Implementing the Five Canonical Offensive Maneuvers in a CGF Environment

David Ezra Sidran
University of Iowa
Iowa City, Iowa, USA

Alberto Maria Segre
University of Iowa
Iowa City, Iowa, USA

{dsidran, segre}@cs.uiowa.edu

Keywords:

Offensive Maneuver, Tactics, Groups, Lines, Frontages, Flanks, Schwerpunkt

ABSTRACT: In this paper we describe the algorithms and underlying machinery necessary to implement the five canonical offensive maneuvers described in U. S. Army Field Manual FM 3-21, Section II, Forms of Maneuver (envelopment, turning movement, infiltration, penetration, and frontal attack) within a computer generated forces environment. We also include descriptions of algorithms for calculating groups and flanks for sets of forces which are a necessary precursor to calculating the five offensive maneuvers.

1. Introduction

It is widely accepted that there is a need for software capable of making high level command decisions within a CGF environment [1]. Currently, the task of making command decisions within such an environment is usually performed by subject matter experts (SMEs) [2]. Traditionally, the military decision making process (MDMP) has seven sequential steps: mission reception, mission analysis, development of the course of action (COA), COA analysis, COA comparison, COA approval, and production of orders [3]. A prerequisite for cognitive models that simulate a commander's decision making process is an understanding of terrain, units' formations and doctrinal templates [3]. We present here a series of algorithms that, together with the necessary underlying machinery, implement the five canonical offensive maneuvers described in U. S. Army Field Manual FM-21 [4]. We suggest that these algorithms will be useful in both the mission analysis, development of COA, COA analysis and COA comparison steps of the MDMP. Furthermore, the utilization of CGF equipped with these algorithms will result in lower cost, requiring fewer human participants in synthetic environment military simulations, while allowing the human participants a greater range of activities [5].

To facilitate our research in this field we have created a test-bed program: the Tactical Inference Generator (TIGER). TIGER employs terrain and elevation layers and allows us to place units of different types within the environment and to observe the results of our algorithms.

2. Building Blocks of Offensive Maneuvers

The U. S. Army Field Manual FM 3-21, Section II, Forms of Maneuver lists five prototype maneuvers: envelopment, turning movement, infiltration, penetration, and frontal attack. The algorithms for implementing the five canonical offensive maneuvers share a common set of building blocks. These building blocks are Range of Influence (ROI), grouping and flank detection, computing flanks and path planning.

2.1. Range of Influence

The concept of a CGF unit's Range of Influence is an extension of the hexagonal board wargame Zone of Control [6] and Influence Mapping [7, 8, 9, 10]. Our implementation of ROI in TIGER allows for each unit type in a simulation to have a unique value which represents the percentage of a unit's strength 'projected' a linear distance from the unit (see Figure 1) to a predefined distance. The ROI for each unit type is stored as an array that also includes a binary value for indicating if a unit's ROI is influenced by an unblocked Line of Sight (LOS) to the target node or not [11].

The most obvious implementation of ROI would be a monotonically decreasing series of numbers indicating that the unit's ROI diminishes as the distance from the unit increases (see Red Group 2, Figure 1). However, our system also allows for representation of other ROIs such as concentric circles (possibly representing a unit with mixed arms, Red Group 1, Figure 1) and a unit with limited offensive power but long range observation abilities (Red Group 0, Figure 1). ROI values are also employed path calculations.

2.2. Groups and Lines

The process of implementing offensive maneuvers begins with sorting the opposing forces (OPFOR) into distinct groups. The earliest known formal description of methods for calculating groups or lines within a 'computer wargame' environment is Crawford's description of his 'Geometric AI' used in the commercial wargame *Patton versus Rommel* circa 1988 [12]. Penner and Steinmetz's

JointAdvisor (2000) employed a method of drawing polygons of different colors representing the probability of accuracy upon a map in response to the query, "Where is the main defense?" [13]

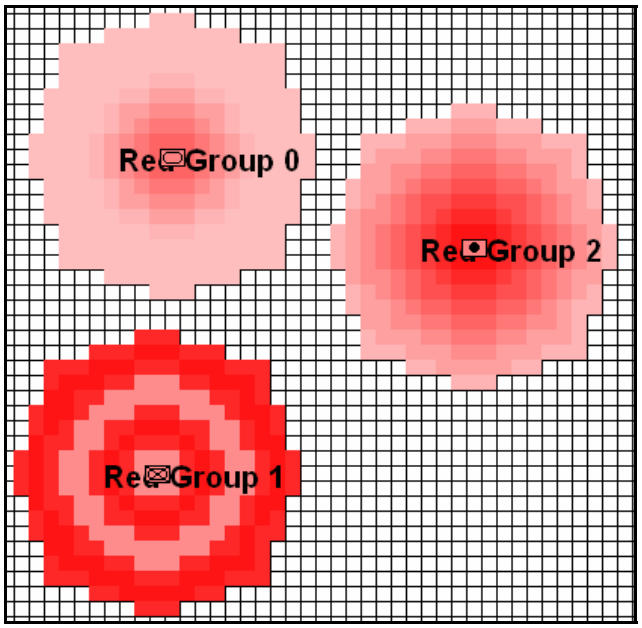

Figure 1. Examples of Range of Influence projected on a grid without terrain or elevation (TIGER screen shot).

