# Modelling Complex Warfare: Cell-DEVS Battlefield Simulation 2 (CBS2)

*Erik Esselaar*

Department of Systems and Computer Engineering
Carleton University
1125 Colonel By Drive
Ottawa, ON, K1S 5B6, Canada
esselaar@sce.carleton.ca

**ABSTRACT**: *The Discrete Event Systems Formalism (DEVS) has been used for a myriad of different applications, ranging from prediction of natural biological and physical phenomenon to modelling complex systems such as power plants. In this paper, we describe how past efforts in modelling land warfare have been extended to include complex warfare, where battle between multiple opponents has been modelled. The approach uses Cellular Automata (CA) to model a two-dimensional battlefield, with up to four "Sides" that could be in conflict. The implementation of this model took advantage of the extensions offered in the Parallel Cell-DEVS Toolkit (PCD++), including multiple state variables for each Cell, as well as multiple input and output ports for each Cell. Simulation results and recommendations for future development are given to highlight the potential future benefit that CA offer to warfare modelling.*

**Keywords**: *DEVS, Cell-DEVS, CD++, PCD++, Cellular Automata, CA, Battle, Warfare, Defence*

## 1. Introduction

Land warfare modelling has been an area of interest of armed forces, businesses and academia worldwide, whether for training, future capabilities development, or a myriad of other applications.

Cellular Automata (CA) have a number of characteristics that lend well to the modelling of combat. Most notably, the ability to deconstruct a complex problem into manageable cells that have a limited area of influence (and hence neighbourhood of influence). The scalability of CA, and the ability to parallelize CA across multiple processors are other features that have immense potential benefit to military applications.

This work focuses on modelling land warfare with the Cell-DEVS Formalism, most notably taking advantage of the latest extensions offered in the Parallel Cell-DEVS Toolkit (PCD++, Version 3).

This paper focuses on the extensions done to R. Madhoun's work, summarized in [5], and does not evaluate other modelling methods for land warfare modelling. Since this study is interested in entity relationships, it assumes a relatively simplified battlefield, without the need to model environmental or detailed physical effects.

This work culminated in a comparison between execution of Madhoun's model, and a similar scenario in this new implementation. As expected, the new implementation performed slightly slower than the previous one. However this slight loss in speed came with greatly increased flexibility in the ability to model more complex warfare situations. The power of this model was shown in a 4-Side combat scenario, with all Sides battling for the same objective.

This paper takes a Practitioner's perspective, as in it describes in detail lessons learned in the construct of the models herein, as well as the testing, execution and analysis of them thereafter.

## 2. Background

### 2.1. Previous Work

Using Cellular Automata (CA) for modelling land warfare increased only relatively recently, most notably with the work documented by Ilachinski [1] whose CA model, ISAAC, can be said to have paved the way for future land warfare models using CA. Other nations took note of Ilachinski's work, and evaluated CA for their own applications as well [2].

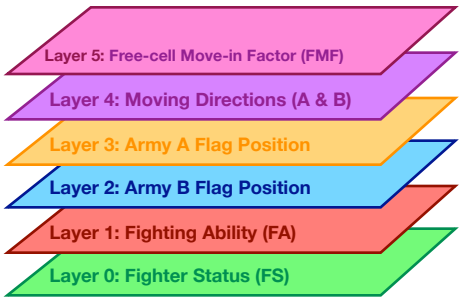


***Figure 1*** *- Previous Implementation (adapted from [5])*

Using work done to detail land warfare models in [3], Madhoun modelled land warfare using the Cell-DEVS

Toolkit (CD++) [5]. This was done with a 6-layer, 3-dimensional Cell-DEVS model, illustrated in Figure 1.

Madhoun took this work, and optimized it using the extensions offered by the new Parallel Cell-DEVS toolkit (PCD++, Version 3) [5]. Of note, he leveraged multiple state variables (denoted in Figure 2 within each cell, preceded by the '$' sign) and multiple ports for each cell (denoted below by the italicized arrows).

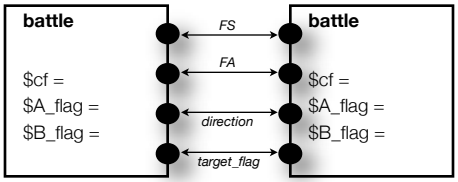


***Figure 2*** *- Previous "Advanced" Implementation (adapted from [5])*

He was able to reduce the problem in complexity significantly, optimizing a 6-layer/zone, 3-D Cell-DEVS model into a 1-layer/zone, 2-D Cell-DEVS model, capitalizing on the power of multiple ports and state variables.

His performance comparison between the "new" and "old" implementations showed that in this case, the Cell-DEVS extensions significantly reduced simulation execution time.

Madhoun's new work also offered improvements to the model, including an expanded battlefield neighbourhood into a full 9-cell Moore's Neighbourhood. This meant that movement could be beyond simple North-South-East-West movement into four adjacent cells, but rather movement into all surrounding 8 cells. Additionally, Madhoun implemented an obstacle avoidance algorithm for movement of entities.

These algorithms were retained in this implementation, with some modification to incorporate the improvements done herein. More details on this implementation are discussed in Section 3.7 below.

### 2.2. Motivation

This work was motivated by modern warfare concepts, whereby there are not traditionally only two enemies in combat with one-another - there are multiple "Sides" in a given conflict.

Inspiration was drawn from actual operations, and the nature of modern warfare whereby there is rarely a case of just one Side versus another, both outright "enemies". The notion of having multiple "Sides" in operations exists, where some Sides will either be *Friendly*, *Hostile* or *Neutral* towards another Side. Additionally, while one side might feel one way to another, the "feeling may not be mutual".

