Formalizing Game-play

Simulation & Gaming
43(2) 157–187

DOI: 10.1177/1046878110388239
http://sag.sagepub.com

Tomas By[1]

> Wouldn't you prefer a good game of Chess?
> Later. Let's play Global Thermonuclear War.
>
> —Wargames (1983)

Abstract

Current computer conflict simulation games, or wargames, are opaque in the sense that most of the game mechanisms are not directly visible to the players and are frequently not described in user accessible documentation, have a transient lifetime that is mainly shaped by the evolution of graphics hardware and processor speed, and do not, in contrast with, for example, the well-known abstract board games CHESS and GO, have the technical prerequisites for critical intellectual discussion that the thought-intensive and knowledge-rewarding character of these games seems to warrant. The main reason for this state of affairs is that many of the mechanisms of the games, and in particular the details of how the game state changes over time, are directly expressed in computer code. This is purely a technical problem, and it has a straightforward solution, namely, treating this information as data by creating a formalism for describing not just the game map and playing pieces but also all the game rules including the "sequence of play." The article suggests such a formalism and shows a complete specification of a simple, but complicated enough for present purposes, "introductory" board wargame. This formalism, with tools that support it, can provide an unambiguous authoritative definition of the rules, accessible by both human and computer players; would allow existing board wargames to be played on a computer, without any simplifications or sacrifices of rule details; and may allow construction of more advanced computer players, since a complete formal specification of the game rules is available as input to them.

Keywords

board games, computer conflict simulation games, computer games, formal specification, formalism, game construction, game design, game development, game formalization, game map, game mechanisms, game-play, game rules, game specification, rules, sequence of play, simulation games, strategy board games, wargame, war gaming

[1]Technische Universität München, München, Germany

Corresponding Author:
Tomas By, Institut für Informatik, Technische Universität München,
Boltzmannstraße 3, 85748 Garching bei München, Germany
Email: tomas.by@in.tum.de

The term *wargaming* covers a wide spectrum of activities, from multiday events involving purpose-built facilities and hundreds of people to inexpensive game software for personal computers. While wargaming in some form has a long history,[1] the traditions that exist today can be traced back to 19th-century Prussia.[2] Parallel to the military use of wargames for training[3] and research,[4] a tradition exists of recreational or hobby wargaming, which started in the early 20th century and grew into a significant industry in the second half of it (Berg, Patrick, Simonsen, Isby, & Dunnigan, 1977, pp. 10-25; Palmer, 1977, pp. 13, 19-20; Sabin, 2002). Today, this type of wargaming is marginal compared with "real-time strategy" and "first-person shooter" games. As objects of analysis, however, games in the "classical" format, characterized by step-wise movement on a hexagonal grid,[5] have some advantages. Because of the discrete, low granularity modeling of time and space, things such as the number of possible unit locations, the number of possible moves at any one time, and the total number of moves in a game are tractable. This means that game states and histories can be described in print using a reasonable amount of space. Detailed, concrete discussions of the play of specific games is quite common in the literature on hobby wargaming,[6] as it is in the literature on the board games CHESS and GO.[7] Wargames differ from the latter, of course, in that each game has its own rules, and these rule sets can differ considerably. With the shift from printed board games to computer implementations the situation worsened dramatically, and very little detailed discussion of this type concerning computer wargames appears in public. One reason for this, no doubt, is that the game rules for the computer versions are typically not available in enough detail. The main outlines may be explained in the manual, but specific information on things such as the cost of movement in various types of terrain under various conditions and exactly how combat is resolved is often not revealed.

What computer wargames need to make the game mechanisms accessible to the user, and what the critical discussions of wargame play need to be independent of physical rule books and human rule interpretation efforts, is a formalism for writing wargame rules. This would include all the conditions, procedures, tables, and so forth that make up the game rules, and a particular game can be encoded as a set of statements in this formalism and will then exist as a logical object that can be read as input by a game-playing program, or some other software tool, and be referred to unambiguously in discussions of technical game-playing details.

Two terms that are commonly heard in discussions of war gaming are *model* and *simulation*. Generally speaking, a model is a simplified representation of something, including only those aspects that are considered relevant, and a simulation is a representation whose behavior is similar, in some respects, to the original over time (Davis & Anderson, 2003, pp. 1, 76; Davis & Blumenthal, 1991, p. 2). So in some sense any wargame is a simulation, and any simulation includes a model.[8]

The remainder of the article has three parts, of two sections each. First section is on the notion of game-play, and the second is on the problems with present-day computer wargames. Then two sections on the proposed solution: the human-friendly external representation and the computer-friendly internal one. Finally, a section on possible future developments of the approach proposed here and a summary.

The Notion of *Game-Play*

Simple computer games such as SPACE INVADERS and PAC-MAN (Crawford, 2003, pp. 18, 19) were originally written for arcade consoles and now exist for almost any combination of computer hardware and operating system. What makes us say that these various pieces of software, which may differ greatly in superficial qualities such as sound and graphics, not to mention the actual program code, implement the "same" game is the fact that the in-game figures and objects, the actions available to the player, and their effects in the game are the same. Similarly, the rules of CHESS are the same whether it is played on a physical board or it is all just a picture in a book or moving images on a computer screen. The game-play of CHESS is well known, since the game is normally played manually and it is recognized that both players need to be familiar with the rules. In the case of wargames, the situation is more complicated. It is probably not the case that players generally know all the rules of the game, for a couple of reasons. Most games have quite a few rules,[9] and because wargames are typically not abstract like CHESS, but intended to portray an actual historical situation, or at least one that could conceivably occur in the real world, it seems often to be possible to play credibly based on general knowledge of warfare and history, without detailed knowledge of the formal game rules. Nevertheless, the games have rules, and together with (some of) the information represented by the mapboard and playing pieces, they constitute what we call the game-play.

Rules and Procedures

In the available literature, the degree of consistency in the detailed classification and terminology is limited. Prados (1987) makes a general observation.

> Typically, a wargame includes systems governing the amount of information players will have, the movement of forces, combat interactions among forces, the method of controlling forces, and the capabilities of military units. (p. 20)

Bowen (1978) is more technical and contrasts *model* with *rules*.

> If the participation of people is overlooked, a game can be thought of as a deterministic model with two main subsystems—a model of the real world (the *Game-world*), and a set of *Rules*, describing its behavior as time passes and as the situation develops. (p. 60)

These rules are then further divided into "system transition rules," which control the updates of the game-world in response to player actions, and "game communication rules," which determine what information the player receives (Bowen, 1978, p. 71). Perla (1990) replaces Bowen's *world model* with a *data base* and a *scenario*, and uses the word model in a more narrow technical sense.

> A set of models, usually a combination of look-up tables and mathematical expressions, translates the game's data and the players' decisions into game events. (p. 165)
>
> But models alone are not enough. A wargame must also have a set of rules or procedures that dictate how and when to apply the models. These rules and procedures help sequence game events, and allow for accurate chains of cause and effect, or action and reaction. They must also ensure that the players receive the appropriate quantity and quality of information during play. (p. 166)

Following McHugh (1966, pp. 3-25, 3-27), the procedures are divided into three types (Perla, 1990):

> those that monitor player actions,
> those that evaluate interactions (i.e., combat), and
> those that provide information to the players (p. 217)

The main reason for having separate rules for the communication of information to the player, which would seem to be a simple matter, is to introduce fog of war[10] effects, where the information that the player gets is partial and/or distorted. For obvious practical reasons, this has seldom been used in board wargames,[11] but it is a standard feature in computational ones (e.g., Darby, 2009, pp. 381-383).

Synthesis

Figure 1 is an attempt at a synthesis of these various theories, showing the major parts of what we want to call the game-play of a wargame. The most basic components are the available terrain types (clear, forest, mountains, etc.) and unit types (infantry, armor, etc.). The available actions would normally be defined in terms of the types of units and terrain, and the "sequence of play" defined in terms of the available actions. These things together constitute the "rules" as opposed to the "scenario." The victory conditions are shown as depending on the units and map, because they normally consist of requirements to occupy certain areas or eliminate specific units (or a certain number of units). Like the authorities say, the central element is the sequence of play (Berg et al., 1977, pp. 78, 108-109; Perla, 1990, pp. 227, 252).