Grouping is also performed by SORTS, a Real Time Strategy bot created by Wintermute, Xu and Laird [14]. Though we employ a different grouping algorithm (SORTS uses the principle of Gestalt grouping and sorts by unit type within a predefined radius) we are, however, in agreement with their statement that groups provide a "key abstraction for tactical reasoning." [14]

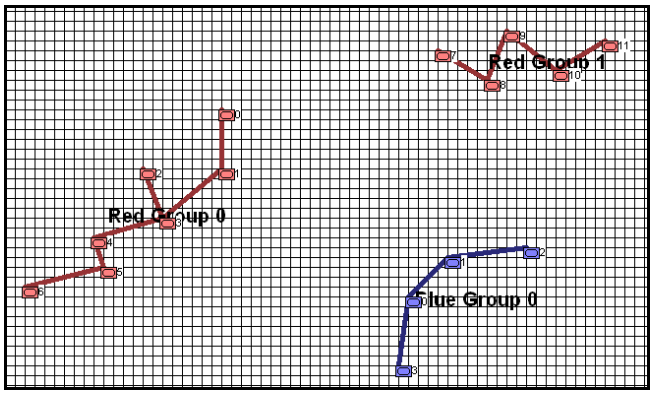

Figure 2. Grouping of units using MST clustering without terrain or elevation (TIGER screen shot).

Our grouping is based on an abstract notion of "proximity" which can be defined in terms of actual distance and/or other measures. From a procedural perspective we employ Kruskal's algorithm to create a minimum spanning tree (MST) of the OPFOR (see Figure 2). [15] Units are divided into groups via MST clustering [16]. Clustering can be performed either by using a minimum threshold between groups (ComputeGroupsByThreshold) or by specifying the number of groups desired (ComputeGroupsByNumber).

In the following algorithms U is the set of *all* units (both opposition, OPFOR, and friendly, MYFOR). ROI values are embedded in the 'world view' W, which also reflects, e.g., a unit's view of the terrain and known opposing forces.

Algorithm for ComputeGroupsByThreshold Function

```
// Group OPFOR units from a set U of units according to
// edge weighting or distance function embedded in 'world
// view' W. Returns a forest of minimum spanning trees
// corresponding to groups separated at least by distance
// threshold D

ComputeGroupsByThreshold(U, W, D)
{
    // Compute MST of OPFOR units in set U
    T ← MST(OPFOR(U))
    // Remove edges longer than distance threshold
    for e in edges (T)
        if (weight(e,W) > D)
            T ← delete (e,T)
    // Return forest of MSTs
    return(T)
}
```

The threshold value can be a user-defined Euclidean distance which may be, e.g., dependent upon unit visibility as calculated by a 3D Bresnham line algorithm [17]. Other values, such as unit ROI, unit type and distance that a unit can travel over terrain within a preset period of time, can also be used to establish a threshold value. When employed as the first step of the MDMP, the returned number of groups (NumGroups(T)) can be used in the process of deciding the appropriate offensive maneuver to implement (see 5, Further Research). In contrast, the penetration maneuver requires exactly two OPFOR groups. Calling ComputeGroupsByNumber with the desired number of groups will result in OPFOR being divided accordingly.

Algorithm for ComputeGroupsByNumber Function

```
// Group OPFOR units from a set U of units into a fixed
// number of N subgroups using edge weighting or distance
// function embedded in 'world view' W. Returns a forest
// of N minimum spanning trees corresponding to groups.

ComputeGroupsByNumber (U, W, N)
{
    // Compute MST of units in set U
    T ← MST(OPFOR(U))
    // Sort edges in T by length, ascending
    E ← sortAscending(edges(T))
```

```
    //Remove N longest edges.
    while (N > 1)
        e ← pop(E)
        T ← delete(e,T)
        N ← N -1
    // Return forest of MSTs
    return(T)
}
```

Once groups have been determined by either method additional factors, e.g. ROI (see Figure 3) can be added to the analysis of the maneuver implementations. Some maneuvers require specific additional information such as "center of mass" of a group or determination of flank units.