Furthermore, this project sought out to generalize some of concepts including seeking out multiple objective types (not just a single *Flag*), the notion of health, and healing.

Having the ability to represent these dynamics are well within the capabilities of the Cell-DEVS formalism, and were the focus of this work.

## 3. Models Defined

### 3.1. Model Overview

The model was named "Cell-DEVS Battlefield Simulation 2" or CBS2. This is a tribute to the pervasive Virtual Battle Space system in use by the Canadian Forces, US DoD, and elsewhere worldwide, named "VBS2", for training, concept development and many other military applications. The '2' also signifies that the model is a two-dimensional Cell-DEVS model.

The model consists of a two-dimensional 10 x 10 Cell-DEVS space. Of note, the model was implemented with PCD++, using multiple ports and multiple state variables per cell.

This model can simulate anywhere from 0 to 4 Sides who may (or may not) be hostile with one another. There are four types of entities: *Combatants*, *Bases*, *Objectives* and *Obstacles*. *Combatants* are given a mission or orders to pursue either a *Base* or an *Objective*, which may be the same as another Side. *Bases* can be defendable, with their own "fighting ability" or health, whereas *Objectives* can be taken without challenge (similar to *Flags* in previous implementations).

The sections below will give details on how this was implemented with PCD++. The next section will guide you through the general approach used in implementing this system.

### 3.2. Implementation Approach

Much focus was put onto not having to modify the model description file (.ma file), where changes to various scenarios could be fed via the separate initial cell value and state variable files (.val and .var respectively). As such, the design of this model imposed a slightly higher degree of attention required to the format of the .val and .var files, in order for the simulation to execute correctly.

In general, this development effort attempts to generalize the work in [5], so that much more complex

conflicts can be simulated, generating more interesting results.

As such, there is no longer a notion of *Flags* or *Army A* and *Army B*, but rather *Objectives* and *Bases*, as well as *Sides*.

Entities in the simulation are now given initial missions / orders through the separate initial values and state variables files (.val and .var respectively).

The final implementation resulted after some significant trials that were later abandoned. The next section quickly describes these trials, which may be a source of future work and development.

#### 3.2.1. Implementation Trials

Initially it was conceived that an entire relationships table could be fed into the model dynamically (ie. a different one, if desired) at the beginning of simulation execution each time, and stored within a special layer/ zone.

The following Table 1, derived from [6] illustrates four different Sides, each having a relationship with another Side, either *Friendly*, *Hostile* or *Neutral*. The relationship with one's own Side will always be *Friendly*. Of note, the complexity of having one side being hostile to another, where this is not reciprocated has been seen in operations, but has been inadequately modelled in other simulations. In the table below, Side 2 is *Neutral* towards Side 3, however Side 3 is *Hostile* towards Side 2, giving them to opportunity to "catch" Side 2 unaware.

| | Side 1 | Side 2 | Side 3 | Side 4 |
|---|---|---|---|---|
| Side 1 | 1 Friendly | 1 Friendly | -1 Hostile | 1 Friendly |
| Side 2 | 1 Friendly | 1 Friendly | 0 Neutral | 1 Friendly |
| Side 3 | -1 Hostile | -1 Hostile | 1 Friendly | 0 Neutral |
| Side 4 | 1 Friendly | -1 Hostile | -1 Hostile | 1 Friendly |

***Table 1** - Side-Side Relationships - adapted from [6] (trial approach not fully implemented)*

To represent these relationships, the above coding schema was developed (ie. 1 = Friendly, -1 = Hostile and 0 = Neutral), and values were to be stored in a separate layer/zone in the model, within a third dimension, and layer in Cell-DEVS.

This entire schema was abandoned in favour of a more simplistic approach, due to the requirement to significantly expand each cell's neighbourhood in order to connect a cell with the relationships data.

Since the rudimentary movement rules for this model took into account only enemy entities, and not collaboration with friendlies, or interaction with neutral forces, a method for tracking relationships in each cell was devised, as discussed below.

Before the rules and PCD++ code is described in detail, a formal model specification is given.

### 3.3. Model Formal Specification

This model is formally specified using the Cell-DEVS formalism, as described in [4] and [7].

#### 3.3.1. Atomic Model Specification

López describes the expanded Cell-DEVS Formalism in his thesis [4], showing how an atomic cell is constructed with multiple ports and multiple state variables. The multiple external input and output ports (listed 1..k and 1..m in Figure 3, below) as well as neighbourhood input and output ports with the *n* neighbours is graphically depicted in Figure 3 as well.

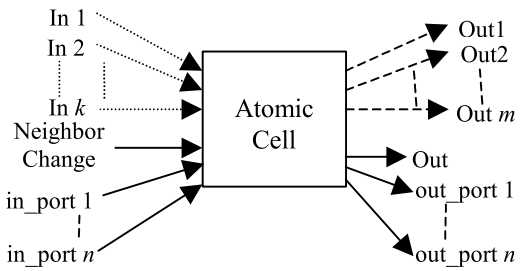


***Figure 3** - Structure of an Atomic Cell (from [4])*

As will be shown, there are no external input or output ports, just neighbourhood ports which are used in this model.