Present Realities

In a board wargame, all the components in Figure 1 are directly accessible to the players. The entire map is visible, with all the units placed on it; the sequence of play, available actions, and victory conditions are described in the rule book, which the players can refer to in case they have not memorized all the details. Typically, a summary of the sequence of play, victory conditions, and terrain and unit types, together with explanations of unit color codes and map symbols, are provided on separate charts. The situation is markedly different in a computer wargame. While the computer relieves the players of some administrative tedium, it also hides information. It is generally

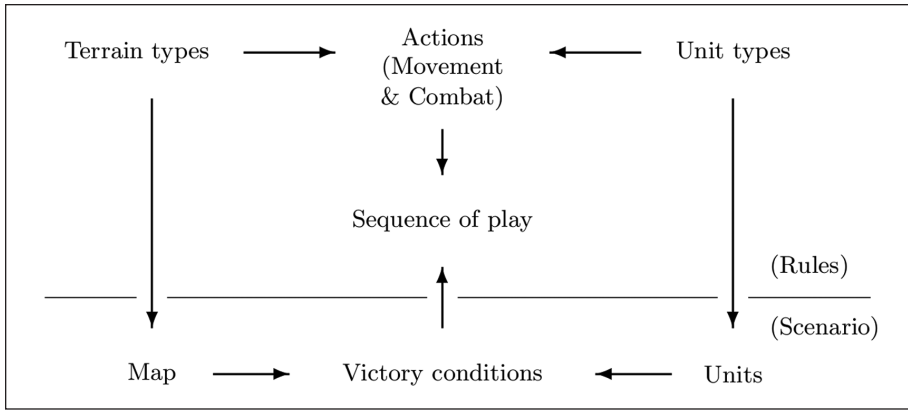
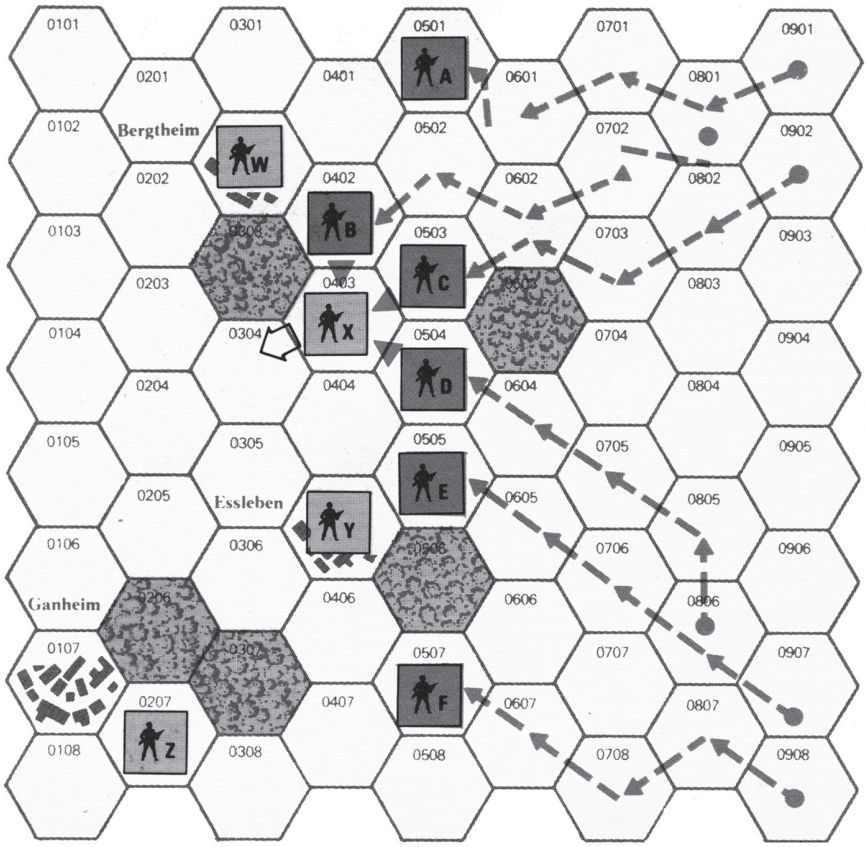


Figure 1. The major components of the wargame "game-play"

harder to get an overview of the map and units in a computer game (pointed out, e.g., by Dunnigan, 2000, p. 74, and Perla, 1990, p. 314). Computer display technology is impressive and shows no signs of ceasing to improve, but it may never match the vividness and immediacy of small physical objects at arm's-length distance. Another related problem, which is not excused by any technical limitations, is that the workings of the game mechanisms are not visible to the players.[12] In a computer wargame, the rules of the game are typically encoded directly in the software (see Dunnigan, 2000, p. 73) and only partially explained in the documentation. It is very seldom that, for example, the exact procedure for combat resolution or the actual probabilities in various situations are given. The player may be given an option to "attack," and be told the numerical strengths of his/her own units, and perhaps an estimate of the opponent's, but not the precise effects of terrain and weather, the full range of possible outcomes, or the exact probabilities of them. This is not primarily a question of player enjoyment but of the fundamental nature of the game. Wargames are not puzzles; they are, or at least should be, tests of skill, knowledge, and tactical and strategic ability. Launching an attack with only a very hazy idea of the likely result may perhaps accurately portray the experience of an incompetent or indifferent commander, but for a game to be an even-sided intellectual duel both players must have full access to the rules.[13]

Basic Events

Perhaps because war itself is often large and complicated, and probably also by way of contrast with the superficially very simple games CHESS and GO, it seems to be generally felt that wargames are complicated and therefore translating them to computer code is necessarily difficult. For a clear appreciation of the technicalities involved, it might be helpful to look at an example, a good source of which is the "introductory"



Movement: On the first turn, all Soviet units move the full four hexes. Note that F cannot move into hex 0506 because it is a Woods hex. A does not move to 0502 because there it might block B, if B were forced to retreat.

Combat: The Soviet player has units B, C and D attack X. He uses the column of the Combat Results Table15 headed 'Three Units.' His die roll results in a '2' (Defender retreats one hex). The U.S. Player retreats his unit to 0304, the only possible hex he can retreat to. The Soviet Player then has C advance into the vacated hex. The Soviet Player decides not to have E attack Y until he can bring more units into the battle.

Figure 2. The first (Soviet) turn of the STRIKE FORCE ONE *sample game*

wargame STRIKE FORCE ONE (Simonsen & Dunnigan, 1975), published as a marketing device by one of the then leading hobby wargame publishers[14] and probably one of the simplest board wargames ever created. Figure 2, taken from the sample game included in the manual, shows the entire first game turn, where six different units move four hexes each, and three of them then make an attack. As can easily be verified by the reader, the game turn displayed in Figure 2 contains six times four separate one-hex

Table 1. The 27 Basic Events During the First Turn of the Sample Game

	Unit	Action		Unit	Action		Unit(s)	Action
1	**A**	0901 → 0801	10	**C**	0802 → 0703	19	**E**	0706 → 0605
2	**A**	0801 → 0701	11	**C**	0703 → 0602	20	**E**	0605 → 0505
3	**A**	0701 → 0601	12	**C**	0602 → 0503	21	**F**	0908 → 0807
4	**A**	0601 → 0501	13	**D**	0806 → 0805	22	**F**	0807 → 0708
5	**B**	0801 → 0702	14	**D**	0805 → 0705	23	**F**	0708 → 0607
6	**B**	0702 → 0602	15	**D**	0705 → 0604	24	**F**	0607 → 0507
7	**B**	0602 → 0502	16	**D**	0604 → 0504	25	**B, C, D**	Attack 0403
8	**B**	0502 → 0402	17	**E**	0907 → 0806	26	**X**	0403 → 0304
9	**C**	0902 → 0802	18	**E**	0806 → 0706	27	**C**	0503 → 0403

movements, plus an attack followed by a one-hex retreat by the defending unit and a one-hex advance by one of the attackers, for a total of 27 separate basic actions. These are listed in Table 1. The events are "basic" in the sense that they cannot be divided into parts. Any shorter movement than from one hexagon to another is not possible, which is of course the purpose of the map grid. Combat between units is simply resolved by a die roll, a common technique in operational and strategic games. In a tactical game, a basic combat action may be the firing of a weapon at a target. In addition to the action types exemplified in Table 1 (one-hex movement; attack), STRIKE FORCE ONE also allows units to be eliminated, but not created, as all units are on the map at the start of the game. The entire sample game, of eight turns, has 68 basic events.

It might be worth pausing and reflecting on the magnitude of this problem. Ten units in the game, 68 hexes on the map, 68 basic events in an entire game. STRIKE FORCE ONE is an extremely simple wargame, and a normal game may perhaps have a few thousand locations and a few hundred units, and might go on for a few tens of turns, involving many thousands of basic actions, but these are still not very intimidating numbers, for a modern personal computer.

In current computer wargames, of course, this notion of a basic game-event has no great significance, as the graphical movements of symbols on the screen may or may not correspond to changes in the game state, and because how and when the game state changes is, as we have already noted, typically revealed to the user only indirectly. For example, the game may have artillery units that can fire a distance or, in other words, make attacks against nonadjacent hexes. Additionally, they may be able to execute "opportunity fire" against enemy actions, such as attacks against other friendly units. The exact details of how all this affects the game state will then typically not be explained in the manual: whether the opportunity attacks are similar to normal artillery attacks, or more, or less, effective; whether they consume the same amount of ammunition; and so on. Implementation-wise, the opportunity attacks may be coded as part of the (enemy) actions they are triggered by, or they may be executed by the same code as normal

artillery fire. This may not be of great interest to players, but for any kind of serious analysis of the gaming, the details must be available.[15]

Formalization

In board wargames, the rules are written in human language, which the computer cannot understand, and in computer wargames, they are directly expressed in computer code, which most human players would probably find hard to read and make use of, even if they had access to it. The obvious way to bring these together and solve the problem of "black box" opaqueness is to design a formal language for writing rules. The analogy between rules writing and computer programming has been noted (Berg et al., 1977).

> [Wargame] rules are not exactly light reading—the number of concepts and procedures to be explained in detail can hardly be dealt with in a few easy paragraphs of colloquial English. The closest analog to a set of rules would be a set of computer program statements. (p. 77)
>
> Another problem with games is that the designer is trying to create, in written form, a mathematical relationship—an algorithm—which is what the rules and Victory Conditions are about. (p. 103)

The rule language we need differs from a programming language, however, in that the formal specification of a game, written in it, is not just an intermediate stage in the production of executable computer code but a logical object that can also be read by human players to, for example, understand the detailed workings of the rules or resolve any conflicts rooted in different interpretations of their expression in English. As noted earlier, the aspect of intellectual competition, which is what many people seek in these types of games, can be most fully realized only if all the players have equal access to the rules.

