THE SUSTAINED COMBAT MODEL:
TANK WARS II PROGRAMMERS' MANUAL

FRED L. BUNN

NOVEMBER 1991

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

91123006
NOTICES

Destroy this report when it is no longer needed. DO NOT return it to the originator.

Additional copies of this report may be obtained from the National Technical Information Service, U.S. Department of Commerce, 5285 Port Royal Road, Springfield, VA 22161.

The findings of this report are not to be construed as an official Department of the Army position, unless so designated by other authorized documents.

The use of trade names or manufacturers' names in this report does not constitute indorsement of any commercial product.
**Title and Subtitle**
The Sustained Combat Model: Tank Wars II Programmers' Manual

**Authors**
Fred L. Bunn

**Performing Organization Name(s) and Address(es)**
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

**Abstract**
This report describes a stochastic simulation of combat between armored combat systems and is oriented toward those who are interested in modifying, debugging, or correcting the model.

The model is routinely used at the Ballistic Research Laboratory, other military installations, and by contractors to evaluate trade-offs in the characteristics of weapon systems candidates. It was specifically designed to evaluate the combat effectiveness of tanks and other armored fighting vehicles but has been adapted to evaluate other weapons systems. The model treats fighting vehicles in meeting engagements, in attack scenarios and in defensive scenarios. It treats multiple systems on each side and has the following features: multiple waves of Red attackers without Blue resupply, Blue attack on multiple Red positions without resupply, individual round or burst fire, kills at four kill levels. The model simulates stationary or moving attackers, and plays guns and missiles but not a mix on a single side. It is written in Fortran 77 for portability and is based on the old TANK WARS model which is still running at over a dozen installations.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>LIST OF FIGURES</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACKNOWLEDGEMENTS</td>
<td>xi</td>
</tr>
<tr>
<td>1. INTRODUCTION</td>
<td>1</td>
</tr>
<tr>
<td>2. TOP LEVEL ROUTINES</td>
<td>11</td>
</tr>
<tr>
<td>3. SEARCH AND DETECTION ROUTINES</td>
<td>27</td>
</tr>
<tr>
<td>4. TARGET SELECTION ROUTINES</td>
<td>43</td>
</tr>
<tr>
<td>5. FIRING ROUTINES</td>
<td>53</td>
</tr>
</tbody>
</table>

## 1. INTRODUCTION

1.1 Hierarchy of Routines ................................................................. 3
1.2 One Line Descriptions of Routines ............................................. 4
1.3 Data Structure ........................................................................... 6

## 2. TOP LEVEL ROUTINES

2.1 Main: Simulate Armored Combat .............................................. 12
2.2 Waves: Loop Through Waves of Red Tanks ............................... 13
2.3 Events: Call Each Event in Sequence ....................................... 15
2.4 Input: Read Game Control File ................................................. 16
2.5 Rdmisc: Read Miscellaneous Tank Characteristics .................. 17
2.6 Prmisc: Print Miscellaneous Tank Characteristics .................... 18
2.7 Init: Initialize for a Single Engagement ................................... 19
2.8 Stats1: Print Summary Statistics .............................................. 21
2.9 Stats2: Print Statistics for a Single Wave .................................. 23
2.10 Stats3: Print Statistics for a Single Engagement ..................... 24

## 3. SEARCH AND DETECTION ROUTINES

3.1 Initinv: Generate Detection Probability Tables ....................... 28
3.2 Nvl: Find Probability of Ever Detecting and of Detecting in Next Second ....................................................... 29
3.3 Eye: Find the Probability the Eye Detects in the Next Second .... 31
3.4 Detrg: Find Maximum Range to Which Each Firer Can Detect .... 33
3.5 Ssearch: Schedule Initial Search Event ..................................... 34
3.6 Search: Find Targets Detected in Next Second ......................... 36
3.7 Searc2: Find if One Searcher Detects One Target in the Next Second ........................................................... 38
3.8 Detect: Find if Target is Detected and Schedule Subsequent Events ............................................................... 40
3.9 Pinpnt: Simulate Firing Signature Detection ............................ 41

## 4. TARGET SELECTION ROUTINES

4.1 Select: Gunner Chooses Most Dangerous Target it Sees ............ 44
4.2 Selecs: Start Target Selection if Appropriate ........................... 46
4.3 Priorn: Select Target with Highest Priority ......................... 47
4.4 Priort: Find Priority of a Single Target .................................. 49
4.5 Engage: Begin Engagement of a New Target by This Firer ....... 52

## 5. FIRING ROUTINES

5.1 Firing Cycles ............................................................................ 54
5.2 Fire: Simulate Firing of a Round and Schedule Effects ............. 56
5.3 Frdssg: Results of Firing a Single Shot Gun ......................... 58
5.4 Frdbst: Results of Firing a Round of a Burst ......................... 60
## TABLE OF CONTENTS (contd)

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.5 Frdmsl: Results of Firing a Missile</td>
<td>61</td>
</tr>
<tr>
<td>5.6 Reload: Bring up Another Pod of Missiles</td>
<td>62</td>
</tr>
<tr>
<td>6. HIT PROBABILITY AND IMPLICATIONS</td>
<td>63</td>
</tr>
<tr>
<td>6.1 Impact: Find What Bullet and Firer Do at Impact</td>
<td>64</td>
</tr>
<tr>
<td>6.2 Mayhit: Find Whether the Round Hits</td>
<td>66</td>
</tr>
<tr>
<td>6.3 Izhit: Find if the Target is Hit</td>
<td>68</td>
</tr>
<tr>
<td>6.4 Bounds: Find the Horizontal Bounds of the Hull or Turret</td>
<td>69</td>
</tr>
<tr>
<td>7. ACCURACY ROUTINES</td>
<td>71</td>
</tr>
<tr>
<td>7.1 RdError: Read Accuracy Data for One Side</td>
<td>72</td>
</tr>
<tr>
<td>7.2 AccErr: Find the Linear Error for a Single Round</td>
<td>73</td>
</tr>
<tr>
<td>7.3 AccSs: Find Angular Accuracy for Stationary Firer Vs Stationary Target</td>
<td>74</td>
</tr>
<tr>
<td>7.4 AccSm: Find Angular Accuracy for Stationary Firer Vs Moving Target</td>
<td>77</td>
</tr>
<tr>
<td>7.5 AccMs: Find Angular Accuracy for Moving Firer Vs Stationary Target</td>
<td>79</td>
</tr>
<tr>
<td>8. DAMAGE ROUTINES</td>
<td>81</td>
</tr>
<tr>
<td>8.1 Damage: Simulate Damage to the Target</td>
<td>82</td>
</tr>
<tr>
<td>8.2 DamagF: Simulate Firepower Damage</td>
<td>84</td>
</tr>
<tr>
<td>8.3 DamagM: Simulate Mobility Damage</td>
<td>85</td>
</tr>
<tr>
<td>8.4 Deaths: Tally Deaths</td>
<td>86</td>
</tr>
<tr>
<td>8.5 LateKl: Simulate Discard of Inactive M&amp;F-Killed Target</td>
<td>87</td>
</tr>
<tr>
<td>9. LETHALITY ROUTINES</td>
<td>89</td>
</tr>
<tr>
<td>9.1 RdPkh1: Read Standard Lethality Data</td>
<td>91</td>
</tr>
<tr>
<td>9.2 MkTbl: Make Table of Head-on Lethality Data</td>
<td>94</td>
</tr>
<tr>
<td>9.3 RdPkh2: Read HEAT and Missile Lethality Data</td>
<td>95</td>
</tr>
<tr>
<td>9.4 RdPkh5: Read Top Attack Lethality Data</td>
<td>96</td>
</tr>
<tr>
<td>9.5 Kill: Find Type of Damage Caused</td>
<td>98</td>
</tr>
<tr>
<td>9.6 Kills: Find Type of Damage Caused by Top Attack Round</td>
<td>99</td>
</tr>
<tr>
<td>10. DISENGAGEMENT ROUTINES</td>
<td>101</td>
</tr>
<tr>
<td>10.1 Disengagement Tactics</td>
<td>102</td>
</tr>
<tr>
<td>10.2 Single-Shot Ballistic Projectiles</td>
<td>103</td>
</tr>
<tr>
<td>10.3 Missile Systems</td>
<td>104</td>
</tr>
<tr>
<td>10.4 Key Disengagement Variables</td>
<td>106</td>
</tr>
<tr>
<td>10.5 Diseng: Attempt to Disengage 1 Firer from 1 Target</td>
<td>109</td>
</tr>
<tr>
<td>10.6 Newtgt: Redirect All Foe to a New Target</td>
<td>111</td>
</tr>
<tr>
<td>10.7 Abort: Abort a Missile in Flight</td>
<td>113</td>
</tr>
<tr>
<td>11. MOTION ROUTINES</td>
<td>115</td>
</tr>
<tr>
<td>11.1 Deploy: Place Combatants at Start of Engagement</td>
<td>116</td>
</tr>
<tr>
<td>11.2 SlowUp: Begin Deceleration</td>
<td>118</td>
</tr>
<tr>
<td>11.3 Halt: Simulate a Tank Halting</td>
<td>119</td>
</tr>
<tr>
<td>11.4 Accel: Begin Acceleration</td>
<td>121</td>
</tr>
</tbody>
</table>
# TABLE OF CONTENTS (contd)

<table>
<thead>
<tr>
<th>Section</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>11.5</td>
<td>MaxVel: Simulate Tank Reaching Combat Cruise Speed</td>
<td>122</td>
</tr>
<tr>
<td>11.6</td>
<td>Path: Find Position and Velocity of Combatant</td>
<td>123</td>
</tr>
<tr>
<td>11.7</td>
<td>Rgf: Find Range to Target, Relative Position, and Velocities</td>
<td>125</td>
</tr>
<tr>
<td>11.8</td>
<td>CanGo: Find if Tank is Stopped by Mobile</td>
<td>126</td>
</tr>
<tr>
<td>12.</td>
<td>OBSCURATION ROUTINES</td>
<td>127</td>
</tr>
<tr>
<td>12.1</td>
<td>RdSmk: Read Intervisibility Data for Smoke</td>
<td>128</td>
</tr>
<tr>
<td>12.2</td>
<td>Smoke: Find When Smoke Will Stop Blocking LOS Between Searchers and Targets</td>
<td>130</td>
</tr>
<tr>
<td>12.3</td>
<td>Appear: Simulate or Reschedule an Appear Event</td>
<td>131</td>
</tr>
<tr>
<td>12.4</td>
<td>Aprsmk: Simulate Target Appearing from Behind Smoke</td>
<td>133</td>
</tr>
<tr>
<td>12.5</td>
<td>Aprter: Simulate Target Appearing from Behind Terrain</td>
<td>134</td>
</tr>
<tr>
<td>12.6</td>
<td>Terain: Find Path Lengths Where Attacker is Masked by Terrain</td>
<td>135</td>
</tr>
<tr>
<td>12.7</td>
<td>Vanish: Simulate or Reschedule Vanish Event</td>
<td>136</td>
</tr>
<tr>
<td>12.8</td>
<td>Vansmk: Simulate Target Vanishing Behind Smoke</td>
<td>137</td>
</tr>
<tr>
<td>12.9</td>
<td>Vanter: Treat Target Vanishing Behind Terrain</td>
<td>138</td>
</tr>
<tr>
<td>12.10</td>
<td>Hide: Simulate Tank Hiding</td>
<td>139</td>
</tr>
<tr>
<td>12.11</td>
<td>PopDn: Simulate Defender Popping Down to Reload Missile Pods</td>
<td>140</td>
</tr>
<tr>
<td>13.</td>
<td>TIME ADVANCE ROUTINES</td>
<td>141</td>
</tr>
<tr>
<td>13.1</td>
<td>Event Handling Using Linked Lists</td>
<td>142</td>
</tr>
<tr>
<td>13.2</td>
<td>Reset: Re-initialize the Event List</td>
<td>143</td>
</tr>
<tr>
<td>13.3</td>
<td>Skedul: Schedule an Event</td>
<td>144</td>
</tr>
<tr>
<td>13.4</td>
<td>Event: Find Next Event</td>
<td>146</td>
</tr>
<tr>
<td>13.5</td>
<td>Cancel: Cancel and Event</td>
<td>147</td>
</tr>
<tr>
<td>14.</td>
<td>OTHER UTILITY ROUTINES</td>
<td>149</td>
</tr>
<tr>
<td>14.1</td>
<td>Create: Find Space to Store Bullet Data</td>
<td>150</td>
</tr>
<tr>
<td>14.2</td>
<td>Anglef: Find the Angle Between Two Vectors</td>
<td>152</td>
</tr>
<tr>
<td>14.3</td>
<td>Confb: Find the 90% Confidence Interval On a Binomial Outcome</td>
<td>153</td>
</tr>
<tr>
<td>14.4</td>
<td>Indexx: Find the Index $j$, Where $a(j) &lt; = &lt; a(j+1)$</td>
<td>154</td>
</tr>
<tr>
<td>14.5</td>
<td>Ranu: Draw a Random Number from the Standard Uniform Distribution</td>
<td>155</td>
</tr>
<tr>
<td>14.6</td>
<td>Rann: Draw a Random Number from a Normal Distribution</td>
<td>157</td>
</tr>
<tr>
<td>14.7</td>
<td>RndAng: Draw a Random Angle from a Cardioid or Other Distribution</td>
<td>158</td>
</tr>
</tbody>
</table>

REFERENCES ........................................................................................................... 161

DISTRIBUTION LIST ................................................................................................. 163
LIST OF FIGURES

1. The Probability of Detecting a Target ................................................................. 29
2. Linear Interpolation .............................................................................................. 32
3. Probability of Detection in the Next Second ..................................................... 38
4. Selection of Highest Priority Target .................................................................. 47
5. Bias and Dispersion Errors .................................................................................. 74
6. Attack Angle Bands Around the Target ............................................................... 82
7. Raw and Specific Probabilities of Kill ................................................................. 92
8. Lethality Fan Above Target ................................................................................. 99
9. Probability LOS exists for t Seconds or More ................................................ 129
10. Overlap of Visibility Segments for Disparate Viewers .................................... 132
11. Alternating in View and Out-of-View Segments ............................................. 135
12. Contents of a Link .............................................................................................. 142
13. The Initial Linked Lists ....................................................................................... 143
14. Scheduling an Event ......................................................................................... 144
15. Selecting the Next Event .................................................................................. 146
16. Cancelling an Event ........................................................................................... 147
17. Ten Pairs of Numbers Drawn from a Uniform Distribution .......................... 155
18. Draw of 20 Random Shots .............................................................................. 157
19. Cardioid Density and Distribution Functions ................................................... 158
INTENTIONALLY LEFT BLANK
LIST OF TABLES

1. Counting Tank Statuses ..................................................................................................................................... 24
2. Factors in Target Selection ............................................................................................................................... 49
3. Selection Priorities ............................................................................................................................................. 50
4. Times From Target Selection to Launch ........................................................................................................... 54
5. Horizontal Boundaries ...................................................................................................................................... 69
6. First Round Accuracy ........................................................................................................................................ 74
7. Stationary-Firer vs Moving-Target Accuracy .................................................................................................. 77
8. Stationary Firer vs Moving Target .................................................................................................................. 78
9. Add-on Dispersions for Moving Firers ............................................................................................................. 79
10. Moving Stationary Input Data (as a function of velocity) ................................................................................ 79
11. Moving Stationary Input Data (as a function of range) ................................................................................... 79
12. Cumulative Damage ........................................................................................................................................ 83
13. Lethality Data Stored in Table(4,12) ................................................................................................................ 91
14. Table Generated by MkTbl ............................................................................................................................. 94
15. Sample Lethality File ..................................................................................................................................... 96
16. Switching Logic for Bullets ............................................................................................................................ 103
17. Switching Logic for Simple Missiles .............................................................................................................. 104
18. Switching Logic for Multi-Target Missile System .......................................................................................... 105
19. Range Bands .................................................................................................................................................. 125
20. Time In-View for a Visual Band Sensor ........................................................................................................... 128
21. Find an Index .................................................................................................................................................. 154
INTENTIONALLY LEFT BLANK
ACKNOWLEDGEMENTS

I would like to thank Ute Abbott and Ingo Rucker, U.S. Army Ballistic Research Laboratory for their careful review of this report. Their suggestions for improving it were most helpful, and their assistance was thoroughly appreciated.
INTENTIONALLY LEFT BLANK
1. INTRODUCTION

This report describes in detail the construction of the Sustained Combat Model: Tank Wars II. It assumes you are familiar with the Users' Manual for Tank Wars.

Tank Wars II: The Sustained Combat Model is a computer simulation of sequential engagements between mechanized combatants; one side of which is not re-supplied. It is routinely used at various military installations and by government contractors for evaluating the combat effectiveness of tanks and other fighting vehicles. The systems being evaluated (usually US systems) defend against one or more waves of attackers without resupply, or on the attack, engage one or more defended positions without being resupplied.

Each engagement is simulated in detail. The critical events in such an engagement include search, detection, selection, acquisition, firing, impact, damage, target disengagement, and re-engagement. Interwoven with these events are motion events and intervisibility events. If desired, the program will print an event history for detailed study.

The model includes three types of engagements, two generic armaments, three categories of functional losses, and two types of false targets. Below is an extensive list of model features. The three scenarios are attack, defense, and a meeting engagement. Guns fire kinetic energy (KE), or high-explosive anti-tank (HEAT) rounds while missiles may be guided-to-impact or fire-and-forget systems. Systems may fire while moving or halt to fire. In either case they may suffer loss of mobility, firepower, or both and may be catastrophically killed. In addition to the weapons systems being evaluated, there may be a number of active or passive decoys and there are generally some false targets in the scenario.

MODEL FEATURES

<table>
<thead>
<tr>
<th>Armament/accuracy</th>
<th>Miscellaneous characteristics</th>
</tr>
</thead>
<tbody>
<tr>
<td>KE, HEAT, missiles (incl STAFF)</td>
<td>multiple kill levels</td>
</tr>
<tr>
<td>time of flight</td>
<td>pinpoint and non-firing detection</td>
</tr>
<tr>
<td>probability of sensing miss</td>
<td>various target disengagement policies</td>
</tr>
<tr>
<td>S-S accuracy depends on prev round</td>
<td>model widely used</td>
</tr>
<tr>
<td>Driver routines loop through</td>
<td>21 levels of tgt priority</td>
</tr>
<tr>
<td>scenarios</td>
<td>simple terrain</td>
</tr>
<tr>
<td>opening ranges</td>
<td>Program design</td>
</tr>
<tr>
<td>force ratios</td>
<td>event sequenced</td>
</tr>
<tr>
<td>multiple replications</td>
<td>time stepping for detection</td>
</tr>
<tr>
<td>waves of threat systems</td>
<td>stochastic</td>
</tr>
<tr>
<td>re-grouping</td>
<td>Fortran 77</td>
</tr>
<tr>
<td>Fire cycle characteristics modeled</td>
<td>modular</td>
</tr>
<tr>
<td>ammo consumption</td>
<td>structured</td>
</tr>
<tr>
<td>burst fire</td>
<td>indented code</td>
</tr>
<tr>
<td>first round time</td>
<td>3900 lines of code</td>
</tr>
<tr>
<td>subsequent round times</td>
<td>Scenario characteristics modeled</td>
</tr>
<tr>
<td>manual loading, load assist, automatic loading</td>
<td>multiple waves of Red threat systems</td>
</tr>
<tr>
<td>Intermediate output</td>
<td>multiple combatants</td>
</tr>
<tr>
<td>event histories</td>
<td>false targets</td>
</tr>
<tr>
<td>logic tracing</td>
<td>passive and active decoys</td>
</tr>
<tr>
<td>event tracing</td>
<td>Vehicle characteristics modeled</td>
</tr>
<tr>
<td>as scheduled, canceled, and retrieved</td>
<td>disjointed turret &amp; hull</td>
</tr>
<tr>
<td>Measures of effectiveness</td>
<td>cardioid or other aspects</td>
</tr>
<tr>
<td>ammo consumption</td>
<td>moving targets</td>
</tr>
<tr>
<td>blue tanks killed</td>
<td>fire on the move</td>
</tr>
<tr>
<td>red tanks killed</td>
<td>halt to fire</td>
</tr>
<tr>
<td>blue win probabilities</td>
<td>gross motion</td>
</tr>
<tr>
<td>red win probabilities</td>
<td>full defilade, hull defilade, fully exposed</td>
</tr>
<tr>
<td>exchange ratios</td>
<td>red tanks killed</td>
</tr>
</tbody>
</table>
blue win probabilities
red win probabilities
exchange ratios

Conventions. All units are meters, seconds, radians, or a combination of these unless otherwise noted.