Formally, our atomic cells are described as follows $M_{cbs}$ =

$<X, Y, I, S, \theta, E, delay, d, \delta_{int}, \delta_{ext}, \tau, \lambda, D>$, where

$X = \{\varnothing\}$;

$Y = \{\varnothing\}$;

$I = <p^x, p^y>$, where

$p^x = \{p_l^{Yk} \mid l \in \{fs, fa, cf, obj, direction, enA, enB, enC \wedge k \in [1..9]\}$;

$p^y = \{p_l^{Yk} \mid l \in \{fs, fa, cf, obj, direction, enA, enB, enC \wedge k \in [1..9]\}$;

$S$ = {0, 1, 2, 3, 4, 10, 20, 30, 40, 50, 69, 70, 71};

$\theta$ = *{s, phase, σqueue,σ, init, initObj, initBase, initEnA, initEnB, initEnC}, w*here

*s* ∈ *S (as above)*

*phase* ∈ *{active, passive}*

$\sigma queue = \{(v_1, \sigma_1), \ldots, (v_m, \sigma_m) \mid m \in N, m < \infty \wedge \forall (i \in N, i \in [1,m]), v_i \in S \wedge \sigma_i \in R_0^+ \cup \infty$

$\sigma \in R_0^+ \cup \infty$

*init* ∈ *{0,1}*

*initObj* ∈ *C (described below)*

*initBase* ∈ *C (described below)*

*initEnA* ∈ *{0..4}*

*initEnB* ∈ *{0..4}*

*initEnC* ∈ *{0..4}*

$E$ = {∅};

*delay* = transport;

$d$ = 100 (milliseconds);

$D = \theta \times N \times d \rightarrow R_0^+ \cup \infty$;

The various functions ($\delta_{int}$, $\delta_{ext}$, $\tau$, $\lambda$) are described in the sections below, as well as the model definition files (*.ma and rules.inc). Please see below and in those files for more details.

### 3.3.2. Coupled Model Specification

The coupled Cell-DEVS Model is described as $M_{cbs2}$ = <*Xlist, Ylist, I, X, Y, n, {t1..tn}, N, C, B, Z, select*>,

where

*Xlist* = {∅};

*Ylist* = {∅};

$I$ = $\langle \eta, \mu^x, \mu^y, p^x, p^y \rangle$, where

$\eta = 9$

$\mu^x$ = *external input ports* = {∅}

$\mu^y$ = *external output ports* = {∅}

$p^x = \mu^x$ + *neighbour ports* = $\{p_{i,j}^{Xk} \mid i \in \{0..9\}, j \in \{0..9\}, k \in \{1..9\}\}$;

$p^y = \mu^y$ + *neighbour ports* = $\{p_{i,j}^{Yk} \mid i \in \{0..9\}, j \in \{0..9\}, k \in \{1..9\}\}$;

$X$ = {∅};

$Y$ = {∅};

$n$ = 2;

$t_1$ = 10;

$t_2$ = 10;

$N$ = {(-1,-1), (-1,0), (-1,1), (0,-1), ***(0,0)***, (0,1), (1,-1), (1,0), (1,1)};

$C$ = $\{C_{ij} \mid i \in \{0..9\}, j \in \{0..9\}\}$;

$B$ = {∅}; (wrapped)

$Z$ =

| | |
|---|---|
| $p_{i,j}^{Y1} \rightarrow p_{i-1,j-1}^{X1}$ | $p_{i,j}^{X1} \leftarrow p_{i+1,j+1}^{Y1}$ |
| $p_{i,j}^{Y2} \rightarrow p_{i-1,j}^{X2}$ | $p_{i,j}^{X2} \leftarrow p_{i+1,j}^{Y2}$ |
| $p_{i,j}^{Y3} \rightarrow p_{i-1,j+1}^{X3}$ | $p_{i,j}^{X3} \leftarrow p_{i+1,j-1}^{Y3}$ |
| $p_{i,j}^{Y4} \rightarrow p_{i,j-1}^{X4}$ | $p_{i,j}^{X4} \leftarrow p_{i,j+1}^{Y4}$ |
| $p_{i,j}^{Y5} \rightarrow p_{i,j}^{X5}$ | $p_{i,j}^{X5} \leftarrow p_{i,j}^{Y5}$ |
| $p_{i,j}^{Y6} \rightarrow p_{i,j+1}^{X6}$ | $p_{i,j}^{X6} \leftarrow p_{i,j-1}^{Y6}$ |
| $p_{i,j}^{Y7} \rightarrow p_{i+1,j-1}^{X7}$ | $p_{i,j}^{X7} \leftarrow p_{i-1,j+1}^{Y7}$ |
| $p_{i,j}^{Y8} \rightarrow p_{i+1,j}^{X8}$ | $p_{i,j}^{X8} \leftarrow p_{i-1,j}^{Y8}$ |
| $p_{i,j}^{Y9} \rightarrow p_{i+1,j+1}^{X9}$ | $p_{i,j}^{X9} \leftarrow p_{i-1,j-1}^{Y9}$ |

*select = {(-1,-1), (-1,0), (-1,1), (0,-1),* ***(0,0)****, (0,1), (1,-1), (1,0), (1,1)}.*

## 3.4. Development Environment

The simulation was executed on a desktop standalone workstation, using Linux Fedora Core 8. The simulation engine was the Parallel Cell-DEVS Toolkit (PCD++ Version 3.0, March 2003).

