausewitz, Prussian military strategist (1780–1831)

In 1914, F. W. Lanchester introduced a set of coupled ordinary differential equations – now commonly called the Lanchester equations (LEs) – as models of attrition in modern warfare [1]. In the simplest case of directed fire, for example, the LEs embody the intuitive idea that one side's attrition rate is proportional to the opposing side's size:

$$\begin{cases} \frac{dR}{dt} = -\alpha_B B(t), & R(0) = R_0, \\ \frac{dB}{dt} = -\alpha_R R(t), & B(0) = B_0, \end{cases} \tag{9.1}$$

where R_0 and B_0 are the initial red and blue force levels, respectively, and α_R and α_B represent the effective firing rates at which one unit of strength on one side causes attrition on the other side's forces. The closed-form solution of these equations is given in terms of hyperbolic functions as

$$\begin{cases} R(t) = R_0 \cosh\left(t\sqrt{\alpha_B \alpha_R}\right) - B_0 \sqrt{\alpha_B/\alpha_R} \sinh\left(t\sqrt{\alpha_B \alpha_R}\right), \\ B(t) = B_0 \cosh\left(t\sqrt{\alpha_B \alpha_R}\right) - R_0 \sqrt{\alpha_R/\alpha_B} \sinh\left(t\sqrt{\alpha_B \alpha_R}\right) \end{cases} \tag{9.2}$$

and satisfies the simple "square-law" state equation

$$\alpha_R \left[R_0^2 - R(t)^2\right] = \alpha_B \left[B_0^2 - B(t)^2\right]. \tag{9.3}$$

Similar ideas were proposed around that time by Chase [2] and Osipov [3]. These equations are formally equivalent to the Lotka–Volterra equations used for modeling the dynamics of interacting predator–prey populations [4]. Despite their relative simplicity, the LEs have since served as the fundamental mathematical models upon which most modern theories of combat attrition are based and are to this day embedded in many "state-of-the art" military models of combat. Taylor [5] provided a thorough mathematical discussion.

On the one hand, there is much to like about the LEs, since they are very intuitive and therefore easy to apply, and they provide relatively simple closed-form solutions. On the other hand, as is typically the case in the more general setting of nonlinear dynamical system theory, knowing the "exact" solution to a simplified problem does not necessarily imply that one has gained a deep insight into the problem. Moreover, almost all attempts to correlate LE-based models with historical combat data have proven inconclusive, a result that is in no small part due to the paucity of data. Most data consist only of initial force levels and casualties, and typically for one side only. Moreover, the actual number of casualties is usually uncertain because the definition of "casualty" varies (killed, killed + wounded, killed + missing, etc.).

Two noteworthy battles for which detailed daily attrition data and daily force levels do exist are the battle of Iwo Jima in World War II and the Inchon-Seoul campaign in the Korean War. While the battle of Iwo Jima is frequently cited as evidence for the efficacy of the classic LEs, the conditions under which it was fought were very close to the ideal list of assumptions under which the LEs themselves are derived. A detailed analysis of the Inchon-Seoul campaign has also proved inconclusive [6]. Weiss [7], Fain [8], Richardson [9], and others analyzed attrition in battles fought from 200 B.C. to World War II.

While the LEs may be relevant for the kind of static trench warfare and artillery duels that characterized most of World War I, they lack the spatial degrees of freedom to realistically model modern combat. They are certainly

too simple to adequately represent the more modern vision of combat, which depends on small, highly trained, well-armed autonomous teams working in concert, continually adapting to changing conditions and environments. The fundamental problem is that the LEs idealize combat much in the same way as Newton's laws idealize physics.

The two most significant drawbacks to using LEs to model land combat are that (1) they are unable to account for any spatial variation of forces (no link is established, for example, between movement and attrition) and (2) they do not incorporate the human factor in combat (i.e., the uniquely individual, often imperfect, psychology and decision-making capability of the human soldier.) While there have been many extensions to and generalizations of the LEs over the years, all designed to minimize the deficiencies inherent in their original formulation (including reformulations as stochastic differential equations and partial differential equations), most existing models remain essentially Lanchesterian in nature, the driving factor being force-on-force attrition.

9.2 Land Combat as a Complex Adaptive System

To address all of these shortcomings, the Center for Naval Analyses and the Office of Naval Research are exploring developments in a complex adaptive systems theory – particularly the set of agent-based models and simulation tools developed in the artificial life community – as a means of understanding land warfare in a fundamentally different way.

Military conflicts, particularly land combat, possess the key characteristics of complex adaptive systems (CASs) [10, 11, 12, 13]: Combat forces are composed of a large number of nonlinearly interacting parts and are organized in a command and control hierarchy; local action, which often appears disordered, induces long-range order (i.e., combat is self-organized); military conflicts, by their nature, proceed far from equilibrium; military forces, in order to survive, must continually adapt to a changing combat environment; and there is no master "voice" that dictates the actions of each and every combatant (i.e., battlefield action effectively proceeds according to a decentralized control).

A number of recent papers discuss the fundamental role that nonlinearity plays in combat. See, for example, Beckerman [14], Beyerchen [15], Hedgepeth [16], Ilachinski [17, 18], Miller and Sulcoski [19], Saperstein [20], and Tagarev and Nicholls [21]. The general approach of the EINSTein project is to extend these largely conceptual and general links that have been drawn between properties of land warfare and properties of complex systems into a set of practical connections and analytical research tools.

9.3 Agent-Based Modeling and Simulation

Models based on differential equations homogenize the properties of entire populations and ignore the spatial component altogether. Partial differential equations – by introducing a physical space to account for movement – fare somewhat better, but still treat the agent population as a continuum. In contrast, agent-based models (ABMs) consist of a discrete heterogeneous set of spatially distributed individual agents, each of which has its own characteristic properties and rules of behavior. These properties can also change as agents evolve in time.

Agent-based models of CASs are becoming an increasingly popular exploratory tool in the artificial life community and are predicated on the basic idea that the (often complicated) global behavior of a real system derives, collectively, from simpler, low-level interactions among its constituent agents [22]. Insights about the real-world system that an ABM is designed to model can then be gained by looking at the emergent structures induced by the interactions taking place within the simulation, as well as the feedback that these patterns might have on the rules governing the individual agents' behavior.[1] Agent-based simulations engender a significant shift in the kinds of questions that are asked of the real system being simulated. For example, where traditional models ask, effectively, "How can I characterize the system's top-level behavior with a few (equally top-level) variables?" ABMs instead ask, *"What low-level rules and what kinds of heterogeneous, autonomous agents do I need to have in order to synthesize the system's observed high-level behavior?"*

Perhaps the most important benefit of using an agent-based simulation to gain insight into why a system behaves the way it does – whether that system is a collection of traders on the stock market floor, neurons in a brain, or soldiers on the battlefield – is that once the simulation is used to generate the desired behavior, the researcher has immediate and simultaneous access to both the top-level (i.e., generated) behavior of the system and a low-level description of the system's underlying dynamics. Because they take an actively generative, or synthesist, approach to understanding a system, from the bottom up, ABMs are thus a powerful methodological tool for not just describing behaviors but also explaining why specific behaviors occur. While an analytical solution may provide an accurate description of a phenomenon, it is only with an agent-based simulation that one can fine-tune one's understanding of the precise set of conditions under which certain behaviors emerge.

[1] Two excellent recent texts on agent-based modeling, as applied to a variety of disciplines, are by Ferber [23] and Weiss [24]. Collections of papers focusing on systems that involve aspects of "human reasoning" are by Gilbert and Troitzsch [25], Gilbert and Conte [26], and Conte et al. [27]. More recently, ABMs have been applied successfully to traffic pattern analysis [28] and social evolution [29, 30].

In the context of modeling combat, agent-based simulations represent a fundamental shift from focusing on simple force-on-force attrition calculations to considering how complex, high-level properties and behaviors of combat emerge out of (sometimes coevolving) low-level rules of behaviors and interactions. The final outcome of a battle – as defined, say, by measuring the surviving force strengths – takes second stage to exploring how two forces might coevolve as a series of firefights and skirmishes unfold. ABMs are designed to allow the user to explore the evolving patterns of macroscopic behavior that result from the collective interactions of individual agents, as well as the feedback that these patterns might have on the rules governing the individual agents' behavior.

9.4 EINSTein

EINSTein (*Enhanced ISAAC Neural Simulation Tool*) is an adaptive ABM of combat and is an outgrowth of a more far-reaching project to develop a complexity-based fundamental theory of warfare [31]. EINSTein [32] builds upon and extends an earlier proof-of-concept, DOS-based combat simulator called ISAAC (*Irreducible Semi-Autonomous Adaptive Combat*), which was developed for the US Marines Corps [33]. All approved-for-public-release documents, project reports and summaries, tutorials, sample runs, and an auto-install program for Windows-based PCs may be downloaded from [70]. Details of the EINSTein toolkit are provided in [34].

EINSTein represents one of the first systematic attempts, within the military operations research community, to simulate combat – on a small to medium scale – by using autonomous agents to model individual behaviors and personalities rather than specific weapons. Because agents are all endowed with a rudimentary form of "intelligence," they can respond to a very large class of changing conditions as they evolve during battle. Because of the relative simplicity of the underlying dynamical rules, EINSTein can rapidly provide outcomes for a wide spectrum of tunable parameter values defining specific scenarios and can thus be used to effectively map out the space of possible behaviors.

Fundamentally, EINSTein addresses the basic question: *"To what extent is land combat a self-organized emergent phenomenon?"* Or, more precisely, "What are the conditions under which high-level patterns (such as penetration, flanking maneuvers, attack, etc.) emerge from a given set of low-lying dynamical primitive actions (move forward, move backward, approach/retreat-from enemy, etc.)." As such, EINSTein's intended use is not as a full system-level model of combat but as an interactive toolbox – or "conceptual playground" – in which to explore high-level emergent behaviors arising from various low-level (i.e., individual combatant and squad-level) interaction rules. The idea behind developing this toolbox is emphatically not to model in de-

tail a specific piece of hardware (an M16 rifle or M101 105mm howitzer, for example). Instead, the idea is to explore the middle ground between – at one extreme – highly realistic models that provide little insight into basic processes and – at the other extreme – ultraminimalist models that strip away all but the simplest dynamical variables and leave out the most interesting real behavior that is, to explore the fundamental dynamical trade-offs among a large number of notional variables.

The underlying dynamics is patterned after mobile cellular automata rules [35] and are somewhat reminiscent of Braitenberg's vehicles [36]. Mobile cellular automata have been used before to model predator–prey interactions in natural ecologies [37]. They have also been applied to combat modeling [38], but in a much more limited fashion than the one used by EINSTein.

9.4.1 Features

EINSTein's major features include the following:

- Dialog-driven I/O, using a Windows graphical user interface (GUI) front-end
- Object-oriented C++ source code base
- Integrated natural terrain maps and terrain-based adaptive decisions
- Context-dependent and user-defined agent behaviors
- Multiple squads, with intersquad communication links
- Local and global command-agent dynamics
- Genetic algorithm toolkit to tailor agent rules to desired force-level behaviors
- Data collection and multidimensional visualization tools
- Mission fitness-landscape profilers
- Over 250 user-programmable functions on the source code level

Fig. 9.1 provides a screenshot of a typical run-session in EINSTein. The screenshot contains three active windows: main battlefield view (which includes passable and impassable terrain elements), trace view (which shows color coded territorial occupancy), and combat view (which provides a gray-scaled filter of combat intensity). All views are simultaneously updated during a run. Toward the right-hand side of the screenshot appear two data dialogs that summarize red and blue agent parameter values. Appearing on the lower left side and along the bottom of the figure are time-series graphs of red and blue center-of-mass coordinates (as measured from the red flag) and the average number of agents within red and blue agent's sensor ranges, and a dialog that allows the user to define communication relays among individual squads.

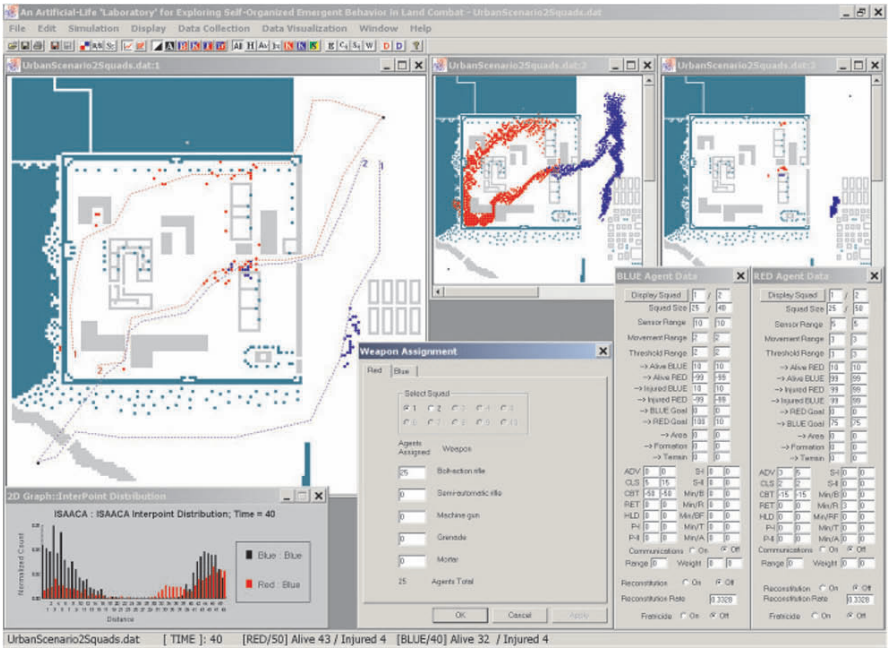

Fig. 9.1 Screenshot of EINSTein's GUI front-end.

9.4.2 Source Code

EINSTein is written and compiled using Microsoft's Visual C++ and makes use of Pinnacle Publishing Inc.'s Graphics Server[2] for displaying time-series plots and three-dimensional (3D) fitness-landscapes. EINSTein consists of roughly 100K lines of object-oriented source code.

The source code is divided into three basic parts: (1) the *combat engine* (parts of which are summarized below); (2) the *graphical user interface*; and (3) the *data-collection/data-visualization functions*. These parts are essentially machine (i.e., CPU and/or operating system) independent and may be compiled separately. EINSTein's source code base is thus highly portable and is relatively easy to modify to suit particular problems and interests. For example, an EINSTein-based combat environment may be developed as a stand-alone program on a CPU platform other than the original MS Windows target machine used for EINSTein's original development. Any developer/analyst interested in porting EINSTein over to some other machine and/or operating system is tasked only with providing his own machine-specific GUI as a "wrap-around" to the stand-alone combat and data-visualization engines (that may be provided as dynamic-link libraries – DLLs). Moreover, it is very

[2] Graphics Server is a commercial plug-in, licensed from Pinnacle Publishing, Inc. [68].

easy to add, delete, and/or change the existing source code, including making complicated changes that significantly alter how agents decide their moves.

At the heart of EINSTein lies the combat engine (discussed later). The combat engine processes all run-time, combat-related logical decisions and is the core script upon which multiple time-series data collection, fitness-landscape sweeps over the agents' parameter space, and genetic algorithm searches all depend.

9.4.3 Design Philosophy

"Things should be as simple as possible, but not simpler." — Albert Einstein

EINSTein's design is predicated upon two guiding principles: (1) to keep all dynamical components and rules as simple as possible (with a view toward optimizing the trade-off between run time and realism) and (2) to treat all forms of information (and the way in which all information is processed locally by agents) in a contextually consistent manner. The meaning of this second principle will become clear in the exposition below.

9.4.3.1 Simplicity

The first guiding principle is to keep things *simple*. Specifically, EINSTein is designed to make it as intuitive as possible for the user to program specific agent behaviors. This is done by deliberately keeping the set of combat and movement rules small and by defining those rules as simply as possible. Thus, the power projection rule is essentially "target and fire upon any enemy agent within a threshold fire range" rather than some other, more complicated (albeit, possibly more physically realistic) prescription. The idea is to qualitatively probe the behavioral consequences of the interaction among a large number of notional variables, not to provide an explicit detailed model of the minutiae of real-world combat.

9.4.3.2 Consistency

The second guiding principle is keep things *consistent*. All dynamical decisions – whether they are made by individual agents, by local or global commanders, or by the user (when scripting a scenario's objectives) – consist of boundedly rational (i.e., locally optimal) penalty assessments. Actions are based on an agent's *personality* (see later), which consists of numerical weights that attach greater or lesser degrees of relative importance to each factor relevant to selecting a particular move in a given local context (from

the point of view of a given agent). It is in this sense that all forms of information, on various levels, are treated on a consistent basis.

The decisions taking place on different levels of the simulation all follow the same general template of probing and responding to the environment. Each decision consists of a personality-mediated "answer" to the following three basic questions:

- What are my immediate and long-term goals?
- What do I currently know about my local environment?
- What must I do to attain my goals?

As we will see in detail ahead, at the most primitive level, each agent cares only about "moving toward" or "moving away from" all other agents and/or his own and the enemy's flag. An agent's personality prescribes the relative weight assigned to each of these immediate "goals." On the other hand, a global commander must weigh such features as overall force strength, casualty rate, rate of advance, and so on in order to attain certain long-term goals. Local and supreme commanders have their own unique concerns. While the actual decisions are different in each case and on each information level – for example, an individual agent's decision to "stay put" in order to survive is quite different and uses a different form of information, from a global commander's drive to "get to the enemy flag as quickly as possible" – the general manner in which these decisions are made is the same.