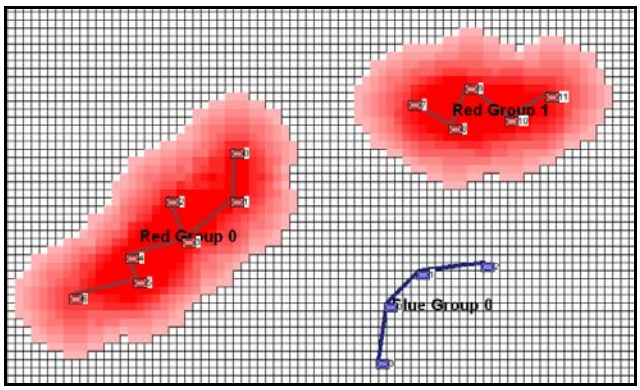

Figure 3. Grouping of units using MST clustering with ROI displayed without terrain or elevation (TIGER screen shot)

2.3. Computing Flanks after Units are sorted into Groups

Computing the flanks of these groups is necessary for, e.g., the implementation of the envelopment and turning maneuvers. The function:

CalculateLeftFlank(U)

returns the flank unit, an element of OPFOR(U). **CalculateRightFlank**(U) is analogous. The flank units of any group are the two units that have the greatest degree of separation as determined by the weighting function. Left and right are determined by the position of MYFOR(U).

2.4. Computing Gap Edges

A gap edge is an edge of the MST that is removed to create a group:

GapEdges(T)

where T is a forest of MSTs; this function returns a set of edges. Since these edges were all part of the original MST, they represent the shortest separating edge between groups.

2.5. Computing Centers

Computing the geographical center of a group is necessary for both the envelopment and the frontal attack maneuvers. The geographical center of a group is calculated by summing the location of every unit in the group weighted by the strength of the unit, returning the resulting average location:

CalculateCenter(U)

where U is a set of units. The function returns a location.

2.6. Pathfinding

Each maneuver described here assumes that we have the ability to plot, for a specified unit, the "best" path to an assigned objective, subject to appropriate constraints and guided by a utility function based on the current world view W (which includes, e.g., enemy ROI, line of sight, etc.):

FindPath(u, G, B, o, W)

where u is a unit, G a collection of "gap edges", B a collection of "barrier edges" o is an objective, given as a location in graph coordinates and W is the unit u's world view. Elements of G and B may be either edges (i.e., line segments defined between two graph coordinates) or rays (i.e., lines rooted at a graph coordinate and extending to infinity along a given direction) which are used to impose restrictions on the legal solutions: A solution path produced by FindPath must traverse at least one of the edges given in G and none of the edges in B. Internally, such a function might use an A* path finding algorithm guided by a heuristic to minimize exposure and maximize concealment of the moving unit, should that be the nature of the prespecified utility function [18] (see also [19]). Finally, the function should return a measure of quality of the path constructed, so that two paths computed separately can be directly compared according to the utility function used to construct them.

These algorithms assume that the ultimate objectives have been set by a higher level in the command structure and that these maneuvers are being implemented in response to OPFOR that are encountered en route to the ultimate objective. These algorithms are implementations of tactical maneuvers within a tactical situation and are not part of the strategic level decision making process per se.

3.0. Algorithms for Implementing the Five Canonical Offensive Maneuvers

We next present the algorithms for implementing the five canonical offensive maneuvers.

3.1. Implementing the Penetration Maneuver

The penetration maneuver can be divided into two phases: in the first phase the attacker concentrates forces to strike at an enemy's weak point; the second phase is to break through the position and rupture the defense. The attacker then passes forces through the gap created to defeat the enemy with attacks into his flanks and rear (see Figure 4) [4], in order to obtain a prespecified objective.

We use the term '*Schwerpunkt*' as the point of attack or the "decisive point" [20]. To implement the penetration maneuver we must calculate the *Schwerpunkt*; in this case the weakest point of the OPFOR.

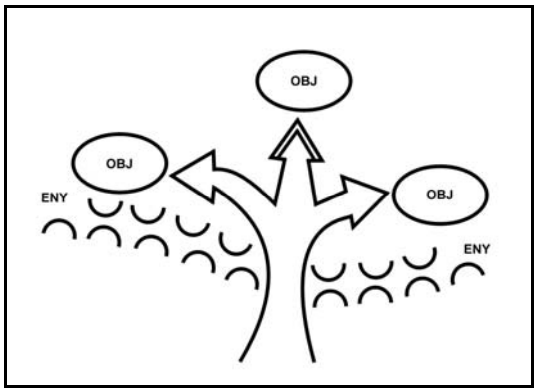

Figure 4. The penetration maneuver from U. S. Army Field Manual 3-21.