### 3.5. System Model Sketch

The following is a sketch of how each cell has been constructed:

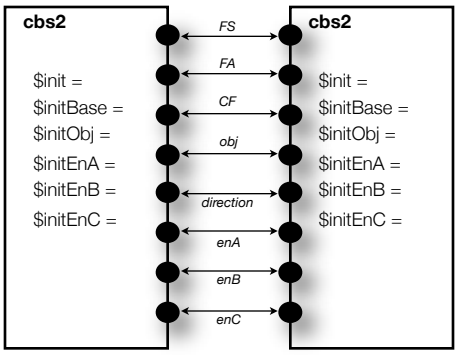


***Figure 4*** *- Implemented Cell Architecture*

As before, the cell state variables are preceded by a dollar sign ($) within each box, and the inter-cell neighbourhood ports are depicted with arrows.

### 3.6. Implementation Details

The following section describes in detail how each aspect has been implemented.

#### 3.6.1. Status and Health

Health converted into a whole-number representing a percentage, ie. ~fa = 100 means 100% health or fighting ability.

A combatant regains health slower than they are damaged. They lose 10 health / fighting ability (~fa) points per superior enemy per time interval, whereas they only heal 5 health points per 100 millisecond interval.

As will be shown later, if their ~fa drops below 0.5, they are considered “combat ineffective” and will not be considered as capable to injure an opponent in their neighbourhood, until their ~fa increases above that value.

Bases also have a notion of health; They are not mere “undefended flags”. They all start with a health or fighting ability (~fa) of 100. However Objectives can be taken without challenge.

The Fighter Status (FS) is communicated through a NeighborPort (~fs). There are four types of entities: they are Combatants, Bases, Obstacles, and Objectives.

Their values are read into the cells, along with any value of health / fighting ability, and then communicated through their NeighborPorts accordingly (as shown in the initialization section, below).

Table 2 gives details on how various values can represent various entities, and their health.

| Value | Meaning |
|---|---|
| 0 | Empty Cell (or dead combatant) |
| 1.99 | Combatant, Side 1, 100% Health |
| 2.99 | Combatant, Side 2, 100% Health |
| 3.99 | Combatant, Side 3, 100% Health |
| 4.99 | Combatant, Side 4, 100% Health |
| 1.85 | Combatant, Side 1, 86% Health |
| 10.99 | Side 1 Base, 100% Fighting Ability |
| 20.99 | Side 2 Base, 100% Fighting Ability |
| 30.99 | Side 3 Base, 100% Fighting Ability |
| 40.99 | Side 4 Base, 100% Fighting Ability |
| 50 | Obstacle |
| 69 | Objective, Utility |
| 70 | Objective, High Feature |
| 71 | Objective, Mobility (ex. bridge) |

***Table 2*** *- Sample initial status and health values read in as initial cell values*

#### 3.6.2. Sides and Enemies

There are up to four possible Sides that could be in conflict with one-another. It is impossible to be in conflict / combat one’s own Side (however accidental fratricide is a large area of interest in the military domain, but is not modelled here).

All cells have their values set to 0 initially, while entities, including Bases, have initial fighter status (~fs, akin to which Side / Entity they are) as well as health or fighting ability (~fa) loaded through an initial values file (*.val). This is shown in Figure 5, below.

Since there are at most four sides, there could be at most three enemies that an entity could be in conflict with. Their enemies are inputted into a .var file, and loaded firstly into the initial enemy variables ($initEnA, $initEnB and $initEnC), and then

transferred to their ports, so they could be "carried" with them through movement in the battlefield (~enA, ~enB and ~enC respectively).

```
[top]
components : cbs2_2

[cbs2_2]
type : cell
dim : (10,10)
delay : transport
defaultDelayTime : 100
border : wrapped

neighbors : cbs(-1,-1) cbs(-1,0) cbs(-1,1)
neighbors : cbs(0,-1)  cbs(0,0)  cbs(0,1)
neighbors : cbs(1,-1)  cbs(1,0)  cbs(1,1)

InitialValue : 0
InitialCellsValue : cbs2_2.val
LocalTransition : battlefield

neighborports : fs fa cf obj direction enA enB
enC

StateVariables : init initBase initObj initEnA
initEnB initEnC
StateValues : 0 0.00 0.00 0 0 0
InitialVariablesValue : cbs2_2.var
```

***Figure 5*** *- PCD++ Model Description*

For example, in Figure 6, we can see that there is a Side 3 Base, with 100% fighting ability / health in cell (1,1). Similarly, in cell (8,8) there is an Objective, in this case a High Feature.

Of note, the different types of high features were not given any specifically different characteristics in this implementation.

```
(1,1) = 30.99
(3,3) = 3.99
(6,6) = 1.99
(6,8) = 10.99
(8,8) = 70
```

***Figure 6*** *- Sample .val InitialCellsValue file*

Figure 7 demonstrates on how this would look in a grid-type layout, based on a *drawlog* output.

```
Line : 1 - Time: 00:00:00:000
     0 1 2 3 4 5 6 7 8 9
   +--------------------+
  0|                    |
  1|  30                |
  2|                    |
  3|       3            |
  4|                    |
  5|                    |
  6|             1  10  |
  7|                    |
  8|                70  |
  9|                    |
   +--------------------+
```

***Figure 7*** *- Tactical Layout of Entities*

Mission parameters, including which entities are hostile or "enemy" to a given Side, as well as their own base location and their assigned mission objective, are loaded from a InitialVariablesValue file. (*.var). Figure 5 above details how this was done.

For example, in Figure 8, we see that the Side 3 base at cell (1,1) has no need to note its own objective nor base location, but it must be aware of which enemies could hurt it. The $init variable is set to 1, to run through the initialization, and it has its $initEnA variable set to Side 1, and $initEnB variable set to Side 2.