A Concrete Suggestion

In the rest of the article, a concrete suggestion for such a formalism is described. It is a first prototype and is only expressive enough to handle the example game STRIKE FORCE ONE, but it seems to indicate that the idea is practical. The two main factors that constrain this wargame rules formalism are the expressiveness and the ease of implementation. It must be possible to express all the game-play (Figure 1), and it should not be possible to say too many other things, because that would just make the implementation more complicated for no benefit. It is perhaps convenient to think of this wargames formalism as a programming language, and Perlis (1967) makes a relevant observation.

> A programming language has a syntax and a set of evaluation rules. They are connected through the representation of programs as data to which the evaluation rules apply. This data structure is the internal or evaluation directed syntax of the language. We compose programs in the external syntax which, for the purposes of human communication, we fix. (p. 13)

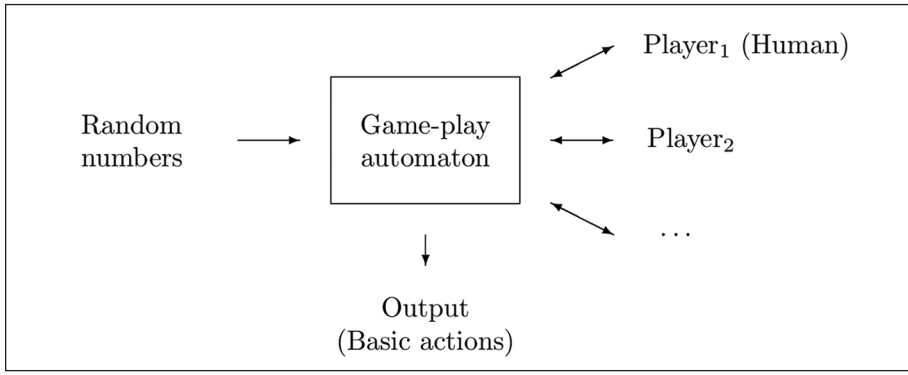


Figure 3. Conceptual structure of the game as a data object

This means that what is needed is an "external" syntax that is human readable (concise and mnemonic) and an "internal" one that is meant for the machine to read, which is to say it should be canonical and as limited as possible in expressiveness.

The external syntax is used to write the rules of the game, to encode the game-play. When playing the game, we eventually end up with a sequence of basic actions that constitute the concrete history of the game (cf. Table 1). Between these two things lies the internal representation of the game-play, which we want to be as simple as possible. Two clear paradigms are available in the history of computing: finite automata[16] and flowcharts.[17] The latter are perhaps closer to what we are looking for, but the former uses a more attractive terminology. Figure 3 shows the conceptual relationships. Given a sequence of player interactions and a source of random numbers, processing of the game-play automaton produces the basic actions that constitute the state history of the game. Since this automaton includes the mapboard and playing pieces in the game, execution of a basic action will typically involve modifications of the automaton.

The Flow of Control

Game-playing software is normally driven by user-interface events, so in terms of control flow, the system will be spending most of the time in the upper right corner of Figure 3. When the user clicks a button, or performs some other interface manipulation, to finish the current phase, the game-play automaton is processed until the next point that requires user input, and then it stops until the user responds again. The double-headed arrows do not simply represent queries and responses but rather turn-taking of initiative. In a wargame, the player who is active in the phase can normally move any number of units, subject to various constraints on which terrain can be entered and how much it will cost, as well as maximum distance for each unit. This means that the active player passes an arbitrary number of interaction commands back to the automaton, before he or she relinquishes control. The form of representation used for the internal syntax, or game-play automaton, will be discussed further, but first we need to consider the external syntax.

External Representation

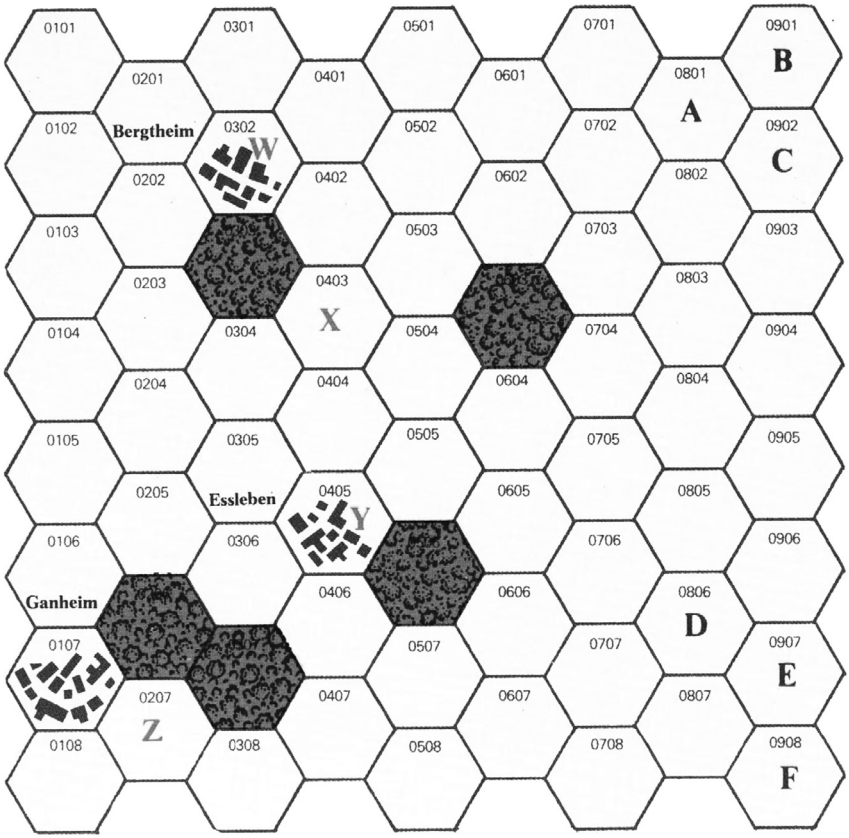
The various types of information that the external formalism must support are those shown in Figure 1: map, units, player actions, sequence of play, and victory conditions. It should also be easy to work with, to try out different methods:

> This is how most development work is accomplished; constant working and reworking to see what happens. Try to visualize the game as a modular set of rules, rules that you can take out and replace with another set to see whether it is that particular section that is at fault. In such a way the developer can see what portions of his game are working and what sections are not. (Berg et al., 1977, p. 53)[18]

Because this formalism only has a single purpose, to encode the game-play, it makes sense to include explicit support for various concepts and techniques that are common in wargame rules but are not normally used in computer programming. Examples of this are a special table syntax for "combat result tables" and "terrain effects charts," shown later in this section. As to the actual syntax, many different paradigms are available to choose from. The format used here, without any further motivation, is inspired by Algol-60 (Naur, 1963) and is characterized by sections delimited by the keywords **"begin"** and **"end,"** and statements generally ending with a semi-colon. Figure 4 shows the map of the example game, in its printed form above and formlized below.[19] Because we are only considering a single example, some details have been omitted. A grid of hexagons has straight lines of hexes running in three different directions, and only one of them can be aligned with the edges of the paper. In Figure 4, one straight line is vertical and two are slanted. Other games have a straight line running horizontally instead. For the numbering of the hexagons, several different systems are used, and we would naturally want the formalism to support all of them so that the formalization of any game can be as near as possible to the printed version. None of the commands and declarations that are needed for these things have been included in the examples here, but they are shown in the appendix. The other major component of wargames, besides the map, is the units. Figure 5 shows the ones in STRIKE FORCE ONE, in the initial deployment.[20]

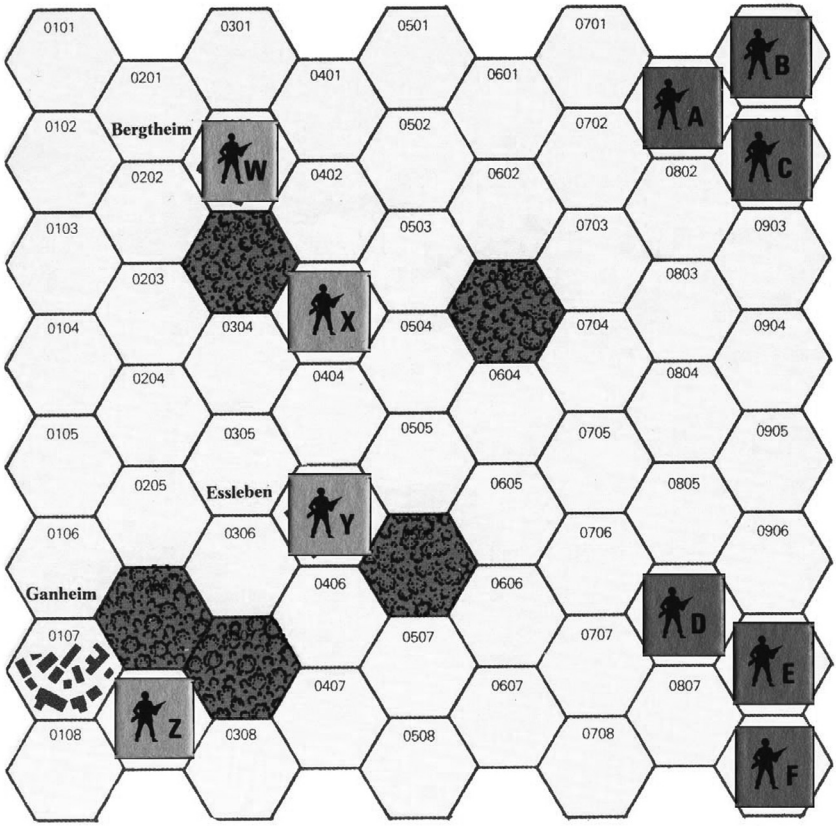
As is commonly the case in computer programming, the same result can be achieved in multiple ways. In the example shown here (Figure 5), the link between a unit and its location on the map is a relation. It would also be possible to use an attribute. Either the units could have a **"location"** attribute or the map hexes an **"occupant"** attribute. STRIKE FORCE ONE has no unit locations other than the map hexes, so attributes would work equally well, but some wargames have special boxes to represent things such as off-map areas, or units that are being transported by ship, and in that case a relation would seem more convenient.