Algorithm for Penetration Maneuver

```
// For every MYFOR unit from a set U of units assign an
// edge (the Schwerpunkt) which is the enemy's weakest
// point and calculate a path from the unit's current
// location to the appropriate objective from the set of
// objectives O via the Schwerpunkt. Return the set of
// paths P.

PenetrationManeuver(U,W,O)
{
     // Divide OPFOR into 2 groups
    T ← ComputeGroupsByNumber(U, W, 2)
    // Edge of lowest weight connecting two groups
    // is the Schwerpunkt
    S ← GapEdges(T)
    P ←{}
    for u in MYFOR(U)
        P ← P ∪ {FindPath(u, S, T,
            mapObjective(u, O,W))}
  return(P)
}
```

Upon completion of the penetration maneuver another maneuver, such as the turning movement using either, or both, OPFOR groups, may be employed to exploit the rupture in the defensive line (see Figure 4).

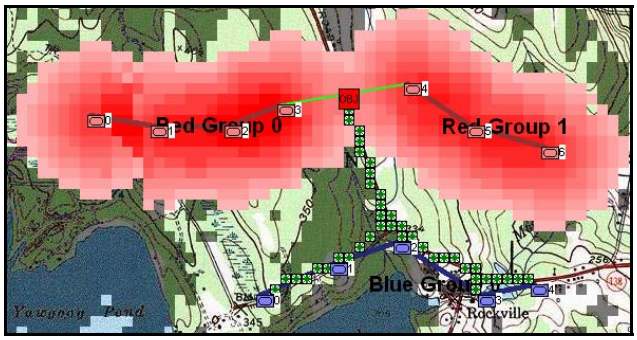

Figure 5. Example of the penetration maneuver from the TIGER test-bed program.

3.2. Implementing the Infiltration Maneuver

The infiltration maneuver is defined as a, "form of maneuver in which combat elements conduct undetected movement (mounted or dismounted) through or into an area occupied by enemy forces to occupy a position of advantage in the enemy's rear." [4] In our implementation we assume that the attacking forces have multiple objectives that are set a priori by SMEs prior to implementation.

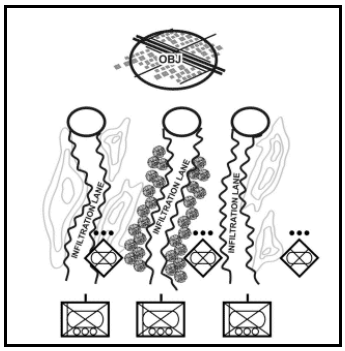

Figure 6. The infiltration maneuver from U. S. Army Field Manual 3-21,

Algorithm for Infiltration Maneuver

```
// For every MYFOR unit from a set U of units assign an
// objective o in a set of objectives O which have been
// placed by an SME and calculate a path from the unit's
// current location to the nearest objective subject to the
// constraint that OPFOR groups are separated by a
// minimum safe infiltration threshold, dMin. Return
// the set of paths P.

InfiltrationManeuver(U, W, O, dMin)
{
    T ← ComputeGroupsByThreshold(U, W, dMin)
    // If the number of trees is less then the size of O + 1
    // attempt to recalculate trees by using alternative
    // method
    if (NumGroups(T) < O + 1 )
        T ← ComputeGroupsByNumber(OPFOR(U),
            W, SizeOf(O)+1)
    // Map units to nearest Objective
    P ← {}
     for u in MYFOR(U)
        P ← P ∪ {FindPath(u, GapEdges(T), T,
            mapObjective(u, O,W))}
     return(P)
}
```

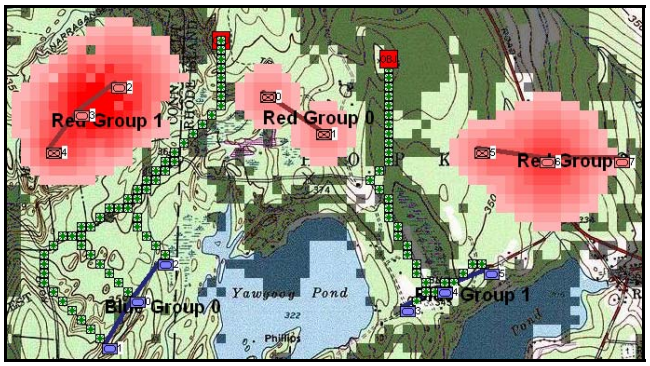

Figure 7. Example of the infiltration maneuver from the TIGER test-bed program.

3.3. Implementing the Turning Movement

The turning movement is designed for the attacking forces to pass around the OPFOR flanks and avoid contact with OPFOR units; it then secures an objective that causes the enemy to move out of its current position or divert forces to meet the threat [4]. In our implementation we assume that the attacking forces have an objective that is set a priori by SMEs prior to implementation.