This process is done for all cells/entities with a non-zero initial value.

```
(1,1) = 1 0.00 0.00 1 2 0
(3,3) = 1 8.08 1.01 1 2 0
(6,6) = 1 6.08 8.08 3 0 0
(6,8) = 1 0.00 0.00 3 0 0
(8,8) = 1 0.00 0.00 0 0 0
```

***Figure 8*** *- Sample .var InitialVariablesValue file*

### 3.6.3. Objectives

Instead of *Flag* locations (which are effectively *Objectives* for the given sides) the new model stores *Objectives*. This distinction is important in that now, more than one side can be striving for the same *Objective*.

These are stored using the same "trick" of storing the row as the integer, and the column as a decimal value, (column divided by 100). That is, in Figure 8 above, cell (3,3)'s assigned objective is in row 8, column 8 - cell (8,8). Objectives (and bases) are not assigned their own $initObj or $initBase values, and they do not move. Of note, Objectives are not given enemy values, since anyone can take them without challenge.

Additionally, Objectives do not have any health / fighting ability value (see Figure 6, cell (8,8)).

Of note, combatants can be given their own "mission" or objective to attack, a capability that should not be understated.

#### 3.6.4. Bases

Combatants and Bases have similar rules for battle; except Bases typically have more defensibility (half the damage incurred by combat-ready / superior enemies).

Bases also track whom their enemy is, so that they can calculate damage when enemy forces are in their midst. Bases, of course, cannot move.

#### 3.6.5. Courage Factor

In the previous implementation, Courage Factor (CF) was a cell state variable, and was re-calculated every time. When "movement" occurred, all that was transferred was fighting ability and fighter status (~fa and ~fs respectively), and the calculated courage value was discarded and re-calculated at each move.

However, in this implementation it was calculated once in the initialization phase, and then "carried" within it through movement. As such, in this implementation, it is represented as a port, ~cf.

This accurately shows that courage is not a random factor that gets set in each engagement, but is rather a function of an individual.

Of note, Bases are assigned a ~cf of 1.

#### 3.6.6. Initialization

There were a number of initialization steps that had to be executed (as with any simulation or program). So as to prevent unnecessary re-calculation of these rules, an initialization state variable ('$init' in Figure 10 below) was deemed necessary to track cell's progress in their initialization. Once initialization steps were complete, this variable was set to 0.

```
% read in fs and fa from values assigned from
InitialCellValues file; for combatants, set ~cf
to normalized value;
rule : { ~fs := trunc((0,0)~fs) ; ~direction :=
0 ; ~fa := (round(fractional((0,0)~fa) * 100)
+1) ; ~cf := normal(0.5,0.15) ; ~obj := 0 ;
~enA := 0 ; ~enB := 0 ; ~enC := 0 ; } 0 { frac-
tional((0,0)~fs) != 0 and trunc((0,0)~fs) <= 4
and $init = 1 }

% read in fs and fa from values assigned from
InitialCellValues file; for obstacles / objec-
tives
rule : { ~fs := trunc((0,0)~fs) ; ~direction :=
0 ; ~fa := (round(fractional((0,0)~fa) * 100)
+1) ; ~cf := 1 ; ~obj := 0 ; ~enA := 0 ; ~enB
:= 0 ; ~enC := 0 ; } 0 {  trunc((0,0)~fs) <= 71
and trunc((0,0)~fs) >= 50 and $init = 1 }

% read in fs and fa from values assigned from
InitialCellValues file; for bases
rule : { ~fs := trunc((0,0)~fs) ; ~direction :=
0 ; ~fa := (round(fractional((0,0)~fa) * 100)
+1) ; ~cf := 1 ; ~obj := 0 ; ~enA := 0 ; ~enB
:= 0 ; ~enC := 0 ; } 0 {  trunc((0,0)~fs) <= 40
and trunc((0,0)~fs) >= 10 and $init = 1 }
```

***Figure 10*** *- Initialization Rules for all potential entities*

For example, after executing these initialization rules, the combatant at cell (3,3) from Figure 7 has a health factor / fighting ability of 100 sent out over its ~fa port, a fighter status of 3 sent out over its ~fs port, and a normally distributed courage factor, sent out over its ~cf port.

#### 3.6.7. The Rule of Combat

The way to determine who wins, who loses in battle has changed to a degree. Instead of it being solely dependent on a random assignment of health at the beginning, depending on the status (ie. whether injured or not), each entity will have a health/fighting ability assigned to it through the InitialVariablesValue file, which will decrease with each "superior" enemy that it encounters (however, fighting ability can increase in time, if it is allowed to "heal").

An enemy is "superior" if its fighting ability (~fa) meets or exceeds it's opponent's. Additionally, if an entity's ~fa is below 0.5, then it is rendered "combat ineffective" in that it will not be counted as capable of injuring an opponent in its neighbourhood.

The following code excerpt shows how winners / losers were determined for combatants (example here just for *Side* 1 - other *Sides* have similar code):

```
rule : { (0,0) } { $initEnA := (((#macro(com-
bat_rule_2)) + (#macro(combat_rule_3)) + (#mac-
ro(combat_rule_4))) * 10) ; } 0 { (0,0)~fs = 1
and $initEnA = 0 }

rule : { ~fs := (0,0)~fs ; ~fa :=
(trunc((0,0)~fa) - $initEnA)  ; ~direction := 0
; } 100 { (0,0)~fs = 1 and (0,0)~fa > $initEnA}

rule : { ~fs := 0 ; ~fa := 0 ; ~direction := 0
;} 100 { (0,0)~fs = 1 and (0,0)~fa <= $initEnA}
```

***Figure 11*** *- Combat Rule*

There were complex concatenated macros, as seen above to calculate whom within a neighbourhood were an enemy. For this reason, the value was calculated and stored once in the $initEnA variable, so as to prevent unnecessary calculation of it repeatedly.