9.4.4 Program Flow

A typical sequence of programming steps during an interactive run consists of multiple loops through the following basic steps:

1. *Initialize battlefield and agent distribution parameters.*
2. *Initialize time-step counter.*
3. *Adjudicate combat.*
4. *Refresh battlefield graphics display.*
5. *Find context-dependent personality weight vector for each red and blue agent.*
6. *Compute local penalty function to determine best move.*
7. *Move agents to their newly selected position (or leave them where they are).*
8. *Refresh graphics display and loop through steps 3–7.*

The most important parts of this skeletal structure are the adjudication of combat, the adaptation of personality weights, and the decision-making process that each agent goes through in choosing its next move. Before describing the details of what each of these steps involves, we must first discuss how each agent partitions its local information. During interactive runs (i.e.,

whenever the fitness-landscape profiler and genetic algorithm breeder batch modes are both inactive), the user can pause the simulation at any time to make on-the-fly changes to any, or all, agent parameters (including adding or subtracting "playing" agents) and then resume the run with the changed values.

9.5 Combat Engine

9.5.1 Agents

The basic element of EINSTein is an agent, which loosely represents a primitive combat unit (infantryman, tank, transport vehicle, etc.) that is equipped with the following characteristics:

- *Doctrine:* a default local-rule set specifying how to act in a generic environment
- *Mission:* goals directing behavior
- *Situational awareness:* sensors generating an internal map of environment
- *Adaptability:* an internal mechanism to alter behavior and/or rules

Each agent exists in one of three states: alive, injured, or killed. Injured agents can (but are not required to) have different personalities and offensive/defensive characteristics from when they were alive. For example, the user can specify that injured agents are able to move half as far, and shoot half as accurately, as their "alive" counterparts. Up to 10 distinct groups (or "squads") of personalities, of varying sizes, can be defined. The user can also specify how agents from one squad react to agents from other squads.

Each agent has associated with it a set of ranges (sensor range, fire range, communications range, etc.), within which it senses and assimilates various forms of local information, and a personality, which determines the general manner in which it responds to its environment. A global rule set determines combat attrition, reconstitution, and (in future versions) reinforcement. EINSTein also contains both local and global commanders, each with their own command radii and obeying an evolving command-and-control (C2) hierarchy of rules.

9.5.2 Battlefield

The putative combat battlefield is represented by a two-dimensional lattice of discrete sites. Each site of the lattice may be occupied by one of two kinds of agents: red or blue. The initial state consists of user-specified formations

of red and blue agents positioned anywhere within the battlefield. Formations may include squad-specific bounding rectangles or may be completely random. Red and blue flags are also typically (but not always) positioned in diagonally opposite corners. A typical goal, for both red and blue agents, is to reach the enemy's flag.

EINSTein includes an option to add terrain elements. Terrain can be either impassable or passable. If passable, the user can also tune an agent's behavior to a particular terrain type. For example, if an agent is positioned within "heavy brush," its movement range and visibility (from other nearby agents) may be curtailed.

9.5.3 Agent Personalities

Each agent is equipped with a user-specified personality – or internal value system – nominally defined by a six-component personality weight vector, $\mathbf{w} = (w_1, w_2, \ldots, w_6)$, where $-1 \leq w_i \leq 1$ and $\Sigma_i |w_i| = 1$. The components of $\mathbf{w}$ specify how an individual agent responds to specific kinds of local information within its sensor range.

The personality weight vector may be health dependent; that is, w_{alive} need not, in general, be equal to w_{injured}. The components of $\mathbf{w}$ can also be negative – in which case they signify a propensity for moving away from, rather than toward, a given entity.

9.5.4 Penalty Function

An agent's personality weight vector is used to rank each possible move according to a penalty function. The simplest penalty function effectively measures the total distance that the agent will be from other agents (including both friendly and enemy agents) and from its own and enemy flags, weighing each component distance by the appropriate component of the personality weight vector, $\mathbf{w}$. An agent moves to the position that incurs the least penalty; that is, an agent's move is the one that best satisfies its personality-driven desire to "move closer to" or "farther away from" other agents in given states and either of the two flags. The general form of the penalty function is given by:

$$Z(B_{x,y}) = \frac{1}{\sqrt{2}r_S}\left[\begin{array}{l} \dfrac{\omega_{AF}}{N_{AF}}\displaystyle\sum_{i\in AF}^{N_{AF}} D_{i,B_{x,y}} + \dfrac{\omega_{AE}}{N_{AE}}\displaystyle\sum_{j\in AE}^{N_{AE}} D_{j,B_{x,y}} \\ \\ +\dfrac{\omega_{IF}}{N_{IF}}\displaystyle\sum_{i\in IF}^{N_{IF}} D_{i,B_{x,y}} + \dfrac{\omega_{IE}}{N_{IE}}\displaystyle\sum_{j\in IE}^{N_{IE}} D_{j,B_{x,y}} \end{array}\right] \tag{9.4}$$

$$+\omega_{FF}\frac{D^{new}_{FF,B_{x,y}}}{D^{old}_{FF,B_{x,y}}} + \omega_{EF}\frac{D^{new}_{EF,B_{x,y}}}{D^{old}_{EF,B_{x,y}}},$$

where $B_{x,y}$ is the (x, y) coordinate of battlefield B; AF, IF, AE, and IE represent respectively the sets of alive friends, injured friends, alive enemies, and injured enemies within the given agent's sensor range, r_S; w_i are the components of the personality weight vector; $\sqrt{2}r_S$ is a scale factor; N_X is the total number of elements of type X within the given agent's sensor range (e.g., N_F is the number of alive friends within range r_S); $D_{A,B}$ is the distance between elements A and B; FF and EF denote the friendly and enemy flags, respectively; and represent distances computed using the given agent's new (candidate move) position and old (current) position, respectively.

A penalty is computed for each possible move; that is, for each of the $N = (2r_m + 1)^2$ possible sites to which an agent can "step" in one time step: $Z_1(B_{x,y})$, $Z_2(B_{x+1,y})$, $Z_3(B_{x-1,y}), \ldots, Z_N(B_{x+n,y+n})$. The actual move is the one that incurs the least penalty. If there is a set of moves (consisting of more than one possible move) all of whose penalties are within $\epsilon_{\text{Penalty}} \geq 0$ of the minimal penalty, an agent randomly selects the actual move among the candidate moves making up that set. Users can also define paths near which agents must try to stay while maneuvering toward their ultimate goal.

The penalty function shown above includes only a few relative-proximity-based weights. In practice, the penalty function is more complicated and incorporates more terms, although its basic form is the same. Additional terms can include the propensity for maintaining the minimum distance from friendly or enemy agents, staying near a designated patrol area, the cost of traversing terrain, desire for finding local cover (from fire) and/or concealment (from enemy sensors), and combat intensity (see Table 9.1).

9.5.5 Meta-rules

An agent's personality may be augmented by a set of meta-rules that tell it how to alter its default personality according to dynamic environmental contexts. A typical meta-rule consists of altering a few of the components of an agent's personality vector according to a set of associated local threshold constraints. The three simplest meta-rule classes effectively define the

Weight	Meaning = Relative Weight for...
w_{AF}	...moving toward/away from alive friendly agents
w_{IF}	...moving toward/away from injured friendly agents
w_{AE}	...moving toward/away from alive enemy agents
w_{IE}	...moving toward/away from injured enemy agents
w_{FF}	...moving toward/away from friendly flag
w_{EF}	...moving toward/away from enemy flag
w_{BB}	...moving toward/away from the boundary of battlefield
w_{area}	...staying near some (squad-specific) area
w_{squad}	...maintaining formation with own squad-mates
$w_{\mathrm{fire-team}}$	...maintaining formation with own fireteam-mates
S_{ij}	...how agents from squad S_i react to agents from squad S_j
SS'_{ij}	...how agents from squad S_i react to agents from enemy squad S_j
w_{LC}	...moving toward/away from local commander
w_{obeyLC}	...obeying orders issued by local commander
w_{terrain}	...moving toward/away from terrain elements
$w_{\mathrm{enemy-fire}}$	...moving toward/away from enemy agents that have fired on agent

Table 9.1 A partial list of EINSTein's *primitive weight* set

local conditions under which an agent is allowed to *advance toward enemy flag* (class 1), *cluster with friendly forces* (class 2), and *engage the enemy in combat* (class 3).

For example, a class-1 meta-rule prevents an agent from advancing toward the enemy flag unless it is locally surrounded by a threshold number of friendly agents; that is, it is a notional indicator of local combat support. A class-2 meta-rule can be used to prevent an agent from moving toward friendly agents once it is surrounded by a threshold number. Finally, a class-3 meta-rule can be used to fix the local conditions under which an agent is allowed to move toward or away from possibly engaging an enemy agent in combat. Specifically, an agent is allowed to engage an enemy if and only if the difference between friendly and enemy force strengths locally exceeds a given threshold.

Other meta-rule classes include *retreat*, *pursuit*, *support*, and *hold position*. A global rule set determines combat attrition (see later), communication, reconstitution, and (in future versions) reinforcement. EINSTein also contains both local and global commanders, each of which is equipped with its own unique command-personality and area of responsibility, and obeys an evolving command and control hierarchy of rules. Table 9.2 summarizes some of EINSTein's meta-rules.[3]

[3] Note that threshold constraints (τ_{Advance}, τ_{Cluster}, and Δ_{Combat}) are explicitly defined only for the first three meta-rules. These meta-rules are used in the sample runs discussed later. In fact, each of the meta-rules appearing in Table 9.2 has one or more threshold constraints associated with it, and the set also requires additional logic to dynamically resolve ambiguities as they arise during the course of a run. Details are in [31].

Meta-rule	Description
w_{AF}	...moving towards/away from alive friendly agents
Advance	Advance to enemy flag if the number of friends $\geq \tau_{\mathrm{Advance}}$
Cluster	Stop seeking friends if number of friends $\geq \tau_{\mathrm{Cluster}}$
Combat	Engage enemy if the $N_{\mathrm{friends}} - N_{\mathrm{enemies}} \geq \Delta_{\mathrm{Combat}}$
Hold	Hold current position
Pursuit-I	Temporarily turn off pursuit of enemy agents
Pursuit-II	Temporarily turn exclusive pursuit on
Retreat	Retreat toward own flag
Run Away	Run away, fast, from enemy agents
Support-I	Provide support for nearby injured
Support-II	Seek support from nearby friends
Min-D Friend	Maintain minimum distance from all friendly agents
Min-D Enemy	Maintain minimum distance from all enemy agents
Min-D Flag	Maintain minimum distance from all friendly flags
Min-D Terrain	Maintain minimum distance from terrain
Min-D Area	Maintain minimum distance from a fixed area on battlefield

Table 9.2 A partial list of EINSTein's *meta-rule* set

9.5.6 Combat

During the combat phase of an iteration step for the whole system, each agent X (on either side) is given an opportunity to fire at all enemy agents Y_i that are within a fire range r_F of X's position. If an agent is shot by an enemy agent, its current state is degraded either from alive to injured or from injured to dead. Once killed, an agent is permanently removed from the battlefield. The probability that a given Y_i is shot is fixed by user-specified single-shot probabilities. Weapons are assigned to individual agents and are either point-to-point (i.e., rifles) or area destruction (i.e., grenades).

By default, all enemy agents within a given agent's fire range are targeted for a possible hit. However, the user has the option of limiting the number of enemy targets that can be engaged simultaneously. If this option is selected and the number of enemy agents within an agent's fire-range exceeds a user-defined threshold number (say N), then N agents are randomly chosen among the agents in this set. Grenades include additional targeting logic (to maximize expected inflicted damage on the enemy).

This basic combat logic may be enhanced by three additional functions: (1) *defense*, which adds a notional ability to agents to be able to withstand a greater number of "hits" before having their state degraded; (2) *reconstitution*, which adds a provision for previously injured agents to be reconstituted to their alive state; and (3) *fratricide* ("friendly fire"), which adds an element of realism by making it possible to inadvertently hit friendly forces.

9.5.7 Run Modes

EINSTein can be run in three basic modes (see EINSTein's *User's Guide* [31]):

- *Interactive mode*, in which the combat engine is run interactively using a fixed set of rules. This mode, which allows the user to make on-the-fly changes to the values of any (or all) parameters defining a given run, is particularly well suited for playing simple *"What if?"* scenarios. The interactive mode also makes it easy to search for interesting emergent behavior.
- *Data-collection mode*, in which the user can (1) generate time series of various changing quantities describing the step-by-step evolution of a battle and (2) keep track of certain measures of how well mission objectives are met at a battle's conclusion. Additionally, the user can generate behavioral profiles on two-dimensional slices of EINSTein's N-dimensional parameter space.
- *Genetic algorithm "breeder" mode*, in which a genetic algorithm is used to breed an agent force that is optimally suited for performing a specific mission against a fixed enemy force. This mode is designed to suggest ways in which ABMs may eventually be used to evolve real-world tactics and strategies.

9.6 Sample Patterns and Behavior

EINSTein possesses a large repertoire of emergent behaviors: *forward advance*, *frontal attack*, *local clustering*, *penetration*, *retreat*, *attack posturing*, *containment*, *flanking maneuvers*, and *"Guerrilla-like" assaults*, among many others. Moreover, behaviors frequently arise that appear to involve some form of intelligent division of red and blue forces to deal with local firestorms and skirmishes, particularly those forces whose personalities have been bred (via a genetic algorithm) to perform a specific mission. It is important to point out that such behaviors are not hard-wired but are, rather, an emergent property of a decentralized, but dynamically interdependent, swarm of agents.

Fig. 9.2 shows screen captures of spatial patterns resulting from 16 different rules and illustrates the diversity of behaviors that emerges out of a relatively simple set of rules. (Note that the sample patterns shown here are for clashing red and blue forces consisting of a *single* squad. Multisquad scenarios, in which agents belonging to different squads obey different rules, and interact with one another according to an additional layer of micro-rules, often result in considerably more complicated emergent behaviors.) An important long-term goal is for EINSTein to be flexible enough to serve as a general tool (that transcends the specific notional combat environment to which it is obviously

tailored) for exploring the still very poorly understood mapping between micro-rules and emergent macro-behaviors in complex adaptive systems.

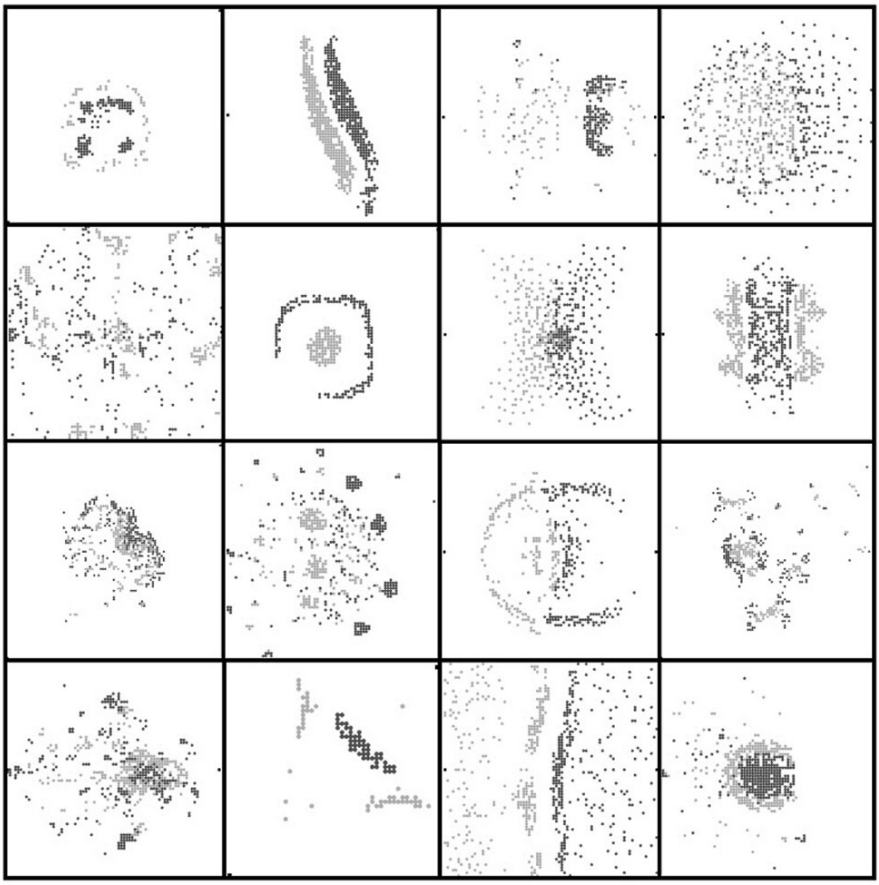

Fig. 9.2 A sampling of emergent spatial patterns of agents obeying EINSTein's micro-rules. Each of the 16 squares represents a different rule and contains a single snapshot of a typical run.

9.6.1 Qualitative Classes of Behavior

Simulations run for many different scenarios and initial conditions suggest that EINSTein's collective behavior generally falls into one of six broad qualitative classes (labeled, suggestively, according to different kinds of fluid flow):

- *Laminar flow*, which typically consists of one (or, at most, a few) well-defined "linear" battlefronts. This class is so named because it is visually suggestive of laminar fluid flow of two fluids and is reminiscent of static trench warfare in World War I. Laminar rules can actually be divided into two types of behaviors, characterized according to a system's overall stability (i.e., according to whether the system is stable, or not stable, to initial conditions).
- *Viscous flow*, in which the unfolding battle typically consists of a single tight cluster (or, at most, a few clusters) of interpenetrating red and blue agents.
- *Dispersive flow*, in which – as soon as red and blue agents maneuver within view of the opposing side's forces – the battle unfolds as a single, explosive, dispersion of forces. Dispersive systems exhibit little, if any, of the "front-like" linear structures that form for laminar-flow rules.
- *Turbulent flow*, in which combat consists of either spatially distributed, but otherwise confined and/or clustered individual combat zones, or a series of close-to space-filling local firestorms. In either case, there is almost always a significant degree of local maneuvering.
- *Autopoeitic flow*, in which agents self-organize into persistent dissipative structures. These formations typically maintain their integrity for long times (on the scale of individual agents entering and leaving the structure) and undergo "higher-level" maneuvering, including longitudinal motion and rotation.[4]
- *Swarming*, in which agents self-organize into nested swarms of attacking and/or defending forces.