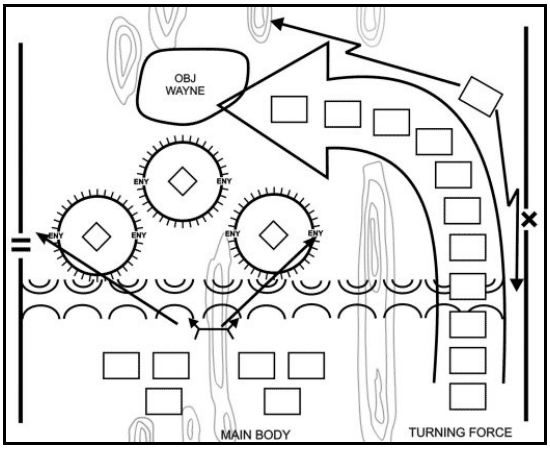

Figure 8. The turning movement from U. S. Army Field Manual 3-21,

Algorithm for Turning Movement

// For every MYFOR unit from a set U of units assign the
// objective o which has been placed by an SME and
// calculate a path from the unit's current location to o
// avoiding OPFOR and their ROI as represented in world
// view W. Returns set of paths P with highest utility
// between left turning paths P_l and right turning paths P_r.

TurningManeuver(U, W, o)
{
  c ← **CalculateCenter**(OPFOR(U));
  l ← **CalculateLeftFlank**(OPFOR(U));
  r ← **CalculateRightFlank**(OPFOR(U));
  P_r ←P_l ← {}
  for u in MYFOR(U)
    P_r ← P_r ∪ {**FindPath**(u, {ray(c, r)}, T,
      mapObjective(u, O, W))}
    P_l ← P_l ∪ {**FindPath**(u, {ray(c, l)}, T,
      mapObjective(u, O, W))}
  if (utility(P_r)>utility(P_l))
    return(P_r)
  else
    return(P_l)
}

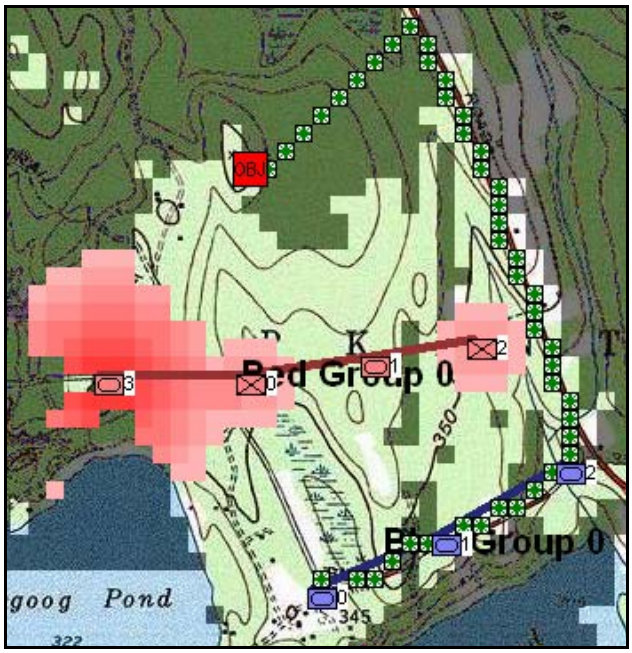

Figure 9. Example of the turning movement from the TIGER test-bed program (note restricted ROI due to limited LOS).

3.4. Implementing the Envelopment Maneuver

In the envelopment maneuver the attacker attempts to fix the defender with supporting attacks (the fixing force) while he maneuvers the main attack around the enemy's defenses to strike at the flanks, the rear, or both (the flanking forces) [4]. The steps for implementing the envelopment maneuver are:

1. Detecting the flanks of OPFOR
2. Calculating the objective for the flanking forces.
3. Calculating the objective for the fixing forces.
4. Divided the offensive forces into flanking and fixing forces based on a percentage set by an SME (the default value in the TIGER test-bed program is 60% of forces assigned to the flanking objective, though this value is adjustable by the user).
5. Plot paths from units to assigned objectives.