Macros were used to count the number of entities of a given enemy side are in a given cell's neighbourhood. Of note, only enemies, with a fighting ability (health) above 0.5 are considered as "effective".

```
#BeginMacro(combat_rule_1)
(
if ( ((0,0)~enA = 1 or (0,0)~enB = 1 or (0,0)~enC =
1), (
  if ( ((-1,-1)~fs = 1 and (-1,-1)~fa > 0.5 and
(-1,-1)~fa >= (0,0)~fa), 1, 0) +
  if ( ((-1,0)~fs = 1 and (-1,0)~fa > 0.5 and
(-1,0)~fa >= (0,0)~fa), 1, 0) +
  if ( ((-1,1)~fs = 1 and (-1,1)~fa > 0.5 and
(-1,1)~fa >= (0,0)~fa), 1, 0) +
  if ( ((0,-1)~fs = 1 and (0,-1)~fa > 0.5 and (0,-
1)~fa >= (0,0)~fa), 1, 0) +
  if ( ((0,1)~fs = 1 and (0,1)~fa > 0.5 and (0,1)~fa
>= (0,0)~fa), 1, 0) +
  if ( ((1,-1)~fs = 1 and (1,-1)~fa > 0.5 and (1,-
1)~fa >= (0,0)~fa), 1, 0) +
  if ( ((1,0)~fs = 1 and (1,0)~fa > 0.5 and (1,0)~fa
>= (0,0)~fa), 1, 0) +
  if ( ((1,1)~fs = 1 and (1,1)~fa > 0.5 and (1,1)~fa
>= (0,0)~fa), 1, 0)), 0)
)
#EndMacro
```

***Figure 12*** *- Combat Rule Macro looking for Side 1 as an Enemy*

### 3.7. Re-used Code

The following implementation details were retained from Madhoun's previous work [5], with some slight alteration to allow for the modifications discussed above.

#### 3.7.1. Neighbourhood

A "Moore's Neighbourhood", of size 9 cells, was used, as shown in Figure 13 below. Relative cell references are shown as well.

| | | |
|---|---|---|
| (-1, -1) | (-1,0) | (-1,1) |
| (0, -1) | (0,0) | (0,1) |
| (1, -1) | (1,0) | (1,1) |

***Figure 13*** *- Moore's Neighbourhood and Relative Cell References*

#### 3.7.2. Directions and Movement

The Moore's neighbourhood was translated into an arbitrary numbering scheme as denoted below. In the rules for movement, these numbers are mapped to cardinal directions (North, East, South, West, etc.).

Madhoun's previous work [5] gives explicit detail on how this movement scheme works.

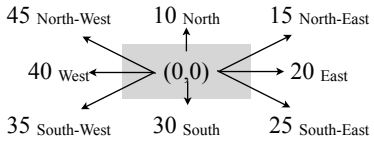


***Figure 14*** *- Movement Directions Numbering*

Of note, the physical movement of an entity from one cell to another was adjudicated by assigning free-cell move-in factors to moving and receiving cells, and then, if a move is permitted, by assigning the port values of ~direction accordingly with +/- values depending on if the cell is receiving movement (free) or is moving.

Obstacles, and the ability to avoid them, was a key feature that was retained from Madhoun's implementation.

The following section will now discuss the testing strategy for simulation execution, as well as simulation scenarios conducted.

## 4. Simulation Results

### 4.1. Testing Approach

This project used an incremental, and sequential testing approach. Firstly, the initialization rules and inputting of initial cell and variable values were tested. Within this, Objective assignment and movement of a single entity to a single objective, unopposed was tested.

Confirmation of movement and obstacle avoidance rules that were ported from the previous implementation were also tested.

Testing then focused on the Combat Rule. This began with the simple scenario of an entity surrounded by 8 enemies, with 100% health. The expected results that the enemy would be killed in two iterations, did occur.

The ability for bases to resist battle, and re-generate was then tested. Fighting Ability regeneration / healing for combatants and bases was also tested.

It was confirmed that if a combatant's Courage Factor was low, that it would be drawn to its own base. Furthermore, when an entity's health was low, it was confirmed that it would not move and would be static.

Lastly, the experimental scenarios described below were tested.

### 4.2. Simulation Run Script

In order to execute the simulation, a Linux script was adapted from [5]. The script is detailed in Figure 15.

Of note, the DEBUG options of -p parser and -v rule evaluation were powerful simulation options that enabled detailed dissection of problems.