We should be quick to point out that this taxonomy is neither complete nor well defined, in a mathematical sense. Because of the qualitative distinctions among classes, there is considerable overlap among them. Moreover, a given scenario, as it unfolds in time, usually consists of several phases of behavior during which one class predominates at one time and other classes at other times. Indeed, for such cases, which occur frequently, it is of considerable interest to understand the nature of the transition between distinct behavioral phases. For example, the initial stages of a scenario may unfold in typically laminar fashion and suddenly transition over into a turbulent phase.

A finer distinction among these six classes can be made on the basis of a more refined statistical analysis of emergent behavior. There is strong evidence to suggest, for example, that while attrition rates for certain classes of rules display smooth Gaussian statistics, other classes (overlapping with viscous-flow and turbulent-flow rules) display interesting fractal power-law scaling behaviors [40]. Insofar as the "box-counting" fractal dimension [41] is useful for describing the degree of agent clustering on the battlefield, it

[4] *Autopoiesis* refers to dynamical systems that are simultaneously self-creating and self-maintaining. It was introduced as an explanatory mechanism within biology by Maturana and Varela [39].

can also be used as a simple discriminant between laminar and turbulent classes of behavior. Measuring temporal correlations in the time series of various statistical quantities describing combat is also useful in this regard. The case studies presented here are selected mainly to highlight the qualitative behavioral classes described previously.

9.6.2 Lanchesterian Combat

On the simplest level, EINSTein is an interactive, exploratory tool that allows users to take conceptual excursions away from Lanchesterian oversimplifications of real combat. It is therefore of interest to first define a Lanchesterian scenario within EINSTein that can subsequently be used as a test bed to which the outcomes of other, non-Lanchesterian, scenarios can be compared. The set of simulation parameters that are appropriate for simulating a maneuverless, Lanchester-like combat scenario in EINSTein includes a red/blue movement range of $r_m = 0$ (so that the position of all agents is fixed) and a red/blue sensor range that is large enough so that all agents have all enemy agents within their view (for the example below, $r_S = 40$).

Fig. 9.3 shows several snapshots of a typical run. Initial conditions consist of 100 red and 100 blue agents (in a tightly packed block formation, with block-centers 15 units distant on a 60-by-60 battlefield) and a red/blue single-shot probability of hit $P_{\text{hit}} = 0.005$. Note that the outcome of the battle is a function of the initial sizes of red and blue forces and P_{hit} alone and does not depend on maneuver or any other agent, squad, or force characteristics.

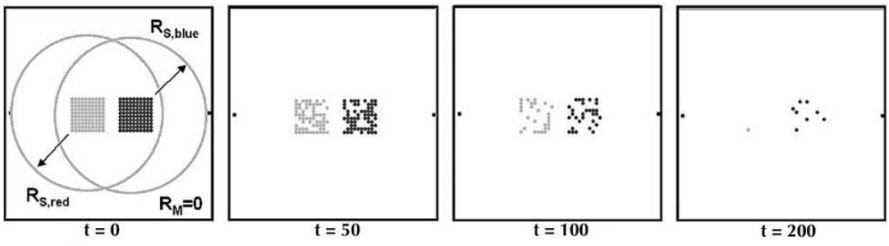

Fig. 9.3 Screenshots of a typical run using an EINSTein rule-set that approximates LE-like combat.

While the Lanchester scenario shown here is highly unrealistic, of course, it is important to remember that most conventional military models (even those that include some form of maneuvering) adjudicate combat by effectively sweeping over a series of similarly idealized maneuverless skirmishes until one side, or both sides, of the conflict decide to withdraw after sustaining a threshold number of casualties. Most models are still almost entirely attrition

driven. The only substantive role that maneuver and adaptability play is in getting the individual combatants into position to fight.

A typical signature of such Lanchesterian-like combat scenarios is a linear dependence of the *mean attrition rate* – defined as the average number of combatants lost, $\langle\alpha\rangle$, during some specified time interval, $\Delta\tau = t - t_0$ – on the *single-shot kill* (or, in our case here, *single-shot hit*) probability, P_{ss}:

$$\langle\alpha\rangle = \left\langle \frac{\Delta n}{\Delta\tau} \right\rangle = \left\langle \frac{n(t_0+t) - n(t_0)}{\Delta\tau} \right\rangle = \sum_{i=1}^{N} P_{ss}(i) = NP_{ss}, \tag{9.5}$$

where N is the total number of agents, $n(t)$ is the number of agents at time t, $P_{ss}(i)$ is the single-shot hit probability of the ith agent, and we have assumed, for the final expression on the right, that $P_{ss}(i) = P_{ss}$ for all i.

What happens if agents are allowed to maneuver? If the maneuver is in any sense "intelligent" (i.e., if agents react reasonably intelligently to changing levels of combat intensity as a battle unfolds), intuitively we should not expect the same linear dependence between $\langle\alpha\rangle$ and P_{ss} to hold. In the extreme case of infinitely timid combatants that run away at the slightest provocation, no fighting at all will occur. In the case where one side applies sophisticated targeting algorithms to maximize enemy casualties but minimize friendly casualties, we might expect a marked increase in that force's relative fighting ability.

To illustrate these ideas, consider an "explosive skirmish" scenario, which is characterized by a rapid, explosive burst of agents as they collide and maneuver close-in during a series of local firefights and skirmishes as the battle slowly dissipates (see screenshots in Fig. 9.4). Table 9.3 lists the parameters values used for these runs.

	Sample 1		Sample 2	
Parameter	Red	Blue	Red	Blue
Agents	250	250	100	100
r_S	5	5	5	5
r_F	3	3	3	3
r_M	2	2	1	1
ω_1	10	10	10	10
ω_2	40	40	40	25
ω_3	10	10	10	10
ω_4	40	40	40	25
ω_5	0	0	0	0
ω_6	25	25	5	50
$\tau_{Advance}$	3	3	3	3
$\tau_{Cluster}$	8	3	5	8
Δ_{Combat}	–99	–3	–10	–3

Table 9.3 Parameter values used for scenarios shown in Fig. 9.4

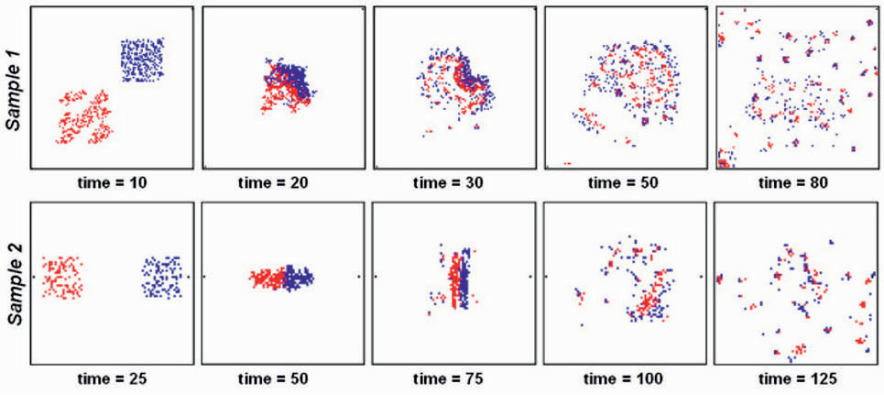

Fig. 9.4 Screenshots of typical runs of the "explosive skirmish" scenario using parameters given in Table 9.3.

9.6.2.1 Fractal Dimensions of Spatial Dispositions

If one were to plot attrition rate, $\langle\alpha\rangle$, versus single-shot probability of hit, P_{ss} – for either of the two "explosive skirmish" scenarios shown in Fig. 9.4 – one would find that $\langle\alpha\rangle \propto P_{ss}^{n}$, where $n \approx 1/2$. Moreover, careful analysis of the spatial patterns, as they emerge in multiple runs, suggests that what lies at the core of a certain class of non-Lanchesterian scenarios is a fractal scaling of combat forces.

Recall that fractal dimensions – such as the *capacity* dimension (or "box-counting" dimension) – are measures that provide useful structural (and/or, in the case of *information* dimension, statistical information) about a given point set. Fractals are geometric objects characterized by some form of self-similarity; that is, parts of a fractal, when magnified to an appropriate scale, appear similar to the whole. Fractals are thus objects that harbor an effectively infinite amount of detail on all levels. Coastlines of islands and continents and terrain features are approximate fractals. A magnified image of a part of a leaf is similar to an image of the entire leaf. Strange attractors also typically have a fractal structure. Loosely speaking, a fractal dimension specifies the minimum number of variables that are needed to specify an object. For a one-dimensional line, for example, say the x-axis, one piece of information, the x-variable, is needed to specify any position on the line. The fractal dimension of the x-axis is said to be equal to 1. Similarly, two coordinates are needed to specify a position on a two-dimensional plane, so that the fractal dimension of a plane is equal to 2. Fractals are objects whose fractal dimension is noninteger-valued.

How might fractals relate specifically to combat? Intuitively, since real combat consists of anything but a series of random skirmishes in which opposing sides constantly shoot at each other, we expect attrition data to contain spa-

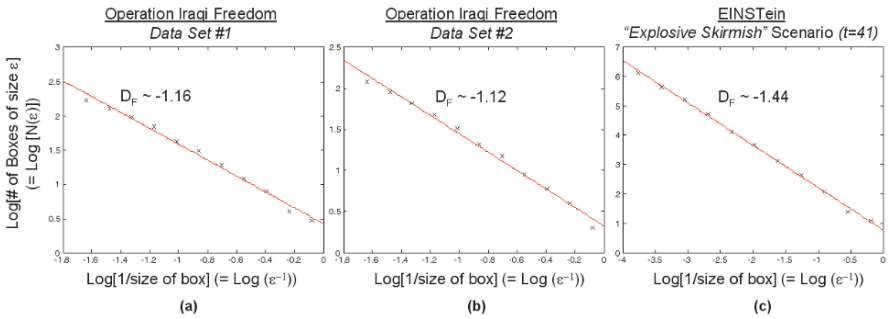

Fig. 9.5 Single-time estimates of D_F for two spatial dispositions of real-world forces and a snapshot of notional forces as arrayed in EINSTein. The (x, y) locations of the real-world data are the longitudes and latitudes of Coalition ground forces during *Operation Iraqi Freedom* in 2003. See the text for additional details.

tiotemporal correlations. Because EINSTein includes rules for maneuver, we expect to see spatiotemporal correlations emerge as a consequence of maneuver (as we would expect to also see in other multiagent-based combat simulations that contain intelligent maneuver).

Spatial distributions of agents on a battlefield are nothing more than sets of abstract points on a lattice. The degree of clustering and maneuver can therefore be measured by calculating a fractal dimension for the distribution in exactly the same way one typically calculates the fractal dimension for point sets representing various *one*- and *two*-dimensional attractors of dynamical systems. The only real difference between using the (x, y) positions of agents and points of an attractor, in practice, is that when dealing with agents, we are naturally limited in the number of "data points" with which we have to work. Even large scenarios are limited to about 500 agents or so to a side. Moreover, agents are allowed to sit only on integer-valued sites, so that the set of possible (x, y)'s is also more limited. Nonetheless, the actual calculation of fractal dimensions proceeds in exactly the same manner for the two cases; just as for continuous dynamical systems, the measures provide an important insight into the geometry of the spatial distributions.

As a concrete example, consider the *capacity* dimension, which is defined by $D_F = \lim_{\epsilon \to 0} \ln[N(\epsilon)] / \ln(1/\epsilon)$, where $N(\epsilon)$ is the number of d-dimensional boxes of side ϵ that contain at least one agent. Alternatively, we can write that $N(\varepsilon) = \varepsilon^{-D_F}$ and call D_F the power-law *scaling exponent*. For a solid block of maneuverless agents, such the solid red and blue blocks of agents dueling it out in the Lanchesterian scenario shown in Fig. 9.3, $D_F \sim 2$; D_F will be *less than* 2 if the agents occupy only a portion of the entire battlefield, as the whole battlefield is be used to estimate D_F.

Fig. 9.5 compares single-time estimates of D_F for (1) two spatial dispositions of real-world forces (Figs. 9.5a and 9.5b) and (2) a snapshot of notional forces as arrayed during the dispersive-flow phase of the explosive skirmish

scenario in EINSTein (Fig. 9.5c). The (x, y) locations of the real-world data consists of the longitudes and latitudes of Coalition ground forces during *Operation Iraqi Freedom* in 2003.[5] In each case, the (x, y) locations are first scaled so that the real and notional battlefields assume the same effective "size" of 1-by-1 sites and the scaled battlefield is then divided into a total of $N_{total} = \varepsilon^{-2}$ boxes of length ε. Several different values of ε are chosen, and for each ε, the number of boxes, $N(\varepsilon)$, that contain at least one agent are counted. Since we are limited by how finely we are able to partition the battlefield (as well as by the relatively limited number of agents that define our set of (x, y) locations: ~ 500), the fractal dimension is estimated by the slope of a linear fit on a plot of $\log[N(\varepsilon)]$ versus $\log[1/\varepsilon]$. (Note that we are not in any way suggesting that a finite distribution of either real or notional combatants represents a genuine fractal in a mathematically rigorous sense. We are only suggesting that for limited domains (in battlefield size, duration of conflict, and number of agents) their distribution is such that it can reasonably well be characterized by a fractal-like power-law scaling.)

The point of Fig. 9.5 is not to compare the absolute values of D_F for the different cases – which we could have anticipated as being different, particularly since EINSTein's explosive skirmish example does not intentionally model any real-world scenario – but rather to illustrate the important fact that EINSTein is able to reproduce a spatial fractal scaling *at all* (at least for the limited spatial ranges being considered). In EINSTein, as in the real world, "intelligent maneuvering" implies an agent's position at time t is strongly correlated with local combat conditions. While it has for a long time been known, on the basis of empirical evidence, that real-world forces tend to arrange themselves in self-organized fractal fashion (see [9] and [44]), no satisfactory generative explanation for why this is so has yet appeared.

How does the fractal dimension change during combat? To illustrate the kinds of spatial configurations that can arise in different combat scenarios, consider Fig. 9.6. It shows three plots each (using different initial configurations of agents) of D_F as a function of time for (1) the *Lanchesterian* scenario, (2) a scenario in which the agents are allowed to move but do so completely randomly (and are initially distributed randomly on the battlefield), and (3) the Sample 1 explosive skirmish scenario defined by the parameters appearing in Table 9.3.

These three cases are defined by first using the explosive skirmish scenario to select a single-shot probability of hit, P_{ss}, for which the mean attrition after 100 time steps is equal to 20%. That same value is then used for the other two cases as well. Also, in order to better place the random scenario in between the Lanchester scenario (in which all agents "see" and "fire at" all other agents at all times) and the explosive skirmish scenario (in which agents' sensor and fire ranges are relatively small), agents in the *random* sce-

[5] These scaled geo-locations are from software databases maintained, and kindly provided to the author, by Dr. Michael Shepko and Dr. David Mazel of the Center for Naval Analyses, Alexandria, Virginia.

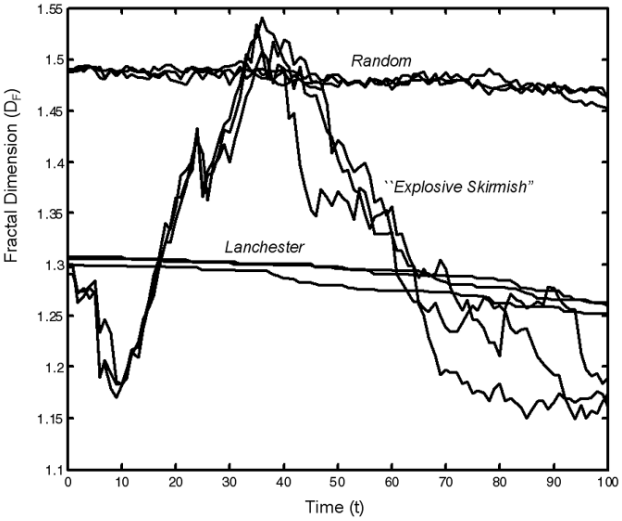

Fig. 9.6 Three sample plots, each (using different initial configurations of agents) of the fractal dimension D_F as a function of time for the *Lanchesterian* (i.e., maneuverless) scenario, a scenario in which the agents are allowed to move but do so completely *randomly*, and the Sample 1 explosive skirmish scenario defined by the parameters appearing in Table 9.3.

nario are assigned sensor and fire ranges equal to three times their value for the explosive skirmish scenario. Because these three cases represent qualitatively different combat scenarios that range from maneuverless, all-seeing/all-shooting agents to intelligently maneuvering agents able to sense, and adapt to, only local streams of information, it is instructive to use them as a basis for comparing simulation output. Except for a small drift of values among different runs for the same scenario and/or differences in precise values at a given time for a given run, the scenarios are each characterized by the unique manner in which the fractal dimension of its associated spatial distribution evolves in time; that is, the time evolution of D_F represents a kind of *behavioral signature* of a given scenario. There is also evidence to suggest a deeper coupling between D_F and combat dynamics.