A commonly used device in board wargame rules are tables, typically showing die roll results along one dimension. STRIKE FORCE ONE contains one such table, the



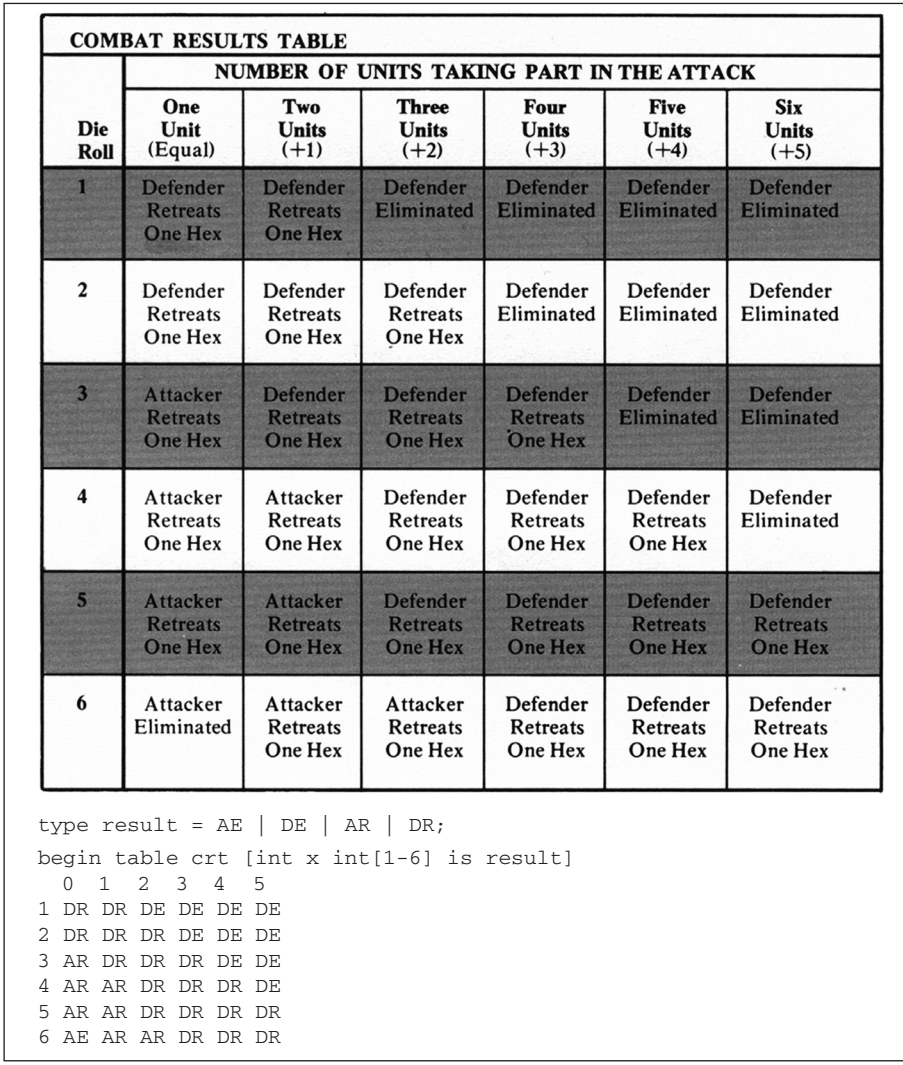

```
begin map sf1
terrain loc clear;
terrain loc town;
terrain loc woods;
clear: 0101, 0102, 0103, 0104, 0105, 0106, 0108, 0201, 0202, 0203,
0204, 0205, 0207, 0301, 0304, 0305, 0306, 0308, 0401, 0402, 0403,
0404, 0406, 0407, 0501, 0502, 0503, 0504, 0505, 0507, 0508, 0601,
0602, 0604, 0605, 0606, 0607, 0701, 0702, 0703, 0704, 0705, 0706,
0707, 0708, 0801, 0802, 0803, 0804, 0805, 0806, 0807, 0901, 0902,
0903, 0904, 0905, 0906, 0907, 0908;
town: 0107 (Ganheim), 0302 (Bergtheim), 0405 (Essleben);
woods: 0206, 0303, 0307, 0506, 0603;
end map
```

Figure 4. The hexagonal map of STRIKE FORCE ONE




```
define unit { player:player };
unit A B C D E F { player = Soviet };
unit W X Y Z { player = "U.S." };
relation location : unit <-> hex;
location : A <-> hex.0801;
location : B <-> hex.0901;
location : C <-> hex.0902;
location : D <-> hex.0806;
location : E <-> hex.0907;
location : F <-> hex.0908;
location : W <-> hex.0302;
location : X <-> hex.0403;
location : Y <-> hex.0405;
location : Z <-> hex.0207;
```

Figure 5. The units and their deployment (STRIKE FORCE ONE)



COMBAT RESULTS TABLE

Die Roll	NUMBER OF UNITS TAKING PART IN THE ATTACK					
	One Unit (Equal)	Two Units (+1)	Three Units (+2)	Four Units (+3)	Five Units (+4)	Six Units (+5)
1	Defender Retreats One Hex	Defender Retreats One Hex	Defender Eliminated	Defender Eliminated	Defender Eliminated	Defender Eliminated
2	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Eliminated	Defender Eliminated	Defender Eliminated
3	Attacker Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Eliminated	Defender Eliminated
4	Attacker Retreats One Hex	Attacker Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Eliminated
5	Attacker Retreats One Hex	Attacker Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex
6	Attacker Eliminated	Attacker Retreats One Hex	Attacker Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex	Defender Retreats One Hex

```
type result = AE | DE | AR | DR;
begin table crt [int x int[1-6] is result]
  0  1  2  3  4  5
1 DR DR DE DE DE DE
2 DR DR DR DE DE DE
3 AR DR DR DR DE DE
4 AR AR DR DR DR DE
5 AR AR DR DR DR DR
6 AE AR AR DR DR DR
```

Figure 6. The "Combat Results Table" of STRIKE FORCE ONE

"Combat Results Table," shown in Figure 6, together with the formalization of it in our system. Having a special table syntax, instead of, for instance, encoding the same information as conditional statements, or as a set of simpler statements as one presumably would if some type of predicate logic was used, makes the rule writing much easier at the comparatively slight cost of some additional work when creating the tools. The table works like a function: two input values are provided and one output value is returned.[21]

Each complete turn of **Strike Force One** proceeds strictly according to the following sequence:

Step 1. The Soviet Player moves any or all of his units, as he wishes, within the limitations of the rules of movement.

Step 2. The Soviet Player may now make attacks against any U.S. units which are in hexes directly adjacent to (next to) Soviet units. Results are applied as each attack is made.

Step 3. The U.S. Player may now move any or all of his units, as he wishes, within the limitations of the rules of movement.

Step 4. The U.S. Player may now make attacks against any Soviet units which are in hexes directly adjacent to U.S. units. Results are applied as each attack is made.

The above four steps make up the complete turn; these steps are repeated in order until four complete turns have been played. The game is then over and the winner is determined.

Figure 7. The "Sequence of Play" in STRIKE FORCE ONE

This table syntax was inspired by the similar feature in the RAND-ABEL programming language (P. D. Allen, 1987, pp. 18-20; Davis, 1990, p. 21f).

Of the game-play components in Figure 1, we have now covered the map and units plus some aspect of combat resolution and movement. What remains are defining the actions and the sequence of play, including the victory conditions. The sequence of play of STRIKE FORCE ONE (sect. 3.0 in the manual) is shown in Figure 7 (printed) and Figure 8 (formalized). The reason for defining "actions" separately instead of as part of the "sequence of play" loop is, apart from good programming practice, to make it possible for software tools, such as the graphical-user interface, to display appropriate graphics and to easily find all points in the game where user input is required.