The program uses a right handed cartesian coordinate system which is standard for test ranges and navigation systems. The x-axis is positive Eastward, the y-axis is positive Northward, and the z-axis is positive upward. Angles in the ground plane are measured clockwise from North.

A second coordinate system is target based, with its origin at the center of the turret ring or what passes for the turret ring. In this coordinate system, the x-axis is positive to the right of the firer, the y-axis is positive upward, and the z-axis is positive going from the target toward the firer.

Some conventions used in the program are:
A two space indentation is used to display organization
Constants are in upper case
Changes in the flow of execution are in upper case
Error messages begin with the routine name
1.1 Hierarchy of Routines. The diagram below shows the organization of the program. The routines in the first three columns are arranged in a hierarchy with called routines indented slightly beneath the calling routines. The routines in the last three columns are utility routines which may be called by many routines. They do not fit well into a hierarchy.

<table>
<thead>
<tr>
<th>Top Level</th>
<th>Events (cont)</th>
<th>Events (cont)</th>
<th>Model Utility</th>
<th>Time Adv.</th>
<th>Utility</th>
</tr>
</thead>
<tbody>
<tr>
<td>Main</td>
<td>Event</td>
<td>Impact</td>
<td>Abort</td>
<td>Reset</td>
<td>Anglef</td>
</tr>
<tr>
<td>Input</td>
<td>Finish</td>
<td>May hit</td>
<td>Diseng</td>
<td>Skedul</td>
<td>Angsum</td>
</tr>
<tr>
<td>Rd misc</td>
<td>Finish2</td>
<td>Acc err</td>
<td>Newtgt</td>
<td>Event</td>
<td>Cango</td>
</tr>
<tr>
<td>Pr misc</td>
<td>Search</td>
<td>Acc asb</td>
<td>Seleca</td>
<td>Cancel</td>
<td>Create</td>
</tr>
<tr>
<td>Rd eror</td>
<td>Searc2</td>
<td>Acc ssa</td>
<td></td>
<td></td>
<td>Interp</td>
</tr>
<tr>
<td>Rd pkh</td>
<td>Detect</td>
<td>Acc sa</td>
<td></td>
<td></td>
<td>Intrap2</td>
</tr>
<tr>
<td>Rd pkh2</td>
<td>Select</td>
<td>Acc sm</td>
<td></td>
<td></td>
<td>Nrgf</td>
</tr>
<tr>
<td>Waves</td>
<td>Priorn</td>
<td>In hit</td>
<td></td>
<td></td>
<td>Ranu</td>
</tr>
<tr>
<td>Reset</td>
<td>Engage</td>
<td>Bounds</td>
<td></td>
<td></td>
<td>Rann</td>
</tr>
<tr>
<td>Init</td>
<td>Fire</td>
<td>Damage</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Deploy</td>
<td>Pinpnt</td>
<td>Kill</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Det rg</td>
<td>Frd ssg</td>
<td>Kill5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Terrain</td>
<td>Frd bat</td>
<td>Damagm</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Smoke</td>
<td>Frd msl</td>
<td>Damagf</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Searc0</td>
<td>Reload</td>
<td>Late kl</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Events</td>
<td>Slow up</td>
<td>Hide</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stats1</td>
<td>Halt</td>
<td>Appear</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stats2</td>
<td>Accel</td>
<td>Vanish</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Stats3</td>
<td>Max vel</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The routines in column one are highest in the calling hierarchy and considered together control the execution of individual engagements. The number of times each is called varies. A few are called only once each time the program is run. At most, some are called twice per simulated engagement; once for each side in the engagement.

The routines in the second and third columns are called by the events routine. They are called many times during a single engagement to simulate the events which occur during a single engagement.

The routines in column four are model utility routines. They are special purpose routines which are called to provide information to other routines but which call no other routines and may be thought of as trig functions or other utility functions.

Column five lists the time advance routines. They or something similar must be included in any event stepped simulation.

Finally, column six lists general utility routines. They can be treated like trig or other library functions and are useful in non-simulation programs.
1.2 One Line Descriptions of Routines. Here is an alphabetical listing of the routines and one line descriptions of what they do.

Abort: Abort a missile in flight.
Acc err: Find the linear error for a single round.
Acc ms: Find angular accuracy for moving firer vs stationary target.
Acc sm: Find angular accuracy for stationary firer vs moving target.
Acc ss: Find angular accuracy for stationary firer vs stationary target.
Accel: Begin acceleration.
Anglef: Find the angle between two vectors.
Appear: Simulate or reschedule an appear event.
Aprsmk: Simulate target appearing from behind smoke.
Aprter: Simulate target appearing from behind terrain.
Bounds: Find the horizontal bounds of the hull or turret.
Can go: Find if tank is stopped but mobile.
Cancel: Cancel an event.
Confb: Find the confidence interval on a binomial outcome.
Create: Find space to store bullet data.
Creset: Clear stored bullet data.
Damag f: Simulate firepower damage.
Damag m: Simulate mobility damage.
Damage: Simulate damage to the target.
Deaths: Tallys deaths.
Deploy: Place combatants at start of engagement.
Det rg: Find maximum range to which each firer can detect.
Detect: Find if target is detected and schedule subsequent events.
Device2: Find the probability device 2 detects in the next second.
Diseng: Attempt to disengage 1 firer from 1 target.
Engage: Begin engagement of new target by this firer.
Event: Find next event.
Events: Call each event in sequence.
Eye: Find the probability the eye detects in the next second.
Fire: Simulate firing of a round and schedule effects.
Frdbst: Results of firing a round of a burst.
Frdmsl: Results of firing a missile.
Frdssg: Results of firing a single shot gun.
Halt: Simulate a tank halting.
Hide: Simulate tank hiding.
Impact: Find what bullet and firer do at impact.
Indexx: Find the index \( j \), where \( a(j) \leq x < a(j+1) \).
Init: Initialize for a single engagement.
Initnv: Generate detection probability tables.
Input: Read game control file.
Iz hit: Find if the target is hit.
Kill5: Find type of damage caused by top attack round.
Kill: Find type of damage caused.
Late kl: Simulate discard of inactive m&f-killed target.
Main: Simulate armored combat.
Max vel: Simulate tank reaching combat cruise speed.
Mayhit: Find whether the round hits.
Mk tbl: Make table of head-on lethality data.
Newtgt: Redirect all foe to a new target.
Nvl: Find probability of ever detecting and of detecting in next second.
Path: Find position and velocity of combatant.
Pinpnt: Simulate firing signature detection.
Pop dn: Simulate defender popping down to reload missile pods.
Pr misc: Print miscellaneous tank characteristics.
Priorn: Select target with highest priority.
Priort: Find priority of a single target.
Ran Ang: Draw a random angle from a cardioid or other distribution.
Rann: Draw a random number from a normal distribution.
Ranu: Draw a random number from the standard uniform distribution.
Rd error: Read accuracy data for one side.
Rd misc: Read miscellaneous tank characteristics.
Rd pkh2: Read HEAT and missile lethality data.
Rd pkh5: Read top attack lethality data.
Rd pkh: Read standard lethality data.
Rd smk: Read intervisibility data for smoke.
Reload: Bring up another pod of missiles.
Reset: Re-initialize the event list.
Rgf: Find range to target, relative position, and velocities.
Schedule: Schedule an event.
Searc0: Schedule initial search event.
Searc2: Find if one searcher detects one target in the next second.
Search: Find targets detected in next second.
Select: Start target selection if appropriate.
Select: Gunner chooses most dangerous target it sees.
Slow up: Begin deceleration.
Smoke: Find when smoke will stop blocking LOS between searchers and targets.
Stats1: Print summary statistics.
Stats2: Print statistics for a single wave.
Stats3: Print statistics for a single engagement.
TDIntp: Interpolate in a two dimensional matrix.
Terrain: Find path lengths where attacker is masked by terrain.
Vanish: Simulate or reschedule vanish event.
Vansmk: Simulate target vanishing behind smoke.
Vanter: Treat target vanishing behind terrain.
Waves: Loop through waves of red tanks.
1.3 Data Structure. The files: common.h and clock.h and the blkdat routine contain code defining global variables. This section discusses common.h and blkdat. The clock.h file is discussed in a later section with the other clock routines.

The following values are communicated to routines via block common statements which are in the common.h file. Constants are all upper case. Normally integers begin with the letters i..n, and reals begin with the remaining letters. If this is not the case, the definitions below tell whether the variable takes an integer, real, logical, or character value.

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ACCELG</td>
<td>= 3 Identifies tank that is accelerating. integer</td>
</tr>
<tr>
<td>ALIVE</td>
<td>= 1 Implies fully functional. integer</td>
</tr>
<tr>
<td>ALL</td>
<td>= 0 Used to schedule or cancel an event for all firers or targets. integer</td>
</tr>
<tr>
<td>BATTAK</td>
<td>= 3 Blue attack scenario. integer</td>
</tr>
<tr>
<td>BLU</td>
<td>= 1 # of blue side. integer</td>
</tr>
<tr>
<td>DEG</td>
<td>= 57.29577951 Degrees/radian.</td>
</tr>
<tr>
<td>FD</td>
<td>= 1 Full defilade. integer</td>
</tr>
<tr>
<td>FE</td>
<td>= 3 Fully exposed. integer</td>
</tr>
<tr>
<td>FKILL</td>
<td>= 3 Firepower kill. integer</td>
</tr>
<tr>
<td>FLSTGT</td>
<td>=-1 ID # for false targets. integer</td>
</tr>
<tr>
<td>HD</td>
<td>= 2 Hull defilade. integer</td>
</tr>
<tr>
<td>HULL</td>
<td>= 2 Identifies hull box. integer</td>
</tr>
<tr>
<td>IKILL</td>
<td>= 5 M&amp;F kill &amp; recognized to be inactive.</td>
</tr>
<tr>
<td>KKILL</td>
<td>= 6 Catastrophic kill.</td>
</tr>
<tr>
<td>MAXVL</td>
<td>= 4 Identifies tank moving at maximum combat velocity.</td>
</tr>
<tr>
<td>MEETNG</td>
<td>= 1 Meeting scenario.</td>
</tr>
<tr>
<td>MFKILL</td>
<td>= 4 Mobility &amp; firepower kill.</td>
</tr>
<tr>
<td>MKILL</td>
<td>= 2 Mobility only kill.</td>
</tr>
<tr>
<td>NN</td>
<td>= Maximum number of tanks playable.</td>
</tr>
<tr>
<td>NULL</td>
<td>= 0 No specific target associated with firer event.</td>
</tr>
<tr>
<td>PI</td>
<td>= 3.141592654 (π).</td>
</tr>
<tr>
<td>RATTAK</td>
<td>= 2 Red attack scenario. integer</td>
</tr>
<tr>
<td>RED</td>
<td>= 2 # of red side. integer</td>
</tr>
<tr>
<td>SLOWNG</td>
<td>= 1 Identifies tank that is slowing up. integer</td>
</tr>
<tr>
<td>STATNY</td>
<td>= 2 Identifies tank that is stationary (halted). integer</td>
</tr>
<tr>
<td>TURRET</td>
<td>= 1 Turret of tank. integer</td>
</tr>
<tr>
<td>TWOPI</td>
<td>= 6.283185308 (2π).</td>
</tr>
<tr>
<td>a(1000)</td>
<td>See routine create. Storage for temporary entities.</td>
</tr>
<tr>
<td>accel(2)</td>
<td>Acceleration of ith side (m/s**2).</td>
</tr>
<tr>
<td>acmax(2)</td>
<td>Maximum lateral acceleration (m/s**2).</td>
</tr>
<tr>
<td>ampl(2)</td>
<td>Amplitude of sinusoidal path (m).</td>
</tr>
<tr>
<td>angle</td>
<td>No longer used. Used to input crossing angle of attacker.</td>
</tr>
<tr>
<td>army(NN)</td>
<td>Army(i) is 1 if ith tank is Blue and 2 if it is Red. integer</td>
</tr>
<tr>
<td>busy(NN)</td>
<td>busy(i)=T iff ith tank is too busy to select a new tgt. logical</td>
</tr>
<tr>
<td>channel(2,NN,5)</td>
<td>channel(i,j,k) contains ID of missile for ith side, jth tank, kth guidance channel.</td>
</tr>
<tr>
<td>color(2)</td>
<td>color(n) is the color of the nth army. character^4</td>
</tr>
<tr>
<td>decel(2)</td>
<td>Deceleration of ith side (m/s**2).</td>
</tr>
<tr>
<td>empty(NN)</td>
<td>empty(i)=T iff out of ammo or empty missile pod. logical</td>
</tr>
<tr>
<td>fot(NN)</td>
<td>fot(i)=T iff ith tank is occupied firing on a target. logical</td>
</tr>
<tr>
<td>history</td>
<td>=T iff event history will be printed. logical</td>
</tr>
</tbody>
</table>
iangd = 1 if using cardioid distribution of aspect angles, 2 if frontal distribution.
idecoy(NN) idecoy(i) = T iff ith tank is a decoy.
iflash(NN) iflash(i) = T iff ith tank is a flashing decoy.
iholy See create. Index of storage space being examined.
invisb = 1 for terrain, 2 for smoke. Cannot use both smoke terrain.
irandm Random number seed.
ishtfs(2) Side is halt to fire. logical
keym(20) = T iff print statement should be executed.
kindrd(2) Side's kind of round.
kneal(NN) kneal(i) is 1 if full defilade, 2 if hull defilade, 3 if fully exposed.
kshot(2) Count of shot results.
kvview(2) kvview(i) = 1 for visual detection, 2 for thermal viewer.
life(NN) life(i) is 1 if ALIVE, 2 if MKILL, 3 if FKILL, 4 if M&FKILL, 5 if IKILL, 6 if KKILL.
loader(2) loader(i) = 1 for manual loader, 2 for load assist, 3 for autoloader.
los(NN,NN) los(i,j) = T iff firer i has line-of-sight to target j.
methsm 1 iff interpolation for crossing angle desired.
mot(NN) mot(i) = T iff ith tank is guiding missile to tgt. logical
motion(NN) motion(i) is 1 if slowing, 2 if stationary, 3 if accelerating, 4 if max combat speed.
nblu # blue tanks.
nbrst(NN) nbrst(i) is # rounds fired by ith tank during current burst.
nbump(2) # of rounds to fire at MF killed tgt before discarding it.
nchan((NN)) # guidance channels for ith side.
nchans(2) # guidance channels for ith side.
nmule # decoys on ith side.
nncri(2) nncri(i) is # rounds in missile pod for ith side.
nprior(2) nprioir(i) = 1 if ith side using priority scheme 1, 2 if using scheme 2.
nrd(20) # rounds fired by ith tank.
nrds(2) # rounds on system for ith side.
nred # red tanks.
nreps # replications.
nrg # of range band. if irginc=500, nrg = 1 for data at 500 meters, 2 at 1000 meters, etc.
nrib(2) # rounds in a burst for ith side.
nrot(NN) # rounds on target for ith tank.
nrpb(2) # rounds per burst for ith side.
nrpt(2) # rounds per target for ith side.
nrui(2) ID # of ith tank's target.
nused(3000) # rounds fired by tank.
nwaves # waves.
pdet(2,3,10) pdet(i,j,k) is probability of detecting in next second for ith side vs tgt in condition j, at kth range.
pfalse(2,2) pfalse(i,1) is prob ith side selects false ID tgt. pfalse(i,2) is for FE tgt.
pfin(2,3,10) Prob tank on ith side detects tgt in cond j at range band k.
pinp(2) Probability of pinpoint detection for ith side.
prevrd(NN) 1 if 1st rd on tgt, 2 if previous was a hit, 3 if previous was sensed miss, 4 if previous was lost miss.

psense(2,8) Probability of sensing miss for tank.
reckn(2) reckn(i) is range cutoff for target selection routine.
reliab(2) Probability round is reliable for ith side.
repeat Repeat search code for all tanks.
rex Random error.
rey Random error.
rg Range from firer to target (m).
rg0 Opening range for engagement (m).
rgincr Range increment in tables (m). Usually 500 meters.
rgvis(3,NN) rgvis(i,j) is range to which jth tank can detect target in ith condition.
rof(2) Rate of fire for ith side.
scene =1 for meeting, 2 for Red attack, 3 for Blue attack.
see(NN,NN) Tank i has seen target j.
share(2) share(i)=T iff side i shares info re targets & spreads fire evenly. logical
speed(2) Combat speed of ith side (m/s).
sysdim(2,8) Dimensions of tanks on side.
t0(NN) Last time position and velocity were updated (sec).
tactic(2) Disengagement tactic for ith side. integer
tbump(2) Time for ith side to bump MF killed tgt to inactive.
tcon(2) Constant time for loaders - for time between rds.
tfire(NN) Time tank fired last.
tfire2(NN,NN) Time tank fired at target last.
thist(2,8) Median time to fire first round for ith side.
thide(2) Time to hide for ith side (sec).
tini(21,5) Table of in view segment lengths for infrared sensors.
tinv(21,5) Table of in view segment lengths for visible band sensors.
tlook(2) Time tank will look before re-engaging old target.
tmax Maximum time (sec). Used to cut off a replication.
tof(2) Time of flight for ith side.
tout(21,5) Table of out-of-view segment lengths for infrared sensors.
toutil(21,5) Table of out-of-view segment lengths for infrared sensors.
toutv(21,5) Table of out-of-view segment lengths for visible band sensors.
toutvl(21,5) Table of first out-of-view segment lengths for visible band sensors.
trace Prints entry to and exit from routines iff true. logical
trelod(2) Time to reload (next missile pod) for ith side.
tvar(2) Median variable time for loaders - for time between rds.
vxO(NN) vxO(i) is the last computed speed of the ith tank. (m/s).
vxO(NN) Velocity of tank.
vWlth(2) Wave length of sinusoidal path (m).
xO(NN) xO(i) is the last computed Easting coordinate of ith tank. (in).
yO(NN) Position of tank.