9.6.2.2 Power-Law Scaling of Attrition

Lauren [40] has used EINSTein (and other ABMs of combat; see [42]) to identify some significant differences between agent-based attrition statistics and results derived from stochastic LE-based models. In particular, he has found evidence to suggest that the *intensity of battles* obeys a fractal power-law dependence on frequency and displays other traits characteristic of high-dimensional chaotic systems, such as fat-tailed probability distributions and

intermittency. Specifically, the attrition rate appears to depend on the cube root of the kill probability, which stands in marked contrast to results obtained for stochastic variants of LE-based models, in which, typically, the attrition rate scales linearly with an increase in kill probability.[6] If the ABM more accurately represents real combat processes, an $\sim 1/3$ power-law scaling implies that a relatively "weak" force, with a small kill probability, may actually constitute a much more potent force than a simple LE-based approach suggests. The potency of the force comes from its ability to maneuver (which is never explicitly modeled by LE-based approaches) and to selectively concentrate firepower on the enemy while maneuvering. This deceptively simple result has an important consequence for peacekeeping activities in the Third World, in which a strong, modern force may (and often, does) significantly underestimate the ability of ostensibly poorly trained and/or poorly armed militia to inflict damage.

The appearance of fractal power-law scaling in EINSTein (and other agent-based combat models) is particularly interesting in light of the fact that it has been observed before in real combat [44]. While it has been previously argued, on intuitive grounds, that this must be due to the dynamical coupling between local information processing and maneuver – features that are completely ignored by Lanchesterian models – no generative "explanation" for why fractal power-law scaling appears in combat has heretofore existed. It is therefore tempting to speculate that there are phases of real combat that are poised at *self-organized critical states* (see e.g. [45, 46]).

9.6.3 A Step Away from Lanchester

With an eye toward exploring non-Lanchesterian scenarios, consider an example that includes both simple maneuver and terrain. Fig. 9.7 shows the initial state, consisting of 12 red and 12 blue agents positioned near their respective "flags" (in the lower left and upper right corners, respectively). The red agents are arrayed along a berm (i.e., a permeable terrain element, which appears green in the figure), whose dynamical effect is to reduce their visibility to the approaching blue enemy agents to 15% of the nominal value. As blue agents approach the red flag, red agents remain fixed at their posi-

[6] The key observation is that the attrition rate generally depends not just on P_{ss} (as in Eq. 9.5), but on both P_{ss} and D_F, the latter measure representing the spatial distribution of agents [34, 42]. To derive Eq. 9.5, for Lanchesterian combat, one assumes that one side's attrition rate is proportional to the opposing side's size (and *nothing else*); in the general case, one must assume that the attrition rate also depends on the probability that an agent actually "sees" an enemy (or cluster of enemy agents) in a given period of time. The likelihood of this happening, in turn, may be expressed in terms of D_F. Lauren et al. [43] have recently introduced the generalized Lanchester equation $\langle \Delta B/\Delta t \rangle \propto k^{D_F/2} \cdot \Delta t^{D_F/2-1} \cdot R(t)$, where B and R are the number of red and blue agents, k is the rate at which red (or blue) kill blue (or red) agents, and t is the time.

tions (simulating a notional "hunkered-down" condition). The red and blue weapon characteristics (probability of hit and range) are equal.

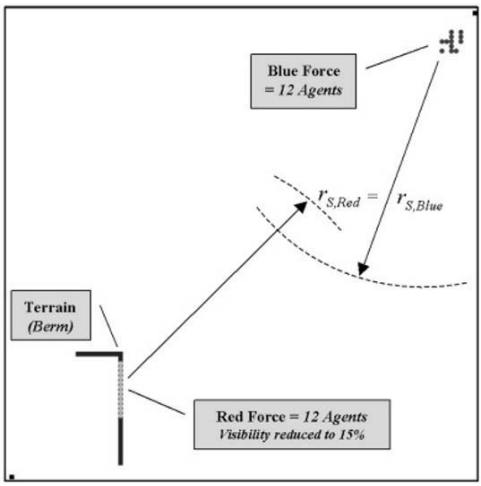

Fig. 9.7 Initial state for simple non-Lanchesterian scenario; see text for details.

Runs typically proceed as follows. Because of the stealth afforded the dug-in red agents by the berm, red agents are targeted and engaged with a much lower probability than the approaching blue force. The attrition of the attacking force (blue) is significantly higher than the attrition of the defending force (red). When the attackers are able to survive (with some of their force intact) – on some particular run of the scenario – it is because they are able to maneuver out of range (which occurs when the force strength drops below the combat effective threshold of 50% and attempts to withdraw) and red is unable to pursue. (As an aside, EINSTein's ability to prescribe retreat conditions adds a certain realism to the model. Faced with mounting attrition, real squads fall back and regroup.)

The red force usually remains at full strength after the engagement (the probability of zero red casualties is about 80%). This result is intuitively satisfying, since, historically (all other factors being equal), defending forces have the advantage over an attacking force traversing open ground. An obvious question to ask is, *"How large must the blue force be in order to overcome the advantage of the red's terrain?"* Fig. 9.8 plots the fraction of the initial forces that remain at the end of the engagement (150 steps) versus the attacker-to-defender force-size ratio (the lines are simple fits to the data to guide the eye). In the runs used to generate this graph, the size of the blue force ranges from 12 to 40 agents, while the red force remains at 12. Note that the red and blue survival curves merge at roughly a 2.8:1 ratio; which is interesting in light of the well-known "rule of thumb" that attackers require a 3:1 force ratio against a defended position [47].

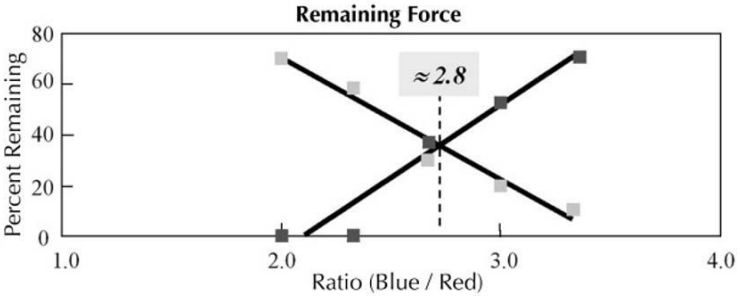

Fig. 9.8 Impact of attacker-to-defender force ratio on survival for the simple non-Lanchesterian scenario shown in Fig. 9.7. The red and blue survival curves merge at about a 2.8:1 ratio, which compares favorably to the well-known "rule of thumb" that attackers require a 3:1 force ratio against a defended position [47].

9.6.4 Swarming Forces

One of the first detailed studies of swarming, as a major theme in military history, was recently conducted by Sean Edwards, as part of the Swarming and the Future of Conflict project at RAND [48]. Edwards' report focuses on 10 carefully selected historical examples of swarming, includes a series of important lessons-learned distilled from these examples about the advantages and disadvantages of swarming, and provides some examples of successful countermeasures that have been used against swarming in the past.

Edwards noted that swarming consists of four overlapping stages: (1) *location*, (2) *convergence*, (3) *attack*, and (4) *dispersion*. Moreover, swarming forces must be capable of a sustainable pulsing; that is, networks of swarming agents must be able to come together rapidly and stealthily on a target, then redisperse, and, finally, recombine for a new pulse:

> The swarm concept is built on the principles of complexity theory, and it assumes that blue units have to operate autonomously and adaptively according to the overall mission statement ... It is important that swarm units converge and attack simultaneously. Each individual swarm unit is vulnerable on its own, but if it is united in a concerted effort with other friendly units, overall lethality can be multiplied, because the phenomenon of the swarm effect is greater than the sum of its parts. Individual units or incompletely assembled groups are vulnerable to defeat in detail against the larger enemy force with its superior fire-power and mass.

The report noted that swarming scenarios have already played a role in certain high-level war-gaming exercises, such as at the *Dominating Maneuver Game*, held at the US Army War College in 1997. Edwards concluded his survey by speculating about the feasibility of a future "swarming doctrine" that would consist of small, distributed, highly maneuverable units converging rapidly on specific targets.

Because of its decentralized rule-base and rich space of behavioral primitives, EINSTein is an ideal test bed with which to explore the nature of

battlefield swarming and the efficacy of swarm-like tactics. Typically, but not always, one side appears to swarm the other when there is a significant mismatch in firepower, total force strength, and/or maneuvering ability. (Swarming also occasionally emerges as a useful "tactic" to use against certain opponents when EINSTein's built-in genetic algorithm is tasked with finding optimal attack strategies.) While it is common to find swarm-like behavior for personalities that include large cluster meta-rule thresholds, τ_{Cluster} (which increases the likelihood that agents will remain in close proximity to friendly agents), the most interesting "self-organized" examples of swarming are those for which τ_{Cluster} is, at most, a few agents.

Table 9.4 lists some of the parameter values defining four representative swarm scenarios (I–IV). In scenario I, blue attacks red; in scenario II, blue defends. Blue agents are more aggressive than red in all four scenarios (as defined by the values of their respective combat meta-rule thresholds, Δ_{Combat}). Note that in scenarios II and III defending blue agents are able to communicate with other blue agents that are within a range $r_C = 25$ of their position. Fig. 9.9 show snapshots of typical runs using parameters for scenarios I–IV.

	I	I	II	II	III	III	IV	IV
Force	Red	Blue	Red	Blue	Red	Blue	Red	Blue
Size	150	225	90	125	25	100	200	200
r_S	5	5	5	10	3	7	3	7
r_F	3	3	3	7	2	5	2	5
r_M	1	1	1	2	1	1	1	1
w_{AF}	25	10	10	0	5	0	5	0
w_{AE}	25	50	40	99	40	5	40	5
w_{IF}	75	0	10	0	5	0	5	0
w_{IE}	25	99	40	99	90	50	90	50
w_{FF}	0	0	0	0	0	0	0	0
w_{EF}	75	25	50	0	0	0	0	0
τ_{Advance}	5	1	3	N/A	N/A	N/A	N/A	N/A
τ_{Cluster}	15	3	3	12	5	5	5	5
Δ_{Combat}	5	−7	0	−15	−5	−10	−5	−10
Comms	no	no	no	yes, $r_C = 25$	no	yes, $r_C = 25$	no	no

Table 9.4 Agent parameter values for scenarios I–IV shown in Fig. 9.9

9.6.5 Nonmonotonicity

For a fixed set of force characteristics, number, type, and lethality of weapon systems, and tactics, one might intuitively expect that as one side's capability is unilaterally enhanced – say, by increasing sensor range or its ability to maneuver – the other side's ability to perform its mission ought to be com-

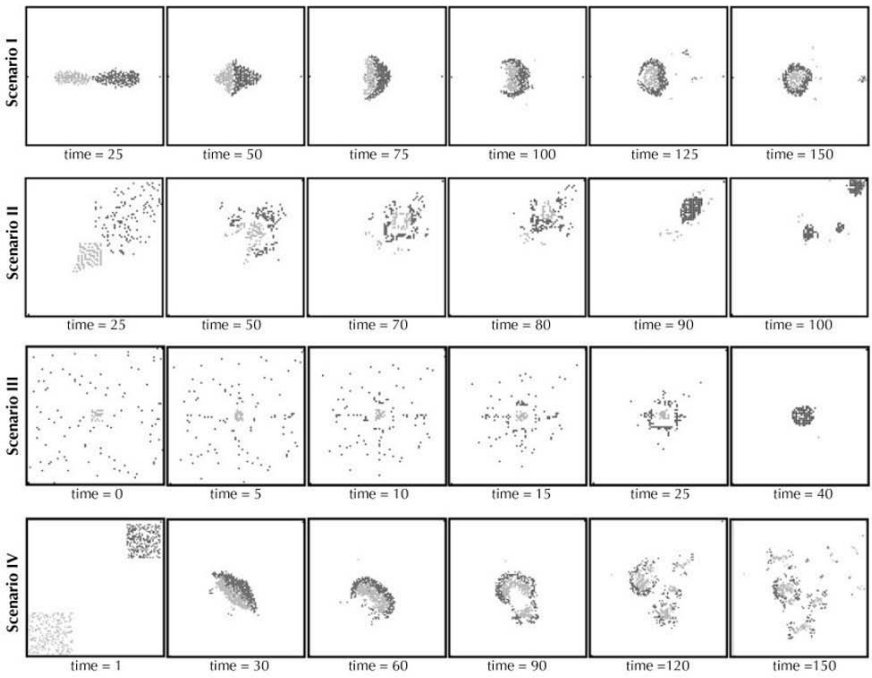

Fig. 9.9 Sample runs of swarm scenarios I–IV. See Table 9.4 for parameter values.

mensurately diminished. In other words, our expectations are that mission success scales monotonically with force capability.

In fact, nonmonotonicities abound in both real-world behavior and simulations. With respect to models and simulations, of course, one must always be on guard against the possibility that nonmonotonic scaling is an artifact of the code and therefore does not represent real processes. As pointed out by a RAND study that addressed this issue [49], "a combat model with a single decision based on the state of the battle ... can produce nonmonotonic behavior in the outcomes of the model and chaotic behavior in its underlying dynamics."

Fig. 9.10 shows an instructive example of genuinely nonmonotonic behavior; genuine in the sense that the nonmonotonicity emerges directly out of the primitive rule set. The three rows in Fig. 9.10 contain snapshots of three separate runs in which red's sensor range is systematically increased in increments of 2: $r_{S,\mathrm{red}} = 5$ for the top sequence; $r_{S,\mathrm{red}} = 7$ for the middle sequence; $r_{S,\mathrm{red}} = 9$ for the bottom sequence. Blue's sensor range, $r_{S,\mathrm{blue}}$, remains fixed at $r_{S,\mathrm{blue}} = 5$ throughout all three runs. The values of other pertinent red and blue agent parameters are given in Table 9.5.

In each of the runs, there are 100 red and 50 blue agents. Red is also the more the aggressive force. Blue engages red in combat if the number of

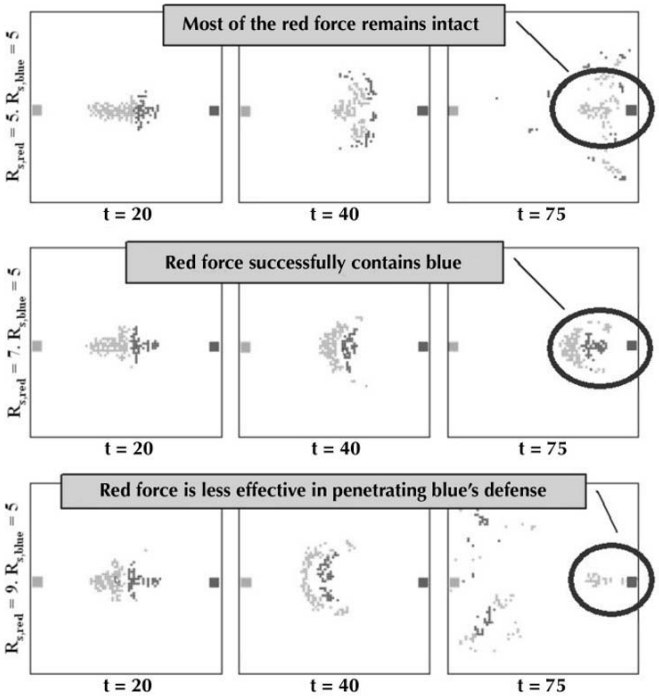

Fig. 9.10 An example of nonmonotonic behavior. The three rows contain snapshots of three separate runs in which red's sensor range is increased in increments of 2 (from $r_{S,\text{red}} = 5$ on the top row to $r_{S,\text{red}} = 9$ on the bottom). Blue's sensor range is fixed at $r_{S,\text{blue}} = 5$ throughout. Comparing the bottom row to the top two rows, we see that increasing red's sensor appears to have a detrimental effect on red's overall ability to penetrate blue's defense.

friendly and enemy agents is locally about even, while red will fight blue even if outnumbered by four enemy combatants. Both sides have the same fire range ($r_F = 4$), and the same single-shot probability ($P_{\text{hit}} = 0.005$) and can simultaneously engage the same maximum of three enemy targets. (Note that the flags for this run are near the middle of the left and right edges of the notional battlefield rather than at the corners.)

The top row of Fig. 9.10 shows screenshots of a run in which red's sensor range is equal to blue's. Here the red force easily penetrates the blue defense as it moves toward the blue flag. During red's advance, a number of agents

	N	r_S	r_F	r_M	$\mathbf{w} = (w_{\text{AF}}, w_{\text{AE}}, w_{\text{IF}}, w_{\text{IE}}, w_{\text{FF}}, w_{\text{EF}})$	τ_{Adv}	τ_{Cluster}	Δ_{Combat}
Red	100	5, 7, 9	4	1	$\mathbf{w}_{\text{Red}} = (10, 90, 10, 50, 0, 99)$	2	4	-4
Blue	50	5	4	1	$\mathbf{w}_{\text{Blue}} = (10, 90, 10, 50, 0, 99)$	2	4	0

Table 9.5 Agent parameter values for *nonmonotonic* run appearing in Fig. 9.10

are "stripped" away from the main red-blue cluster in the center as they respond to the presence of nearby blue agents. The snapshots in the middle row of Fig. 9.10 show that when red's sensor range is two units greater than blue's, red is not only able to mass almost its entire force on the blue flag (by $t = 90$ – not shown – blue's flag is completely enveloped by red forces), but to also defend its own flag from all blue forces as well. In this instance, the red force knows enough about and can respond quickly enough to enemy action such that it is able to march into enemy territory effectively unhindered by enemy forces and "scoop up" blue agents as they are encountered.