The GNU plugin 'time' was not used for this implementation, however it is an area for future work to determine the computational resources consumed by the simulation, and compare to other implementations.

```
#!/bin/sh
#
SIM=cbs2_3

SIMU_DIR=/home/erik/cd++

SIMU=$SIMU_DIR/cd++
DRAWLOG=$SIMU_DIR/drawlog
#LOGBUFFER=$SIMU_DIR/logbuffer


MA=$SIM.ma
#EV=$SIM.ev
LOG=./logs1/$SIM.log
DRW=$SIM.drw
#OUT=$SIM.out
#LOG_FULL=$LOG.full
TIME=00:00:05:000
STEP=00:00:00:100

#DEBUG="-p$SIM.parser -v$SIM.eval"

rm -f $LOG* $DRW $OUT

SIMU_ARGS="-m$MA"
[ -n "$TIME" ]  &&  SIMU_ARGS="$SIMU_ARGS -t$TIME"
[ -n "$EV" ]    &&  SIMU_ARGS="$SIMU_ARGS -e$EV"
[ -n "$LOG" ]   &&  SIMU_ARGS="$SIMU_ARGS -l$LOG"
[ -n "$OUT" ]   &&  SIMU_ARGS="$SIMU_ARGS -o$OUT"
[ -n "$DEBUG" ] &&  SIMU_ARGS="$SIMU_ARGS $DEBUG"


if [ -z "$DEBUGGER" ]; then

    time $SIMU $SIMU_ARGS

else
    $DEBUGGER $SIMU --pargs $SIMU_ARGS
fi

echo Running drawlog
$DRAWLOG -m$MA -c$SIM -l$LOG -i$STEP -0 -w2 -p0
-nfs > $DRW

vi $DRW
#echo Collapsing log files
#cat $LOG?* | $LOGBUFFER > $LOG_FULL
```

***Figure 15*** - *Simulation Run Script*

### 4.3. Scenario 1 - Compete for Same Objective

The first Scenario used to demonstrate the power of this model is one where two opposing entities fight for the same objective. In this case, the Side 1 combatant is stronger (~fa at full 100) than the Side 2 combatant, who starts with only 60 health.

As expected, they first "battle it out" and after Side 1 wins, they move in to take the objective.

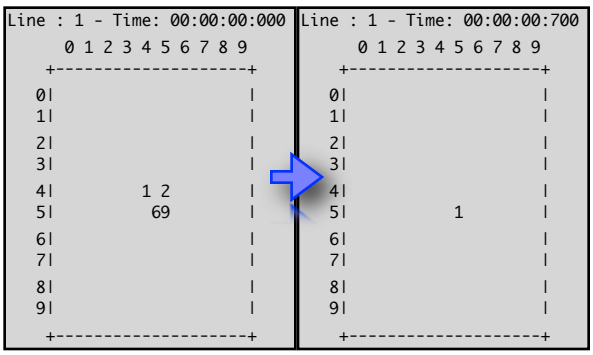


***Figure 16*** *- Scenario 1 Start and End States*

### 4.4. Scenario 2 - "Ganging up on the Little Guy"

The scenario was tested where a single entity would be surrounded by different enemies. In this case, a Side 3 combatant is surrounded by its enemies from Side 2 and Side 1. As expected, in three turns, the Side 3 entity is eliminated.

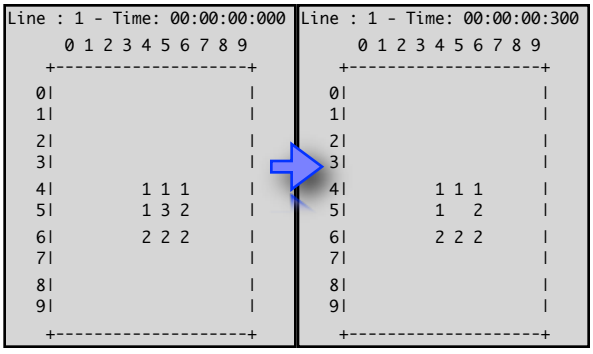


***Figure 17*** *- Scenario 2 Start and End States*

### 4.5. Scenario 3 - Battle Royale

A full-up multi-sided conflict, competing for two different objectives was set up. As expected, Sides 1 and 4 won out over Sides 2 and 3 in pursuit of their assigned objectives.

We see that right off, the objectives 69 and 71 at (3,3) and (6,6) respectively are "taken" by the neighbouring combatants.

This magnificent battle shows examples of courage failing certain combatants. We see certain combatants from Sides 1, 2, 4 retreating to their designated "bases" or rendez-vous points at (0,0), (5,5) and (9,9) respectively.

As expected, once combatants are wounded to ~fa < 0.5, they are not "combat effective" and they cannot move.

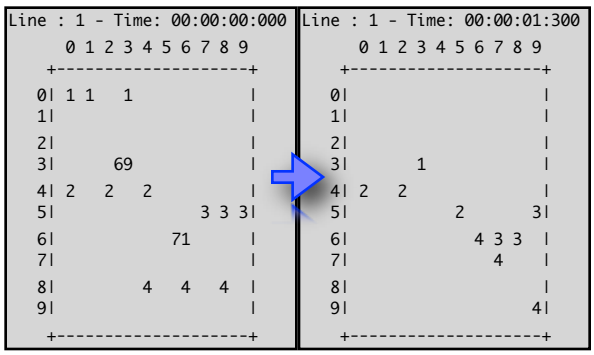


***Figure 18*** *- Scenario 3 Start and End States*

### 4.6. Cross-Implementation Performance Comparison

Performance comparison between old model and newly implemented one with two Sides, both enemies to one-another, both going for opposite objectives. Figure 19 shows the Start and End states for Madhoun's ***battle*** model, while Figure 20 shows the Start and End states for the ***cbs2*** model.