The common.h file and the blkdat routine are listed below. The data statements in the blkdat routine set values for the constants in labeled common. While many compilers do not require these data statements to be placed in a separate block data routine, the Fortran 77 standard and the current Microsoft Fortran compiler do.
common /const2/ ALL, NULL, FLSTGT,
1 FD, HD, FE, TURRET, HULL, BLU, RED, MEETNG, RATTAK,
2 BATTAK, ALIVE, MKILL, FKILL, MFKILL, IKILL, KKILL,  
3 SLOWNG, STATNY, ACCELG, MAXVL
integer scene, army
common /scene, army
const2/ nreps, keym(20), scene, tmax, mth sm
common /eshot/, kshot(2,20)
logical trace, histry
common /trace/, trace, histry
common /sys/, nblu, nred
common /states/, army(NN), life(NN), nrtgt(NN)
c Less used:
common /aspekt/, iangd
common /store/, a(1000), iholy
common /vars6/, irginc, ngincr
c Vehicle:
common /endgam/, sysdim(2,8)
common /state2/, idecoy(NN), iflash(2), ndeco(2), nflash(2)
c Round:
common /round/, kindrd(2), nrd(2), nd(2), relia(2),
1 nipods(2), nipod(2), nflash(2), nrt(2,8)
c Detection:
logical los, see, repeat
1 integer prevrd
common /xx/, invisb, kview(2), ndets(2), ndet(NN),
1 pinp(2), look(2), los(NN,NN),
2 knocal(2), prevrd(2,8), rgvis(3,NN), see(2,NN,NN)
c common /sensor/, psm(2,8), pinp(2,3,10), pdet(2,3,10).
1 repeat
1 c Selection:
logical busy, fot, mot, share
common /choose/, busy(2), fot(2,NN), mot(2,NN,NN),
1 nprior(2), pfals(2,2), reknz(2), share(2,2), tfire(2,NN), tfire2(2,NN)
2 c Fire cycle:
integer chanel
logical empty
1 common /cycle/, ishft(2,8), locader(2), tfirst(2,8),
1 rot(2), tvar(2), teon(2), nbs(2,NN), empty(2,NN),
2 chanel(2,NN,8), ncchan(2), ncchan(2)
c Target discard:
integer tactic
common /policy/, tactic(2), nrot(2), nrot(NN), nhrot(2,NN),
1 tbump(2), nbump(2)
c Motion:
common /path/, acc(2), dec(2),
1 sped(2), angle(2), acmax(2), v(2), v(2)
1 motion(NN), to(2,NN), x0(2,NN), y0(2,NN), v(2,NN)
2 common /where2/, nrg, rg0, rg, s(3)
c common /yy/, thide(2)
c V7.4
BLOCK DATA BLKDAT
include 'common.h'
data color, pi, two, deg
1 'Blue', 'Red', 3.141592654, 6.283185308, 57.29577951,
data ALL, NULL, FLSTGT /0, 0, -1/
data FD, HD, FE /1, 2, 3/
data TURRET, HULL /1, 2/
data BLU, RED /1, 2/
data MEETNG, RATTAK, BATTAK /1, 2, 3/
data ALIVE, MKILL, FKILL, MFKILL, IKILL, KKILL /1,2,3,4,5,6/
data SLOWNG, STATNY, ACCELG, MAXVL
1 /1,2,3,4/
data keym /20*0/ END

9
2. TOP LEVEL ROUTINES

The Tank Wars routines are in a hierarchy. The routines discussed in this section are at the top of the hierarchy. They read input, loop through parameters, and produce summary statistics, rather than dealing with the specifics of a single engagement. Main, and waves take the burden of changing parameters off the user. They simply read the input and vary parameters. Then with the parameters set, execute the combat model proper. The parameters varied are the scenario, the number of tanks on each side, the opening range, the number of threat units (Red) met, and the number of replications.

The diagram below shows the relationship between the driver routines discussed in this section. Each routine is called by the one directly above it.

Input calls several routines not shown in the diagram which read lethality and accuracy data. They are discussed in the sections describing the lethality and accuracy routines.

Init also calls several routines not shown in the diagram. They initialize event routines and since they are conceptually linked with those routines they are discussed with them.

Events is called once for each engagement. It is at the top of the hierarchy for all remaining routines. They will be discussed in later sections.

![Diagram of routine hierarchy]

11
2.1 Main: Simulate Armored Combat. Main is where the simulation starts; it is the main program at the top of the routine hierarchy. It prints the version header, and calls the input routines, executing perhaps 1,000 replications for each scenario and opening range. Then it loops through the scenarios and the set of opening ranges. Main is small so that code controlling other interesting parameters can be added for parametric studies.

The end of the Game file controls the opening ranges and the scenarios to be played. Suppose the game file ends with the following data:

<table>
<thead>
<tr>
<th>Game file</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000, 3000, 500</td>
<td>min range, max range, range increment</td>
</tr>
<tr>
<td>1, 5, 5</td>
<td>scene, #blue, #red</td>
</tr>
<tr>
<td>2, 4, 12</td>
<td>scene, #blue, #red</td>
</tr>
<tr>
<td>3, 12, 4</td>
<td>scene, #blue, #red</td>
</tr>
</tbody>
</table>

The first line shown controls the opening range for the battle. It contains a minimum range, maximum range, and a range increment. If, for example, the line contains these values: 1000,3000,500, then Tank Wars would simulate combat at 1000, 1500, 2000, 2500, and 3000 meters opening range.

The lines following this (three in the example) describe three scenarios to be played. Each of these lines contain the scene, the number of Blue combatants, and the number of Red combatants. The codes for the scene are as follows:

| Scene = 1 | A meeting engagement                        |
| Scene = 2 | A Red attack (the threat system)            |
| Scene = 3 | A Blue attack (the system being evaluated)  |

For our example, Tank Wars would simulate a meeting engagement between 5 Blue and 5 Red tanks at each of the opening ranges. It would simulate perhaps 1000 replications for each opening range. Then it would repeat the process for a 12 Red tanks attacking 4 Blue attacks. Finally, it would repeat the process for 12 Blue tanks attacking 4 Red tanks. It then runs out of scenario data and quits.

Code.

c V7.10
MAIN ROUTINE

character string(3)*11
include 'common.h'
format( ' #Blues=',j3,' #Reds=',i3,2x,a)
data string /'Meeting','Red attack','Blue attack'/
print*, 'The Sustained Combat Model: Tank Wars II'
print*, 'by Fred Bunn, ph (301) 278-6676, autovon 298-6676'
print*, 'Ballistic Research Laboratory'
print*, 'Aberdeen Proving Ground, MD 21005'
print*, 'Version 7.10 Created 4/2/89'
call input
read*, minrg, maxrg, incr
CONTINUE
read(5, IOSTAT=i) scene, nblu, nred
if (io.gt.0) print*, 'End of run.'
if (io.gt.0) print*/'Can' t read data.'
IF (io.ne.0) STOP
print 3, nblu, nred, string(scene)
DO 30 irg = minrg, maxrg, incr
   rg0 = irg
   call waves
30 CONTINUE
2.2 Waves: Loop Through Waves of Red Tanks. Waves initializes and computes summary statistics and loops through one or more sets of replications. The summary statistics are generated by the subsidiary subroutines; \texttt{state1}, \texttt{state2}, and \texttt{state3}. Later sections discuss these routines and the summary statistics. Normally, waves loops through a single set of engagements each time it is called. This set may be 1,000 replications of a case, where a case is a single scenario with a single opening range. However, the user may specify multiple sets where each set of engagements represents unresupplied Blue tanks in combat against a ‘wave’ of fresh Red tanks.

The waves routine was developed to analyze ammo consumption. If you aren’t interested in that, just set the number of waves to one. The routine initially pits \( N \) blue against \( M \) red and does this for perhaps 1,000 replications. This completes the simulation of the first wave of Reds. It then regroups all survivors with ammo in groups of \( N \) blues and pits them against \( M \) reds and does this say 500 times. This regrouping continues until not enough Blues are left to form a group or the routine completes \texttt{nwaves} of waves.

Normally, Blue tanks with many cannon rounds will win or lose before hardly any of them run out of ammo. This, of course, would not be the case if the Blue systems are armed with just a few missiles; many of them might run out of ammo before the engagement ends.

Waves checks to see if multiple waves of Red tanks are being simulated and if there will be sufficient space to record the ammo expended by the Blue tanks. If there’s not enough space and multiple waves are being simulated, waves prints a message and skips the current case. Otherwise, it runs the current case. For example, if \texttt{nwaves} > 1, multiple waves are being simulated. If the number of replications for the first wave is \texttt{nreps}=1000, and if the number of blue combatants in an engagement is \texttt{nblu}=4, waves will skip this case because only 3,000 ammo consumption values can be saved but there may be as many as 4,000 survivors whose ammo consumption must be recorded.

If there’s no problem recording ammo consumption, the code proceeds to run the current case. It initializes scenario statistics, executes multiple replications of the engagement and generates summary statistics.

Key variables and relations are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>nreps</td>
<td># replications (engagements) to simulate during the first wave.</td>
</tr>
<tr>
<td>nblu</td>
<td># of Blue tanks in an engagement.</td>
</tr>
<tr>
<td>nwaves</td>
<td>Maximum # of waves of Red units Blue systems will combat.</td>
</tr>
<tr>
<td>kshot(2,20)</td>
<td>Table of shot outcome statistics.</td>
</tr>
<tr>
<td>kount(2,20)</td>
<td>Table of survivor statistics.</td>
</tr>
<tr>
<td>nused(3000)</td>
<td>Array of ammo consumption for fully functional Blue survivors.</td>
</tr>
<tr>
<td>nreps3</td>
<td>Total # replications fought in all waves.</td>
</tr>
<tr>
<td>nreps2</td>
<td># replications fought in current wave.</td>
</tr>
<tr>
<td>nsurv</td>
<td># fully functional Blue survivors in current wave.</td>
</tr>
<tr>
<td>nwave</td>
<td># of current wave.</td>
</tr>
<tr>
<td>iseed2</td>
<td>Saves starting random number seed to print with summary statistics.</td>
</tr>
<tr>
<td>nrg</td>
<td>Range band. =1 for opening range near 500 meters, 4 near 2000 meters, etc.</td>
</tr>
<tr>
<td>iseed3</td>
<td>Saves starting random number seed of wave.</td>
</tr>
<tr>
<td>keyd(5).gt.0</td>
<td>Implies events scheduled, canceled, and executed should be printed when scheduled, canceled, or executed.</td>
</tr>
<tr>
<td>nsurv.lt.nblu</td>
<td>True when not enough fully functional survivors to send \texttt{nblu} tanks into the next engagement.</td>
</tr>
<tr>
<td>nwave.eq.nwaves</td>
<td>True when the current wave equals the maximum # of waves desired.</td>
</tr>
<tr>
<td>iuse</td>
<td># of survivors available for regrouping during later replications of the current wave.</td>
</tr>
</tbody>
</table>
SUBROUTINE WAVES

Waves: loop thru waves of red tanks.

include 'common.h'

common /crandm/ irandm
common /inpwav/ keyd(6), nwaves, neval, nused(3000)
integer kount(2,20)
logical done

if (trace) print *, '>waves'

IF (nreps*nblu.gt.3000 .and. nwaves.gt.1) THEN
  print *, 'WAVES: Too many reps or blues.', nreps, nblu
ELSE
  Initialize scenario statistics
  DO 10 i=1,20
      kshot(1,i) = 0
      kshot(2,i) = 0
      kount(1,i) = 0
      kount(2,i) = 0
  10 CONTINUE
  DO 30 i=1,3000
      nused(i) = 0
  30 CONTINUE
  print*', 'Starting seed=', irandm
  nreps2 = nreps
  nsurv = 0
  nwave = 0
  40 CONTINUE

Loop thru up to n waves of red tanks
  nwave = nwave+1
  iseed2 = irandm
  nrg = rg0/irginc
  DO 50 i=6,20
      kount(2,i) = 0
  50 CONTINUE
  DO 60 i=1,nreps2
      iseed3 = irandm
      call reset(keyd(5).gt.0)
      call creset
      call init
      call events
      call stats3 (keyd(1),nrep,kount,iseed3,nsurv,nused)
  60 CONTINUE

Simulate a single engagement (replication).
    iseed3 = irandm
    call reset(keyd(5).gt.0)
    call creset
    call init
    call events
    call stats3 (keyd(1),nrep,kount,iseed3,nsurv,nused)
  80 CONTINUE

Update statistics and see if all waves are done.
  nreps3 = nreps+nreps2
  call stats2(nwave,nwaves,nreps2,kount)
  done = nsurv.lt.nblu .or. nwave.eq.nwaves
  IF (done) GOTO 80

Find #reps, #Blue tanks, #unused Blues for next wave.
  nreps = nsurv/nblu
  iuse = nreps2*nblu
  kount(1,1) = kount(1,1)-iuse
  nsurv = nsurv-iuse
  GOTO 40

ENDIF

IF (trace) print *, '<waves'
END
2.3 Events: Call Each Event in Sequence. Events is the heart of the program because it controls the simulation of single engagements. It loops until it finds a finish event. Until then, it finds the most imminent event on the event list and branches to the appropriate event routine. The event routine then simulates the event. The process of finding the next event and executing it continues until a finish event occurs.

Events is a simple loop which calls event to return the next event and branch to one of the many events.

Key variables are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>what</td>
<td>The event name. character*6</td>
</tr>
<tr>
<td>who</td>
<td>The ID of the tank (or bullet). integer</td>
</tr>
<tr>
<td>whom</td>
<td>The ID of the target (if any). integer</td>
</tr>
<tr>
<td>t</td>
<td>Simulated time (sec).</td>
</tr>
</tbody>
</table>

Code.

```
SUBROUTINE EVENTS
  include 'common.h'
  character*6 what
  integer who, whom
  CONTINUE
  call event (who, what, whom, t)
  IF (what.eq.'search') THEN
    call search (t)
  ELSEIF (what.eq.'vanish') THEN
    call vanish (t,who,whom)
  ELSEIF (what.eq.'appear') THEN
    call appear (t,who,whom)
  ELSEIF (what.eq.'detect') THEN
    call detect (t,who,whom)
  ELSEIF (what.eq.'select') THEN
    call select (t,who)
  ELSEIF (what.eq.'fire ') THEN
    call fire (t,who,whom)
  ELSEIF (what.eq.'impact') THEN
    call impact (t,who)
  ELSEIF (what.eq.'damage') THEN
    call damage (t,who,whom)
  ELSEIF (what.eq.'slowup') THEN
    call slowup (t,who)
  ELSEIF (what.eq.'halt ') THEN
    call halt (t,who)
  ELSEIF (what.eq.'accel ') THEN
    call accel (t,who)
  ELSEIF (what.eq.'maxvel') THEN
    call maxvel (t,who)
  ELSEIF (what.eq.'kill ') THEN
    call kill (t,who,whom)
  ELSEIF (what.eq.'hide ') THEN
    call hide (t,who)
  ELSEIF (what.eq.'reload') THEN
    call reload (t,who)
  ELSEIF (what.eq.'popdn ') THEN
    call popdn (t,who)
  ELSEIF (what.eq.'finish') THEN
    GOTO 90
  ELSE
    print*,'EVENTS: what=',what,' who=',who,' whom=',whom,' time=',t
    print*,'Contact Fred Bunn'
    STOP
  ENDIF
  GOTO 10
  GOTO 90
```
2.4 Input: Read Game Control File. This routine reads game control information from what we call the Game File which is assumed to be the standard input (Fortran unit 5). The Game File lists other files that contain accuracy, lethality, and miscellaneous data for the Blue and Red tanks. When the input routine reads these file names, it calls lower level routines to read these files.

For the most part the values read here are described in the earlier section entitled: Data Structure. They are further described in the *The Sustained Combat Model: Tank War II Users' Manual*.

Code.

```fortran
SUBROUTINE INPUT
    include 'common.h'
    character*32 fname
    integer indx(S)
    common /crandm/ irandm
    common /inpwav/ keyd(S), nwaves, neval, nused(3000)
    format(1,1x,a32)
    format(2x,,fG.2)
    read(5,*)keyd(i),i=1,5
    trace=keyd(4).gt.0
    history=keyd(1).ge.2
    iecho=keyd(2)
    read(5,*),INDX
    DO 20 i=1,5
         IF (indx(i).gt.1 .and. indx(i).le.20) keym(indx(i))=1
    CONTINUE
    read(5,*) ,irginc
    rgincr=irginc
    read(5,*) ,nreps, nwaves, iangd, METHSM, irandm
    read(5,*) ,tmax
    read *, ,sun
    read(5,1) k, fname
    invib = k
    IF (iecho.gt.0) THEN
       print *, 'ENVIRONMENT:'
       print 2, 'Illumination is ',sun,' ft-candles.'
       if (iangd.eq.1) print *, 'Using cardioid distribution.'
       if (iangd.eq.2) print *, 'Using frontal distribution.'
       print *, 'Rg increment for all tables is ',irginc,' metres.'
    ENDIF
    IF (k.le.1) THEN
       print *, 'Terrain parameters are hardwired now'
    ELSE
       print *, 'Playing smoke'
       call rdsmk(fname)
    ENDIF
    read(5,1) k, fname
    call rdmisc(fname,BLU,sun,iecho)
    read(5,1) k, fname
    call rderror(fname,BLU,iecho)
    Read pkh data for Blue.
    read 1, k, fname
    if (k.eq.1) call rdpkh1(fname,BLU,iecho)
    if (k.eq.2) call rdpkh2(fname,BLU,iecho)
    if (k.eq.5) call rdpkh5(fname,BLU,iecho)
    read(5,1) k, fname
    call rdmisc(fname,RED,sun,iecho)
    read(5,1) k, fname
    call rderror(fname,RED,iecho)
    Read pkh data for Red.
    read 1, k, fname
    if (k.eq.1) call rdpkh1(fname,RED,iecho)
    if (k.eq.2) call rdpkh2(fname,RED,iecho)
    if (k.eq.5) call rdpkh5(fname,RED,iecho)
    if (trace) print *, '<input'
END
```
2.5 Rdmisc: Read Miscellaneous Tank Characteristics. This routine simply reads data values and if an echo is desired, calls the prmisc routine. It then prints the name of the miscellaneous data file used.

An earlier section entitled: Data Structure, and the Users' Manual describes the variables read by rdmisc.

c
V7.7
SUBROUTINE RDMISC (fname,n,sun,iecho)
c 0 Rd misc: read miscellaneous tank characteristics.
c fname - file name.
c n - # of side. Blue=1, Red=2
include 'common.h'
character fname*32
real high(2)
c
if (trace) print *, '>Rdmisc'
open(4, file=fname, status='old')
rewind 4
c Read vehicle characteristics.
read(4, *) (sysdim(3-n,i),i=1,8)
read(4, *) ndecoy(n), nflash(n)
c Read round characteristics.
read(4, *) kindrd(n), nrd(n), reliab(n)
read(4, *) (tof(n,i),i=1,8)
c Read acquisition characteristics.
read(4, *) kview(n), ndet(n)
read(4, *) pfase(n,1), pfase(n,2), pin(n)
read(4, *) share(n)
c Read target selection criteria.
read(4, *) nprior(n), reck(n)
c Read fire cycle characteristics.
read(4, *) nhfs(n), loader(n), tcon(n), tvar(n)
read(4, *) nchan(n)
read(4, *) (ftfirst(n,i),i=1,8)
read(4, *) nrnb(n), rof(n)
read(4, *) nipods(n), treod(n)
c Read disengagement policy.
read(4, *) ibump, nbump(n)
tbump(n) = ibump
read(4, *) tactic(n), nrpt(n), tlook(n)
c Read motion characteristics.
read(4, *) accel(n), decel(n), speed(n), thide(n)
close (4)
high(1) = sysdim(3-n,1)
high(2) = high(1) + sysdim(3-n,5)
call initnv(n,kview(n),sun,high)
if (iecho.gt.0) call prmisc(n)
print *, 'Misc file is:',fname
if (trace) print *, <rdmisc'
END
2.8 Prmisc: Print Miscellaneous Tank Characteristics. This routine simply prints the miscellaneous data with labels. It documents the values read in and allows the user to verify that the input was correctly prepared.

c V7.9

SUBROUTINE PR MISC(n)

i if (k.eq.2) print 4, 'Selects new tgt over old hit tgt.'
if (k.eq.1) print 4, 'Selects old, hit tgt over new tgt.'

if (k.eq.2) print 4, 'Selects new tgt over old hit tgt.'
if (k.eq.1) print 4, 'Selects old, hit tgt over new tgt.'