What happens as red's sensor range is increased still further? One might intuitively guess that red can only do at least as well, certainly no worse; that is, red's mission performance scales monotonically with the amount of information that each red agent is allowed to have about the engagement. However, as the snapshots for bottom row of Fig. 9.10 reveal, when red's sensor range is increased to $r_{S,\mathrm{red}} = 9$ – *so that all red agents are locally aware of more information* – red, as a force, turns in an objectively weaker mission performance than on the preceding runs. "Weaker" here meaning that red is less effective in (1) establishing a presence near the blue flag and (2) defending blue's advance toward the red flag.

The nonmonotonic behavior is immediately obvious from Fig. 9.11, which shows a 3D fitness landscape for mission objective = *maximize number of red agents near blue flag* (where "near" is defined as anywhere within 10 battlefield units). The landscape sweeps over $r_{S,\mathrm{red}}$ $(= 1, 2, \ldots, 16)$ and the red combat meta-rule threshold Δ_{Combat} $(= -15, -14, \ldots, +15)$. Higher-valued fitness values translate to mean better performance.

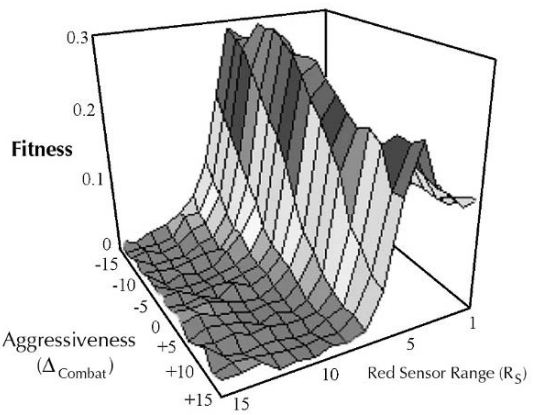

Fig. 9.11 Fitness landscape for mission = *maximize number of red agents near blue flag*, as a function of combat aggressiveness (Δ_{Combat}) and red sensor range ($r_{S,\mathrm{red}}$). Higher-valued fitness values translate to mean better performance. Note that (this particular fitness measure) does not scale monotonically with sensor range.

This example illustrates that when the resources and personalities of both sides remain fixed in a conflict, how well side X does over side Y does not necessarily scale monotonically with X's sensor capability. As one side is forced to assimilate more and more information (with increasing sensor range), there will inevitably come a point when the available resources will be spread too thin and the overall fighting ability will therefore be curtailed. Agent-based models such as EINSTein are well suited for providing insights into more operationally significant questions such as, *"How must X's resources and/or tactics (i.e., personality) be altered in order to ensure at least the same level of mission performance?"*

9.7 Genetic Algorithm Breeding

One of EINSTein's most powerful built-in features is a genetic algorithm "breeder" run-mode. Genetic algorithms (GAs) are a class of heuristic search methods and computational models of adaptation and evolution based on natural selection. In nature, the search for beneficial adaptations to a continually changing environment (i.e., evolution) is fostered by the cumulative evolutionary knowledge that each species possesses of its forebears. This knowledge, which is encoded in the chromosomes of each member of a species, is passed on from one generation to the next by a mating process in which the chromosomes of "parents" produce "offspring" chromosomes. GAs mimic and exploit the genetic dynamics underlying natural evolution to search for optimal solutions of general combinatorial optimization problems. They have been applied to the traveling salesman problem, VLSI circuit layout, gas pipeline control, the parametric design of aircraft, neural net architecture, models of international security, and strategy formulation [50].

Fig. 9.12 illustrates how GAs are used in EINSTein. Chromosomes define individual agents. Genes encode the components of the personality weight vector, sensor range, fire range, meta-rule thresholds, and so forth. The initial GA population consists of a set of randomly generated chromosomes. The fitness function represents a user-specified mission "fitness" (see later). The target of the GA search is, by default, the red force. The parameter values defining the blue force – once they are defined at the start of a search – are held fixed.

9.7.1 Search Space

EINSTein uses up to 80 genes to conduct a GA search; the actual number depends on the particular region of the parameter space the user wishes to explore. Some genes are integer-valued (such as the agent-to-agent commu-

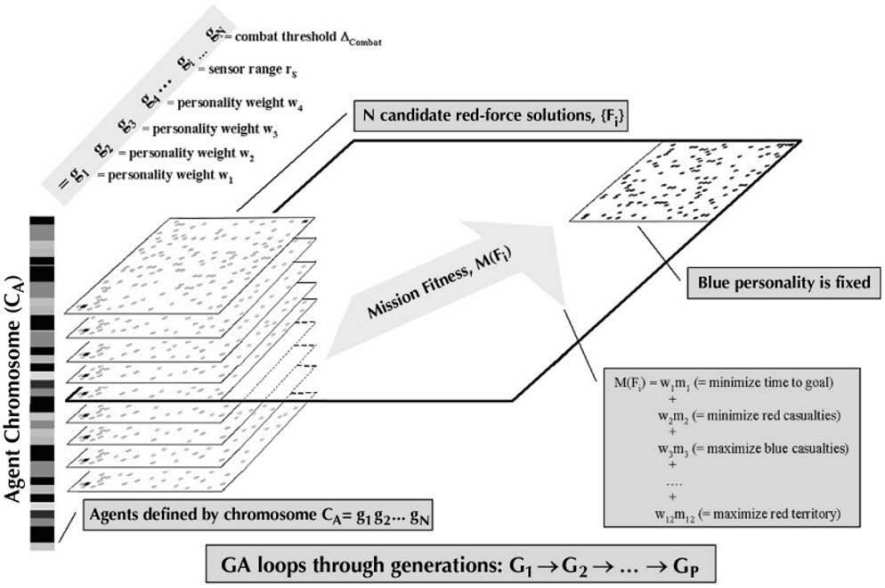

Fig. 9.12 Schematic of EINSTein's GA. The blue force and mission fitness are both fixed by the user. The GA encodes components of the agents' personality weight vector, sensor range, fire range, meta-rule thresholds, and so forth and breeds the "best" red force using populations of N red force "candidate" solutions; see the text for details.

nication links), while others are real-valued. All appropriate translations to integer values and/or binary toggles (*on/off*) are performed automatically by the program. Typically, each gene encodes the value of a basic parameter defining the red force. For example, g_1 encodes red's sensor range when an agent is in the alive state, g_3 encodes red's alive-state fire range, and so on. Some special genes encode the sign (+ or −) of an associated parametric gene. Thus, the actual value of each of the components of red's personality weight vector, for example, is actually encoded by *two genes*: one gene specifying the component's absolute value, and the other gene specifying its sign.

EINSTein's GA can conduct its search over five spaces:

- *Single-squad personality:* GA searches over the personality-space defining a single squad.
- *Multiple-squad personality:* GA searches over the personality-space defining multiple squads. The number of squads and the size of each squad remain fixed throughout this GA run mode.
- *Squad composition:* GA searches over squad composition space. The personality parameters defining squads 1 through 10 are fixed according to the values defined in the default input data file used to start the interactive run. The GA searches over the space defined by the number of squads

(1–10) and size of each squad (constrained by the total number of agents as defined by the data file).
- *Intersquad communications connectivity:* GA searches over the zero-one entries defining the communications matrix. The number of squads and the number of agents per squad are kept fixed at the values defined in the default input data file used to start the interactive run.
- *Intersquad weight connectivity:* GA searches over (real-valued) entries defining the squad interconnectivity matrix. The number of squads and the number of agents per squad are kept fixed at the values defined in the default input data file.

9.7.2 Mission Fitness

The mission fitness (MF) is a measure of how well agents perform a user-defined mission. Typical missions are *"Get to blue flag as quickly as possible," "Minimize red casualties,"* and *"Maximize the ratio of blue to red casualties,"* or some combination of these. MFs are always defined from red's perspective. The user assigns weights ($0 \leq w_i \leq 1$)[7] to represent the relative degree of importance of each mission-fitness primitive, m_i (see Table 9.6). While the mission primitives are relatively few in number and simple, they can be combined to define more complicated multiobjective functions.

Weight	Primitive	Description
w_1	m_1	Minimize time to goal
w_2	m_2	Minimize friendly casualties
w_3	m_3	Maximize enemy casualties
w_4	m_4	Maximize friendly-to-enemy survival ratio
w_5	m_5	Minimize friendly center-of-mass distance to enemy flag
w_6	m_6	Maximize enemy center-of-mass distance to friendly flag
w_7	m_7	Maximize N_{friends} within distance D of enemy flag
w_8	m_8	Minimize N_{enemy} within distance D of friendly flag
w_9	m_9	Minimize number of friendly fratricide hits
w_{10}	m_{10}	Maximize number of enemy fratricide hits
w_{11}	m_{11}	Maximize friendly territorial possession
w_{12}	m_{12}	Minimize enemy territorial possession

Table 9.6 EINSTein's GA mission-fitness primitives

The mission-fitness function, M, used by the GA, is a weighted sum of mission primitives: $M = \sum_i m_i$. (It is left up to the user to ensure that mission objectives are both logically consistent and amenable to a "solution.")

[7] *Mission fitness* weights must not be confused with the *personality* weights; agent personalities discussed earlier.

Future versions of EINSTein will include a richer set of mission-fitness primitives, including: locate and kill enemy squad leaders, stay close to friends, stay away from enemies, have combat efficiency (as measured by cumulative number of hits on enemy), clear specified area of enemy agents, occupy area for specified period of time, take the enemy flag under specific conditions (e.g., the user is asked to specify the number of agents that must occupy a given area around the enemy flag for a given length of time), among others.

9.7.3 EINSTein's GA Recipe

The GA uses EINSTein's agent-movement/combat engine to conduct its searches. In pseudocode, the main components of EINSTein GA recipe are as follows:

```
for generation=1,G_max
    for personality=1,P_max
        decode chromosome
        for initial_condition IC=1 to IC_max
            run combat engine
            calculate fitness (for given IC)
        next initial_condition
        calculate mission fitness
    next personality
    find the best personality
    select survivors from population
    perform (single-point) crossover operation
    perform mutation operation
    update progress/status
next generation
write best personality to file
```

In words, the GA uses a randomized pool of chromosomes to define an initial generation of red personalities. For each red personality, and for each of the $IC_{\max}$ initial spatial configurations of red and blue forces, the program then runs EINSTein's combat engine to determine the mission fitness. After looping through all personalities and initial conditions, the GA first sorts and ranks the personalities according to their mission-fitness values, then selects some to be eliminated from the pool and others to breed. The GA then performs the basic operations of crossover and mutation. Finally, after defining a new generation of red personalities, the entire process is repeated until either the user interrupts the evolution or the maximum generation number has been reached (see Fig. 9.12).

9.7.4 Sample GA Breeding Experiment #1

Consider the following mission (as stated from the *red* force's point of view): *"Keep blue agents as far away from the red flag as possible, for as long as possible (up to a maximum 100 iteration steps)"*; that is, set all GA mission weights to zero, except for $w_6 = w_8 = 1/2$; see Table 9.6. This means that the mission fitness M will be close to its maximal value *one* only if red is able to keep all blue agents pinned near their own flag (at a point farthest from the red flag) for the entire duration of the run, and M will be near its minimal value *zero* if red allows blue agents to advance completely unhindered toward the red flag. Combat unfolds on a 40-by-40 battlefield, with 35 agents per side. The GA is run using a pool of 50 red personalities for 50 generations, and each personality is averaged over 25 initial spatial configurations. Blue agents are each assigned (a fixed) personality weight vector $w_{\text{Blue}} = (w_{\text{AF}}, w_{\text{IF}}, w_{\text{AE}}, w_{\text{IE}}, w_{\text{FF}}, w_{\text{EF}}) = (0, 10, 0, 10, 0, 90)$.

Fig. 9.13 shows a typical *learning curve*, where "Best" refers to the fitness of the highest-ranking candidate solution and "Average" refers to the average fitness among all candidate solutions per generation. The GA run described here (using a 1-GHz Pentium IV PC) each requires roughly an hour to complete.

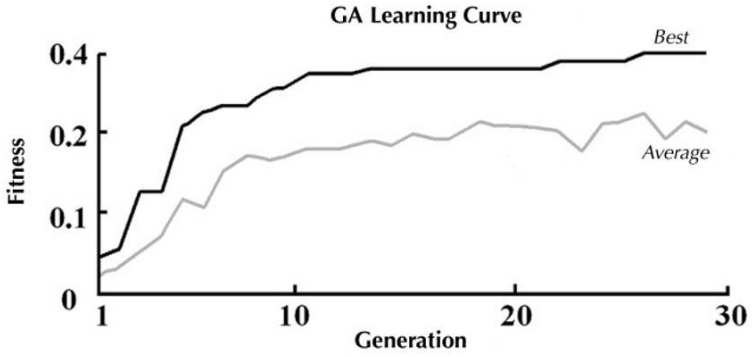

Fig. 9.13 Typical GA learning curve for GA breeding experiment discussed in the text.

Screenshots from a typical run using the highest-ranked red personality (as sampled from "solution" pool representing generation 30) that the GA is able to find for this mission are shown along the top row of Fig. 9.14. They show that red is very successful at keeping blue forces away from its own flag; the closest that red permits blue agents to approach the red flag – during the entire allotted run time of 100 iteration steps – is some point roughly near midfield. In words, the "tactic" here seems to be – from red's

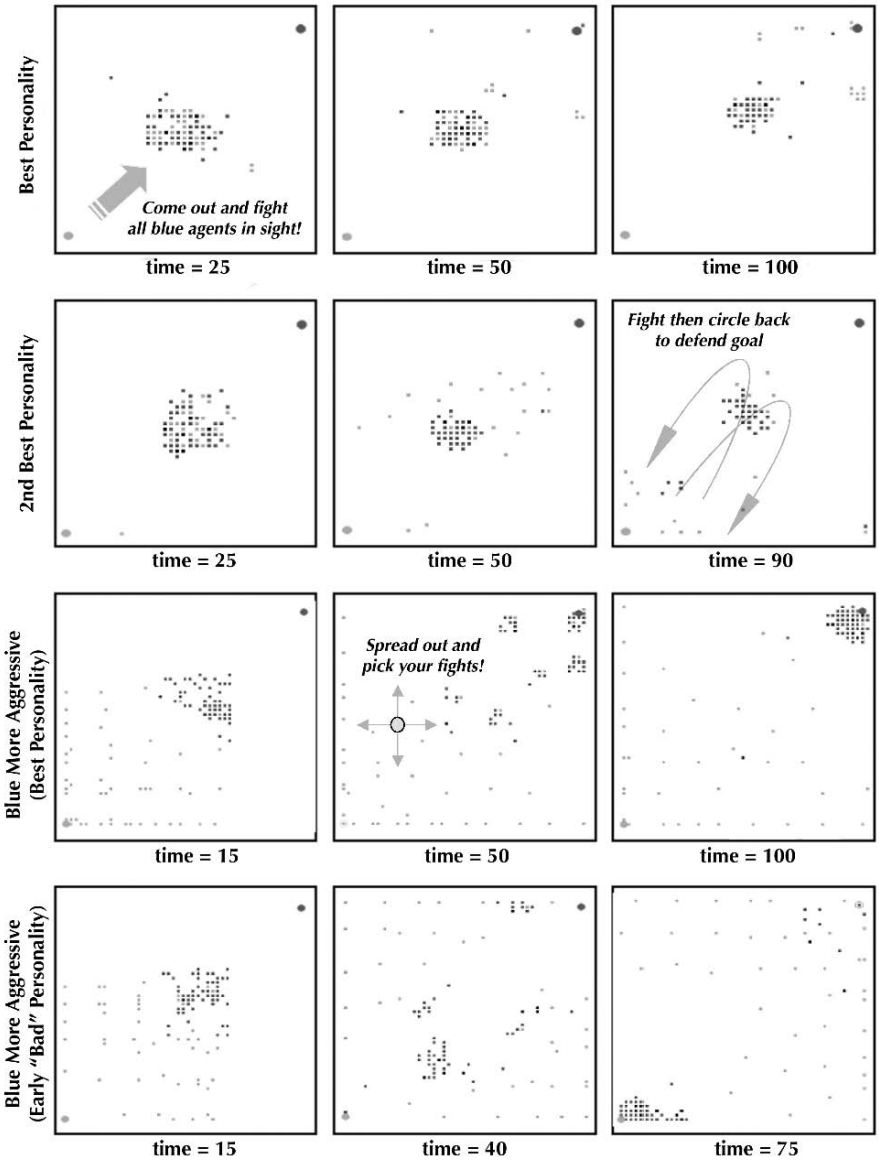

Fig. 9.14 Screenshots from several sample runs of GA breeding experiment #1. The top row shows a run using the highest-ranked red agents after 50 generations. The second row shows the second highest-ranked red force. The third row shows how the red force adapts to a more aggressive blue force. Finally, the fourth row shows an example of how a suboptimal red force performs representing a pool of agents occupying an early portion (generation 10) of the GA's learning curve.

perspective – *"fight all enemy agents within sensor range, and move toward the enemy flag slowly enough to drive the enemy along."* Note that this emergent tactic is also fairly robust, in the sense that if the battle is initialized with a different spatial disposition of red and blue forces (while keeping all personality parameters fixed), red performs this particular mission about as well, on average, as evidenced by these screenshots.

Screenshots from a typical run using the second highest-ranking red personality are shown along the second row of Fig. 9.14. These show a slightly less successful, but nonetheless innovative, alternative tactic. Initially, red agents move away from their own goal to meet the advancing blue forces, just as in the first case (at $t = 25$). Once combat ensues, however, any red agents that find themselves locally isolated now "double back" toward their own flag (positioned in the lower left corner of the battlefield) to regroup with other remaining friendly agents. The red force thus, effectively, forms an impromptu secondary defense against possible blue leakers. Because a few blue agents do manage to fight their way near the red flag at later times (at least in the particular run these screenshots have been taken from; see snapshot for $t = 90$), the red agent parameter values underlying this emergent tactic are not as highly ranked as the parameter values underlying the run shown in the top row.