Relations have their own syntax, using the operator "**<->.**" The commands **"assert"** and **"retract"** are used to add and delete individual relation instances (cf. Figure 8). What is mainly missing from the formalism, compared with the discussion above, is control of the information that is given to the players (cf. "fog of war," p. 4). The example game STRIKE FORCE ONE, like most board wargames, makes all the information available to all the players. This is not always a satisfactory solution, and with computational help many other options are available, but this is not discussed further here.

```
begin sop
  variable winner : player;
  variable ocs : set(unit);
  begin for [i:int, 1..4]
    begin foreach unit u:
      u.ap :=4;
      u.hat=false; u.atd=false;
    end foreach
    dialogue many {move(Soviet)}
                  {stacking};
    dialogue many {combat(Soviet)};
    dialogue many {move(US)}
                  {stacking};
    dialogue many {combat(US)};
  end for
  begin findall unit y = ocs :
    variable x : hex;
    location(y <-> x);
    member(x,{0107,0302,0405});
    y.player = Soviet;
  end findall
  begin if ( ocs > 1 ) then
    winner := soviet;
  else
    winner := US;
  end if
end sop

begin action move(player p,
                  unit u,
                  hex to)
  u.player = p;
  variable from : hex;
  u.ap > 0;
  location( u <-> from );
  distance(sf1,from,to) = 1;
  not ( zoc(u,from) & zoc(u,to) );
  not impassable(to);
  do_move(u,to);
  begin if zoc(u,to) then
    u.ap := 0;
  else
    u.ap := u.ap + 1;
  end if
end action
```

There are four turns.

Each unit starts with four movement points.
`hat` = has attacked
`atd` = (has been) attacked

The Soviet player goes first.
First movement, then combat.
The U.S. moves second.

The variable `` `ocs' `` holds the number of Soviet occupied cities.

This is executed when the user moves a unit.

The unit must have action points available; it must move exactly one hex, not directly from ZOC to ZOC (cf. note 29).

Figure 8. The formalized "Sequence of Play" (STRIKE FORCE ONE)

```
begin action combat(player p,
                    set(unit) us,
                    unit t)
  subset( us, neighbours(t) );
  begin foreach unit u in us :
    u.player = p;
    u.hat = false;
  end foreach
  t.atd=false;
  begin case crt[count(us),
                 random(1,6)]
    AE :
      begin foreach unit uu in us :
        eliminate(uu);
      end foreach
    DE : eliminate(t); advance(us,x);
    AR :
      begin foreach unit vv in us :
        retreat(vv);
      end foreach
    DR : retreat(t); advance(us,x);
  end case
  begin foreach unit v in us :
    v.hat := true;
  end foreach
  t.atd := true;
end action
```

One or more units attack a hex. They must be adjacent.

This checks that none of the units have been in combat.

The `crt' table (fig. 6) has number of attackers and die roll (D6) on the axes.
AE = attacker(s) eliminated.
DE = defender eliminated.
AR = attacker(s) retreat.
DR = defender retreat.

This sets the variables, so that these units cannot fight again.

Figure 8. (Continued)

Internal Representation

The external representation of the game-play is primarily meant as a medium of game development and secondarily as unambiguous game documentation, but the internal representation, or "game-play automaton" (Figure 3), is for machine rather than human consumption. This means that instead of conciseness and close analogy to traditional wargame rules, as with the external syntax, the major desiderata of the internal syntax are appropriate expressiveness (enough, but not too much) and simplicity.

As mentioned above, the closest analogy in the computational literature is probably the "flow diagrams" (Goldstine & von Neumann, 1963, pp. 84ff) that were popular in the early days of computing. Goldstine (1972) explains:

> The purpose of the flow diagram is to give a picture of the motion of the control organ as it moves through the memory picking up and executing the instructions it finds there. The flow diagram also shows the states of the variables at various key points in the course of the computation. (p. 267)

Formally, the flowchart is a directed graph, with nodes representing various types of operations and edges indicating the flow of control. Traditionally, four types of nodes are used: operation, alternative, start/exit, and join of control flow (Floyd, 1967, p. 23; Goldstine & von Neumann, 1963, pp. 87-89; Scott, 1967, p. 188). For the purposes of the wargaming formalism, it seems expedient to slightly modify the first two, as follows:

- *Operation node.* Has one input and one output edge. Any changes to variables happen here. Shown as a rectangle.
- *User action node.* Similar to operation node, but involves user interaction. Needs to be distinct to allow automatic processing tools (e.g., user interfaces) to find it easily.
- *Alternative node.* Encodes the control structure. If two outgoing edges, shown as diamond. If more than two, shown as oval.

In addition to these, we need entry and exit points (circles) and joins (points). Figure 9 shows the top-level[22] of the flowchart of the example game STRIKE FORCE ONE, automatically generated from the external representation, which is partially shown in Figures 4, 5, 6, and 8 and completely listed in the appendix. At any time during the playing of the game, exactly one of the nodes in this graph is "active." This node, together with the values of the variables and the instances of the relations, defines the state of the game. Playing the game means moving through the graph. At an operation node, computations are made and data values updated; at an action node, user input is received; at an alternative node, one of the available paths is selected.

It is worth noting that the game graph (Figure 9) is not merely a "visual model" of some aspects of the program code, as is used in various modern software engineering systems (see Anderson, Bankes, Davis, Hall, & Shapiro, 1993, p. 18; Davis & Anderson, 2003, p. 57). The game graph, by itself, is the complete formalization of the game-play, in the machine-friendly internal representation. The graph is (partially) shown in Figure 9 only for illustration. When playing a game the graph exists as a data structure not as a visual representation. However, having this type of data structure means, among other things, that general wargaming software can be written, to handle any game, however complex, as long as it is written in the external formalism described in this article.

Related Work and Future Directions

Existing software tools for creating wargames or military simulations fall mainly into two categories: either complete applications where certain aspects are configurable[23] or extensions of general purpose programming languages.[24] Those in the first category suffer from being limited in which aspects of the game can be modified (typically only the "scenario"

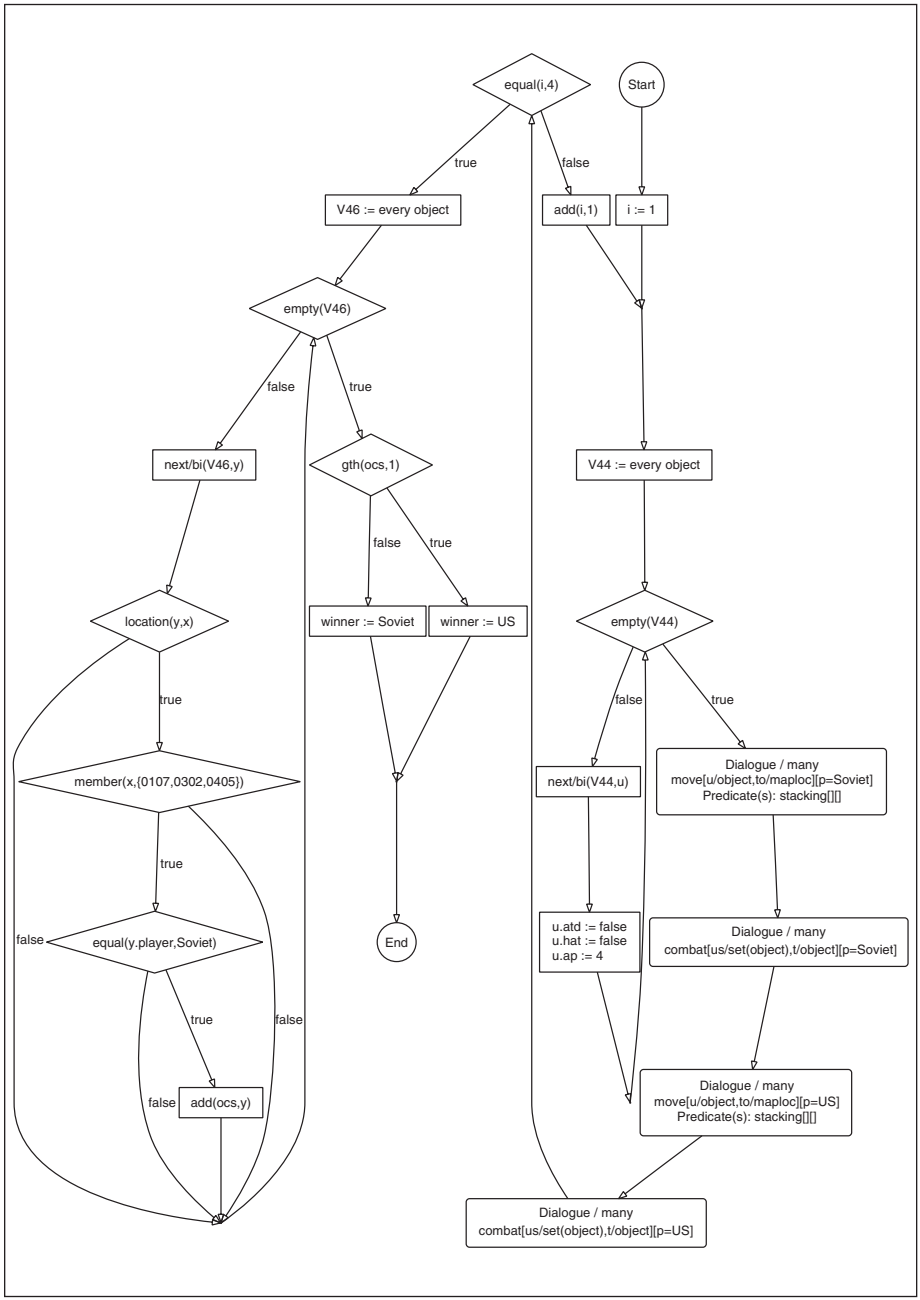


Figure 9. Auto-generated game graph (STRIKE FORCE ONE, top-level only)

part of the game-play; see Figure 1) and from being tied to the specific application used to create them. Those in the second category, programming language extensions, has none of those drawbacks but are instead too general to be used as data objects in the way described earlier for the internal formalism proposed in this article and too complicated to be used by game designers who are not also computer programmers.[25]