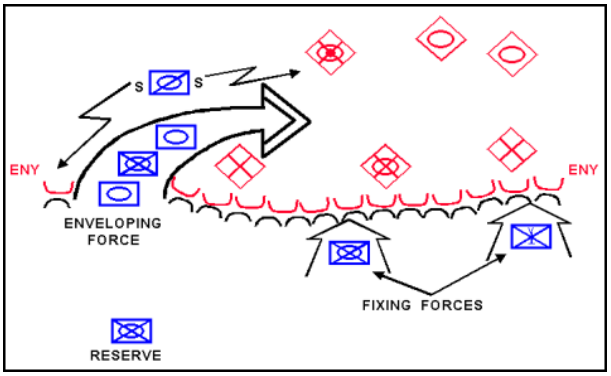

Figure 10. The envelopment maneuver from U. S. Army Field Manual 3-21,

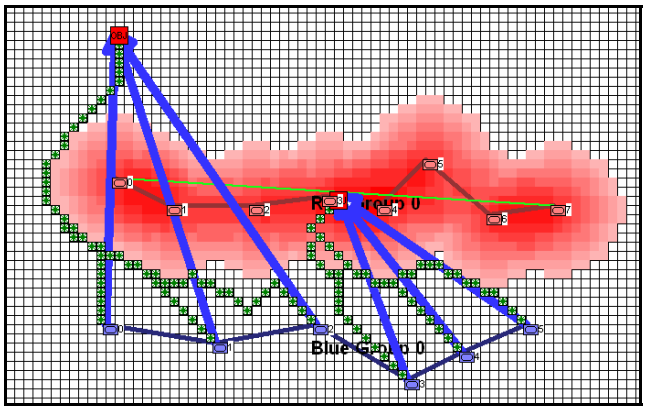

Figure 11. The envelopment maneuver without terrain or elevation displayed. Unit objectives shown as thick blue lines. The thin green line is the longest edge; its vertices are the flank units. TIGER screen shot.

Algorithm for Envelopment Maneuver

// Determine the number of MYFOR(U) assigned to, N_{fix}
// the Fixing Force, and N_{flank} the Flanking Force based on
// the FlankingForcePercentage (which is SME defined).
// Calculate the left and right flank units of OPFOR.
// Determine the optimality of either P_l or P_r to their
// respective objectives. For every unit assigned to the
// Flanking Objective plot a path from the unit's current
// location to the objective avoiding OPFOR ROI.
// For every unit assigned to the Fixing Force plot a path to
// the Fixing Objective c. Return the paths in P, a set of
// paths.

EnvelopmentManeuver(U, W)
{
    // Calculate size of flanking and fixing forces.
    N_{flank} ← (FlankingForcePercentage/100)*|MYFOR(U)|
    N_{fix} ← |MYFOR(U)| - N_{flank}
    // Find OPFOR center
    c ← **CalculateCenter**(OPFOR(U))
    // Find left and right flanks of OPFOR(U)
    // with respect to MYFOR(U)
    l ← **CalculateLeftFlank** (U)
    r ← **CalculateRightFlank** (U)
    // Initialize alternate left and right path sets
    P_l ← P_r ← {}
    // Find paths around both left and right
    // flanks of OPFOR; only one will eventually
    // be used.
    for u in MYFOR(U)
        P_l ← P_l ∪ {**FindPath**(u, {ray(c,l)}, T,
            mapObjective(u, O, W)}
        P_r ← P_r ∪ {**FindPath**(u, {ray(c,r)}, T,
            mapObjective(u, O, W)}
    // Evaluate the utility of the top
    // FlankingForcePercentage paths in both path sets.
    // The higher scoring direction will be used.
    U_r ← U_l ← 0
    for p in sortDescending(P_l) as i from 1 to N_{flank}
        U_l ← U_l + utility(p)
    for p in sortDescending(P_r) as i from 1 to N_{flank}
        U_r ← U_r + utility(p)
    // Pick flank with best utility value for flanking
    // forces based on best FlankingForcePercentage
    // paths. Return set of paths including new paths
    // for the fixing forces.
    P ← {}
    if (U_l > U_r)
    // Go left; accept top N_{flank} flanking paths
        for p in sortDescending(P_l) as i from 1 to N_{flank}
            P ← P ∪ {p}
    // Add fixing paths for N_{fix} units with worst
    // flanking paths
        for (p in sortAscending(P_l) as i from 1 to N_{fix}
            P ← P ∪ {**FindPath**(unit(p), {}, {}, c)}
    else
    // Go right; accept top N_{flank} flanking paths
        for p in sortDescending(P_r) as i from 1 to N_{flank}
            P ← P ∪ {p}
    // Add fixing paths for N_{fix} units with worst
    // flanking paths
        for (p in sortAscending(P^r) as i from 1 to N_{fix}
            P ← P ∪ {**FindPath**(unit(p), {}, {}, c)}
    // Done; return collection of paths.
    return(P)
}

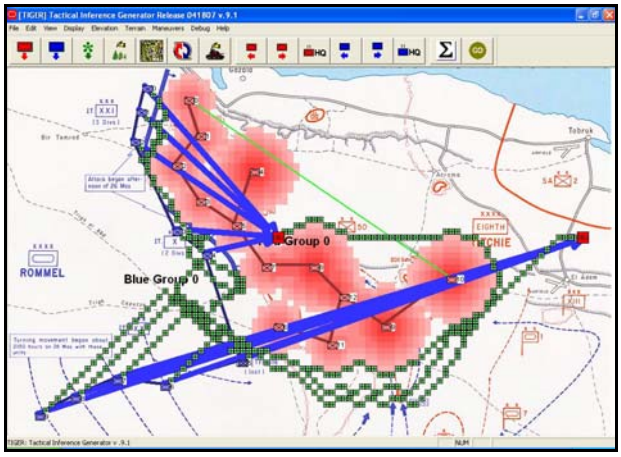

Figure 12. Rommel's envelopment maneuver (Gazala June 1942) imported from the West Point Atlas[20] into TIGER. TIGER's version of the envelopment maneuver has been overlayed on the same map. TIGER screen shot.