The series of screenshots appearing in the third row of Fig. 9.14 show the emergent tactic used by the highest-ranked red personality found by the GA after the blue force is made more *aggressive*. For this case, prior to initializing the GA search, blue's personality weight-vector components for moving toward red (i.e., $w_{AE} = w_3$, and $w_{IE} = w_4$) are first increased by 50%. We see that EINSTein's GA discovers an entirely different (and more effective) tactic to use. Here, the red force quickly spreads out to cover as much territory as possible and individual agents attack the enemy as soon as they come within view. As red agents' local territorial coverage is thinned – either through attrition or gradual advance toward the blue flag – other red agents (namely agents that had previously been positioned near the periphery of the battlefield) move closer to the center of the battlefield, thus filling emerging voids. This tactic succeeds in preventing any blue agents from reaching the red flag and also manages to push most of the surviving blue force back toward its own flag (near the top right corner of the battlefield)! As is true of the other cases in this experiment, this tactic is also fairly robust and is not a strong function of the initial spatial disposition of red and blue forces.

The last row of plots in Fig. 9.14 contains snapshots from a run using interim red agent parameter values, *before* the GA has had a chance to evaluate a large number of candidate solutions. This example illustrates how an obviously *sub*-optimal pool of agents behaves differently from their optimized counterparts. The mission parameters and blue force agent personalities are the same as in the case represented by the screenshots in the third row. We see that, initially at least, there does not seem to be much difference in the optimal and sub-optimal behaviors; red agents quickly disperse outward to

cover a large area. However, because the GA has not yet had the time to fine-tune all of red's genes, the red force is, in this instance, unable to prevent blue agents from penetrating deeply into its territory. The defensive tactic, however it may be characterized, is obviously ineffective.

9.7.5 Sample GA Breeding Experiment #2

Consider a scenario in which the blue force is tasked with defending its flag against a smaller attacking red force. We use the GA to find a red force that is able to penetrate the blue defense. Table 9.7 lists some pertinent parameter values defining the two forces. The third column of the table (i.e., red trial values) lists baseline red force parameter values (as defined by us, not the GA) used to test the scenario. The fourth and fifth columns (i.e., GA-bred values) list the GA-bred red force "solution." Notice that, in both cases, the number of agents is the same and is fixed (with blue outnumbering red, 100 to 50 in all runs). All baseline red-*trial* alive and injured parameter values are equal.

	Blue	Red *Trial*	Red *GA Bred*	
	Agents	Agents	*Alive* Agents	*Injured* Agents
N_{Agents}	100	50	50	50
r_S	5	5	8	5
r_F	3	3	8	5
r_M	2	2	2	2
w_{AF}	0	10	3	−22
w_{AE}	100	40	40	95
w_{IF}	0	10	46	−86
w_{IE}	100	40	38	−14
w_{FF}	0	0	−70	14
w_{EF}	0	25	65	31
τ_{Advance}	N/A	3	3	1
τ_{Cluster}	5	10	13	17
Δ_{Combat}	−20	0	−19	+20

Table 9.7 Agent parameter values for GA sample run appearing in Figs. 9.15 and 9.16

Fig. 9.15 shows screenshots from a typical run using the red-trial values. Red agents attack, unsuccessfully, in a tight cluster. The larger blue force (whose agents initially move about randomly around their starting position until a red agent comes within their sensor range) dispels the red force rather easily (within the 30 time steps shown here).

The GA-bred parameters listed along the bottom row in Table 9.7 define the highest-ranked red force that EINSTein's GA is able to find (after 30 generations) with respect to performing the mission = *"maximize the number of red agents able to penetrate within a distance* $d = 7$ *units of the blue flag*

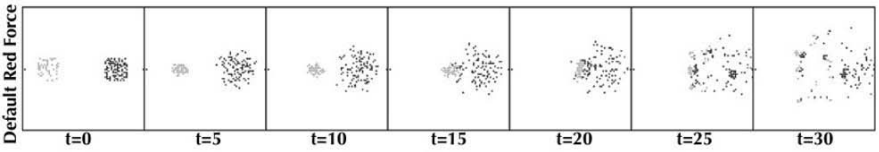

Fig. 9.15 Trial red attacking force (consisting of typical parameter values that are not explicitly tuned for performing any specific mission). Red performance is used simply as a reference for interpreting the output of the sample GA breeding experiment discussed in the text.

within 40 time steps." A population size of 75 was used (i.e., each generation of the GA search consists of 75 red force candidate "solutions") and mission fitness, for a given candidate solution, is averaged over 10 initial configurations of red and blue forces. The fitness equals *one* if a candidate solution performs the specified mission in the best possible manner (i.e., if the red force sustains zero casualties and all agents remain within $d = 7$ of the blue flag starting from the minimal possible time at which they move to within that distance of the flag, for all 10 initial states) and equals *zero* if a candidate solution fails to place a single red agent within $d = 7$ of the blue flag for all 10 initial states (within the mission time limit). Fig. 9.16 shows screenshots from a typical run using the GA-bred red force values. (The arrows are included as visual aids and simply trace the motion of the red agent clusters.) Comparing this sequence of steps to those in the trial run shown in Fig. 9.15, it is obvious that the respective "attack strategies" in the two cases are very different. Indeed, the GA has found just the right mix of agent-agent proximity weights and meta-rules to define a red force that effectively exploits a relative weakness in the randomly maneuvering blue defenders. The emergent "tactic" is to *separate into two roughly equal-sized units, regroup beyond enemy sensor range, and then simultaneously strike, as a pincer, into the heart of the defending enemy cluster.*

Apart from the anecdotal evidence supplied by screenshots of this particular run, the efficacy of this simple GA-bred tactic is illustrated by comparing graphs of the number of agents near the blue flag (averaged over 50 runs)

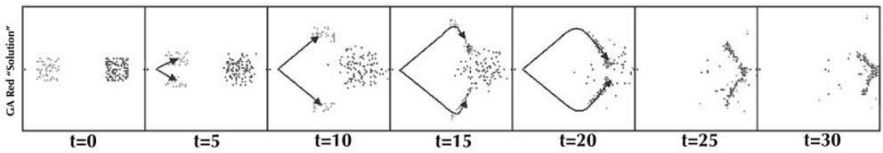

Fig. 9.16 Screenshots from a typical run using the GA-bred red force for the sample GA breeding experiment discussed in the text. Red agents are defined by GA-bred parameter values that are the highest ranked (after 30 generations) with respect to performing the mission = *"maximize the number of red agents able to penetrate within a distance d=7 units of the blue flag within 40 time steps."*

as a function of time for the red-*trial* and *GA-bred* cases. Fig. 9.17 shows that whereas fewer than three red-trial agents, on average, penetrate close to the blue flag (Fig. 9.17a), almost 80% of the entire GA-bred red force is able to do so (Fig. 9.17b) and begins penetrating at an earlier time. Other (well-performing) tactics are possible, of course. A representative sampling is generally provided by looking at the behaviors of some of the higher-ranking red forces remaining at the end of a GA search. It is interesting to run a series of GA runs to systematically probe how red forces "adapt" to different blue personalities. What one observes, typically, is that as the behavior of the blue agents changes, various – often dramatically different – GA-bred red personalities emerge to exploit any new weaknesses in the blue's defensive posture.

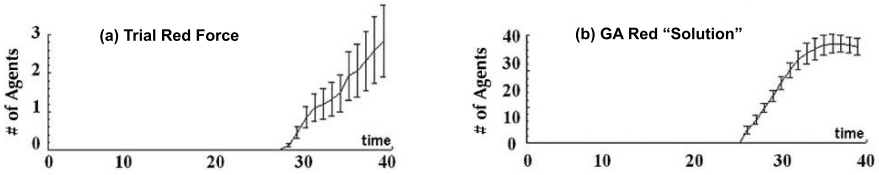

Fig. 9.17 A comparison between the average number of red agents that approach within a distance $d = 7$ of the blue flag for (a) *trial* and (b) *GA-bred* red forces. We see that the GA-bred red force typically performs this mission an order of magnitude more successfully than the trial force.

9.8 Discussion

"The musical notes are only five in number,
but their melodies are so numerous that one cannot hear them all.
The primary colors are only five in number,
but their combinations are so infinite that one cannot visualize them all.
In battle there are only the normal and extraordinary forces,
but their combinations are limitless; none can comprehend them all."

— Sun Tzu, *The Art of War*

The high-level, or poetic, description of EINSTein owes much to the suggestive metaphors appearing in the quote from *The Art of War*. In the same way as, for Sun Tzu, rainbows and melodies are all natural outcomes of combining primary colors and musical notes, EINSTein may be viewed as an "engine" that converts a primitive grammar (i.e., a grammar composed of the basic notes and colors of combat) into the limitless patterns and possibilities of war. The researcher chooses and/or tunes primitive, low-level agents

and rules; EINSTein provides the dynamic arena within which these rules interact and spawn high-level patterns and behaviors. On a more practical level, EINSTein was developed with these three important goals in mind:

1. To demonstrate the efficacy of agent-based simulation alternatives to more traditional Lanchester-equation-based models of combat [5].
2. To be used as a general prototype artificial life model/toolkit that can be used as a testbed for exploring self-organized emergent behavior in complex adaptive systems.
3. To provide the military operations research community with an easy-to-use, intuitive agent-based combat-simulation laboratory that – by respecting both the principles of real-world combat and the dynamics of complex adaptive systems – may lead researchers one step closer to a fundamental theory of combat.

To better appreciate how each of these motivations has contributed to EINSTein's (still evolving) architecture, consider the conceptual map of its design, as illustrated schematically in Fig. 9.18. Self-organized patterns emerge out of a set of primitive local rules of combat, both on the individual agent level – via interactions among internal motivations (on the Phenotype-I level, which appears as the middle level in Fig. 9.18) – and squad and force levels (labeled Phenotype-II in Fig. 9.18, and which appears as the topmost level in the figure) – via mutual interactions among many agents in a changing environment.

Of course, a deeper understanding of phenomena governing behaviors on the topmost level can only be achieved by developing a suite of appropriate pattern recognition tools (the need for which is indicated symbolically at the top of Fig. 9.18). Although a number of interesting, and highly suggestive, high-level patterns have already been discovered, much still remains to be done. Consider, for example, the frequent appearance of various power-law scalings and fractal dimensions describing space-time patterns and attrition rates ([34, 40]). The existence of power-law scalings, in particular, strongly suggests that a self-organized, critical-like dynamical mechanism might govern turbulent-like phases of combat. However, the data collection and analysis necessary to rigorously establish the nature of these findings (as well as to establish a mathematically precise set of conditions under which power-law scalings either *do* or *do not* occur) has only just started.

One of the directions in which EINSTein's design is moving (some details of which are described in the next section) is toward a fully developed ontological architecture that assigns specific meaning to the symbolic relationship between *environment* and *action*. The hope is to be able to explore the complementary problem of *reverse behavior engineering*; that is, the problem of finding an appropriate set of primitives (properties and rules) that lead either to empirically observed or desired macroscopic patterns of combat (or, in Fig. 9.18, of finding ways of going from either phenotype level I or II to the genotype level).

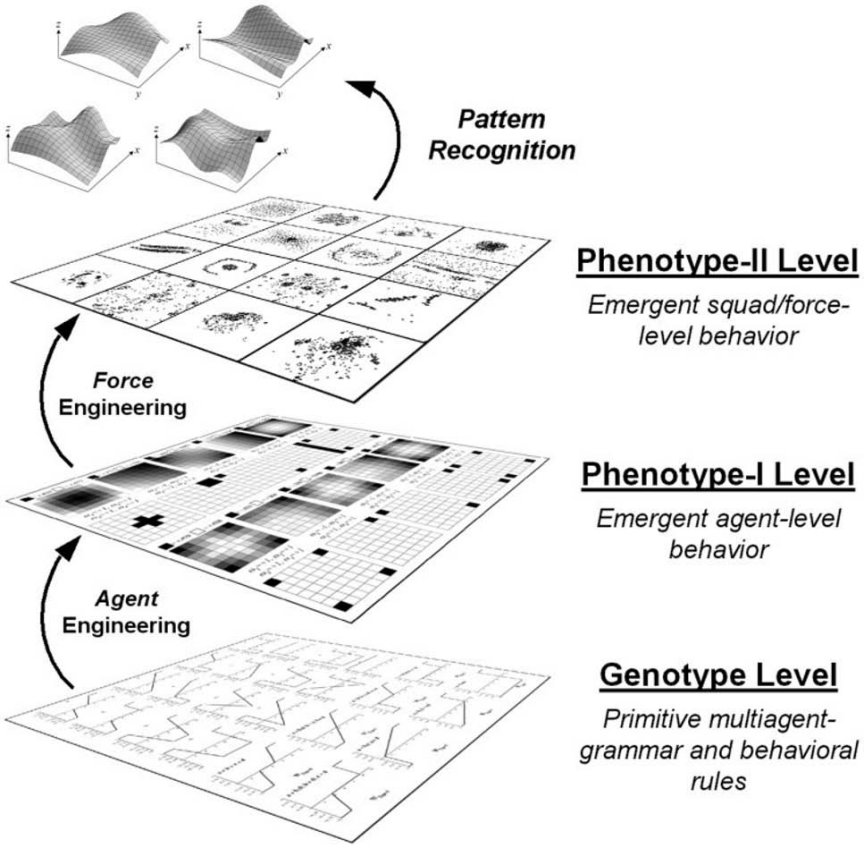

Fig. 9.18 A hierarchy of conceptual levels that illustrate EINSTein's core design.

9.8.1 Why Are Agent-Based Models of Combat Useful?

The most important immediate payoff to using EINSTein is the radically new way at looking at fundamental issues. However, agent-based models are best used to enhance understanding, not as prediction engines. Specifically, EINSTein is being designed to help researchers do the following:

- Understand how all of the different elements of combat fit together in an overall combat phase space: *"Are there regions that are 'sensitive' to small perturbations, and, if so, might there be a way to exploit this in combat (as in selectively driving an opponent into more sensitive regions of phase space)?"*
- Assess the value of information: *"How can I exploit what I know the enemy does not know about me?"*

- Explore trade-offs between centralized and decentralized command-and-control (C2) structures: *"Are some C2 topologies more conducive to information flow and attainment of mission objectives than others?" "What do emergent forms of a self-organized C2 topology look like?"*
- Provide a natural arena in which to explore consequences of various qualitative characteristics of combat (unit cohesion, morale, leadership, etc.).
- Explore emergent properties and/or other "novel" behaviors arising from low-level rules (even combat doctrine if it is well encoded): *"Are there universal patterns of combat behavior?"*
- Provide clues about how near-real-time tactical decision aids may eventually be developed using evolutionary programming techniques.
- Address questions such as *"How do two sides of a conflict coevolve with one another?"* and *"Can one side exploit what it knows of this coevolutionary process to compel the other side to remain 'out of equilibrium?' (or be otherwise trapped in a supoptimal dynamical combat state)."*

EINSTein has been used to explore the following: patrol dynamics, security, and ambush tactics [52]; reconnaissance [53]; counter reconnaissance [54]; communications [55]; distributed operations in open [56] and urban environments [57]; the historical evolution of squad and fire-team composition and weapon-mix [47]; small unit combat [59]; C2 [60]; situational awareness [58]; civil disobedience [61]; peacekeeping operations [62]; and maritime ship stationing [63]. In all, there are some 800 registered users of EINSTein, including researchers from the US Department of Defense, academia, research and development centers and private companies.

9.8.1.1 Command and Control

EINSTein contains embedded code that hardwires in a specific set of C2 functions (i.e., both contain a hierarchy of local and global commanders), so that it can be used to explore the dynamics of a given C2 structure. However, a more compelling question is, *"What is the best C2 topology for dealing with a specific threat, or set of threats?"* One can imagine using a genetic algorithm, or some other heuristic tool to aid in exploring potentially very large fitness landscapes, to search for alternative C2 structures. What forms should local and global command take, and what is the optimal communications matrix among individual combatants, squads, and their local and global commanders?

9.8.1.2 Pattern Recognition

An even deeper issue has to do with identifying the primitive forms of information that are relevant on the battlefield. Traditionally, the role of the

combat operations research analyst has been to assimilate and provide useful insights from certain conventional streams of battlefield data: attrition rate, posture profiles, available and depleted resources, logistics, rate of reinforcement, Forward Edge of the Battle Area (FEBA) location, morale, and so forth. While all of these measures are obviously important, and will remain so, having an ABM of combat permits one to ask the following deeper question: *"Are there any other forms of primitive information – perhaps derived from measures commonly used to describe the behavior of nonlinear and complex dynamical systems – that might provide a more thorough understanding of the fundamental dynamical processes of combat?"* We have already mentioned, for example, that evidence suggests that the intensity of battles – both in the real world and in ABMs of combat – obeys a fractal power-law dependence on frequency and displays other traits characteristic of high-dimensional chaotic systems. Are there other, similar but heretofore unexamined, measures that may provide insight into the dynamics of real-world combat?

9.8.1.3 "What If?" Experimentation

The strength of ABMs lies not just in their providing a potentially powerful new general approach to computer simulation but also in their infallible ability to prod researchers into asking a host of interesting new questions. This is particularly apparent when EINSTein is run interactively, with its provision for making quick "on-the-fly" changes to various dynamical parameters. Observations immediately lead to a series of *"What if?"* speculations, which in turn lead to further explorations and further questions. Rather than focusing on a single scenario and estimating the values of simple attrition-based measures of single outcomes ("Who won?"), users of agent-based simulations of combat typically walk away from an interactive session with an enhanced intuition of what the overall combat fitness landscape looks like. Users are also given an opportunity to construct a context for understanding their own conjectures about dynamical combat behavior. The agent-based simulation is therefore a medium in which questions and insights continually feed off one another.