Game-Play in General

In the field of computer gaming generally, the problem of specifying the game-play independently of any particular implementation is receiving increasing attention. Morgan (2009, p. 705), for example, seeks "a language capable of describing interaction" but seems to suggest that existing protocols for commercial transactions over a network could be a solution, which appears unlikely in the general case. Zhu and Morgan (2008, pp. 4-5) describe another approach, more similar to the one advocated in the present article. An existing financial "rule engine" is used to implement a simple game (TIC TAC TOE). It seems improbable, though, that adopting a tool from another domain has any benefits, apart from cost, over creating a new tool specifically designed for the task at hand. Another solution, scripting, is discussed by White, Koch, Gehrke, and Demers (2008), who say that "one of the major problems with traditional scripting languages is that the programmer must be explicitly aware of low-level processing issues that have little to do with gameplay" (p. 21) and then suggest some partial solutions. It is probably fair to say that all these contributions, as well as the present article, strive toward the same distant goal, a formalism specifically designed for describing game-play and nothing else.[26]

Managing Complexity

Perhaps inevitably, both military simulations and commercial computer wargames seem to have followed a monotonic trajectory toward greater complexity. While, for example, Smith (2010, pp. 6-7) apparently considers this an undoubted improvement, more thoughtful observers have pointed out that the increasing sophistication mainly concentrates on physical factors that can be measured or counted, while ignoring or marginalizing other factors such as training, morale, and surprise, which are equally, or more, relevant (Davis & Anderson, 2003, pp. 26-27; Davis & Blumenthal, 1991, p. 8; Shubik, 2009, pp. 588, 590; cf. Sabin, 2007, pp. 19, 24-25). An interesting side note is that "hobby" board wargames have always included these "soft" factors (P. D. Allen, 1987, p. 5) to a greater extent.[27]

Another problem is that increasing complexity is likely to mean that the workings of the games or simulations are harder to understand. Related to this is that they will also be harder to create, in accordance with specifications. This issue was noted long ago[28] and it seems that it has not yet been solved to full satisfaction (Anderson et al., 1993,

pp. 11-12; Davis, 1992, pp. 14, 35; Davis & Anderson, 2003, pp. 55-56). Although these problems are probably less of a worry for commercial gaming, it could be that they contribute to the continuing popularity of table-top games, which are often less complex (see also Note 27).

By creating a formalism that is expressive enough to encode the game-play in existing table-top hobby wargames, but not more, the approach followed in this article goes some way toward solving this particular problem once and for all. Since the formalism is known to work for at least some games, it is safe to use that as a starting point, and the task has then become limited to those particular extensions that are needed for the game at hand. Using a wargame-specific formalism rather than a general programming language also eliminates many possible mistakes, simply because a much more limited set of possible instruction sequences is available.

Other Benefits of the Approach

As mentioned above, the external specification of the game-play is meant not only as input to game-playing software but also as the ultimate documentation of the rules of the game. This can then be seen as a realization of the dictum of Davis and Blumenthal (1991, pp. 14, 22) and Davis and Anderson (2003, p. 39n) that a major function of games/models is embodiment of knowledge. In the normal situation today, the game or simulation exists as a piece of software and associated documentation. Having a complete formalization in a wargame-specific formalism is better because there cannot be any discrepancy between the human-readable version and the one that is processed when executing the software, as they are the same.

A further possibility is improved computer opponents, or "wargame AI." Since the complete formalization of the game can be easily read and processed by these pieces of software, and a separate "automaton" can be instantiated with, for example, only a subset of the mapboard and units, they can use this to evaluate hypothetical situations, such as an attack or defense with more or less units than are in fact in that location on the actual game map. This kind of thing is not so easily done with the traditional type of system architecture, where the game rules are directly encoded in the software.

Software Tools for the Suggested Formalism

The external game definition format is meant for writing the game formalization. Even though this formalism is very simple and limited compared with general programming languages, the idea conflicts, to some extent, with the principle that game design and game programming be kept separate (cf. Note 25). A possible solution to this problem would be to create tools that generate game definitions from some less technical specification. For the mapboards and unit locations, a

graphical editor is the obvious choice. The rest of the game definition is mainly the actions and the sequence of play. In some cases, concepts that are fairly clear and well defined in traditional board game rules would seem to require a more complicated expression in the formalism. One example of this is "zone of control" (Berg et al., 1977, p. 177; Palmer, 1977, p. 31).[29] By cataloging such concepts and their implementation, game definition "wizards" can be created to simplify the process of encoding the game rules.

Another purpose for mechanical processing of the game definition might be to determine, for example, the maximum rates of movement and combat for various units under different circumstances. Limited examples of this type of analysis, done by hand, has appeared in the literature, but an automatic solution will be easier and more reliable.

Summary

A formal language for describing the game-play makes the game definition an entity that is separate from any particular tools, which means that games can survive through generations of software products. It also allows the game rules to be publicly examined and discussed in detail, something that is common with board wargames, but almost unheard of for computational ones. In addition, specific game situations can be described, as well as stored and reproduced, independently of any particular software implementation, and perhaps most important, it provides an accessible, unambiguous, definition of the rules of the game so that all players, human and mechanical, can have an equal, complete understanding of them.

This article proposes such a game-play definition formalism[30] and gives an example of how to use it to describe a simple board wargame. The formalism is similar to computer programming languages but limited to those features that are needed for game definitions. Some special facilities, such as simple ways to enter map and table data, are included that are not common in general programming languages.

Appendix

Complete Formal Rules for the Board Wargame STRIKE FORCE ONE

```
begin map sf1
type hexagonal vertical 10 10;
coordinates spi;
terrain loc clear;
terrain loc town;
terrain loc woods;
clear: 0101, 0102, 0103, 0104, 0105, 0106, 0108, 0201, 0202, 0203,
0204, 0205, 0207, 0301, 0304, 0305, 0306, 0308, 0401, 0402, 0403, 0404,
0406, 0407, 0501, 0502, 0503, 0504, 0505, 0507, 0508, 0601, 0602, 0604,
0605, 0606, 0607, 0701, 0702, 0703, 0704, 0705, 0706, 0707, 0708, 0801,
0802, 0803, 0804, 0805, 0806, 0807, 0901, 0902, 0903, 0904, 0905, 0906,
0907, 0908;
town: 0107 (Ganheim), 0302 (Bergtheim), 0405 (Essleben);
woods: 0206, 0303, 0307, 0506, 0603;
end map

type player = Soviet | US;
define unit { player:player, str:int, ap:int, hat:bool, atd:bool };
relation location : unit <-> hex;
unit A { player=Soviet }; location : A <-> sf1.0801;
unit B { player=Soviet }; location : B <-> sf1.0901;
unit C { player=Soviet }; location : C <-> sf1.0902;
unit D { player=Soviet }; location : D <-> sf1.0806;
unit E { player=Soviet }; location : E <-> sf1.0907;
unit F { player=Soviet }; location : F <-> sf1.0908;
unit W { player="U.S." }; location : W <-> sf1.0302;
unit X { player="U.S." }; location : X <-> sf1.0403;
unit Y { player="U.S." }; location : Y <-> sf1.0405;
unit Z { player="U.S." }; location : Z <-> sf1.0207;

begin sop
  variable winner : player;
  variable ocs : set(unit);
  begin for [i:int, 1..4]
    begin foreach unit u :
      u.ap:=4; u.hat=false; u.atd=false;
    end foreach
    dialogue many {move(Soviet)} {stacking};
    dialogue many {combat(Soviet)};
    dialogue many {move(US)} {stacking};
```

(continued)

Appendix (continued)

```
    dialogue many {combat(US)};
  end for
  begin findall unit y = ocs :
    variable x : hex;
    location(y <-> x);
    member(x,{0107,0302,0405});
    y.player=Soviet;
  end findall
  begin if ( ocs > 1 ) then
    winner := Soviet;
  else
    winner := US;
  end if
end sop

begin action move ( player p, unit u, hex to )
  u.player = p;
  variable from : hex;
  u.ap > 0;
  location ( u <-> from );
  distance(sf1,from,to) = 1;
  not ( zoc(u,from) & zoc(u,to) );
  not impassable(to);
  do_move(u,to);
  begin if zoc(u,to) then
    u.ap := 0;
  else
    u.ap =- 1;
  end if
end action

begin function stacking = bool
  variable b : bool; b := true;
  variable us : set(unit);
  begin foreach hex x :
    begin findall unit u = us :
      location(u <-> x);
    end findall
    begin if (us > 1) then b := false; end if
  end foreach
  return(b);
end function
```

(continued)