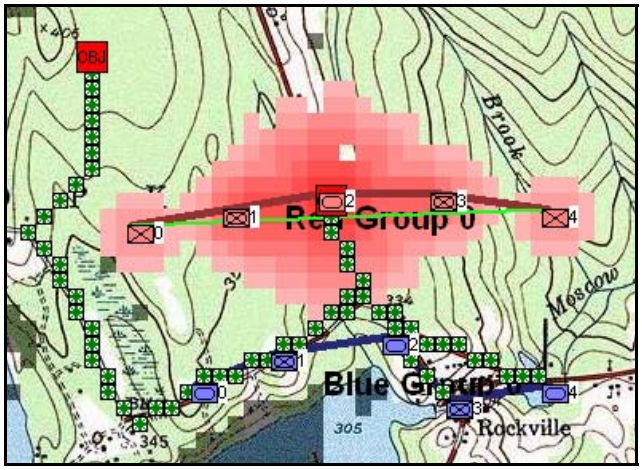

Figure 13. The envelopment maneuver from the TIGER test-bed program. TIGER screen shot.

3.5. Implementing the Frontal Attack Maneuver

The frontal attack is the least desirable form of maneuver because it exposes the majority of the offensive force to the concentrated fires of the defenders [4]. To minimize attacker casualties would require a detailed analysis of terrain, elevation, weapons, covering and suppressing fire, as well as other factors and, consequently, is beyond the scope of this paper. Our simplified algorithm for movement of units follows.

Algorithm for Frontal Attack Maneuver

```
// For every MYFOR unit from a set U of units assign a
// goal (the weighted center of OPFOR) and calculate a
// path from the unit's current location to the goal and
// return the paths in P a set of paths.
FrontalAttack(U, W)
{
    c ← CalculateCenter(OPFOR(U))
    P ← {}
    for u in MYFOR(U)
        P ← P ∪ {FindPath(u, {}, {}, c, W)}
    return(P)
}
```

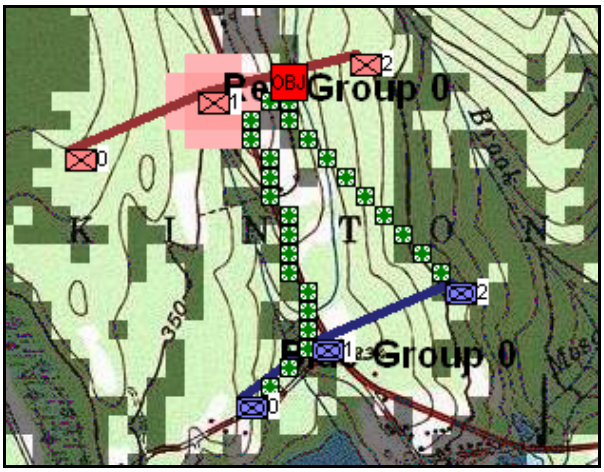

Figure 14. The frontal attack maneuver from the TIGER test-bed program. TIGER screen shot.

4. Conclusions

We have presented here a basic set of algorithms that can be utilized in any land warfare CGF environment in which subordinate units are required to implement the offensive orders of a superior unit or Command Entity. We have implemented and tested these algorithms within TIGER, our test-bed environment. We have also begun experimenting with importing maps from the West Point Atlas [21], placing units accordingly, and observing the results (see Figure 12). Preliminary results seem to confirm that these algorithms can accurately recreate historical offensive maneuvers under appropriate conditions.

5. Further Research

It has been said that the "holy grail of CE designers is a system that can learn tactics and doctrine" [22]. We intend to expand our TIGER test-bed program and incorporate learning capabilities (via a Support Vector Machine system) which is reactive, and can learn, from experience or textbooks: specifically utilizing digitized images of battlefield maps from the West Point Atlas as a training set [21]. Our preliminary experiments with importing West Point Atlas maps suggest this approach may well be feasible (see Figure 12 in which we have imported the West Point Atlas map of Rommel at Gazala, June 1942 and TIGER issued similar, but not identical, unit orders). We also anticipate further work on fine-tuning our algorithms for implementing the five canonical offensive maneuvers.