SUBROUTINE PR

1 CONTINUE

if (k.eq.1) print 4, 'Selects old, hit tgt over new tgt.'

if (k.eq.2) print 4, 'Selects new tgt over old hit tgt.'
if (k.eq.1) print 4, 'Selects old, hit tgt over new tgt.'

print*, 'ACQUISITION CHARACTERISTICS:'

print*,'FIRE CYCLE CHARACTERISTICS:'

print*, 'TARGET SWITCHING CRITERIA:'

print*,'TARGET SELECTION CRITERIA:'

k=prnror(n)

END
2.7 Init: Initialise for a Single Engagement. Init initializes many variables to the values appropriate to the beginning of an engagement. It then calls specialized initialization routines. These lower level routines are tightly coupled to other routines and will be discussed in the appropriate sections.

The diagram below shows the relationship of the various initialization routines. Those in dashed boxes are discussed elsewhere.

![Diagram](image)

First, the code schedules a `finish` event to make sure the engagement ends in a reasonable time.

Then values are set for each tank and tank/target pair as shown in the table below.

<table>
<thead>
<tr>
<th>CODE</th>
<th>INITIAL VALUE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy</td>
<td>F</td>
<td>FOR EACH TANK 1</td>
</tr>
<tr>
<td>empty</td>
<td>F</td>
<td>Tank is not busy. (busy inhibits selecting a new target)</td>
</tr>
<tr>
<td>idecoy</td>
<td>0</td>
<td>Tank is not a decoy.</td>
</tr>
<tr>
<td>iflash</td>
<td>0</td>
<td>Tank is not a flashing decoy.</td>
</tr>
<tr>
<td>life</td>
<td>ALIVE</td>
<td>Tank is mobile and lethal</td>
</tr>
<tr>
<td>ndet</td>
<td>0</td>
<td>Tank has detected zero targets.</td>
</tr>
<tr>
<td>nhot</td>
<td>0</td>
<td>Tank has achieved no hits on a target.</td>
</tr>
<tr>
<td>nrd</td>
<td>0</td>
<td>Tank has fired zero rounds.</td>
</tr>
<tr>
<td>nrtgt</td>
<td>0</td>
<td>ID of target is NULL.</td>
</tr>
<tr>
<td>nchan</td>
<td>0</td>
<td>Tank has no guidance channels busy.</td>
</tr>
<tr>
<td>nrot</td>
<td>0</td>
<td>Tank has achieved no rounds on target yet.</td>
</tr>
<tr>
<td>tfire2</td>
<td>0</td>
<td>Time tank last fired is NULL.</td>
</tr>
<tr>
<td>los</td>
<td>F</td>
<td>CLEAR NN by NN MATRICES</td>
</tr>
<tr>
<td>missed</td>
<td>F</td>
<td>Line-of-sight does not exist.</td>
</tr>
<tr>
<td>mot</td>
<td>F</td>
<td>Tank has not missed target.</td>
</tr>
<tr>
<td>fot</td>
<td>F</td>
<td>Firer has no missile assigned to target.</td>
</tr>
<tr>
<td>see</td>
<td>F</td>
<td>Firer is not engaging target.</td>
</tr>
<tr>
<td>tfire</td>
<td>F</td>
<td>Firer doesn’t see target.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Time tank last fired at tgt is NULL.</td>
</tr>
</tbody>
</table>
CODE | INITIAL VALUE | COMMENT
--- | --- | ---
chanel | 0 | FOR EACH TANK'S GUIDANCE CHANNEL
                  Channel assigned to guide NULL missile.
los | T | FOR EACH BLUE-RED PAIR
                  Line-of-sight exists between foes.

Next, the code calls `deploy` to define the initial position, velocity, side, and cover for each tank. Then it calls `destr` to find how far each tank can see. After that, it calls `terrain` and `smoke` to set up terrain and smoke conditions. Next, it resets some values if some of the systems are decoys. Finally, it calls `search` so that each tank will begin searching for targets.

c 7.5
SUBROUTINE INIT
1 = nblu+j
idecoy(i) = 1
if (j.le.fflash(RED)) iflash(i) = 1
if (j.gt.fflash(RED)) ard(i) = 999
60 CONTINUE

END
2.8 Statal: Print Summary Statistics. Statal generates and prints the final statistics for a case. Its output summarizes the results of combat for hundreds of engagements at a single opening range for a single scenario. The combat summarized here may include many engagements for multiple waves of Red systems.

Statal generates and prints summary results as shown below:

<table>
<thead>
<tr>
<th>RESULTS</th>
<th>#</th>
<th>90% CONFIDENCE</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blue won</td>
<td>54</td>
<td>0.540 0.454 0.625</td>
</tr>
<tr>
<td>Red won</td>
<td>46</td>
<td>0.460 0.376 0.547</td>
</tr>
<tr>
<td>Draw</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>All dead</td>
<td>0</td>
<td>0.000</td>
</tr>
<tr>
<td>TOTAL REPS</td>
<td>100</td>
<td>1.000</td>
</tr>
</tbody>
</table>

(\text{Red k-killed})/(\text{Blue k-killed}) = 3.112

<table>
<thead>
<tr>
<th>ROUNDS FIRED BY</th>
<th>Blue</th>
<th>Red</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fired</td>
<td>809</td>
<td>709</td>
</tr>
<tr>
<td>Wasted</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Aborted</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>False Tgts</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Hidden Tgts</td>
<td>6</td>
<td>0</td>
</tr>
<tr>
<td>Impacting</td>
<td>803</td>
<td>709</td>
</tr>
<tr>
<td>Misses</td>
<td>4</td>
<td>393</td>
</tr>
<tr>
<td>Hits</td>
<td>799</td>
<td>316</td>
</tr>
<tr>
<td>Duds</td>
<td>6</td>
<td>4</td>
</tr>
<tr>
<td>No damage</td>
<td>65</td>
<td>41</td>
</tr>
<tr>
<td>M-kill only</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>F-kill only</td>
<td>29</td>
<td>8</td>
</tr>
<tr>
<td>M&amp;F-kill only</td>
<td>163</td>
<td>90</td>
</tr>
<tr>
<td>K-kill</td>
<td>534</td>
<td>173</td>
</tr>
</tbody>
</table>

First, Statal generates and prints the 6 upper lines summarizing the outcomes of the engagements. Then it generates a few final numbers including the tank total and damage total under SYSTEM STATUS. Finally, the code prints these numbers, along with the summary of shot results.

After printing some header lines, Statal generates and prints the four possible outcomes; Blue won, Red won, draw with survivors on both sides, and draw with no survivors. For each of these outcomes, it gives the number of engagements with that outcome, the fraction of engagements with that outcome, and perhaps the confidence interval. In the table above, Statal is 90% confident that the probability of blue winning is between the confidence interval .454 to .625. If the sample size is too small, Statal cannot generate a confidence interval.

All this is done in the DO 50 loop. The tally of outcomes is stored in kount(1,i), i=15,18. Frac is the fraction of engagements with each outcome. Conf is called to find the binomial confidence interval. Near the end of the loop, the result line is printed.

If any Blue tanks were k-killed, Statal finds the exchange ratio. The exchange ratio is the number of Red tanks k-killed per Blue k-killed. If Blue tanks were k-killed, the exchange ratio is printed, otherwise a zero is printed.

The DO 30 loop adds all tanks on each side that are alive, m-killed only, f-killed only, m&f-killed only, and k-killed to get the TOTAL line. The number of damaged tanks is the total number of tanks less the number alive. The number of Blue alive with 1-5 rounds is then copied from kount(1.19) to kount(1.8) and the number of Blue alive with no rounds is copied from kount(1.20) to kount(1.9).

Finally, Statal prints the results of the rounds and the status of the tanks (or other weapon system).
CODE | MATH | COMMENT
---|---|---
nwave | number of current wave
nreps3 | total number of replications executed
kount(i,j) | table of tank status at end of combat
hi | upper limit of confidence interval
lo | lower limit of confidence interval
fail | True if there is not enough data to find confidence interval

### V1.2

**SUBROUTINE STATS1** (nwave,nreps3,kount)

**Stats1:** Update and print statistics for 1 wave.

```fortran
include 'common.h'

integer kount(2,20)
character str(3)*4, str2(4)*g
character str3(14)*20, str4(14)*20
logical rail
real lo

data str /'Mtg ','Ratk','Batk'/
data str2/Blue won ','Red won ','Draw ', 'All dead '/
data str3/’Fired’, ’Wasted’, ’Aborted’, ’False Tgt’s’,
1 ’Hidden tgt’s’, ’Impacting’, ’Misses’, ’Hits’,
2 ’Duds’, ’No damage’, ’M-kill only’,
3 ’F-kill only’, ’M&F-kill only’, ’K-kill’/
data str4/’Alive’, ’M-kill only’, ’F-kill only’,
1 ’M&F-kill only’, ’K-kill’, ’TOTAL’, ’damaged’,
3 ’future’, ’future’, ’future’/

format(2x a~14,i5,’ or’,3r7.3)
2 format(2x :at7,t3)
3 format(2x,a,t23,2i6,t42,a,t62,2i6)
4 format(’
ROUNDS FIRED BY’,8x,’Blue Red’)

if (trace) print *,’statsl'
c0 Print summary of outcomes.
print*,’SUMMARY’
print*,’RESULTS # 90% CONFIDENCE’
DO 50 i=1,4
   n = kount(i,i+14)
   frac = float(n) /nreps3 + 0.0004999
   call confb(frac,nreps3,hi,lo,fail)
   if (fail) print 1, str2(i),nfrac
   if (notfail) print 1, str2(i),n,frac,lo,hi
  50 CONTINUE

c Print round results and weapon system status
print*,’TOTAL REPS’,nreps3,1.00
exch = 0.0
if (kount(1.5).gt.0) exch=0.0
print 2,[Red k-killed]/[Blue k-killed] = ’exch

Find system status totals (6th line of numbers)
DO 30 i=1,5
   kount(1.5)=kount(1.6)+kount(1.1)
   kount(2.5)=kount(2.6)+kount(2.1)
 30 CONTINUE

c Fill lines 7-9
kount(1.6)=kount(1.6)+kount(1.1)
kount(2.6)=kount(2.6)+kount(2.1)
kount(1.8)=kount(1.10)
kount(1.9)=kount(1.20)
c Print results
print *
print 3,’str3(i), (kshot(i,j),i=1,2),
1 str4(i), (kount(i,j),i=1,2),j=1,14)
if (trace) print *,’<stats1’

END
```
2.9 Stats2: Print Statistics for a Single Wave. If several waves of Reds are simulated stats2 prints the results of the current wave. Then it adds the results of the current wave to any previous waves.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>nwave</td>
<td>number of current wave</td>
</tr>
<tr>
<td>nw</td>
<td>maximum number of waves</td>
</tr>
<tr>
<td>nreps2</td>
<td>number of replications in current wave</td>
</tr>
<tr>
<td>kount(i,j)</td>
<td>table of systems status</td>
</tr>
</tbody>
</table>

If there is more than one wave stats1 prints them as shown below:

Wave 1: Blue, Red won 53 46 Draws, all dead= 1 0 Low, no ammo= 0 0
Wave 2: Blue, Red won 18 13 Draws, all dead= 0 0 Low, no ammo= 0 0
Wave 3: Blue, Red won 7 6 Draws, all dead= 0 0 Low, no ammo= 0 0
Wave 4: Blue, Red won 2 4 Draws, all dead= 0 0 Low, no ammo= 0 0
Wave 5: Blue, Red won 0 2 Draws, all dead= 0 0 Low, no ammo= 0 0

The DO 70 loop sums the results by adding the results for this wave to the results of previous waves. kount(1,15) is the number of times Blue won in all engagements, kount(1,16) is the number of times Red won in all engagements, and so on. The first column tends to act as a final counter of all waves performed, while the second column acts as a counter for one wave.

`V1.2
SUBROUTINE STATS2 (nwave,nw,nreps2,kount)
include 'common.h'
integer kount(2,20), n(4)

! Wave2: Update and print statistics for I wave.
! include 'common.h'
! integer kount(2,20), n(4)
1 2i3,' Wave',i2,: Blue, Red won',2i4,' Draws, all dead=',
1 2i3,' Low, no ammo=',2i3)
if (trace) print *, '>stats2'
if (nw.gt.1) print 1, nwave,(kount(2,i),i=15,20)
c Add results of current wave to results from previous waves.
DO 70 i=15,20
kount(1,i) = kount(1,i) + kount(2,i)
70 CONTINUE
if (trace) print *, '<stats2'
END`
2.10 Stats3: Print Statistics for a Single Engagement. Stats3 summarizes the results of a single engagement. It counts the number of blue and red tanks in each of 5 states, and finds which side won the engagement. If the user wishes, it prints a one line summary of the engagement. Then it adds the single engagement results to the results for previous engagements. Finally, it decides which blue tanks survived with enough ammunition to fight again.

The first three loops count the number of tanks in each damage state. The DO 5 loop clears the \( n_k \) array which will be used for counting. The DO 10 array count the blues in each state and the DO 20 loop counts the reds.

The 6 possible tank states are counted into 5 'buckets' in the array \( n_k \) as shown in table 1 below. The tanks in state 4 and 5 are both M&F killed but not K-killed. The only difference is that the foe knows the tanks in state 5 can’t shoot or move. But they are unaware that the tanks in state 4 can’t shoot or move (so they’re still considered threats.) At the end of combat, we’re not interested in the distinction, so the two tank states are counted together.

Table 1. Counting Tank Statuses

<table>
<thead>
<tr>
<th>life(j)</th>
<th>Status</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Alive</td>
<td>( n(1,k) = n(1,k)+1 )</td>
</tr>
<tr>
<td>2</td>
<td>M-only killed</td>
<td>( n(2,k) = n(2,k)+1 )</td>
</tr>
<tr>
<td>3</td>
<td>F-only killed</td>
<td>( n(3,k) = n(3,k)+1 )</td>
</tr>
<tr>
<td>4</td>
<td>M&amp;F-only killed</td>
<td>( n(4,k) = n(4,k)+1 )</td>
</tr>
<tr>
<td>5</td>
<td>M&amp;F-only killed</td>
<td>( n(4,k) = n(4,k)+1 )</td>
</tr>
<tr>
<td>6</td>
<td>K-killed</td>
<td>( n(5,k) = n(5,k)+1 )</td>
</tr>
</tbody>
</table>

The next section of code finds which side won (if any). To do this, it checks to see if there are tanks on each side that have received no firepower damage. The conditions and results are:

1 Draw if all blue and red tanks are firepower killed.
2 Red win if all blue but not all red tanks are firepower killed.
3 Blue win if all red but not all blue tanks are firepower killed.
4 Draw if not all blue and not all red tanks are firepower killed.

This win criteria can be modified to consider tanks with no ammunition and crews abandoning mobility killed tanks. However, it leads to many complications and should not be done until the model is thoroughly understood.

Next, if the user desires, the code prints a one line summary of the engagement. The following shows the first five and the last one line summary from a set of 1,000 engagements.

```
#Blues= 3 #Reds= 7 Red attack
Starting seed= 1111111
Rep Result AL MO FO MF K AL MO FO MF K seed
1 Red won 0 0 0 0 3 3 0 0 1 3 1111111
2 Red won 0 0 0 0 3 3 0 0 1 3 32219259
3 Red won 0 0 0 0 3 2 0 0 0 5 20450295
4 Red won 0 0 0 0 3 2 0 0 1 4 45588611
5 Red won 0 0 0 0 3 1 0 0 2 4 54179171
......
1000 Blue won 2 0 1 0 0 0 0 0 0 7 36495111
```

The DO 30 loop adds the results of the current engagement to the results of previous engagements so that a summary of all replications can be printed out later.
Finally, the DO 40 loop finds which blue tanks are available to fight a subsequent wave of red attackers. If a tank is fully functional and has at least 5 rounds left, it can play in a subsequent engagement. If fewer than 5 rounds are left, these tanks are tallyed as having no or low ammo.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>key</td>
<td>Prints history iff key &gt; 0.</td>
<td></td>
</tr>
<tr>
<td>nrep</td>
<td>Number of the replication just completed.</td>
<td></td>
</tr>
<tr>
<td>kount(i,j)</td>
<td>Count of tanks in status i for side j for all replications completed in this set.</td>
<td></td>
</tr>
<tr>
<td>iseed</td>
<td>Random number seed at start of engagement.</td>
<td></td>
</tr>
<tr>
<td>nsurv</td>
<td>Number of fully functional tanks with &gt; 4 rounds remaining.</td>
<td></td>
</tr>
<tr>
<td>nused</td>
<td>Number of rounds remaining for each blue tank in the next wave.</td>
<td></td>
</tr>
<tr>
<td>n(i,k)</td>
<td>Number of tanks in status i for side k.</td>
<td></td>
</tr>
<tr>
<td>nblu</td>
<td>Number of tanks in blue army</td>
<td></td>
</tr>
<tr>
<td>i, life(j)</td>
<td>Status of tank</td>
<td></td>
</tr>
<tr>
<td>nred</td>
<td>Number of tanks in red army</td>
<td></td>
</tr>
<tr>
<td>balive</td>
<td>Number of blue tanks not F-killed.</td>
<td></td>
</tr>
<tr>
<td>ralive</td>
<td>Number of red tanks not F-killed.</td>
<td></td>
</tr>
<tr>
<td>result</td>
<td>1=draw w/ all F-killed, 2=Red win, 3=Blue win, 4=draw w/ F-alive on both sides.</td>
<td></td>
</tr>
<tr>
<td>nrd(j)</td>
<td>Number of rounds remaining for each blue tank in the next wave.</td>
<td></td>
</tr>
<tr>
<td>ammo</td>
<td>Number of rounds available for fire</td>
<td></td>
</tr>
</tbody>
</table>

SUBROUTINE STATS3 (key, nrep, kount, iseed, nsurv, nused)
    print 2, nrep, str(result). n, iseed
ENDIF
character str(4)'8
Update statistics for all replicatioi~s at 1 opening range.
integer ammo, n(5,2), kount(2,20), balive, ralive, result
 dope 1,6
if (ammo.ge.5) nsurv = nsurv+1
n(1) = n(1)-0
if (ammo < 5.and. nr(j).gt.0) kount(2, 19) = kount(2, 19)+1
5 CONTINUE

DO 5 j=1,nblu
Count tanks in each damage status at the end of an engagement.
    Data str /'Blue won','Red won ','Draw ','All dead'/
7 CONTINUE

DO 10 j=1,nred
Count tanks by status.
    life(j) = i-1
if (i.gt.5) i = i+1
n(j,1) = n(j,1)+1
10 CONTINUE

DO 20 j=1,nred
Count red tanks by status.
    life(j) = i+nblu
if (i.gt.5) i = i+1
n(j,2) = n(j,2)+1
20 CONTINUE

Find who won engagement, if anybody.
    balive = n(1,1)+n(2,1)
ralive = n(1,2)+n(2,2)
if (balive.gt.0 .and. ralive.ge.0) result = 1
if (balive.ge.0 .and. ralive.gt.0) result = 2
if (balive.eq.0 .and. ralive.eq.0) result = 3
if (balive.eq.0 .and. ralive.gt.0) result = 4
ENDIF

Print results of 1 engagement.
3. SEARCH AND DETECTION ROUTINES

The search and detection routines simulate acquisition of non-firing targets. When undetected, unmasked targets are within detection range, the search routine is called once for each second of simulated time to find if a searcher detects a target in the next second.