Appendix (continued)

```
type cmbt_res = AE | DE | AR | DR;

begin action combat ( player p, set(unit) us, unit t )
  location ( t <-> x );
  subset( us, neighbours(t) );
  begin foreach unit u in us : u.player = p; u.hat=false; end foreach
  t.atd=false;
  begin case crt[count(us),random(1,6)]
    AE : begin foreach unit uu in us : eliminate(uu); end foreach
    DE : eliminate(t); advance(us,x);
    AR : begin foreach unit vv in us : retreat(vv); end foreach
    DR : retreat(t); advance(us,x);
  end case
  begin foreach unit v in us : v.hat:=true; end foreach
  t.atd:=true;
end action

begin procedure retreat ( unit u )
  variable xs : set(hex); xs := empty;
  begin foreach hex x in neighbours(u) :
    begin if not ( impassable(x) | zoc(u,x) ) then
      insert(x,xs);
    end if
  end foreach
  begin if empty(xs) then
    eliminate(u);
  else
    variable xx : hex;
    dialogue select xs xx;
    do_move(u,xx);
  end if
end procedure

begin procedure advance ( set(unit) us, hex to )
  variable u : unit;
  dialogue optional us u;
  do_move(u,to);
end procedure
```

(continued)

Appendix (continued)

```
begin procedure eliminate ( unit u )
  retract ( location ( u <-> _ ) );
end procedure
begin procedure do_move ( unit u, hex to )
  location ( u <-> from );
  retract ( location ( u <-> from ) );
  assert ( location ( u <-> to ) );
end procedure

begin function zoc ( unit u, hex x ) = bool
  variable b : bool; b := false;
  variable e : player; e := enemy(u.player);
  begin foreach hex n in neighbours(x) :
    begin if ( location( u <-> n ) & u.player=e ) then
      b := true;
    end if
  end foreach
  return(b);
end function

begin function impassable ( hex x ) = bool
  return ( terrain(x) = woods );
end function
begin function enemy ( player p ) = player
  begin case c
    US : return(Soviet);
    Soviet : return(US);
  end case
end function

begin table crt [int x int[1-6] is cmbt_res]
  0  1  2  3  4  5
1 DR DR DE DE DE DE
2 DR DR DR DE DE DE
3 AR DR DR DR DE DE
4 AR AR DR DR DR DE
5 AR AR DR DR DR DR
6 AE AR AR DR DR DR
end table
```

Acknowledgement:
Paul K. Davis (RAND corporation), Philip Sabin (King's College London), and one anonymous reviewer gave helpful comments.

Declaration of Conflicting Interests

The author declared no conflicts of interest with respect to the authorship and/or publication of this article.

Funding

The author received no financial support for the research and/or authorship of this article.

Notes

1. CHESS evolved from an Indian game called CHATURANGA, a Sanskrit word for "army" (Murray, 1913, p. 42). This game dates back to at least the 7th century AD (Murray, 1913, p. 32). The game GO is first mentioned in the 6th century BC (Parlett, 1999, p. 168). The oldest board wargames for which solid evidence exists are apparently from third millenium BC Egypt (Murray, 1951, pp. 13-14, 229).
2. Young (1955, p. 6); Weiner (1959, p. 6), Thomas (1961, p. 426), McHugh (1966, pp. 1-3, 2-6), Wilson (1970, p. 17), and Hausrath (1971, p. 5).
3. Cf. "War Gaming is amazingly effective for teaching obvious ideas that people have resisted because they run counter to doctrine, or are unpleasant e.g., if one's airforce is exposed to destruction, the enemy may destroy it" (Kahn & Mann, 1957, p. 5).
4. For example, "Writing and teaching, together with wargaming, seem to have played an important part not only in the dissemination but also in the evolution of Guderian's military thought." From the foreword (p. 10) of Guderian (1992).
5. Hexagons are preferred because distances are more equal in all directions (Helmer, 1960, p. 8; McHugh, 1966, p. 4-27; Page, 1952, p. 85). According to Berg Patrick, Simonsen, Isby, and Dunnigan, (1977, p. 12), Freeman (1980, p. 15), Gush and Finch (1980, p. 30), and Perla (1990, p. 116), the hobby wargames industry copied this from the RAND Corporation sometime between 1953 and 1958. T. B. Allen (1987, p. 96) thinks it was the other way around, which seems less likely. The originator may have been George Gamow, whose game TIN SOLDIER, created in 1951 at the Operations Research Office at the Johns Hopkins University, used a hexagonal grid (Hausrath, 1971, p. 65; Page, 1952). Apparently, this game was initially played on a square grid (Gamow, 1970, p. 151).
6. For example, in the wargaming magazine *The General* (`http://www.ahgeneral.org`), published 1964-1998 (Perla, 1990, p.167). Dunnigan (2000, pp. 12-33) describes two turns of an example game, illustrated by pictures of the map with playing pieces at different points in the game. Palmer (1977) includes concrete discussions, with pictures, of several games. Sabin (2007) uses a similar approach as background to a historical discussion of ancient battles. See also Figure 2.
7. The similarity in the nature of the challenge offered by these ancient wargames and the modern ones has been noted by, for example, Palmer (1977, p. 14), Gush and Finch (1980, p. 13), and Prados (1987, p. ix).

8. According to noted military combat theorist Trevor Dupuy, quoted in T. B. Allen (1987), "the terms models, wargaming, simulation are used synonymously" (p. 64). Brewer and Shubik (1979) use the term "MSG" throughout, standing for "model, simulation, and game" (p. 3).
9. The example game STRIKE FORCE ONE, described elsewhere in this article, which is probably one of the simplest manual hobby wargames, has 26 numbered rules, taking up three pages of print. Some games published in books (Dunnigan, 2000, pp. 174-189; Freeman, 1980, pp. 50-61; Prados, 1987, pp. 65-75) are around a dozen pages. The complete rules of the game ASL (`http://www.advancedsquadleader.net`) require approximately 200 pages.
10. This term is from Carl von Clausewitz (1993): "War is the realm of uncertainty; three quarters of the factors on which action in war is based are wrapped in a fog of greater or lesser uncertainty" (p. 117).
11. Methods that have been tried include duplicate mapboards separated by a screen (Perla, 1990, p. 143; Setear, 1989, pp. 6-8) and inverted counters (Berg et al., 1977, pp. 50-51, 114-115; Freeman, 1980, p. 43; Setear, 1989, p. 9).
12. For example, "The complicated program and the large computing machine have taken on quasi-religious overtones. Offerings are put into the black box by acolytes who are never sure what is going to come out; those who come to worship are often not sure what has happened either. (Brewer & Shubik, 1979, p. 25)"
    Dunnigan (2000) puts it more succinctly: "The computer does some mumbo jumbo inside the box and gives you an answer based on who knows what" (p. 108). See also Davis and Blumenthal (1991, p. 6n) and Davis and Anderson (2003, pp. 69, 106-107).
13. "All [wargames] share one feature in common: competition. The free competition between Red and Blue exposes weak points in an argument, vague points in a plan . . . The essential element in any war game is free competition—the intelligent and obnoxious opponent." (Specht, 1957, p. 17)

    "A war game is a model of military reality set up by a judicious process of selection and aggregation, yielding the results of *the interactions of opponents with conflicting objectives* as these results are developed under more or less definite rules." (Paxson, 1963, p. 1)

    "The gaming technique, first and most fully developed in war gaming, establishes an environment that challenges and motivates a responsible participant. He must bring all past learning and his most mature judgment to bear on analyzing the situation confronting him and then employ the best possible approach in meeting that situation. Moreover, he knows that he is matched against a competent and resourceful opponent." (Hausrath, 1971, p. 11)
14. Berg et al. (1977) contains an offer of a free copy of this game, sent by mail. The producers, Simulations Publications Inc., went bankrupt in 1982. The game has been reprinted and is available from `http://victorypointgames.com/`.
15. The difficulty of analyzing the game-play of computer games generally is cited by Zagal, Rick, and Hsi (2006) as the reason to study board games instead. "Computer games represent closed systems that are highly complex as well as opaque to in-depth analysis," whereas "the nature of board games implies a transparency regarding the core mechanisms of the game and the way they are interrelated" (p. 26). Morgan (2009) points out that "realizing the ramifications on game rules even the smallest change to implementation may cause is

difficult to judge" (p. 690). This is a strong argument for the type of formalism suggested in the present article.