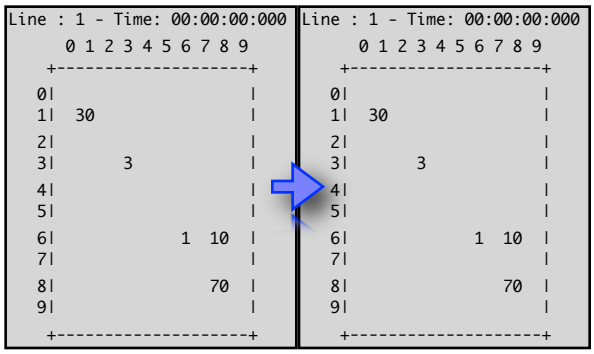


***Figure 19*** *- Start and End States for battle model*

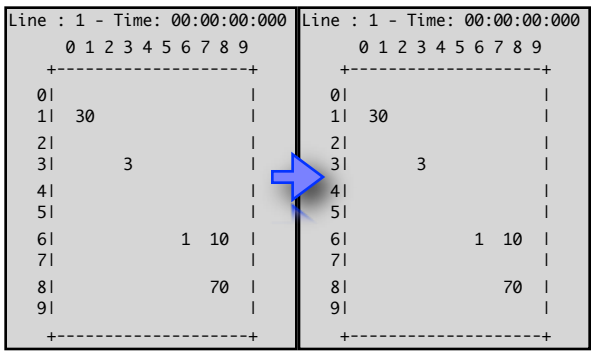


***Figure 20*** *- Start and End States for cbs2 model*

When compared to execution run times using Madhoun's original model, it was found that it performed slightly slower than Madhoun's model. This was expected due to the number of additional ports

(and messages) as well as state variables that were used.

The following chart shows a performance comparison in runtime execution between Madhoun's original model, and the newly expanded model.

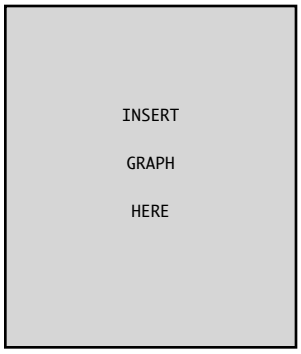


***Figure 21*** *- Performance Comparison between Madhoun's battle and cbs2*

## 5. Conclusion

The Cell-DEVS Formalism has immense potential in modelling modern warfare. This body of work focused on the modelling of complex land warfare, where up to four different Sides could be in combat with one-another.

This "more complex" warfare is more representative of actual operations, where various factions can be in combat or collaboration or mere tolerance of one another, depending on their pre-existing relationships.

Additionally, this model architecture has the potential to be simplified back down to the last implementation done by Madhoun.

## 6. Recommendations

### 6.1. Toolkit Improvement

While there the PCD++ toolset is incredibly flexible and powerful, there are some improvements which could be of great benefit.

The toolkit PCD++ offers many great benefits that promise to (potentially) significantly reduce simulation execution time. However, this toolkit should be distributed for use by more platform users. It would be worthwhile to compile the toolkit into executables that will run on more modern OS such as Mac OS X, Windows 7 and Fedora 12.

Further expand "new" CD++ tool with multiple ports and state variables into Windows environment, with the Eclipse plugin.

Develop ability to more easily extract data from variables.. implementation of movement painful

Develop ability to read in / compare with static data in external files. For example, if a relationships table were to exist, have it stored externally as an input file to the simulation, from which to base the simulation relationships.

Develop ability to dynamically set port values through an external file. This was also noted in López' thesis [4].

### 6.2. Model Development

Additionally, the cbs2 model could be further developed to increase its utility. For example, the movement rules could be adapted to take into account collaboration with *Friendly* forces, avoidance of *Neutral* forces, and combat with *Hostile* forces.

Of interest to military forces worldwide is the case of accidental fratricide (accidentally injuring / killing an entity from one's own side). Modelling this with an evolved cbs2 model is entirely possible.

Lastly, it seems that the multi-dimensional nature of the Cell-DEVS formalism lends itself well to modelling a third spacial dimension. Implement a third dimension, taking into account a rudimentary "air picture" and air elements (UAVs, helicopters, jets). Having these air entities interact with the ground entities (this work could be entitled "cbs3".

## 7. References

(1) Ilanchinski, A. 2000. Irreducible Semi-Autonomous Adaptive Combat (ISAAC): An Artificial-Life Approach to Land Combat. Military Operation Research, 5 (3), 29-46. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.75.6947&rep=rep1&type=pdf Accessed: November 9, 2009.

(2) Lauren, M.K. 2002. Firepower concentration in cellular automaton combat models-an alternative to Lanchester. The Journal of the Operational Research Society, 53(6), 672-679. http://www.jstor.org/stable/823011 Accessed: November 9, 2009.

(3) Lim Kew Kia, J. 2002. Land warfare modelling. USSC 3001 Complexity Term Paper. National University of Singapore. http://staff.science.nus.edu/~parwani/project1/jeff.html [Accessed December 7, 2009]

(4) López, A. 2003. Extending CD++ Specification Language for Cell-DEVS Model Definition. Thesis. University of Buenos Aires: Argentina.

(5) Madhoun, R, and Wainer, G.A. 2005. Modelling space-shaped defense applications with Cell-DEVS. Proceedings of SISO Fall Interoperability Workshop, San Diego. http://cell-devs.sce.carleton.ca/publications/2005/MW05b/SIW-05s-Battle.pdf Accessed: November 9, 2009.

(6) Surdu, J.R., Parsons, D., Tran, O. 2005. The Three-Block War in OneSAF. In Proceedings of I/ITSEC 2005. Orlando, FL. Website: http://www.onesaf.net/community/documents/Papers_Presentations/Published/IITSEC05_1977_ThreeBlockWar.pdf [Accessed December 7, 2009]

(7) Wainer, G.A. 2009. Discrete-Event Modeling and Simulation: A Practitioner's Approach. Boca Raton: CRC Press.