Acknowledgements

We would like to thank Professors Jim Cremer, Joe Kearney, Hwan Jo Yu and Steve Bruell of the University of Iowa Department of Computer Science and Professor Rosemary Moore of the University of Iowa Department of Classics for their insightful comments and suggestions. We would like to thank LTC John R. "Buck" Surdu, Ph.D. who first suggested this line of research.

References

[1]. Calder, Robert B, et al. "Architecture of a Command Forces Command Entity." Paper presented at the 6th Conference on Computer Generated Forces and Behavioral Representation. Orlando, FL, July 23-25, 2004.

[2]. Hunter, Keith O., and William E. Hart. *A Naturalistic Decision Making Model for Simulated Human Combatants*. Sandia National Laboratories no. SAND2000-0974. Albuquerque, NM: Sandia National Laboratories, 2000.

[3]. Jaurez-Espinosa, Octavio, and Cleotilde Gonzalez. "Situation Awareness of Commanders: A Cognitive Model." Paper presented at the 7th Conference on Computer Generated Forces and Behavioral Representation, May, 2005.

[4]. U. S. Army Field Manual 3-21.21. *U. S. Army Field Manual (FM) 3-21.21 The Stryker Brigade Combat Team Infantry Battalion*. Washington DC: Headquarters, Department of the Army, 2003.

[5]. Pongracic, Helen, Peter Clark, and Arvind Chandran. *Integrating Intelligent Agents with a Human-in-the-Loop Simulation*. Air Operations Division. Melbourne, Victoria, Australia: Defence Science and Technology Organisation, 2000.

[6]. Dunnigan, James. F. *The Complete Wargames Handbook*. New York: Quill, 1992.

[7]. Tozour, P. "Influence Mapping." *Game Programming Gems 2*. Ed. M. Deloura. Hingham, MA: Charles River Media, 2001. 287-97.

[8]. Sweetser, Penny. "Strategic Decision-Making with Neural Networks and Influence Maps." *AI Game Programming Wisdom 2*. Ed. Steve Rabin. Hingham, MA: Charles River Media, 2004. 439-46.

[9]. Sweetser, Penelope. "An Emergent Approach to Game Design - Development and Play." Diss. School of Information Technolgy and Electrical Engineering, The University of Queensland, 2006.

[10]. Sidran, David Ezra. "A Calculated Strategy: Readings Directed Towards the Creation of a Strategic Artificial Intelligence." Computer Science, University of Iowa, 2004. Viewed May 3, 2007 http://www.cs.uiowa.edu/~dsidran/ReadingsForResearch2.pdf.

[11]. ---. "Good Decisions Under Fire: Human-Level Strategic and Tactical Artificial Intelligence in Real-World Three-Dimensional Environments." Computer Science, University of Iowa, April 23 2007. Viewed May 3, http://cs.uiowa.edu/~dsidran/GoodDecisionsUnderFire.pdf

[12]. Crawford, Chris. *Chris Crawford on Game Design*. Indianapolis, IN: New Riders, 2003.

[13]. Penner, Robin R., and Erik S. Steinmetz. "JointAdvisor: An Intelligence Analysis Agent." Paper presented at the ARL Consortium Conference. Army Research Labs, 2000..

[14]. Wintermute, Sam, Joseph Xu, and John E. Laird. "SORTS: A Human-Level Approach to Real-Time Strategy AI." *Association for the Advancement of Artificial Intelligence* (Paper not yet published) (2007).

[15]. Kruskal, J. B. "On the Shortest Spanning Subtree and the Traveling Salesman Problem." *Proceedings of the American Mathematical Society* 7 (1956): 48-50.

[16]. Zahn, C. T. "Graph Theoretic Methods for Detecting and Describing Gestalt Clusters." IEEE Transactions on Computing, 1971

[17]. Bresenham, Jack E. "Algorithm for Computer Control of a Digital Plotter." *IBM Systems Journal* 4(1) (1965): 25-30.

[18]. Hart, P. E., N. J. Nilsson, and B. Raphael. "A Formal Basis for The Heuristic Determination of Minimum Cost Paths." *IEEE Transactions on Systems, Science and Cybernetics* 4(2) (1968).

[19]. Beeker, Emmet. "Potential Error in the Reuse of Nilsson's A Algorithm for Path-Finding in Military Simulations." *JDMS* 1.2 (April 2004): 91-97.

[20]. von Mellenthin, F. W. *Panzer Battles: A Study of the Employment of Armor in the Second World War*. New York: Ballantine Books, 1956.

[21]. Griess, Thomas. E. *The West Point Military History Series Atlas for Thge Second World War (Europe and the Mediterranean)*. Wayne, NJ: Avery Publishing Group, 1953.

[22]. Howard, Michael D. "Modeling Command Entities." *Planning and Scheduling*. International Joint Conference on Artificial Intelligence, 1997.

.