Originally, all the search routines were called once per second of simulated time and search was taking 90% of the run time. They have been re-written so that the code runs five times faster.

Now, the initnv routine and its subordinate routines are called twice per run, once for Blue and Red. They generate tables containing the probability of ever detecting a target and the probability of detecting the target in the next second - given that it can be detected. These tables give the probabilities as a function of range, target exposure, and target motion. This greatly reduces the great deal of run time by avoiding repeated calculation of exponentials later.

Further, the det rg and searcO routines are called at the beginning of each engagement. The former is called once for each side. It finds the range within which each tank can acquire moving or stationary targets in hull defilade or fully exposed. These maximum acquisition ranges are stored for later use.

Search generates a table containing the range between each searcher-target pair. If the scenario is stationary, this table will not have to be updated so the computation of many square roots will be avoided. If the scenario has moving tanks, the table avoids the calculation of some square roots. The routine then finds the first time when targets are within detection range and schedules search to begin at that time. This avoids repeated calls to search when no targets can be detected.

* Scheduled by: aprsmk, aprter, diseng, newtgt, searcO, search, and select.
3.1 Initnv: Generate Detection Probability Tables.

Initnv simply calls the nvl routine with the appropriate arguments for detection level, illumination, target size, and sensor type.

The variables are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>=1 for Blue searchers, 2 for Red searchers.</td>
</tr>
<tr>
<td>kind</td>
<td>=1 for visual detection, 2 for device 2.</td>
</tr>
<tr>
<td>alumin</td>
<td>Illumination (ft-candles).</td>
</tr>
<tr>
<td>high(2)</td>
<td>Heights of the hull-defilade and fully-exposed target (meters).</td>
</tr>
<tr>
<td>job</td>
<td>Acquisition level required for sensor.</td>
</tr>
</tbody>
</table>

```fortran
V7.3
SUBROUTINE INITNV (n,kind,alumin,high)
    real high(2), job
    if (kind.eq.1) job=3.0
    if (kind.eq.2) job=4.0
    c Find values for HD stationary target.
    call nvl (n, 1, alumin, high(1), job, kind)
    c Find values for FE stationary target.
    call nvl (n, 2, alumin, high(2), job, kind)
    c Find values for FE moving target.
    call nvl (n, 3, alumin, high(2), 0.667*job, kind)
END
```
3.2 Nvl: Find Probability of Ever Detecting and of Detecting in Next Second. Nvl uses
the Night Vision code for visual detection to generate a table of probabilities. The table contains probabili-
ties that the target will ever be detected as a function of range and condition. The other table contains
the probabilities that the target will be detected in a given second. This is also a function of range and
condition.

Nvl is code that was stripped from a much larger program developed by the Electro-Optical and
Night Vision Laboratory. The code is considered correct, however details are unavailable.

Assumptions. It has the following built-in assumptions:

1. Acquisition is divided into the following categories:
   - Detection - there's something there.
   - Classification - it's tracked.
   - Recognition - it's a tank.
   - Identification - it's a T80.
   The program assumes acquisition at the recognition level.

2. The size of the search field is 225 degrees squared, e.g. 5 degrees high and 45 degrees wide.

3. The visibility range is 7 kilometers.

Input. As an example, suppose the ambient light level is 300 ft-candles, the tank turret is 0.8
meters high, and the total tank is 2.2 meters high. The single input line would then be: '1 300. 0.8 2.2'.
The 1 means that the sensor is the human eye. If you are interested in other targets, just input the
appropriate heights. Typical light levels are:

<table>
<thead>
<tr>
<th>Ft-candles</th>
<th>Typical Day</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>Clear day</td>
</tr>
<tr>
<td>100</td>
<td>Overcast day</td>
</tr>
<tr>
<td>10</td>
<td>Heavy overcast day</td>
</tr>
<tr>
<td>1</td>
<td>Sunset overcast day</td>
</tr>
</tbody>
</table>

Output. Figure 1 illustrates the output. It shows the probability of ever detecting as a function of
range, and the median time to detect given detection is possible.

![Figure 1. The Probability of Detecting a Target.](image)

The output consists of 12 lines of input echo, then 7 lines of output proper. Line 13 is ranges in
kilometers. Lines 14-16 are median (τ) times to detect given that detection is possible. The 14th line is for
a stationary, hull-defilade target; the 15th for a stationary, fully-exposed target; and the 16th for a mov-
ing, fully-exposed target. Lines 17-19 are the corresponding probabilities that the target will ever be
detected. The probability of detecting in a time \( t \) is then:

\[
p_d = p_\infty e^{-t/\tau}
\]
SUBROUTINE NVL (n,j,alumnc, dim, ajob, kind)

NVL: find p-infinity, pdetect in 1 sec.

n - # of army
j - range band
alumnc - illumination in ft-candles (sun?)
dim - target height (m)
ajob - acquisition level
kind - kind of sensor (1=eye)
common /sensor/ psense(2,8), pinfn(2,3,10), pdet(2,3,10),

ndets(2), tlock(2), pinp(2), repeat, reckn(2), pfalse(2,2)
save rinfd, zone, visrg
data zone, rinfd /225. ,1/
data visrg /7,

DO 20 i=1,10
Find probabilities and median times for 2 ranges.
rc = 0.
pinf = 0.
tbarr = 999.
range = 0.5!
attn = 3.912 / visrg
IF (kind.eq.1) THEN
rc = eye (alumnc, attn, range, visrg)
fov = 24.5
ELSEIF (kind.eq.2) THEN
rc = devic2 (attn, range)
fov = 11.98
ENDIF
rc = rc * dim / range
IF (rc.ge.rinfd) THEN
x = rc / ajob
y = 2.7 * y * x
z = x ** y
pinf = z / (z + 1.0)
pinf = amni (pint, 99)
tau = zone / (fov * amni(5, fov))
tbarr = 3.4 * tau / pinf
if (pinf.gt.0.9) tbarr = tau * ajob * 6.8 / rc
ENDIF
if (tbarr.gt.99.0) tbarr = 999.0
pinfn(n,j,i) = pinf
pdet(n,j,i) = 1.0 - exp(-1.0/tbarr)
20 CONTINUE
END
3.3 Eye: Find the Probability the Eye Detects in the Next Second. Eye can be used to find detection probabilities for various types of targets; however, we generally use it for tank targets. The calling routine, \texttt{nvl}, finds the actual probabilities after \texttt{eye} finds the resolvable cycles. Resolvable cycles are akin to scan lines used to paint an object on a TV screen.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>alumn</td>
<td>$f$</td>
<td>Illumination (ft-candles).</td>
</tr>
<tr>
<td>attn</td>
<td>$a$</td>
<td>Attenuation (?).</td>
</tr>
<tr>
<td>range</td>
<td>$r$</td>
<td>Range to target (km).</td>
</tr>
<tr>
<td>visrg</td>
<td>$v$</td>
<td>Visibility range (km).</td>
</tr>
<tr>
<td>sog</td>
<td>$s = (v+1)/3$</td>
<td>Sky over ground ratio. ($1 \leq s \leq 3$)</td>
</tr>
<tr>
<td>cntrst</td>
<td>$c = \frac{0.4}{1+s(e^r-1)}$</td>
<td>Target to background contrast ratio.</td>
</tr>
<tr>
<td>$i$</td>
<td></td>
<td>Lower column for interpolation. $1 \leq i \leq 4$</td>
</tr>
<tr>
<td>$j$</td>
<td></td>
<td>Upper column for interpolation. $2 \leq j \leq 4$</td>
</tr>
<tr>
<td>$k$</td>
<td></td>
<td>Row index.</td>
</tr>
<tr>
<td>clog</td>
<td>$C = \ln c$</td>
<td>Natural logarithm of contrast ratio.</td>
</tr>
<tr>
<td>rlo</td>
<td>$r_l = \sum_{k=1}^{7} a_{i,k} C^{k-1}$</td>
<td>Resolvable cycles for lower power of 10.</td>
</tr>
<tr>
<td>rhi</td>
<td>$r_h = \sum_{k=1}^{7} a_{j,k} C^{k-1}$</td>
<td>Resolvable cycles for higher power of 10.</td>
</tr>
<tr>
<td>eye</td>
<td>$r_e = r_l + (r_h - r_l)(f/10^{1/10})/(0.9 \times 10^4)$</td>
<td>Resolvable cycles interpolated for intermediate illumination.</td>
</tr>
</tbody>
</table>

It's not obvious that the equations for $r_l$ and $r_h$ above correspond to the calculation of $r_{l0}$ and $r_{hi}$. The summation:

$$ r_l = \sum_{k=1}^{7} a_{i,k} C^{k-1} $$

may be expanded as:

$$ r_l = a_{i,1} + a_{i,2} C + a_{i,3} C^2 + a_{i,4} C^3 + a_{i,5} C^4 + a_{i,6} C^5 + a_{i,7} C^6 $$

and re-written for efficiency as:

$$ r_l = a_{i,1} + C(a_{i,2} + C(a_{i,3} + C(a_{i,4} + C(a_{i,5} + C(a_{i,6} + C(a_{i,7})))))) $$

The code uses the DO 20 loop to evaluate the equation above. The first iteration finds the value of the innermost parenthesis, and each following iteration finds the value of the next innermost parenthesis, until it finds the entire value on the sixth iteration. Since the calculation of $r_h$ is similar to the calculation of $r_l$, the code finds it at the same time. The relevant portion of the code is:

```c
  clog = alog(cntrst)
  rlo = a(i,7)
  rhi = a(j,7)
  DO 20 k=6,1,-1
  rlo = rlo*clog + a(i,k)
  rhi = rhi*clog + a(j,k)
  20 CONTINUE
```

It is also, not obvious that the equation for $r_e$ above yields an interpolated value between $r_l$ and $r_h$. To understand how $r_e$ is interpolated, we'll have to do some backtracking. The equation used is:

$$ r_e = r_l + (r_h - r_l)(f/10^{1/10})/(0.9 \times 10^4) $$

$$ r_e - r_l = (r_h - r_l)(f/10^{1/10})/(0.9 \times 10^4) $$
This is simply the relationship for similar triangles used to perform linear interpolation in Figure 2 below.

![Figure 2. Linear Interpolation](image)
3.4 Detrg: Find Maximum Range to Which Each Firer Can Detect. Detrg is called for each side at the beginning of each engagement to generate detection ranges. For each tank on the side, it finds the ranges within which the tank is able to detect 1) stationary, hull defilade targets, 2) stationary, fully exposed targets, and 3) moving, fully exposed targets.

The DO 80 loop loops through all the tanks on the side. For each tank it draws a random number from a uniform distribution. Then for each of the three types of targets, it interpolates on the appropriate curve to find the range within which the firer is able to detect that type of target. The figure below illustrates the procedure.

The DO 70 loop chooses the curve for each of the three types of targets: moving fully-exposed, stationary fully-exposed, and stationary hull-defilade.

The DO 60 loop finds the appropriate interval to interpolate in.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>narmy</td>
<td>n</td>
<td>1 if Blue, 2 if Red</td>
</tr>
<tr>
<td>first</td>
<td>ID of first tank on the side</td>
<td></td>
</tr>
<tr>
<td>last</td>
<td>ID of last tank on the side</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>Random draw from a uniform distribution.</td>
<td></td>
</tr>
<tr>
<td>con</td>
<td>c</td>
<td>1 if stationary HD, 2 if stationary FE, 3 if moving FE.</td>
</tr>
<tr>
<td>krg</td>
<td></td>
<td>Index such that $p_{e,c,k} \leq p &lt; p_{e,c,k+1}$.</td>
</tr>
<tr>
<td>p1</td>
<td>$p_1$</td>
<td>Lower bound of interpolation interval</td>
</tr>
<tr>
<td>p2</td>
<td>$P_{n.c.k}$</td>
<td>Higher bound of interpolation interval.</td>
</tr>
<tr>
<td>r1</td>
<td></td>
<td>Range at lower bound (m).</td>
</tr>
<tr>
<td>incr</td>
<td>$\Delta r$</td>
<td>Range increment in table (m).</td>
</tr>
<tr>
<td>r</td>
<td>$r = \frac{P_{1-p}}{P_{1-p_2}}$</td>
<td>Range chosen.</td>
</tr>
</tbody>
</table>

```
33
```
### 3.5 Search: Schedule Initial Search Event

The model executes the **search** routine once at the beginning of each engagement. **search** initializes a table containing the ranges between Red and Blue tanks, selects the appropriate detection ranges for each tank, and then finds the first possible time that detection can occur. It schedules **search** to begin at that time. This eliminates the repeated simulation of search before detections are possible.

**search** initializes the table containing ranges between combatants as follows:

\[
rgtbl(i,j) = r_{i,j} = \sqrt{(x_i-z_j)^2+(y_i-y_j)^2)}
\]

Where,

- \(x_i, y_i\) are the coordinates of the \(i\)th system.

Combatants are not ignored until they are killed to some level. So initially, \(\text{ignore}(i)\) is set to .false.

The range to which each tank can see is copied from the \(rgvis\) table into the \(rgvs\) vector. The appropriate column copied depends on whether the target is stationary or moving, or whether the searcher is moving or stationary.

**search** finds the longest detection range of all the tanks and uses this plus the speed of any attacker to determine when the first target can be detected by an observer. If the sides are within detection range at the start of the game, search begins immediately (at \(t = 0\)). If they are beyond detection range and neither side is moving they will never detect each other, so **search** schedules a finish event immediately. If they are beyond detection range and one side is moving, **search** finds when the sides are within detection range and schedules a search event at that time.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>i</td>
<td>ID of Blue tank.</td>
</tr>
<tr>
<td>n</td>
<td>n</td>
<td>ID of Red tank.</td>
</tr>
<tr>
<td>x0(i), y0(i)</td>
<td>(x_i, y_i)</td>
<td>Coordinates of Blue tank (m).</td>
</tr>
<tr>
<td>x0(j), y0(j)</td>
<td>(x_j, y_j)</td>
<td>Coordinates of Red tank (m).</td>
</tr>
<tr>
<td>x, y</td>
<td>(x, y)</td>
<td>Coordinate of Blue tank W.R.T. to Red tank (m).</td>
</tr>
<tr>
<td>r</td>
<td>(r = \sqrt{x^2+y^2})</td>
<td>Distance between tanks (m).</td>
</tr>
<tr>
<td>rgtbl(i,n)</td>
<td>Cell in table storing distance from tank (i) to tank (n) (m).</td>
<td></td>
</tr>
<tr>
<td>kred(scene)</td>
<td>Initial exposure of Red tanks in scenario 'scene'.</td>
<td></td>
</tr>
<tr>
<td>ni</td>
<td>(k)</td>
<td>Initial exposure of Blue tanks.</td>
</tr>
<tr>
<td>ignore(i)</td>
<td>True IFF tank (i) is no longer a threat.</td>
<td></td>
</tr>
<tr>
<td>rgvis(i)</td>
<td>Range to which tank (i) can detect (m).</td>
<td></td>
</tr>
<tr>
<td>dmax</td>
<td>(d)</td>
<td>Farthest distance any tank can detect a target (m).</td>
</tr>
<tr>
<td>nj</td>
<td>(r_0)</td>
<td>Initial exposure of Red tanks.</td>
</tr>
<tr>
<td>rg0</td>
<td>(r_0)</td>
<td>Opening range (m).</td>
</tr>
<tr>
<td>dist</td>
<td>(d = d - r_0)</td>
<td>Farthest distance a tank can detect beyond opening range (m).</td>
</tr>
<tr>
<td>time</td>
<td>(r_0 &lt; d)</td>
<td>Implies at least 1 tgt in detection range at time zero.</td>
</tr>
<tr>
<td>scene</td>
<td>=1 for Meeting, 2 for Red attack, 3 for Blue attack</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE SEARCO

Searc0: schedule initial search event.
ni, nj - 1 if HD, 2 if FE & stationary, 3 if FE & moving.
dist - distance to travel before entering detection range.
dmax - maximum distance any combatant can detect.
time - time to travel before entering detection range.

integer kred(3), kblu(3)
logical ignore(NN)
include 'common.h'
common /cserch/ rgtbl(NN,NN), ignore(NN), rgvs(NN), time, ni, nj
save /cserch/
data kred /2,3,1/, kblu /2,1,3/

if (trace) print *, 'searc0'

Set up tables.
DO 30 i=1, nb
DO 20 j=1, nred
   n = j+nb
   x = xo(j)-xo(n)
   y = yo(j)-yo(n)
   r = sqrt(x**2+y**2)
   rgtbl(i,n) = r
30 CONTINUE

Find maximum distance combatants can detect.
ni = kred(scene)
dmax = 1e10
DO 50 i=1, nb
   ignore(i) = false.
   rgvs(i) = rgvis(ni,i)
   dmax = amax1(dmax,rgvs(i))
50 CONTINUE

nj = kblu(scene)
DO 60 i=1, nred
   ignore(i+nb) = false.
   rgvs(i+nb) = rgvis(nj,i+nb)
   dmax = amax1(dmax,rgvs(i+nb))
60 CONTINUE

Find when search should begin.
dist = rgtbl(0, dmax)
IF (rg0.lt.dmax) THEN
   In detection range at time 0.0
time = 0.0
ELSEIF (scene.eq.MEETING) THEN
   Outside & never enters detection range.
time = 1.0e10
call skedul(0.0, NULL, 'finish', NULL)
ELSEIF (scene eq RATTAK) THEN
   Outside and red enters
time = dist/speed(RED)
ELSE
   Outside and blue enters.
time = dist/speed(BLU)
ENDIF

if (trace) print *, '<searc0'
END
3.6 Search: Find Targets Detected in Next Second. **Search** makes sure the positions of the tanks are up-to-date. Then it checks all searchers to see which targets are within detection range. When a target is within range, it calls searc2 to find whether detection will occur.

To update the tank positions, **search** first updates the ignore array. All K-killed tanks and perhaps some MF killed tanks are ignored. This depends on user input. Then it updates the positions of any moving tanks except those that are to be ignored. (This seems redundant. But maybe some K-killed tanks haven't halted quite yet.) Finally, the distance between each Blue and Red tank is stored in the rgtbl array.

The next step is to loop through all the Blue and Red searchers to see which foes are within detection range. This is done in the DO 40 loop. The DO 40 loop loops through the Blue tanks. If the ith Blue tank is not to be ignored, we find the range (rgi) to which it can detect. We then check the Blue tank against each Red tank. This is done in the DO 30 loop. If the jth Red tank is not to be ignored, we find the range (rgj) to which it can detect. If the range between the two tanks is less than either of the detection ranges then we consider the pair further. The next step is to find if they are in line-of-sight of each other. If so, we treat Blue as the searcher and then Red as the searcher.