16. The origin of finite automata theory is traditionally considered to be McCulloch and Pitts (1943). Modern work is more directly based on Kleene (1956).
17. Flowcharts were apparently used earlier in electrical engineering and business analysis, but their first application to computer programming was by Goldstine and von Neumann (1947), more readily available as Goldstine and von Neumann (1963).
18. Also: "Rule-making brings its devotees so many fascinating problems of history, of mathematics and probabilities, definition, tabulation and clarity, that it could almost stand as a hobby on its own" (Gush & Finch, 1980, p. 19).
19. In Figure 4, **"begin map"** and **"end map"** surround the map definition, **"sf1"** is the name of the map, and the **"terrain"** statement defines the terrain types. The words **"clear"**, **"town"**, and **"woods"** are not predefined, but introduced by the terrain statement. The words within round brackets are just names of the locations and have no function in the game-play.
20. In Figure 5 the only predefined words are **"define"** and **"relation"**. The words **"unit"** and **"location"** are defined and then used in the other statements. The single-letter names are the units, which are placed on the map by the location statements. Technically, relations are declared to exist between each unit and the hex location.
21. The function is used in the **"combat"** action in Figure 8, where the meanings of the result values are defined.
22. In total, the game-play graph for STRIKE FORCE ONE has about 300 nodes.
23. For example, ADC (`http://www.hpssims.com`) and the editors that come with many computer wargames. More limited tools are also available that only display the map and units for human–human play over a network.
24. Such as the RAND-ABEL programming language (Davis, 1990).
25. Dunnigan (2000, pp. 354, 380) suggests that a board wargame should be used as prototype for a computer implementation and warns against letting the game be designed by "the programmer" (pp. 251-252). Crawford (2003) says that "game development shares nothing with game programming; they are completely separate fields of endeavor" (p. 2). Davis (1992, p. 35f) also advocates a separation between model design and programming.
26. Crawford (2003) should perhaps also be added to the list, although he is somewhat more ambitious. Talking about adventure games, or in modern parlance, "interactive storytelling," he says that "verbs don't fall into some simple pattern or system that makes it possible to put them in a simple database" so "the big parts, like level maps and weapons characteristics, are all table-driven, while the verbs, being few in number, are always handled with code." What is needed then, he thinks, is "a database manager for verbs with an embedded programming language for interactive storytelling" (pp. 166-167). Davis and Anderson (2003) say that "among the cutting-edge issues" for developing "composable systems describing modern military operations" [is creating] a "rigorous language for describing models, simulations, and many of the subtleties therein" (p. 55).

27. Davis and Anderson (2003) even say that "forward-looking warfighters have often ignored [Modeling & Simulation]-based work while using old-fashioned tabletop games to conceive and think about new concepts" (p. 59).
28. "I feel that the real limitation of the (computer) war game [i.e., what we would call "simulation"] is the difficulty of translating the intention of the model to the computer." (Thomas, 1961, p. 443; quoted in McHugh, 1966, pp. 6-28).
29. The "zone of control" rules are implemented by the function **`"zoc"`**, which is used in the **`"move"`** action (Figure 8) and the **`"retreat"`** procedure.
30. Additional information about this formalism, tentatively christened "G-code," can be found at `http://www.basun.net/software/gcode/`.

References

Allen, P. D. (1987). *The secondary land theater model* (RAND Note N-2625-NA). Santa Monica, CA: RAND Corporation.

Allen, T. B. (1987). *War games*. London, England: William Heinemann.

Anderson, R. H., Bankes, S. C., Davis, P. K., Hall, H. E., & Shapiro, N. (1993). *Toward a comprehensive environment for computer modeling, simulation, and analysis* (RAND Note N-3554-RC). Santa Monica, CA: RAND Corporation.

Berg, R., Patrick, S. B., Simonsen, R. A., Isby, D. C., & Dunnigan, J. F. (1977). *Wargame design. The history, production and use of conflict simulation games*. New York, NY: Simulations Publications.

Bowen, K. (1978). *Research games*. London, UK: Taylor & Francis.

Brewer, G. D., & Shubik, M. (1979). *The war game: A critique of military problem solving*. Cambridge, MA: Harvard University Press.

von Clausewitz, C. (1993). *On war*. London, England: David Campbell.

Crawford, C. (2003). *Chris Crawford on game design*. Indianapolis, IN: New Rider.

Darby, J. (2009). *Going to war: Creating computer war games*. Boston, MA: Course Technology.

Davis, P. K. (1990). *An analyst's primer for the RAND/ABEL programming language* (RAND Note N-3042-NA). Santa Monica, CA: RAND Corporation.

Davis, P. K., & Blumenthal, D. (1991). *The base of sand problem* (RAND Note N-3148-OSD/DARPA). Santa Monica, CA: RAND Corporation.

Davis, P. K. (1992). *Generalizing concepts and methods of verification, validation, and accreditation (VV&A) for military simulations* (RAND Report R-4249-ACQ). Santa Monica, CA: RAND Corporation.

Davis, P. K., & Anderson, R. H. (2003). *Improving the composability of department of defense models and simulations* (RAND Monograph MG-101-OSD). Santa Monica, CA: RAND Corporation.

Dunnigan, J. F. (2000). *Wargames handbook* (3rd ed.). Lincoln, NE: Writers Club Press.

Floyd, R. W. (1967). Assigning meanings to programs. In J. Schwartz (Ed.), *Proceedings of symposium on applied mathematics* (Vol. 19, pp. 19-32). Providence, RI: American Mathematical Society.

Freeman, J. (1980). *The complete book of wargames*. New York, NY: Simon & Schuster.

Gamow, G. (1970). *My world line*. New York, NY: Viking Press.
Goldstine, H. H. (1972). *The computer from Pascal to von Neumann*. Princeton, NJ: Princeton University Press.
Goldstine, H. H., & von Neumann, J. (1947). *Planning and coding problems for an electronic computing instrument* (Tech. Rep.). Princeton, NJ: Institute for Advanced Study.
Goldstine, H. H., & von Neumann, J. (1963). Planning and coding problems for an electronic computing instrument. In A. A. Taub (Ed.), *von Neumann, J., collected works* (pp. 80-151). Oxford, England: Pergamon.
Guderian, H. (1992). *Achtung—Panzer!* London, England: Arms & Armour Press.
Gush, G., & Finch, A. (1980). *A guide to wargaming*. London, England: Croom Helm.
Hausrath, A. H. (1971). *Venture simulation in war, business, and politics*. New York, NY: McGraw-Hill.
Helmer, O. (1960). *Strategic gaming* (Paper P-1902). Santa Monica, CA: RAND Corporation.
Kahn, H., & Mann, I. (1957). *War gaming* (Paper P-1167). Santa Monica, CA: RAND Corporation.
Kleene, S. C. (1956). Representation of events in nerve nets and finite automata. In C. E. Shannon & J. McCarthy (Eds.), *Automata studies* (pp. 3-41) Princeton, NJ: Princeton University Press.
McCulloch, W. S., & Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Medical Biophysics*, *5*, 115-133.
McHugh, F. J. (1966). *Fundamentals of war gaming* (Technical Report Z801686). Newport, RI: Naval War College.
Morgan, G. (2009). Challenges of online game development: A review. *Simulation & Gaming*, *40*, 688-710.
Murray, H. (1913). *A history of chess*. Oxford, England: Oxford University Press.
Murray, H. (1951). *A history of board-games other than chess*. Oxford, England: Oxford University Press.
Naur, P. (1963). Revised report on the algorithmic language Algol 60. *Computer Journal*, *5*, 349-367.
Page, T. L. (1952). A tank battle game. *Journal of the Operations Research Society of America*, *1*, 85-86.
Palmer, N. (1977). *The comprehensive guide to board wargaming*. New York, NY: McGraw-Hill.
Parlett, D. (1999). *The oxford history of board games*. Oxford, England: Oxford University Press.
Paxson, E. W. (1963). *War gaming* (Research Memorandum RM-3489-PR). Santa Monica, CA: RAND Corporation.
Perla, P. P. (1990). *The art of wargaming*. Annapolis, MD: Naval Institute Press.
Perlis, A. J. (1967). The synthesis of algorithmic systems. *Journal of the ACM*, *14*, 1-9.
Prados, J. (1987). *Pentagon games: Wargames and the American military*. New York, NY: Harper & Row.
Sabin, P. (2002). Playing at war: The modern hobby of wargaming. In T. J. Cornell & T. B. Allen (Eds.), *War and games* (pp. 193-222). Suffolk, England: Boydell Press.
Sabin, P. (2007). *Lost battles*. London, England: Hambledon Continuum.
Scott, D. S. (1967). Some definitional suggestions for automata theory. *Journal of Computer and System Sciences*, *1*, 187-212.

Setear, J. K. (1989). *Simulating the Fog of War* (Paper P-7511). Santa Monica, CA: RAND Corporation.
Shubik, M. (2009). It is not just a game! *Simulation & Gaming*, *40*, 587-601.
Smith, R. (2010). The long history of gaming in military training. *Simulation & Gaming*, *41*, 6-19.
Specht, R. D. (1957). *War games* (Paper P-1041). Santa Monica, CA: RAND Corporation.
Simonsen, R., & Dunnigan, J. (1975). *STRIKE FORCE ONE. A conflict simulation introductory game*. New York, NY: Simulations Publications.
Thomas, C. J. (1961). Military gaming. In R. L. Ackoff (Ed.), *Progress in operations research* (Vol. I, pp. 421-463). New York, NY: John Wiley.
Weiner, M. G. (1959). *An introduction to war games* (Paper P-1773). Santa Monica, CA: RAND Corporation.
White, W., Koch, C., Gehrke, J., & Demers, A. (2008). Better scripts, better games. *ACM Queue—Game Development*, *6*, 18-25.
Wilson, A. (1970). *War Gaming*. Middlesex, England: Penguin Books.
Young, J. P. (1955). *A brief history of war gaming* (Staff Memorandum 41). Fort Monroe, VA: Combat Operations Research Group.
Zagal, J. P., Rick, J., & Hsi, I. (2006). Collaborative games: Lessons learned from board games. *Simulation & Gaming*, *37*, 24-40.
Zhu, L., & Morgan, G. (2008). *Runtime evolution for online gaming: A case study using JBoss and Drools*. Paper presented at the proceedings of the Sixth International Conference on Game Design and Technology, Liverpool, England.

Bio

Tomas By is a computer science researcher with degrees from Gothenburg, Sweden and Sheffield, England.

Contact: tomas.by@in.tum.de.