9.8.1.4 Validation

Before any combat model – agent based or not – is judged "useful" to a military operations researcher, it must pass two important tests in the affirmative: (1) Does it provide insight into the specific set of problems the researcher is interested in studying, in a well-defined and self-contained conceptual context? (which is an obvious requirement of even the most basic mathematical model) and (2) Is its output consistent with behavior that is either accepted to be true or has otherwise been observed to occur in the real-world? Most

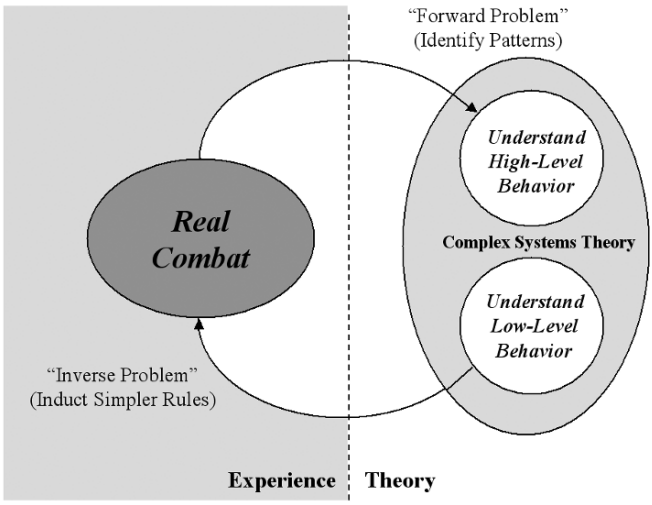

Fig. 9.19 Schematic of the interplay between experience and theory in the *forward* and *inverse* problems of simulating combat.

importantly, a successful model of combat must respect the critical interplay between real-world experience and simulation outcome. That is to say, the model must at some point be *validated*.

Fig. 9.19 illustrates, schematically, the interplay between the *forward* problem of using the model to predict behaviors and identify patterns and the *inverse* problem of using experience and real-world data to add to, change, and/or refine the rules and behaviors that define the model. The forward problem consists of observing real-world behavior, with the objective being to identify any emergent high-level behavioral patterns that the system might possess. The inverse problem deals with trying to induct a set of low-level rules that describe observed high-level behaviors. Starting with observed data, the goal is to find something interesting to say about the properties of the source of the data. Solving the forward problem requires finding the right set of theoretical tools that can be used to identify patterns, while the inverse problem needs tools that can be used to induct low-level rules (or models) that generate the observed high-level behaviors.

Since EINSTein was conceived primarily as an exploratory model, validation is less of an issue than it might be for more ostensibly realistic models. As long as EINSTein's outcomes are intuitive and its inputs are easy to generalize (or translate) for use by more realistic models, EINSTein will remain useful for many different kinds of exploratory analyses. Nonetheless, because EINSTein is already powerful enough to simulate many real-world scenarios, it is proper to ask about how the model may be validated.

One such recent attempt to validate (a beta version of) EINSTein was undertaken at the United States West Point Military Academy's Operations

Research Center for Excellence by Klingaman and Carlton [64], by comparing its output to that of another well-established combat simulation model, called JANUS (a high-resolution conventional – i.e., not agent based – simulation of red and blue forces, with resolution down to the individual platform and soldier). The study endeavored to establish the combat effectiveness of EINSTein agents executing a standard National Training Center combat scenario that consists of a single armored company of 14 blue force tanks engaged versus a similar size force of 14 red battle tanks. One set of blue agents is allowed to gain knowledge (or "learn") by using EINSTein's built-in GA agent breeder function. The behavior of another set of blue agents is kept fixed, and the agents are not allowed to learn. In both cases, EINSTein's combat results for all agent actions are recorded. These *observed actions* are then programmed into JANUS (EINSTein's automatic record of center-of-mass positions are used to define the routes in JANUS), and, for each case, the combat effectiveness resulting from JANUS is compared to the outcome in EINSTein. The validation test consisted of verifying the reasonable expectation that the knowledgeable agents exhibit noticeably different (and, hopefully, improved) behavior and have a significantly better loss-exchange ratio (LER) in both EINSTein and JANUS.

Using a general linear model analysis of two factors (agent type and model) with two levels each (default/GA-bred and EINSTein/JANUS), the study found that although the LER was different for the two models, the LER data in both models follow similar trends. The standard deviations of the mean LER also decrease from the default agents to GA-bred agents in both models. Overall, the study found that EINSTein's agents may be used to portray similarities of combat vehicles reasonably well and that learning as portrayed in EINSTein can be transferred into another combat model. However, citing various limitations due to model-specific constraints on translating agent and environmental characteristics from one model to the other, as well as unavoidable conceptual differences between the two models, Klingaman and Carlton concluded their report by offering three suggestions: (1) that ABMs need increased fidelity in terms of terrain and weapons performance (limitations that are no longer an issue for newer versions of EINSTein; see discussion in the next section), (2) traditional models, such as JANUS, ought to incorporate ABM-like personality traits and decision-making algorithms to allow for more realistic combatant actions, and (3) traditional models ought to incorporate some mechanism to allow for adaptive learning.

One other important suggestion that belongs on Klingaman and Carlton's list of general recommendations is due to Axtell et al. [65], who argued for the need to *align* (or "dock") simulation models. Docking refers to a process of testing to see whether two models can produce the same results, or using one model to check the output of another, but in a more general and systematic manner than was used in the one-time EINSTein-to-JANUS comparison discussed above. The idea is similar to a common practice among technical researchers to use not one, but two or more mathematical packages

(such as *Mathematica* [66] and *Maple* [67]) to help check their calculations. The authors illustrate this concept by using, as their testbed simulations, a model of cultural transmission designed by Axelrod [76] and the *Sugarscape* multiagent-based simulation of evolution that takes place on a notional sugar field, developed by Epstein and Axtell [29]. Since the models differ in many ways (and have been designed with quite different goals in mind), the comparison was not an especially easy one to make. Nonetheless, the authors report that the benefits of the sometimes arduous process of alignment far outweighed the hardships. In the end, the user communities of both models benefited from the alignment process by gaining a deeper understanding of how each program works, of their similarities and differences, and of how the inputs and outputs of each program must be modified before a fair comparison of what really happens in either model can be made.

As the list of combat agent models grows beyond those that have already appeared (such as CROCADILE, MANA, SEM, Socrates, and SWarrior; see [34]), the need to "align" the output of these models – in the sense defined by Axtell et al. [65] – can only become more important.

9.8.1.5 Universal Grammar of Combat?

What lies at the heart of an artificial life approach to simulating combat is the hope of discovering a fundamental relationship between the set of higher-level emergent processes (penetration, flanking maneuvers, containment, etc.) and the set of low-level primitive actions (movement, communication, firing at an enemy, etc.). Wolfram [77] has conjectured that the macro-level emergent behavior of all cellular automata rules falls into one of only four universality classes, despite the huge number of possible local rules. While EINSTein's rules are obviously more complicated than those of their elementary cellular automata brethren, it is nonetheless tempting to speculate about whether there exists – and, if so, what the properties are – a *universal grammar of combat*. A step toward achieving this far-reaching goal is being taken with EINSTein's recent integration within *Python*, a scripting language that allows programming-savvy users to extend EINSTein (see discussion below).

9.9 Overview of Features in Newer Versions

EINSTein has evolved considerably beyond the snapshot of the program that appears in the previous sections of this chapter, although the core of the model remains essentially the same and is consistent with everything thus far discussed.[8] This section summarizes the state of the model as it appears

[8] Version 2.1 may be downloaded from [71], and older versions may be downloaded from [72].

at the time of this writing (April 2008) and highlights some of the main changes and additions that have been made to the program since the first edition of this book came out in 2005.

Loosely speaking, EINSTein's emphasis as a CAS-based "combat simulator" has gradually shifted away from describing the mutual interactions among many *simple* agents (although EINSTein's original capability in this regard is still there, of course) to describing interactions among a relatively few, but *complex* agents that are also endowed with a richer internal structure and dynamics. Thus, where earlier versions have focused on the complexity of emergent behaviors on the system level, more recent work adds the ability to explore emergent behaviors on the individual agent level as well. Details are discussed in [34].

Apart from (mostly self-explanatory) changes to the GUI and file I/O (see below), the newest version of EINSTein includes three general classes of enhancements over earlier releases: (1) agent attributes, (2) battlefield environment and behavior functions, and (3) integration within *Python*.

9.9.1 Agent Attributes

An agent's "personality" may now be tuned using many more primitive attributes than previously available. For example (in alphabetical order),[9] (1) ***Acuity***, which regulates the strength of an agent's focus on its priorities and is used to "sharpen" decisions that are made probabilistically (its range of possible values varies from +1, for which an agent only considers options of maximum value, to +1, for which an agent gives equal consideration to all options); (2) ***Camouflage***, which defines how well an agent can hide from other agents in the context of a local environment (if the value is 0, an agent is 100% visible to other agents; if the value is 1, an agent is effectively invisible to both friends and enemies; and intermediate values determine an agent's inherent probability of being detected); (3) ***Frazzle***, which specifies how much an agent's firing accuracy degrades when the agent is presented with multiple simultaneous targets (and increases linearly with the number of simultaneous targets an agent fires at, reaching its maximum value at the weapon's maximum target limit); (4) ***Mass***, which specifies the rate at which agents burn their energy (all agents expend energy while moving, the amount being determined by their "velocity" – i.e., how far they move in a single time step – and by the nature of the terrain over which they are moving); (5) ***Memory***, which regulates how long does an agent retains information about certain internal state variables (and is computed using a time-averaging technique: the contribution of past values decays exponentially, with the value of memory controlling the rate of that exponential decay); (6) ***Recovery_rate***, which

[9] This is only a partial list of agent attributes that have been added to EINSTein since version 1.0. For a complete list, please refer to EINSTein's programming guide [75].

defines how much health an agent regains in a single time step (a value of 0 means an agent never recovers from injury and a value of 1 means an agent fully recovers from all injuries in a single time step; all intermediate values are interpreted as the probability an agent will recover from injuries); (7) ***Resupply_rate***, which specifies how frequently an agent reloads its weapon with new ammunition (a value of 0 means that an agent never reloads; a value of 1 means that an agent reloads at the start of every time step; intermediate values are interpreted as the probability of reload); (8) ***Stamina***, which is the rate at which agents regain their energy (energy expenditure is proportional to agent mass, and the amount recovered is proportional to stamina, if ***mass*** = ***stamina*** then an agent can traverse 100% passable terrain indefinitely); and (9) ***Tremor***, which is the degree to which an agent's "inherent fallibility" reduces its weapon firing accuracy (if the value is 0, then weapon accuracy is unaffected; if the value is 1, then one standard deviation is added to the agent's weapon's average hit distance from an intended target). ***Health*** has also been generalized from a binary variable (healthy/not-healthy) to a continuous one (ranging in value from 0 to 1). And ***movement_range*** is now effectively free to vary from 0 to the size of the battlefield.

9.9.2 Environment and Behavior Functions

There are six major additions to EINSTein's battlefield environment and repertoire of behavior functions and modifiers: (1) ***Pathfinding***, which uses a priority-queue variant of Dijkstra's optimal path algorithm [51] to give agents an innate "intelligence" to find paths between any two points on the battlefield (an ability which, among other things, prevents agents from becoming trapped at corners of an obstacle and generally yields more realistic "flow" around terrain elements); (2) ***Waypoint scripting***, which allows the user to define arbitrarily complex paths (or roads) on the battlefield and tune the way in which agents traverse them (that also allows for far more complex scenarios to be constructed than before)[10]; (3) ***Obstacles***, which are fixed objects in the terrain that interfere with weapon fire and agent movement (and can therefore be used as building blocks to populate a battlefield with notional buildings). A weapon's *loft* and an obstacle's *height* properties determine whether a weapon round reaches an intended target[11]; (4) ***Weapon-construction class***, which allows users to design their own

[10] The "battlefield" that appears on the upper left of the screenshot shown in Fig. 9.1 contains two red and two blue user-defined paths defined using waypoints.

[11] *Impenetrability* defines how much energy an obstacle may absorb from a weapon blast. If an obstacle exists along the straight-line path between a weapon and its target and $height > loft$, then all rounds fired by the weapon will be blocked from reaching its target. If $height \leq loft$, a weapon's blast energy is reduced by a factor $\propto$ *Impenetrability*. If there are multiple obstacles between a weapon and its kill-zone center, then the reduction in

weapons-of-choice[12] using a palette of 10 primitives (*range, firing rate, capacity, blast radius, power, armor, deviation, reliability, loft,* and *ammunition capacity*); (5) ***Weapon-targeting logic***, which provides agents with an intelligent targeting capability (and with which agents can discriminate targets by weighing the relative potential benefit of firing at the given coordinate on the battlefield). Agents consider factors such as expected blast size, the damage likely to be inflicted on friends and enemies near the target coordinate, and the value or threat that specific enemy agents represent. A targeting penalty function is used to evaluate each of the possible targeting strategies that may be used in a given context and is an analog of the movement penalty function defined in Eq. 9.4; and (6) ***Trigger states***, which generalize EINSTein's older meta-rules (which are still available) by allowing users to associate certain predefined environmental conditions with agent behaviors (i.e., agent actions may now be adaptively triggered by dynamic contexts).

Meta-rules have always allowed agents to tailor their behavior to simple contexts – the Δ_{Combat} meta-rule, for example, defines the conditions under which agents either engage (or do not engage) the enemy (see Table 9.2) – but also constrain the user to making basic *either/or* decisions (and limit an agent's context-specific behavior modification to changing a single component of its default personality weight vector). EINSTein's newer trigger-state logic is vastly more flexible and allows essentially arbitrary modifications of an agent's behavior to be made contingent upon arbitrary environmental conditions. Aside from obviously adding a great deal of realism to scenarios, the new logic also allows analysts to more deeply explore interactions between agent personalities and their dynamic environment.

Fig. 9.20 shows a screenshot of a sample work session in which the user has elected to define a trigger state called "Hunker Down." We see that it depends on three activation conditions (health, energy, and fire density) and that the response (shown along the right-hand side of the figure) consists of changing the value of 7 of 12 personality features (maximize rest and propensity to flock, replace values for alive ally, own flag, and enemy flag with 0, replace the value for alive foe with −100, and replace value for moving toward the local commander with +100). The probability that a given condition becomes "active" may be defined as a piecewise discontinuous probability density function using another dialog (not shown) that pops up by pressing the "Define PDF..." button associated with a given condition. The probability, P_A, that a trigger state (in this case, "Hunker Down") is active is the product of the conditional probability distribution functions, P_a^i, applied to

blast energy is cumulative (additive) along the straight-line path connecting the site and the kill-zone center.

[12] The user is presented with a default – but modifiable – selection of *bolt-action rifle* (accurate, reliable, low range, low power); *semiautomatic rifle* (higher rate, lower range, less accurate); *machine gun* (high rate, higher power, lower reliability); *grenade* (lower range, larger spread, hard to target); and *mortar* (long range, large spread, high power, accurate).

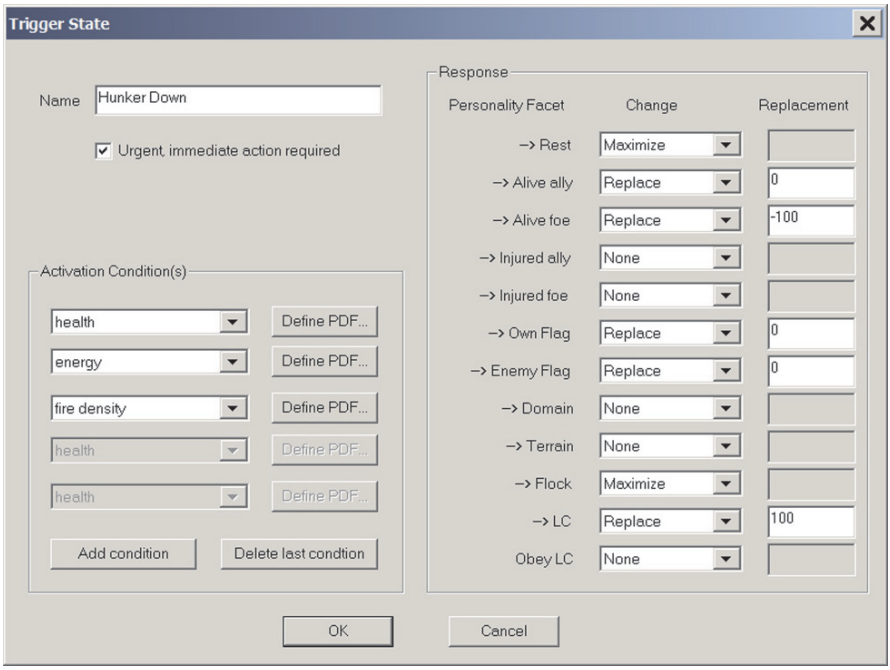

Fig. 9.20 Screenshot of EINSTein's new trigger-state edit dialog.

their corresponding activation conditions, C_i, is $P_A \equiv \prod_i P_a^i(C_i)$. Up to five activation conditions may be active at one time. However, there is no limit on the number of trigger states that may be defined in a scenario for any given agent.