If the Red tank is in detection range of the Blue searcher and the Blue searcher has not already detected the Red tank and the Blue searcher can handle another detection, detection is possible. If detection is possible, searc2 is called to find whether it occurs and if so, when. If detection does not occur on the Red tank in the next second, the repeat flag is set so that **search** will be rescheduled in the next second. Identical logic determines whether the Red tank detects the Blue one.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>Current time (sec).</td>
</tr>
<tr>
<td>repeat</td>
<td></td>
<td>True IFF <strong>search</strong> should reschedule itself.</td>
</tr>
<tr>
<td>time</td>
<td></td>
<td>First time any target can be detected (sec).</td>
</tr>
<tr>
<td>i</td>
<td>i</td>
<td>ID of tank.</td>
</tr>
<tr>
<td>ignore(i)</td>
<td></td>
<td>True IFF tank i is no longer a threat. (logical)</td>
</tr>
<tr>
<td>life(k)</td>
<td></td>
<td>Status of ith tank ((\geq) KILL implies it's known dead.)</td>
</tr>
<tr>
<td>x0(i), y0(i)</td>
<td></td>
<td>Coordinates of ith tank (m).</td>
</tr>
<tr>
<td>rgtbl(i,j)</td>
<td></td>
<td>Distance between ith &amp; jth tanks (m).</td>
</tr>
<tr>
<td>ndet1</td>
<td></td>
<td>Maximum detections for a Blue tank.</td>
</tr>
<tr>
<td>ndetj</td>
<td></td>
<td>Maximum detections for a Red tank.</td>
</tr>
<tr>
<td>rgi</td>
<td></td>
<td>Distance Blue can detect Red target (m).</td>
</tr>
<tr>
<td>rgj</td>
<td></td>
<td>Distance Red can detect Blue target (m).</td>
</tr>
<tr>
<td>ok</td>
<td></td>
<td>Implies tgt is in detection range and not yet seen, and searcher is not loaded with detects.</td>
</tr>
</tbody>
</table>
SUBROUTINE SEARCH (t)

Search: see if any targets are detected in the next second.
include 'common.h'
logical ignore, ok
common /cserch/ rgtbl(NN,NN), ignore(NN), rgvs(NN), time, ni, nj
save /cserch/
rs(x,y) = sqrt(x*x+y*y)

if (trace) print *,'>search'
repeat = .false.
IF (tlt-tine) RETURN

Update status of tanks.
DO 5 i=1,nblu+nred
  (Next line should eventually be updated in damage.f, ltkill.)
  ignore(i) = ignore(i).or.life(i).lt.0.5
  if (.not.ignore(i).and. motion(i).ne.STATNY)
    call path(i,t,motion(i),0.0,dml,dm2,dm3,dm4)
  5 CONTINUE

CONTINUE
DO 20 i=1,nblu
  IF (.not.ignore(i)) THEN
    IF (.not.ignore(j)) THEN
      DO 10 j=1,nblu+nred
        rgtb(i,j) = rss(x0(i)-x0(j),y0(i)-y0(j))
      10 CONTINUE
      rgtb(i) = amax1(rgtbi,rgtbi)
      IF (rg.lt.rgmax) THEN
        IF (los(ij)) THEN
          Treat Blue tank as searcher
          ok = rg.lt.rgj.and. not.see(i,j).and. ndet(i).lt.ndeti
          if (ok) call search2(tjij,BLU,ni,dt)
          if (.not.ok) repeat = .true.
        ELSE
          repeat = .true.
        END IF
      END IF
    ELSE
      Treat Red as searcher
      ok = rg.lt.rgj.and. not.see(i,j).and. ndet(j).lt.ndetj
      if (ok) call search2(tjij,RED,nj,dt)
      if (.not.ok) repeat = .true.
    ELSE
      repeat = .true.
  END IF
  20 CONTINUE

END
ENDIF
30 CONTINUE
END

if (repeat) call skedul(t+1.0,’search’, NULL)
if (trace) print *,’<search'
if (trace) print *,’<search'
END
3.7 **Searc2: Find if One Searcher Detects One Target in the Next Second.** *Searc2* finds whether a specific observer detects a specific target in the next second, and if so when during that second. If the observer doesn’t detect the target in the next second, *searc2* sets the repeat flag so that *search* will re-occur in one second.

The probability of detecting in the next second is a function of range, target motion, and target exposure. This data is stored in the array `pdet(2,3,8)`, where `pdet(1,i,j)` is the detection probability for Blue searchers against targets in condition `i`, at range `j`. A sample of `pdet(1,i,j)` might look as follows:

<table>
<thead>
<tr>
<th>index</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rg (m)</td>
<td>100</td>
<td>200</td>
<td>300</td>
<td>400</td>
<td>500</td>
<td>600</td>
<td>700</td>
<td>800</td>
</tr>
<tr>
<td>1</td>
<td>Stationary, Hull defilade</td>
<td>.775</td>
<td>.514</td>
<td>.372</td>
<td>.286</td>
<td>.230</td>
<td>.190</td>
<td>.160</td>
</tr>
<tr>
<td>2</td>
<td>Stationary, Fully exposed</td>
<td>.635</td>
<td>.461</td>
<td>.342</td>
<td>.272</td>
<td>.222</td>
<td>.185</td>
<td>.158</td>
</tr>
<tr>
<td>3</td>
<td>Moving, Fully exposed</td>
<td>.490</td>
<td>.367</td>
<td>.286</td>
<td>.228</td>
<td>.190</td>
<td>.160</td>
<td>.138</td>
</tr>
</tbody>
</table>

The first half of *searc2* simply interpolates in the appropriate row of this matrix to find the detection probability. *Searc2* then draws a random number; if it’s less than the probability of detection *searc2* schedules a detection randomly in the next second.

![Figure 3. Probability of Detection in the Next Second](image)

**Figure 3. Probability of Detection in the Next Second**

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Simulation time (sec)</td>
<td></td>
</tr>
<tr>
<td>firer</td>
<td>ID of firer (integer)</td>
<td></td>
</tr>
<tr>
<td>tgt</td>
<td>ID of target (integer)</td>
<td></td>
</tr>
<tr>
<td>narmy</td>
<td>1 if firer is Blue, 2 if Red.</td>
<td></td>
</tr>
<tr>
<td>cond</td>
<td>1 if tgt is stationary HD, 2 if stationary FE, 3 if moving FE. (integer)</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>Delay for rescheduling search (sec).</td>
<td></td>
</tr>
<tr>
<td>rg</td>
<td>Range from searcher to target (m).</td>
<td></td>
</tr>
<tr>
<td>rginr</td>
<td>Range increment in table (m).</td>
<td></td>
</tr>
<tr>
<td>temp</td>
<td>Index of lower bound of range interval.</td>
<td></td>
</tr>
<tr>
<td>z</td>
<td>Probability of detecting at lower bound of range interval.</td>
<td></td>
</tr>
<tr>
<td>i</td>
<td>Index of lower bound of range interval.</td>
<td></td>
</tr>
<tr>
<td>tlo</td>
<td>Probability of detecting at upper bound of range interval.</td>
<td></td>
</tr>
<tr>
<td>thi</td>
<td>Fraction of distance into range interval.</td>
<td></td>
</tr>
<tr>
<td>frac</td>
<td>Probability of detection in current 1 second interval.</td>
<td></td>
</tr>
<tr>
<td>pdet</td>
<td>Flag to reschedule search. Set to .true. if detection does not occur. (logical)</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE SEARC2 (t, firer, tgt, narmy, cond, dt)

Searc2: see if a tank detects a target during this second.

include 'common.h'
integer firer, tgt, cond

Find where to interpolate.
    temp = rg/rgincr
    indx = int(temp)
    IF (indx .lt. 1) THEN
        tlo = 1.0
        thi = pdet(narmy,cond,1)
    ELSEIF (indx .lt. 8) THEN
        tlo = pdet(narmy,cond,indx)
        thi = pdet(narmy,cond,indx+1)
    ELSE
        tlo = pdet(narmy,cond,8)
        thi = 0.0
    ENDIF

Interpolate in interval.
    frac = temp-aint(temp)
    pdetct = tlo + frac*(thi-tlo)
    IF (ranu(0.0) .gt. pdetct) THEN
        repeat = .true.
        dt = 1.0
    ELSE

Set flag to repeat search. (At least one searcher didn't detect tgt.)
    repeat = .true.
    dt = 1.0
ENDIF

Schedule search randomly in next second. (Searcher may detect.)
    call skedul(t+ranu(0.0),firer,'detect', tgt)
ENDIF

if (trace) print *, '<searc2'
END
3.8 Detect: Find if Target Is Detected and Schedule Subsequent Events. Detect simulates detection if a) line-of-sight still exists, b) the observer hasn't detected the target, and c) the observer is not loaded with detections. (The user specifies how many targets each tank can detect simultaneously.) If all these conditions hold, detect increments the number of targets the observer knows about, marks that this observer has detected this target, and schedules the observer to select a target. (The select event controls whether a selection actually happens.)

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Simulation time (sec)</td>
</tr>
<tr>
<td>I</td>
<td>ID of observer.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>m</td>
<td>1 if observer is Blue, 2 if Red.</td>
</tr>
<tr>
<td>n</td>
<td>1 if target is Blue, 2 if Red.</td>
</tr>
<tr>
<td>los(I,it)</td>
<td>True iff line of sight exists between observer &amp; target (logical).</td>
</tr>
<tr>
<td>see(I,it)</td>
<td>True iff observer already sees target (logical).</td>
</tr>
<tr>
<td>ndet(I)</td>
<td>Number of targets observer is aware of.</td>
</tr>
<tr>
<td>ndets(m)</td>
<td>Maximum number of targets observer on side m can remain aware of.</td>
</tr>
<tr>
<td>thuman</td>
<td>Randomly chosen time required for human to select a target (sec).</td>
</tr>
</tbody>
</table>

Detect contains two assumptions. The first is that the observer has a fixed limit (ndets(m)) on the number of targets it can maintain cognizance of. The second is that the observer requires a lognormally distributed time to select a target. The author has chosen to make the median time of this distribution to be 2 seconds and the standard deviation of the underlying normal distribution to one-half.

V7.3
SUBROUTINE DETECT (t, I, it)
if (trace) print *,’>detect’
m = army(I)
n = 3-m IF los(I,it) .and. .not. see(I,it) .and. ndet(I).lt.ndets(m) THEN
if (history)print t,color(m),I,color(n),it
ndet(I) = ndet(I)+1
see(I,it) = true.
thuman = 2.0*exp(rann(0.5))
call select(*,I,thuman)
ENDIF
if (trace) print *,’<detect’
END

40
3.9 Pinpnt: Simulate Firing Signature Detection. Pinpnt simulates detection of a tank due to its firing signature. The program executes this routine every time a tank fires. For each foe, the routine draws a random number and schedules detection if a) the random draw was less than the pinpoint detection probability, b) the foe can still shoot, c) line-of-sight exists, and d) the foe hasn’t already detected the firer.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td>Simulation time (sec).</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>ID of firer.</td>
</tr>
<tr>
<td>first</td>
<td></td>
<td>ID of first foe.</td>
</tr>
<tr>
<td>last</td>
<td></td>
<td>ID of last foe.</td>
</tr>
<tr>
<td>k</td>
<td></td>
<td>ID of foe.</td>
</tr>
<tr>
<td>pinpxx</td>
<td></td>
<td>Probability of pinpoint detection.</td>
</tr>
<tr>
<td>wilsee</td>
<td></td>
<td>True IFF foe detects muzzle flash or smoke (logical).</td>
</tr>
<tr>
<td>life(k)</td>
<td></td>
<td>Status of foe k. Foe doesn’t detect if it is firepower killed.</td>
</tr>
<tr>
<td>ndet(k)</td>
<td></td>
<td>Number of targets foe k is cognizant of.</td>
</tr>
<tr>
<td>ndets()</td>
<td></td>
<td>Maximum number of targets foe k can maintain cognizance of.</td>
</tr>
<tr>
<td>los(k,l)</td>
<td></td>
<td>True IFF line of sight exists between k and l (logical).</td>
</tr>
<tr>
<td>see(k,l)</td>
<td></td>
<td>True IFF k already sees l (logical).</td>
</tr>
<tr>
<td>thuman $t_h = 2e$</td>
<td>$N(0.5)$</td>
<td>Time required for k to select a target (sec).</td>
</tr>
<tr>
<td></td>
<td>$N(0.5)$</td>
<td>Random draw from a normal distribution with mean zero and standard deviation of one-half.</td>
</tr>
</tbody>
</table>

SUBROUTINE PINPNT (t,l)
include 'common.h'
integer first
logical wilsee
format (f8.2,1x,a4,i3,' sees ',a4,i3,' muzzle flash')
if (trace) print *, '>'
pinpxx = pinp(army(first))
DO 20 k-first, last
wilsee = pinpxx.gt.ranu(0.0)
IF (life(k).lt.FKILL .and. wilsee .and. ndet(k).lt.ndets(army(k)) .and. los(k,l) .and. .not.see(k,l)) THEN
if (history) print 1,
t, color(army(k)), k, color(army(l)), l
see(k,l) = true
ndet(k) = ndet(k) + 1
thuman = 2.0*exp(ranu(0.5))
call selecs(t,k,thuman)
ENDIF
20 CONTINUE
if (trace) print *, '<'
END
4. TARGET SELECTION ROUTINES

The target selection routines decide which of any targets in an area has the highest priority. The subroutine priort assigns each target a priority number. The integer function priorn then decides which target has the highest priority, and breaks any ties that may occur. The subroutine select uses priorn to decide which target will be selected. The subroutine selecs determines when the program will begin selection, and the subroutine engage sets up the gunner to aim and fire at the target that has been chosen.

The diagram below shows the relationship between the routines discussed in this section. The many routines calling selecs, the selection start event, will be discussed in other sections.
4.1 Select: Gunner Chooses Most Dangerous Target It Sees. In the subroutine select the gunner chooses the most dangerous target it sees. If the firer cannot select a target because it cannot see any, the subroutine moves the gunner. When it is possible for the firer to shoot, the subroutine begins to select a target. After a target is chosen, select finds whether the target has previously been fired on by the gunner. If the target is new, it may be made into a false target. When the target has been classified as old or new, the target is engaged.

The subroutine select first calculates if the level of damage to the firer is less than FKILL. If the code determines that the gunner still has firepower, then it calls priorn to choose a target. If priorn cannot select a target because there are none in view, the code moves the gunner to another position.

If the target that is selected is a new target, the code may make it into a false target. This is done to better simulate the actual conditions of combat, because gunners will often mistake land formations as targets. The code restarts the search for targets if the search has been turned off.

If the target that is selected has been fired on previously, the code will print the target's history. Finally, select calls engage.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer. 1 if Blue, 2 if Red</td>
</tr>
<tr>
<td>kind</td>
<td>Kind of round. 1 if KE, 2 if HEAT, 4 if missile, 5 if top attack.</td>
</tr>
<tr>
<td>alive</td>
<td>True if firer is not firepower killed (logical).</td>
</tr>
<tr>
<td>level</td>
<td>Priority level (1..22).</td>
</tr>
<tr>
<td>it</td>
<td>ID of target with highest priority.</td>
</tr>
<tr>
<td>busy(I)</td>
<td>If firer is busy, new tgt selection is inhibited (logical).</td>
</tr>
<tr>
<td>color</td>
<td>Color of target is 'Blue' or 'Red' (character*4).</td>
</tr>
<tr>
<td>k</td>
<td>Concealment. 1 if FD, 2 if HID, 3 if FE.</td>
</tr>
<tr>
<td>pf</td>
<td>Draw from uniform random distribution.</td>
</tr>
<tr>
<td>tgtfls</td>
<td>True IFF this is a false target (logical).</td>
</tr>
<tr>
<td>see(I,it)</td>
<td>True IFF firer sees target (logical).</td>
</tr>
<tr>
<td>flstgt</td>
<td>ID of false target is always -1 (integer).</td>
</tr>
<tr>
<td>repeat</td>
<td>True IFF search is to be rescheduled in 1 second (logical).</td>
</tr>
<tr>
<td>fot(I,it)</td>
<td>True IFF firer is on (servicing) target (logical).</td>
</tr>
<tr>
<td>nrtgt(I)</td>
<td>ID of firer I's target.</td>
</tr>
</tbody>
</table>

**SUBROUTINE SELECT** (t, I)

Select: gunner chooses most dangerous target it sees.

include 'common.h'

character*4 color
t logical tgt fls, f alive, can go
integer I, it, prior, m

format('t, it), format('t, it, i), format('t, it, i), format('t, it, i)

1 format('t, it), format('t, it), format('t, it, i), format('t, it, i)

2 format('t, it), format('t, it, i), format('t, it, i), format('t, it, i)

3 format('t, it, i), format('t, it, i), format('t, it, i), format('t, it, i)

4 format('t, it, i), format('t, it, i), format('t, it, i), format('t, it, i)

if (trace) print *, '>select' m = army(I)
kind = kind(m)
if alive = if alive(I), it, FKILL
if (f alive) THEN
  Firer can shoot, so have him select.
it = prior(I, level)
  if (it.eq.NULL) THEN
    Firer has no targets to select so he moves if possible
    if (history) print 3, t, color(m), 1
    busy(I) = false
    if (kind eq 4) nchan(I) = nchan(I)-1
    IF (can go(I, t) and (kind eq 2 or...
      kind.eq.5) or. nchan(I).eq.0)) THEN
        call cancel(I, 'halt', NULL)
        call cancel(I, 'accel', NULL)
        call schedul(t, I, 'accel', NULL)
      ENDIF
    ELSE
      Tgt has been selected
      color = color(army(it))
      if (tfire(I, it).eq.0.) THEN
        if (tfire(I, it).eq.0.) THEN
          Tgt is new; replace with false tgt randomly.
          k = kneal(it)-1
          tgt fls = pf. true.
          IF (tfire(I, it).eq.0.) THEN
            if (history) print 3, t, color(m), 1
            color, color, it, nchan(I)
            if (history) print 3, t, color(m), 1
            color, color, it, nchan(I)
            t = fls tgt
            Restart search if it is turned off
            repeat = true.
            call schedul(t, 'search', NULL)
          ENDIF
        ELSE
          fot(I, it) = true.
        ENDIF
      ELSE
        Firl can still shoot, so have him select.
        if alive = if alive(I), it, FKILL
        if alive = if alive(I), it, FKILL
        if (alive) THEN
          Firer can shoot, so have him select.
          if alive = if alive(I), it, FKILL
          if alive = if alive(I), it, FKILL
          if (alive) THEN
            Firer can still shoot, so have him select.
            if alive = if alive(I), it, FKILL
            if alive = if alive(I), it, FKILL
            if (alive) THEN
              Firer can still shoot, so have him select.
              if alive = if alive(I), it, FKILL
              if alive = if alive(I), it, FKILL
              if (alive) THEN
                Firer can still shoot, so have him select.
                if alive = if alive(I), it, FKILL
                if alive = if alive(I), it, FKILL
                if (alive) THEN
                  Firer can still shoot, so have him select.
if (history) print 1, t, color(m),
   k, color(I, I), level, nchan(I)
ENDIF
ELSE
  Fire has previously serviced this target.
  fot(I, I) = true.
  if (history) print 1, t, color(m),
     k, color(I, I), level, nchan(I)
ENDIF
          call engage (t, t, 1, it)
ENDIF
  arg(I) = it
ENDIF
if (trace) print *, '<select'
END
4.2 Select: Start Target Selection if Appropriate. The subroutine `selec` determines whether the program will start the selection of a target immediately or wait. The program will pause if the gunner is already selecting, if the channels are full, or if the pod is empty.

The subroutine `selec` is the subroutine which calculates whether or not the target selection routines will be called. When `selec` is called by other routines in the Tank Wars II program, it first checks to see if the firer is busy, the firer has no missiles, or the channels are full. If any of the previous situations exist, then `selec` prints out a message stating that the selection routines will not start and gives the reason for the delay. If the firer is free and ready to start selection, then `selec` changes the status of the firer to busy and calls the subroutine `select` to start choosing a target.

```
CODE | COMMENT
--- | ---
t | Time (sec).
I | ID of firer (integer).
dt | Time required to select a target (sec).
m | Side of firer. 1 if Blue, 2 if Red. (Integer)
kind | Kind of round.
busy() | True IFF firer is too busy selecting a target already (logical).
empty() | True if raised missile pod is empty (logical).
also | True if entirely out of ammo??
loaded | True IFF all missile guidance channels are loaded (logical).
nchan(firer) | Number of busy missile guidance channels.
```

C V7.2
SUBROUTINE SELECS (t, l, dt)
include 'common.h'
logical loaded
1 format (f8.2,1x,a4,i3,' does not select; selecting already.')}}</code>
2 format (f8.2,1x,a4,i3,' does not select; channels full.')</code>
3 format (f8.2,1x,a4,i3,' does not select; pod empty.')</code>
4 format (f8.2,1x,a4,i3,' begins selection.')</code>
c if (trace) print *,'>selecs'
m = army(I)
k = kindrd(m)
if (kind.eq.4) loaded = nchan(l).ge.nchans(m)
if (kind.ne.4) loaded = nrtgt(l).ne.0
IF (busy(l) or. empty(l)).or. loaded) THEN
Wait cause bus selecting, pod empty, or channels full.
IF (busy(l)) THEN
print 1, t, color(m), 1
ELSEIF (loaded) THEN
print 2, t, color(m), 1
ELSEIF (empty(l)) THEN
print 3, t, color(m), 1
ENDIF
ENDIF
ELSE
Start selection: none in progress and a channel is free.
busy(l) = true.
if (kind.eq.4) nchan(l) = nchan(l)+1
call sched(t+dt,l,'select', NULL)
if (history) print 4, t, color(m), 1
ENDIF
if (trace) print *,'<selecs'
END
4.3 Priorn: Select Target With Highest Priority. Priorn simulates a firer selecting its next target. After removing any targets already being engaged, the program compares the remaining targets if 1) it has detected the target, 2) the target is not dead, 3) and is within 4 km. If any two targets have the same priority, the program selects the one that was least recently engaged. If neither has been engaged, the program selects the closest target.

Figure 4 illustrates the order in which priorn executes:

![Diagram](image)

Figure 4. Selection of Highest Priority Target

The subroutine priorn creates a dummy target that is given an extreme range, and made to be more recently engaged than all other possible targets. The code assigns the dummy target a priority of 1000 and then compares all other tanks, each time selecting the one with the highest priority (lowest priority number).

The DO 30 loop compares each tank in turn with the last selected target and selects the higher priority of the two. First it checks to see if either a friendly tank or the firer is guiding or firing a missile to the target. If the target is being serviced, it is not considered in the comparisons. This option should only be used to conserve expensive missiles, and this only happens with missile systems having more than one guidance channel. The code also makes sure the firer does not select targets that it doesn't see, ones it knows are dead, and ones beyond 4 km. When a target that is in view, is alive, and is not being engaged has been chosen, the subroutine calls priort to assign the target a priority number.

After the priority number is assigned, the code 'fuzzes' the range, finds how recently the firer has fired on the target, and determines whether this is a 'better' target. (Better means higher priority.)

The range is 'fuzzed' because the crew cannot estimate range perfectly and will tend to pick the target they think is closer. This avoids several firers selecting the same tank simply because it is a tiny distance closer. The 'fuzzing' is done by adding a random amount to the range. This random amount is chosen from a normal distribution with mean zero and standard deviation equal to 5% of the true range.

If the current target and best previous target have equal priority, the code breaks ties. If the targets are new, then the closest one will be given higher priority. If the targets have previously been fired on, the code chooses the one that has been least recently fired upon. This spreads the fire over targets instead of concentrating on a single target.

Finally, if the current target has a lower priority number, it replaces the previous best choice.
CODE

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>levold</td>
<td>Priority level of highest priority target (1..22).</td>
</tr>
<tr>
<td>armyf</td>
<td>Side of firer. 1 if Blue, 2 if Red (integer).</td>
</tr>
<tr>
<td>rgold</td>
<td>Range to highest priority target (m).</td>
</tr>
<tr>
<td>told</td>
<td>Time firer last serviced highest priority target (sec).</td>
</tr>
<tr>
<td>priorn</td>
<td>Priority of current candidate (integer).</td>
</tr>
<tr>
<td>pick</td>
<td>True IFF the tank is a candidate for selection (logical).</td>
</tr>
<tr>
<td>share()</td>
<td>True IFF tanks on a side know which targets are engaged by friends (logical).</td>
</tr>
<tr>
<td>mot(i,j)</td>
<td>True if missile is on (assigned to) a target (logical).</td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>True if firer i is on (servicing) target j (logical).</td>
</tr>
<tr>
<td>see(i,j)</td>
<td>True IFF firer i sees target j (logical).</td>
</tr>
<tr>
<td>life(j)</td>
<td>Status of target j. IFF less than IKILL, it's considered threatening.</td>
</tr>
<tr>
<td>level</td>
<td>Priority level of current candidate (1..22).</td>
</tr>
<tr>
<td>rg tgt</td>
<td>Approximate range to target (m).</td>
</tr>
<tr>
<td>t tgt</td>
<td>Time firer last serviced target (sec).</td>
</tr>
<tr>
<td>better</td>
<td>True IFF priority of current candidate is highest found so far (logical).</td>
</tr>
</tbody>
</table>

c V7.3 INTEGER FUNCTION PRIORN (t, I, lev old)

ENDIF

ENDIF

CONTINUE

if (trace) print *, 'priorn'

END

c 6 Priorn: Select target with highest priority.
include 'common.h'
logical better, ck tgt, pick
integer I, armyf

if (trace) print *, 'priorn'
armyf = army(I)

'make' dummy tgt for comparison
rg old = 1.e35
t old = 1.e35
lev old = 1000
priorn = NULL
last = nblu+nred
DO 30 mtgt = 1, last

30 CONTINUE

ELSE

DON'T select this tgt if anyone is already servicing it.
DO 20 jfirer = 1, last

i f (mot(j firer, mtgt). or. fot(j firer, mtgt)) pick = false.

20 CONTINUE

ENDIF

g t g t = r g (t, I, mtgt)
ck tgt = see(I, mtgt) and. life(mtgt).lt. IKILL

and. r g tgt .le. 4000.0 and. pick

IF (ck tgt) THEN

Firer sees tgt, it's threatening. & he's not firing at it.
call priort(I, mtgt, rg tgt, t, level)

c Now pick the tgt with highest priority
rg tgt = rg tgt * (1+.05*ran(1.0))
t tgt = tref(I, mtgt)
better = level .lt. lev old

IF (lev old .eq. level) THEN

Same priority class; now break ties

c if new tgt is closer
if (t tgt .le. 0) better = rg tgt .lt. rg old
c if old tgt is closer (least recently fired on)
if (t tgt .gt. 0) better = t tgt .lt. t old

ENDIF

IF (better) THEN

lev old = level
t old = t tgt
rg old = rg tgt
priorn = mtgt
4.4 Priort: Find Priority of a Single Target. Each time priort is called, it assigns a priority number to a single target. A target is given a priority after the consideration of whether or not the target has been shot at previously, if it has been hit, how close it is, whether or not it has fired recently, and if it is moving, slowing, or stationary.

Table 2 shows the factors taken into account when assigning priority to a target.

Table 2. Factors in Target Selection

<table>
<thead>
<tr>
<th>Preferred Choice</th>
<th>Less Desirable Choice</th>
<th>Rationale</th>
<th>How Modeled</th>
</tr>
</thead>
<tbody>
<tr>
<td>Close tgt</td>
<td>Far tgt</td>
<td>1 Easier to hit</td>
<td>1 Tgts within 1.5 km given higher priority</td>
</tr>
<tr>
<td>A tgt that fired recently</td>
<td>A tgt that hasn't</td>
<td>2 More dangerous</td>
<td>2 If tgts have equal priority, select closest</td>
</tr>
<tr>
<td>A tgt you missed</td>
<td>A tgt you hit</td>
<td>A firing tgt is more dangerous</td>
<td>Tgts that fired in the last 30 sec given higher priority</td>
</tr>
<tr>
<td>A tgt not being approached by a missile</td>
<td>A tgt being approached by a missile</td>
<td>1 Conservation of missiles 2 Tgt being approached has less chance of shooting back.</td>
<td>Tgts being approached by a missile will not be selected!</td>
</tr>
<tr>
<td>An old tgt</td>
<td>A new tgt</td>
<td>A new tgt may be a false tgt</td>
<td>Old tgt given higher priority *IF USER DESIRES</td>
</tr>
<tr>
<td>A new tgt</td>
<td>An old tgt</td>
<td>Old tgt is partially serviced so it may be dead</td>
<td>New tgt given higher priority *IF USER DESIRES</td>
</tr>
<tr>
<td>A tgt that is stopped or slowing</td>
<td>A tgt that is accelerating</td>
<td>1 Tgt that is stationary is easier to hit 2 Tgt may be stopping to shoot</td>
<td>Stopped or slowing tgt is given higher priority</td>
</tr>
<tr>
<td>A tgt that has a tgt</td>
<td>A tgt that has no tgt</td>
<td>1 Tgt that has tgt is threatening 2 Tgt that has tgt is known to be active</td>
<td>A tgt that has a tgt is given higher priority</td>
</tr>
</tbody>
</table>

*A target that has been previously fired on should be given higher priority if the probability of an F-kill is low. A new target that has not been engaged should be given higher priority if the probability of it being a false target is low.

After all factors are taken into account, a list of selection priorities can be made. The list combines the preferences found in Table 2 with information about the targets’ movements (stationary, slowing, or active). Table 3 lists the priority for each set of target conditions. The first column should be used when the probability of F, M, or K kill for the target is low. The second column should be used if the probability of the target being a false target is low.
<p>| | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Close old tgt missed</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>Close old tgt missed</td>
<td>that has tgt</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>Close old tgt missed</td>
<td>that is stopped or slowing</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Far old tgt missed</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>5</td>
<td>Close old tgt missed</td>
<td>all others</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>6</td>
<td>Far old tgt missed</td>
<td>that is stationary</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>7</td>
<td>Far old tgt missed</td>
<td>all others</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>8</td>
<td>Close new tgt</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>9</td>
<td>Far new tgt</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>10</td>
<td>Close old tgt hit</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>11</td>
<td>Close old tgt hit</td>
<td>that has tgt</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>12</td>
<td>Close old tgt hit</td>
<td>that is stopped or slowing</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>13</td>
<td>Far old tgt hit</td>
<td>that fired in last 30 sec</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>14</td>
<td>Close old tgt hit</td>
<td>all others</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>15</td>
<td>Far old tgt hit</td>
<td>that is stationary</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>16</td>
<td>Close new tgt</td>
<td>that is stationary</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>17</td>
<td>Close new tgt</td>
<td>that has tgt</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>18</td>
<td>Close new tgt</td>
<td>that is stopped or slowing</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>19</td>
<td>Close new tgt</td>
<td>all others</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>20</td>
<td>Far new tgt</td>
<td>all others</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>21</td>
<td>Far new tgt</td>
<td>all others</td>
<td></td>
</tr>
</tbody>
</table>

The code calculates whether or not the target has been shot at previously or not. If it has been shot at, the code then finds out if it was hit. If the target was not hit, then the subroutine checks to see if the target is within recognition range. If the target is within 1500 meters, then the code determines if the target is slowing or stationary. If it is, it is assigned a priority of 3. If the target appears to be preparing to engage, it is given a priority of 2. If the target has fired within the last 30 seconds, the code assigns it a priority of one. Any other target within 1500 m that has been shot at but not hit is assigned a priority of 5.

If the target has been shot at and missed but is beyond recognition range, then the code determines if the target is stopped. The code does not determine whether or not the target is slowing, because at that range it would not be possible to tell. If the target is stationary, it is given a priority of 6. It is also impossible to tell if the target is aiming from beyond recognition range, so the code does not determine if the target is preparing to engage. However, if the target has fired a shot in the last 30 seconds, the code gives it a priority of 4. Any other target beyond 1500 m that has been fired on but not hit is assigned a priority of 7.

If the target was hit when it was fired on, the code determines if the target is within recognition range. If the target is within 1500m, then the code calculates if it is stationary or slowing. If it is, then it is given a priority of 12. If the target appears to be preparing to engage, it is given a priority of 11. If the target has fired in the last 30 seconds, the code assigns a priority of 10. Any other target within 1500 m that has been hit receives a priority of 14.

If the target was hit but is beyond recognition range, the code assigns it a priority of 16. If it is stationary, the priority is set at 15. If it has fired in the last 30 seconds, the priority is raised to 13.

If the target has never been fired upon, the code first calculates if the target is within recognition range. If it is within 1500m and is either stationary or slowing down, it is assigned a priority of 18. If the target appears to be preparing to engage a new target, then it is assigned a priority of 17. If the target has fired in the last 30 seconds, the priority is 8. Any other targets that are new and close are given a priority of 19.
If the target has not previously been engaged, and is far away, the code assigns it a priority of 21. If it is stationary, then the target’s priority is changed to 20. If the target has recently fired, the priority is set at 9.

```fortran
SUBROUTINE PRIORT(i, it, rgtgt, t, L)
  include 'common.h'
  logical missed
  common /MayPri/ missed(NN,NN)
  dimension lev(21,2)
  save /MayPri/, lev
  data lev/1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16,17,18,19,20,21,1,2,3,4,5,6,7,8,9,10,11,12,13,14/  
  format('PRIORT:', a4,i3,' considrs ',a4,i3,' with priority',  
  1 i:4, ('(',j2,')'))
  i = nprior(army(l))
  m = motion(it)
  tactiv = 1.0e5
  if (tfire2(t),gt0.) tactiv = t-tfire2(it)
  IF (tfire2(it),gt0.) THEN
    c I have already shot at this target previously.
    IF (missed(i,it)) THEN
      c Missed target with last round fired at it
      IF (rgtgt(it).lt.recknz(army(l))) THEN
        c Target is within recognition range.
        L = 5
        if (m.eq.STATNY .or. m.eq.SLOWNG) L = 3
        if (nrtgt(it).ne.0) L = 2
        if (tactiv .lt. 30.) L = 1
        ELSE
          c Target is beyond recognition range.
          L = 7
          if (m.eq.STATNY) L = 6
          if (tactiv .lt. 30.) L = 4
        ENDIF
      ELSE
        c I hit target with last round fired at it.
        IF (rgtgt(it).gt.recknz(army(l))) THEN
          c Target is within recognition range.
          L = 14
          if (m.eq.STATNY .or. m.eq.SLOWNG) L = 12
          if (nrtgt(it).ne.0) L = 11
          if (tactiv .lt. 30.) L = 10
          ELSE
            c Target is beyond recognition range.
            L = 16
            if (m.eq.STATNY) L = 15
            if (tactiv .lt. 30.) L = 13
          ENDIF
        ENDIF
      ENDIF
    ELSE
      c Target is a new target
      IF (rgtgt(it).eq.army(l)) THEN
        c Target is within recognition range.
        L = 19
        if (m.eq.STATNY .or. m.eq.SLOWNG) L = 18
        if (nrtgt(it).ne.0) L = 17
        if (tactiv .lt. 30.) L = 8
        ELSE
          c Target is beyond recognition range.
          L = 21
          if (m.eq.STATNY) L = 20
          if (tactiv .lt. 30.) L = 9
        ENDIF
      ENDIF
      L = lev(L,j)
      if (trace) print *, '<priort'
    END
  END
```
4.5 Engage: Begin Engagement of a New Target by This Firer. Engage starts the engagement of the newly selected target by the firer. It sets the firer in a position to fire, determines the range to the target, and then engages.

Engage begins by determining if the firer still has the ability to fire and has any rounds left. If the gunner is capable of engaging, then the subroutine calculates its velocity. The code will slow down any firer still in motion. When the firer has become stationary, engage finds the range to the target, and then prepares to fire.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t1, t2</td>
<td>Current time (sec).</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>Side of firer. 1 if Blue, 2 if Red.</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>Side of target. 1 if Blue, 2 if Red.</td>
<td></td>
</tr>
<tr>
<td>life(I)</td>
<td>Status of firer. Fully alive, mobility killed, etc.</td>
<td></td>
</tr>
<tr>
<td>nrd(I)</td>
<td># rounds fired by firer.</td>
<td></td>
</tr>
<tr>
<td>nrds(m)</td>
<td># rounds on board tanks on side m.</td>
<td></td>
</tr>
<tr>
<td>nbst(I)</td>
<td># Rounds fired in burst.</td>
<td></td>
</tr>
<tr>
<td>ishtfs(m)</td>
<td>1 if tanks on side m halt to fire. Zero otherwise.</td>
<td></td>
</tr>
<tr>
<td>motion(I)</td>
<td>1..4 if tank I is braking, stationary, accelerating, cruising</td>
<td></td>
</tr>
<tr>
<td>speed(m)</td>
<td>Combat cruise speed for tanks on side m.</td>
<td></td>
</tr>
<tr>
<td>rg</td>
<td>Range to target (m). Use opening range if false target.</td>
<td></td>
</tr>
<tr>
<td>nrg</td>
<td>Range band.</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>$\Delta t = t_{fire} - t$</td>
<td></td>
</tr>
<tr>
<td>prevrd(I)</td>
<td>Time to fire first round (sec).</td>
<td></td>
</tr>
<tr>
<td>nrib(I)</td>
<td>=1 implies this is the first round fired at the target.</td>
<td></td>
</tr>
<tr>
<td>nrot(I)</td>
<td>=0 implies firer I is just beginning a burst (if it fires bursts).</td>
<td></td>
</tr>
<tr>
<td>COUNT of rounds on target.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

SUBROUTINE ENGAGE (t1, t2, I, it)

if (trace) print *,’<engage'
include 'common.h'

m = army(I)
$n =inan(3, m)
IF (life(I).lt.FKILL and nrd(l).lt.nrds(m)) THEN
nbst(I) = 1
IF (ishtfs(m).gt.0 .AND. motion(I).ne.STATNY)
1
and. speed(m).gt.0) THEN
HALT to fire
CALL cancel (I,’maxvel’,NULL)
call cancel (I,’accel’,’NULL’)
call skedul (t1,’slowup’,NULL)
ELSE
ENDIF
END

if(kindrd(rn).eq.4) dt-0.1
call skedul (t2+dt,’fire’,I)
ENDIF
5. FIRING ROUTINES

The event subroutine `fire` simulates firing a round and schedules the effects. The appropriate events depend on the ammunition status, number of shots fired, and type of round: gun burst, single shot gun, or missile. If the firer is out of ammo, it will either attempt to hide or schedule a reload if it is a missile system.