9.9.3 GUI and File I/O

While most of the older elements of EINSTein's GUI (see Fig. 9.1) remain intact, albeit in a modified form to accommodate the growing palette of agent attributes and behaviors, entirely new elements have been added to make it easier for the user to interact with the program as it is running. A click of the *right* mouse button now calls up a series of menus (see Fig. 9.21) that provide quick access to essentially all of EINSTein's key data-entry dialogs. There are also additional options to both probe and alter the environment. For example, the user may choose to "inspect a site" to see the properties of a particular terrain element (if the site is "empty") or to be given a complete list of agent attributes if the site is occupied by an agent. Or, the user can pick up an agent (or a goal), move it from one part of the battlefield to another, and

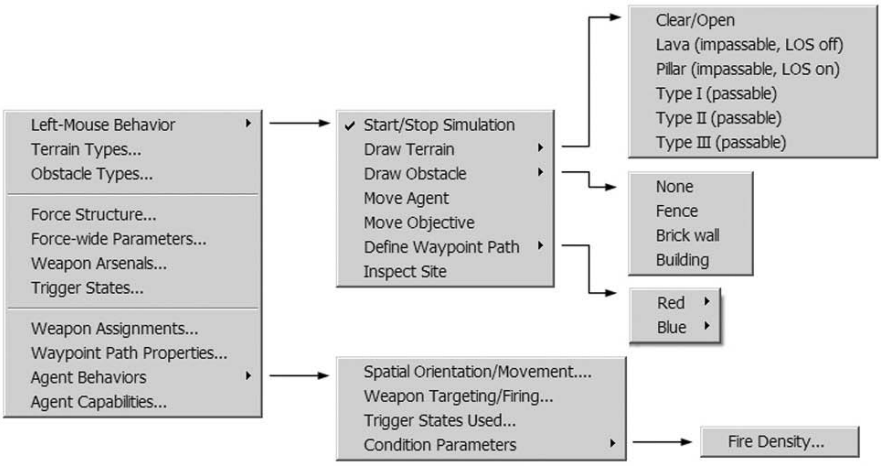

Fig. 9.21 Screenshots of pop-up menus that appear after clicking with the right mouse button in EINSTein versions 2.0 and higher.

continue running the same scenario. Terrain elements, obstacles, waypoints, and paths can also all now be drawn freehand directly on the battlefield.

The types and contents of the scenario and data files depends on the version of the program. Versions 1.0 and older use basic ASCII text files. Starting with versions 1.1 (and newer), EINSTein uses Apache Software's open-source Xerces XML (eXtensible Markup Language) parser in C++ (version 2.3 [73]) to provide XML I/O functionality. Compatibility with EINSTein's older *.dat file format has been retained in the form of I/O functions that appear under the Load and Save options of the main File menu. EINSTein has also been significantly enhanced with data logging and data exporting functions.

9.9.4 Python and wxWidgets

One of the most exciting changes that has been made to EINSTein in recent years is its integration within Python. Python is an interpreted, interactive, object-oriented programming scripting language. It is freely available in source or binary form from the Python developer homepage [69]. Since Python is easily extended with new functions and data types implemented in C or C++ (or other languages callable from C), when combined with the 300+ source-level functions of EINSTein's core engine, Python becomes a powerful programming environment in which to build custom "combat scenario" applications.

PyE (version 2.1) is an extension of EINSTein that allows the simulation to be run directly within the Python programming environment. On the

GUI level, PyE builds on (the pre-Python) EINSTein source code but is without the earlier versions' embedded graphical visualization tools (although the data-extraction routines remain the same). The idea is to first use PyE interactively to generate (and/or fine-tune) desired scenarios and then run PyE in batch mode to generate statistics for follow-on analysis by some other program. A sample Python-script for multiple time-series runs is included in the automatic Windows install file [71].

Future versions of EINSTein (and PyE) will migrate entirely away from EINSTein's original Windows MFC codebase and use wxWidgets [74] as the development platform. wxWidgets is a cross-platform, open-source GUI library that provides a single API for simultaneously developing applications on multiple platforms (such as Win32, Linux/Unix, Mac OS X, X11, and Motif) using a single codebase.

9.10 Next Step

While EINSTein continues to be developed and new capabilities are still being added to the program as of this writing (April 2008), a new direction has recently been taken in a related project. The project's goal is to develop an EINSTein-based model of terrorist-counterterrorist coevolutions called SOTCAC (= *S*elf-*O*rganized *T*errorist-*C*ounterterrorist *A*daptive *C*oevolutions).

SOTCAC leverages and generalizes EINSTein's "combat agents" to a class of "terrorist agents" that live in an abstract social-information space – that is, a mathematical graph whose vertices possess an inner semantic structure, and assimilate, process, and adapt to various forms of information (such as a multidimensional feature space that describes the properties of the Al Qaeda terrorist network, as derived from intelligence sources). SOTCAC uses adaptive agents to describe the self-organized, emergent behavior of terrorist networks – conceived as complex adaptive systems – on three interrelated dynamical levels: (1) *dynamics on networks*, in which notional terrorist agents process and interpret information, search and acquire resources, and adapt to other agents' actions; (2) *dynamics of networks*, in which the terrorist network itself is a fully dynamic, adaptive entity and whose agents build, maintain, and modify the network's local (and therefore, collectively, its global) topology; and (3) dynamics *between networks*, in which the terrorist network and counterterrorist network mutually coevolve; the terrorist network's "goal" is to achieve the critical infrastructure (of manpower, weapons, financial resources, and logistics) required to strike, while the counterterrorist network's mission is to prevent the terrorist network from achieving its goal.

For all of EINSTein's "complexity" on the source code level (EINSTein consists of roughly 150K lines of C++ code, contains over 350 agent functions, and requires a programming manual that is about 400 pages long to define its class structure and primitive behaviors), EINSTein's basic dynamics

is – conceptually speaking – very simple: Red and blue agent swarms interpenetrate on a notionally physical battlefield with the "goal" of annihilating the other side and/or "controlling" a certain region of the battlefield (such as the opponent's "flag"). There are effectively two – and only two – kinds of actions that agents must adjudicate as a battle unfolds: (1) *where to move* and (2) *whom to fight*; and everything "happens" in a single notional space (i.e., the "battlefield").

In contrast, the behaviors in SOTCAC depend on (and take place in) four coupled spaces: (1) a *physical space*, analogous to EINSTein's "battlefield"; (2) a *social space*, in which terrorist agents forge/break social bonds and exchange information and materiel resources; (3) an *"image" of the physical space* that represents the counterterrorist network's "best guess" view of the terrorist's activity in the physical space; and (4) an *"image" of the social space* that represents the counterterrorist network's "best guess" view of the terrorist network's activity in its social arena. Not only are the physical and social spaces obviously coupled, but because all counterterrorist actions are fundamentally grounded on what the counterterrorist network "believes" the terrorist network is "doing" at any given time (possibly erroneously), the coupled terrorist physical/social space ("ground truth") is also tightly coupled with the counterterrorist belief network.

SOTCAC's agents must fuse multiple streams of (often conflicting information), decide to whom (and when, and for how long) to establish communication (and/or material flow) links, and – in the case of counterterrorist forces – decide on courses of action based on an incomplete "snapshot" view of what is currently known about (and/or can be inferred from) existing data. Consequently, SOTCAC's effective "space of possible interactions" (i.e., its emergent fitness landscape) is significantly larger than EINSTein's. SOTCAC is currently in development.

Acknowledgements US Marine Corps LtGen (Ret) Paul van Riper's vision of applying the lessons of complexity theory to warfare directly inspired the ISAAC and EINSTein projects. I would like to thank Lyntis Beard (who coined both of the names ISAAC and EINSTein), Rich Bronowitz, David Broyles, Dave Kelsey, David Mazel, Katherine McGrady, Igor Mikolic-Torriera, Mike Shlesinger, Jonathan Schroden, Greg Swider, David Taylor, and Wendy Trickey, all of whom helped form and shape EINSTein over the years. Programming support for the EINSTein project continues to be very skillfully provided by Fred Richards. Funding was provided, in part, by the Office of Naval Research (Contract No. N00014-96-D-0001).

References

1. Lanchester F W (1995) Aircraft in Warfare. Lanchester Press, Sunnyvale, California
2. Chase J V (1902) A Mathematical Investigation of the Effect of Superiority in Combats Upon the Sea. In: Fiske B A (1994) The Navy as a Fighting Machine. U.S. Naval Institute Press, Annapolis, Maryland

3. Osipov M (1995) The Influence of the Numerical Strength of Engaged Forces in Their Casualties. Naval Research Logistics 42:435–490
4. Hofbauer J, Sigmund K (1988) Evolutionary Games and Population Dynamics. Cambridge University Press, Cambridge, United Kingdom
5. Taylor J G (1983) Lanchester Models of Warfare. Operations Research Society of America, Arlington, Virginia
6. Hartley D S, Helmbold R L (1995) Validating Lanchester's Square Law and Other Attrition Models. Naval Research Logistics 42:609–633
7. Weiss H K (1957) Lanchester-Type Nodels of Warfare. Proceedings of 1st Conference on Operations Research, Operations Research Society of America. Arlington, Virginia
8. Fain J (1975) The Lanchester Equations and Historical Warfare. In: Proceedings of the 34th Military Operations Research Symposium. Military Operations Research Society, Alexandria, Virginia
9. Richardson L F (1960) Statistics of Deadly Quarrels. Boxwood Press, Pittsburgh, Pennsylvania
10. Cowan G A, Pines D, Meltzer D (1994) Complexity: Metaphors, Models and Reality. Addison-Wesley, Reading, Massachusetts
11. Kauffman S (1993) Origins of Order. Oxford University Press, New York
12. Langton C G (1995) Artificial Life: An Overview. MIT Press, Cambridge, Massachusetts
13. Mainzer K (1994) Thinking in Complexity. Springer-Verlag, New York
14. Beckerman Linda P (1999) The Non-Linear Dynamics of War. Science Applications International Corporation, `http://www.calresco.org/beckermn/nonlindy.htm`
15. Beyerchen A (1992) Clausewitz, Nonlinearity, and the Unpredictability of War. International Security 17:59–90
16. Hedgepeth W O (1993) The Science of Complexity for Military Operations Research. Phalanx 26:25–26
17. Ilachinski A (1996) Land Warfare and Complexity, Part I. Center for Naval Analyses, Alexandria, Virginia
18. Ilachinski A (1996) Land Warfare and Complexity, Part II. Center for Naval Analyses, Alexandria, Virginia
19. Miller L D, Sulcoski M F (1995) Foreign Physics Research with Military Significance. Defense Intelligence Reference Document, NGIC-1823-614-95
20. Saperstein A (1995) War and Chaos. American Scientist 83:548–557
21. Tagarev T, Nicholls D (1996) Identification of Chaotic Behavior in Warfare. In: Sulis W, Combs A (eds) Nonlinear Dynamics in Human Behavior. World Scientific, Singapore
22. Maes P (1990) Designing Autonomous Agents. MIT Press, Cambridge, Massachusetts
23. Ferber J (1999) Multi-Agent Systems. Addison-Wesley, Reading, Massachusetts
24. Weiss G (1999) Multiagent Systems. MIT Press, Cambridge, Massachusetts
25. Gilbert N, Troitzsch K G (1999) Simulation for the Social Scientist. Open University Press, Philadelphia, Pennsylvania
26. Gilbert N, Conte R (1995) Artificial Societies. UCL Press, London
27. Conte R, Hegselmann R, Terna P (1997) Simulating Social Phenomena. Springer-Verlag, New York
28. Barrett C (1997) Simulation Science as it Relates to Data/Information Fusion and C2 Systems. Los Alamos
29. Epstein J M, Axtell R (1996) Growing Artificial Societies. MIT Press, Cambridge, Massachusetts
30. Prietula M J, Carley K M, Gasser L (1988) Simulating Organizations. MIT Press, Cambridge, Massachusetts
31. Ilachinski A (2000) EINSTein. Center for Naval Analyses, Alexandria, Virginia
32. Ilachinski A (1999) EINSTein User's Guide. Center for Naval Analyses, Alexandria, Virginia

33. Ilachinski A (1997) Irreducible Semi-Autonomous Adaptive Combat (ISAAC). Center for Naval Analyses, Alexandria, Virginia
34. Ilachinski A (2004) Artificial War. World Scientific, Singapore
35. Ilachinski A (2001) Cellular Automata. World Scientific, Singapore
36. Braitenberg V (1984) Vehicles. MIT Press, Cambridge, Massachusetts
37. Boccara N, Roblin O, Roger M (1994) Automata Network Predator–Prey Model with Pursuit and Evasion. Physical Review E 50:4531–4541
38. Woodcock, A E R, Cobb L, Dockery J T (1988) Cellular Automata: A New Method for Battlefield Simulation. Signal 1:41–50
39. Varela F J, Maturana H, Uribe R (1974) Autopoiesis. Biosystems 5:187–196
40. Lauren M K (2002) Firepower Concentration in Cellular Automata Models. Journal of the Operations Research Society 53:672–679
41. Krantz H, Schreiber T (1997) Nonlinear Time Analysis. Cambridge University Press, Cambridge
42. Lauren M K (1999) Characterizing the Difference Between Complex Adaptive and Conventional Combat Models. Defense Operational Technology Support Establishment, New Zealand
43. Lauren M K, McIntosh G, Moffat J (2007) Fractals: From Physical Science to Military Science. Phalanx 40:8–10
44. Dockery J T, Woodcock A E R (1993) The Military Landscape. Woodhead Publishing Limited, Cambridge
45. Bak P (1996) How Nature Works. Springer-Verlag, New York
46. Roberts D C, Turcotte D L (1988) Fracticality and Self-Organized Criticality of Wars. Fractals 6:351-357
47. Taylor D, Schmal C, Hashim A (2000) Ground Combat Study: Summary of Analysis. Center for Naval Analyses, Alexandria, Virginia
48. Edwards S J A (2000) Swarming on the Battlefield. RAND Corporation, Santa Monica, California
49. Dewar J A, Gillogly J, and Juncosa M (1991) Non-Monotonicty, Chaos, and Combat Models. RAND Corporation
50. Mitchell M (1996) An Introduction to Genetic Algorithms. MIT Press, Cambridge, Massachusetts
51. Thulasiraman K, Swami M (1992) Graphs: Theory and Algorithms. John Wiley and Sons, New York
52. Schroden J, Broyles D (2005) EINSTein Simulations of Squad and Fire Team Operations. Report 11705, Center for Naval Analyses, Alexandria, Virginia
53. Gill A (2001) Impact of Reconnaissance on Mission Success. Defense Science and Technology Organization, Australia, Briefing Slides
54. Baigent D, Lauren M (2000) Exploring the Recce-Counter Recce Scenario with ISAAC. Defense Operational Technology Support Establishment, DOTSE Report 171, NR 1349
55. Schroden J, Broyles D (2005) An EINSTein Simulation of a Conventional Ground Combat Scenario. Report 10873, Center for Naval Analyses, Alexandria, Virginia
56. Broyles D, Trickey W, Schroden J (2005) Ground Combat Modeling in EINSTein. Report 13192, Center for Naval Analyses, Alexandria, Virginia
57. Broyles D, Trickey W (2006) Modeling Dispersed Units in Open and Urban Environments with EINSTein, Volume I. Report 14187, Center for Naval Analyses, Alexandria, Virginia
58. Perla P, Ilachinski A, Hawk C, Markowitz M, Weuve C (2002) Using Gaming and Agent Technology to Explore Joint Command and Control Issues. Report 7164, Center for Naval Analyses, Alexandria, Virginia
59. Woodaman, R (2000) Agent-Based Simulation of Military Operations Other Than War, Small Unit Combat. Master's Thesis, Naval Postgraduate School, Monterey, California

60. Kewley R (2001) Combat Operations Support System for Adaptive Command and Control. United States Military Academy, Briefing Slides
61. Freedman T, Ilachinski A (2004) Riots and Civil unrest: An Agent-Based Model Approach. Center for Naval Analyses Briefing Slides
62. Lauren, M, Stephen R (2000) Modeling Patrol Survivability in a Generic Peacekeeping Setting Using ISAAC. Defense Operational Technology Support Establishment, DOTSE Report 172, NR 1350
63. Cox G (2002) EINSTein Visits the Persian Gulf (via Monterey, CA): A Naval Requirements War "Gamelet" Held at the Naval Postgraduate School, Monterey, California. Annotated Brief, Center for Naval Analyses, Alexandria, Virginia
64. Klingaman, Randall R, Carlton B (2002) EINSTein Model Validation, Military Academy, West Point New York, Department of System Engineering, Operations Research Center of Excellence Technical Report DSE-TR-02-03
65. Axtell R, Axelrod R, Epstein J, Cohen M (1996) Aligning Simulations Models: A Case Study and Results. Computational and Mathematical Organization Theory, 1:123–141
66. Wolfram Mathematica website, http://www.wolfram.com/products/mathematica/index.html
67. Maple website, http://www.maplesoft.com/products/Maple11/professionals/
68. Graphics Server website, http://www.graphicsserver.com/
69. Python developer homepage, http://www.python.org/
70. ISAAC website, http://www.cna.org/isaac/
71. EINSTein website, http://www.cna.org/isaac/PyE_setup.htm
72. EINSTein previous releases website, http://www.cna.org/isaac/einstein_install.htm
73. Apache Software's Xerces XML parser, http://xml.apache.org/xerces-c/index.html
74. wxWidgets developer homepage, http://www.wxwidgets.org/
75. EINSTein's programming guide, http://www.cna.org/isaac/EINSTein/PyEProgRef.pdf
76. Axelrod R (1997) The Dissemination of Culture: A Model with Global Polarization. Journal of Conflict Resolution, 41:203–226
77. Wolfram S (1994) Cellular Automata and Complexity: Collected Papers. Addison-Wesley, Reading, Massachusetts