The diagram below shows the major routines called by `fire`. This section discusses the ones in solid boxes; they are most closely related to firing. The `pinpnt` routine is discussed with the other detection routines, `create` is discussed with the utility routines, and `selecs` is discussed with the target selection routines. The arrow leading into the `reload` box indicates `frdmsl` calls `reload` indirectly rather than directly. It does this via the clock routines.
5.1 Firing Cycles. Each type of weapon on armor has its own timing characteristics. We'll discuss some or all of these:

- Cannon w/manual loader
- Cannon firing 'bursts'
- Cannon w/load assist
- Guided missiles
- Cannon w/auto loader
- Beam Weapons

Human reaction times. Delay times in the firing cycle include human reaction times. These times are approximately log-normally distributed. That is, the logarithms of the reaction times are normally distributed, typically with $\mu = 0$, and $\sigma = 0.5$.

If $N(0.5)$ is a random draw from such a normal distribution, $t_m$ is the median time for the log-normal, and $t_h$ is a randomly chosen human reaction time, then:

$$t_h = t_m e^{N(0.5)}$$

First round time. The model assumes that the first round is loaded when the tank engages a target. The time to launch the first round at a target is a function of the range to the target. Why? Perhaps the gunner considers a distant target less threatening and more difficult to hit, so he takes more time to aim carefully. Table 4 contains sample values against a stationary target. The time to fire the first round at a moving target is proportionally longer and should be added to the model in the future.

### Table 4. Times From Target Selection to Launch

<table>
<thead>
<tr>
<th>Range (m)</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
<th>3000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Time (sec)</td>
<td>10.0</td>
<td>11.0</td>
<td>12.0</td>
<td>13.0</td>
<td>14.0</td>
<td>15.0</td>
</tr>
</tbody>
</table>

Time to subsequent launch. The time between rounds fired at the same target depends on the type of armament and load mechanism. The model uses eight values for fixed times as a function of range. It also uses the median, minimum, and intra-burst times.

**Manual loader.** For manually loaded guns, the time between rounds is a random human reaction time. The delay time is:

$$\Delta t = t_m e^{N(0.5)}$$

If the median time between rounds is 10 seconds then the mean time between rounds will be approximately 11.4 seconds.

**Load assist.** Certain tank guns have a load assist mechanism which performs its task in a fixed time $t_c$ in series with a human who performs his task in a log-normally distributed time. For this type of system:

$$\Delta t = t_c + t_m e^{N(0.5)}$$

**Auto-loader.** Other tank guns have an auto loader which performs its task if a fixed time in parallel with a human who performs his task in a log-normally distributed time. For these times the delay is:

$$\Delta t = \max(t_c, t_m e^{N(0.5)})$$

**Burst fire cannon.** Tank cannons which fire, say 3 rounds, in a burst have been postulated. These have a fixed time between rounds in a burst on the order of, say, two seconds. The time between the last round in a burst and the first round in a subsequent burst at the same target is random, based on human response times. The time between rounds in a burst is:
The time between bursts is:

$$\Delta t = t_{ref}$$

$$t = t_m e^{N(0,0.5)}$$

The code for burst firing must be re-introduced into Tank Wars.

**Guided missiles.** Missile systems may guide $n$ missiles to $n$ targets, where $n$ is the number of guidance channels. Many missile systems have only a single guidance channel. In any case, when the guidance channels are full, the system must wait until a missile impacts or is aborted before firing another missile. For systems with a single channel, the time between rounds is the time of flight, which in turn is a function of the range to the target.

$$t = t_f(rg) = \text{time of flight}$$

Some conceptual systems have multiple guidance channels and are able to fire on $n$ targets simultaneously. For such systems the time between launches may be a fraction of a second until all $n$ guidance channels are busy, then the launch of the $n+1$st missile must wait until a channel is free. If a guidance channel is free, the time between launches is:

$$t = t_c$$

Otherwise, the time between launches depends on when a channel becomes free and is not a direct input to the model.

These conceptual systems may hold fire until $n$ targets are designated and then launch $n$ missiles a fraction of a second apart. Tank Wars doesn’t yet model this hold-fire technique but it should be added. It will require careful thought about what happens when there are fewer than $n$ targets or fewer than $n$ missiles remaining.

**Beam weapons.** If such weapons fire several times at one target, the time between shots will depend primarily on a fixed time to recharge. If $t_r$ is the recharge time, then perhaps:

$$\Delta t = \max(t_r, t_m e^{N(0,0.5)})$$

55
5.2 Fire: Simulate Firing of a Round and Schedule Effects. Fire simulates the firing of a round, updates and saves related values, and schedules effects of the firing. These effects include impact of the round, detection of the firer by its firing signature, and the next activities of the firer. The firer's next activities depend on whether it fires single shots, bursts, or guided missiles. The firer may fire again, switch targets or move. Missile firers may also replace an empty missile pod (reload) or simply wait until impact.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
<td></td>
</tr>
<tr>
<td>I</td>
<td>firer</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>Target</td>
<td></td>
</tr>
<tr>
<td>busy(l)</td>
<td>T/F. Tentatively set to false. This permits the tank to select a new target.</td>
<td></td>
</tr>
<tr>
<td>m,n</td>
<td>Side of firer, target</td>
<td></td>
</tr>
<tr>
<td>nrd(l)</td>
<td>Number of rounds fired by tank</td>
<td></td>
</tr>
<tr>
<td>nrib</td>
<td>Number of rounds fired in burst</td>
<td></td>
</tr>
<tr>
<td>nrpb</td>
<td>Number of rounds per burst</td>
<td></td>
</tr>
<tr>
<td>nrot(l)</td>
<td>Number of rounds fired at current target</td>
<td></td>
</tr>
<tr>
<td>it &gt; 0</td>
<td>true if a real target. ID of false tgt is -1</td>
<td></td>
</tr>
<tr>
<td>tfire(l,it)</td>
<td>Save time firer fired at target. When switching targets, firer will give priority to least recently serviced tgt.</td>
<td></td>
</tr>
<tr>
<td>tfire2(l)</td>
<td>Save time firer last fired. When selecting tgt, foes will give priority to tgts that fired recently.</td>
<td></td>
</tr>
<tr>
<td>it = FLSTGT</td>
<td>Implies target is a false target. Code must generate position and set velocity to zero.</td>
<td></td>
</tr>
<tr>
<td>rg</td>
<td>Range of target (m).</td>
<td></td>
</tr>
<tr>
<td>nrg</td>
<td>Number of range band</td>
<td></td>
</tr>
<tr>
<td>irginc</td>
<td>Size of range bins (m).</td>
<td></td>
</tr>
<tr>
<td>s(i)</td>
<td>Position of tank in question. i=1,2,or3</td>
<td></td>
</tr>
<tr>
<td>iflash(l)</td>
<td>T for flashing decoy, F for passive decoy.</td>
<td></td>
</tr>
<tr>
<td>bullet</td>
<td>ID of bullet.</td>
<td></td>
</tr>
<tr>
<td>tfly</td>
<td>Time of flight (sec).</td>
<td></td>
</tr>
<tr>
<td>tof</td>
<td>Time of flight table (sec).</td>
<td></td>
</tr>
<tr>
<td>psense</td>
<td>Probability of sensing the impact location of a miss</td>
<td></td>
</tr>
<tr>
<td>vx0,vy0</td>
<td>v</td>
<td>Current velocity of firer or tgt (m/s).</td>
</tr>
<tr>
<td>kindrd</td>
<td>1 for KE,2 for HEAT,4 for missile,5 for STAFF</td>
<td></td>
</tr>
</tbody>
</table>

Initially busy is set to false, although this may not be true for missile systems. We may have to reset it to true depending if it is a simple or multi-missile system.

By counting the number of rounds fired in a burst(nrib) and the number of rounds per burst(nrpb), we can determine if the burst is just starting, in the midst, or over. If the burst is just starting we must draw errors for the burst, but if the burst is over the firer can switch targets or pop-down. The firer may want to change targets based on the policy of firing a fixed number of rounds at a target, therefore, we need to count the number of rounds fired at the current target(nrot).

Tfire and tfire2 are used in a target’s selection process based on previous set priorities. The last time the firer fired at a target (tfire) is needed if the firer returns to service the least recently engaged target. Tfire2 saves the last time the firer fired at any target. This is used by the target’s foes and priority is given to recent firers.

Velocities, positions are updated and used to find the time of flight of round, and saved for use at impact time. No velocity or position, however, is calculated for false targets, although a dummy position is picked at x=0,z=0, and y=+/- opening range. A false target is a natural object such as a bush or stone.
mistaken for a real target. A real target updates its position by using rgf(range of firer).

*Pinpnt* is called to find out if any foes detected the firer due to its muzzle flash.

If iflash(I) = 0, this is a real firer, otherwise it is a flashing decoy. In this branch, we are only concerned with a real firer and the round it fires. **Fire** calls **create** to set aside space for information about the round. In this space, **Fire** stores: 1) target ID, 2) firer ID, 3&4) predicted (x,y) position of target at impact, 5&6) unused, 7) probability that the impact location of the bullet will be sensed, 8) unused, 9) speed of target when round was fired at it, and 10) speed of firer.

The number of shots fired by the firer's side is counted for output statistics by kshot(m,1). Impact for the round is then scheduled. For missiles systems, assign a guidance channel to the missile.

This completes calculations for the bullet that was fired. Non-missile systems must select single shot or burst fire code, while missile systems must select missile code.

```fortran
SUBROUTINE FIRE (t,I,it)
integer bullet if (nichan(1).lt.nichans(m)) call frdmsl(t,f,itm)
1 format(f8.2, iX, a4, i3, 'fires at', a4, i3)
2 format(f5.2, iX, a4, i3, 'ran out of ammo.')

if (trace) print 1,'<fire'
END

busy(I)=false.
m = army(I)
if (history) print 1,1,color(m),I. color(n),it

Update rd counts, time of last fire.

nrd(I) = nrd(I)+1
nrb(I) = nrb(I)+1
if(nrb(I).gt.nrb(m)) nrb(I)=1
nrot(I) = nrot(I)+1
if (it.gt.0) tfire[I,lt] = t
End tfire[I] = t

Update positions & velocities.

If (it.eq.FLST) THEN
    rg = rg0
    s(1) = 0.0
    s(2) = 0.0
    s(3) = 0.0
    IF (m.eq.BLU .and. scene.eq.BATTACK) or ...
    ELSE
    dm = rg(t,lt,1)
    ENDIF
    call pinpnt (I)
    IF (iflash(I).eq.0) THEN
    c Branch for real firer (do nothing if firer is flashing decoy)
    c Create round with various attributes
    call create (10,bullet)
    a(bullet+1) = it
    a(bullet+2) = I
    tfly = tof(m,nrg)
    a(bullet+3) = s(I)+ttfly*x0(I)
    a(bullet+4) = s(I)+ttfly*y0(I)
    a(bullet+7) = pspeed(m,nrg)
    a(bullet+9) = sqrt(xv0(I)**2+yy0(I)**2)
    if (it.eq.1) a(bullet+10) = 0.0
    if (it.gt.0) a(bullet+10) = sqrt(xv0(I)**2+yy0(I)**2)
    kshot(m,I) = kshot(m,I) + 1
    c Schedule impact for rd & allot guidance channel.
    call skedul (tt,ttfly,bullet,'impact',it)
    IF (kindrd(m).eq.4) THEN
    IF (it.eq.0) mot[I,lt] = .true.
    DO 20 k=1,5
    IF (chanel(m,k).eq.0) GOTO 25
    20 CONTINUE
    25 chanel(m,k) = bullet
    ENDIF
    ENDIF
    c Move, fire, or switch targets as required
    IF (kindrd(m).eq.2 .or. kindrd(m).eq.5) THEN

```
5.3 Frdssg: Results of Firing a Single Shot Gun. Frdssg schedules what the firer does after firing a single shot gun. The primary consideration is whether it has more rounds or not. If so, it will switch to a new target or continue to fire at the current target. If not, it will hide if it can move.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrd(I)</td>
<td>r</td>
<td># rounds fired by tank.</td>
</tr>
<tr>
<td>nrds(n)</td>
<td>( r_{\text{max}} )</td>
<td>magazine capacity (Ammo load).</td>
</tr>
<tr>
<td>tactic(n)</td>
<td>3</td>
<td>firer has more rounds to shoot.</td>
</tr>
<tr>
<td>nrot(I)</td>
<td>Number of rounds fired at current target.</td>
<td></td>
</tr>
<tr>
<td>nrotpt(n)</td>
<td>Number of rounds per target before switching targets.</td>
<td></td>
</tr>
<tr>
<td>busyn(I)</td>
<td>False for tank not busy; True for tank is busy.</td>
<td></td>
</tr>
<tr>
<td>loader(n)</td>
<td>k 1 for manual loader, 2 for automatic loader then manual, 3 for automatic loader parallel with manual gunning.</td>
<td></td>
</tr>
<tr>
<td>tvar</td>
<td>Median between rounds of tank cannon.</td>
<td></td>
</tr>
<tr>
<td>dm</td>
<td>Random human reaction time (sec).</td>
<td></td>
</tr>
<tr>
<td>tcon(n)</td>
<td>Minimum time to fire tank cannon (sec).</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>For ( k = 1 ) (manual loading). For ( k = 2 ) (series auto-loading). For ( k = 3 ) (parallel auto-loading).</td>
<td></td>
</tr>
<tr>
<td>empty(I)</td>
<td>No ammo.</td>
<td></td>
</tr>
</tbody>
</table>

**Does the firer have more rounds to fire?** The number of rounds it fired is \( \text{nrd}(I) \) and the number of rounds it started with is \( \text{nrds}(n) \). It has more rounds if \( \text{nrd}(I) < \text{nrds}(n) \).

**Should it switch targets?** If the firer has more rounds to shoot, the next consideration is whether it switches targets or continues to fire at the current target. If \( \text{tactic}(n).eq.3 \), then the policy is to fire a fixed number of rounds at a target and then switch targets. The number of rounds to fire is \( \text{nrot}(I) \), so if \( \text{nrot}(I).eq.\text{nrpt}(n) \), the policy has been satisfied and the firer attempts to switch targets.

**Switching targets.** Upon disengaging, a halt-to-fire system that can still move will move before firing at the new system, so this kind of firer is scheduled to accelerate now.

**Firing again at the current target.** If the tank switches targets after firing a fixed number of rounds at the target and has done so, it will switch targets. Otherwise, the code schedules the next fire at the target. The time the next round will be fired depends on the loader type.

**Out of ammo.** The code for a system that has ammo ends here and the code now treats the tank that is out of ammo. The tank is out of ammo when \( \text{empty}(I) = .true. \). If the tank is not going but can move, the code schedules an acceleration event right away and a hide event in \( \text{thide}(n) \) seconds. Since the tank cannot shoot any more it seeks cover.

```fortran
SUBROUTINE FRD SSG (t, I, it, n)
   t - time (sec).
   I - firer
   it - target
   \( n \) - side firer is on
   include 'common.h'
   logical cango, done, tactc3
   1 format('FRD SGG: t,I,it,n-',7.2,3i3)
   2 format(f8.2,ix,a4,i3,'is out of ammo. Will attempt',
      \quad 'to hide if mobile.')
   c if (trace) print *,>'frd ssg'
   IF (\text{nrds}(n)) THEN
   IF (\text{nrds}(I).lt.\text{nrds}(n)) THEN
   c Have ammo branch
   tactc3 = tactic(n).eq.3
   done = nrot(I).eq.\text{nrpt}(n)
   IF ((tactc3 .and. done)) THEN
   c Switch targets after firing a fixed nr of rds at it
   busyn(I) = .false.
   ENDIF
   ELSEIF (\text{empty}(I)) THEN
   c Out-of-ammo branch
   \text{empty}(I) = .true.
   IF (\text{cango}(I)) THEN
   call skedul(t+dt,l,['fire','it])
   ENDIF
   ELSE
   c Schedule effects after firing single shot gun.
   call dis eng (t, I, it,.,true.,true.)
   if (\text{can go}(I.t)) and .\text{ishfs}(n).eq.1
   c No other it and can move, skedul acceleration
   1 call skedul(t,1,'accel',NULL)
   nrot(I) = \text{nrpt}(n)
   
   c If no other it and can move, skedul acceleration
   if (\text{can go}(I,t) .and. .\text{ishfs}(n).eq.1)
   c out of ammo branch
   \text{empty}(I) = .true.
   else
   c Schedule next round fired
   \( k = \text{loader}(n) \)
   \( \text{dm} = \text{tvar}(n) * \exp(\text{rann}(0.5)) \)
   if (k.eq.1) \( \text{dt} = \text{dm} \)
   if (k.eq.2) \( \text{dt} = \text{tcon}(n) + \text{dm} \)
   if (k.eq.3) \( \text{dt} = \text{max}(\text{tcon}(n),\text{dm}) \)
   call skedul(t+dt,l,['hide',NULL])
   ENDIF
   ELSE
   c Schedule next round fired
   \( k = \text{loader}(n) \)
   \( \text{dm} = \text{tvar}(n) * \exp(\text{rann}(0.5)) \)
   if (k.eq.1) \( \text{dt} = \text{dm} \)
   if (k.eq.2) \( \text{dt} = \text{tcon}(n) + \text{dm} \)
   if (k.eq.3) \( \text{dt} = \text{max}(\text{tcon}(n),\text{dm}) \)
   call skedul(t+dt,l,['fire','it'])
   ENDIF
ENDIF
```

58
ENDIF
if (trace) print *, ', <frd ssg'
END
5.4 Frb: Results of Firing a Round of a Burst. Frb schedules what the firer does after firing a round in a burst. If the firer is out of ammo and is mobile, it will attempt to hide. When the system has ammo it either disengages the old target after a certain number of rounds and searches for a new target or the firer schedules to fire the next round.

SUBROUTINE FRD.BST (t, firer, tgt, armyf)

Out-of-ammo branch
IF (ishtfs(armyf).eq.1 .and. can go(firer,t))
CALL skedul(t,firer,'accel','NULL')
ELSE
SWITCH targets after firing a fixed nr of rds at it
CALL dis eng(t, firer, tgt, true.,true.)
IF (halt-to-fire & no tgts & cango, skedul acceleration
if (ishtfs(armyf).eq.1 and can go(firer,t))
CALL skedul(t, firer, 'accel', 'NULL')
nrot(firer) = 0
ELSE
Schedule next round fired
timea = tcon(armyf)
timeb = tcon(armyf)
timec = tvar(armyf) * exp(rann(0.5))
dt = amax1(timea,timb+timec)
CALL skedul(t+dt,firer,'fire',tgt)
ENDIF
IF (trace) print *,'<frd bst'
END
5.5 Frdmsl: Results of Firing a Missile. Frdmsl schedules what the firer does after firing a missile. If the firer has more ammo, it may 'reload' (replace an empty missile pod), fire again at the current target, or switch targets. Otherwise, it does nothing further.

This routine is called after a missile is fired but not until a guidance channel is available for the next missile. If another guidance channel is available immediately after firing a round, the fire routine calls this routine. Otherwise, a guidance channel will become available at impact, so the impact routine calls this routine. (If the missile is aborted it is because the target went behind the terrain. The routine abort should be called immediately so it can fire again.)

If the system is out of ammo, it does nothing further.

If the current missile pod is empty, the firer is considered temporarily empty, any fire and select events are cancelled, and a reload is scheduled. Since the current target is discarded, the code resets the number of rounds on target (nrot=0).

If ammo is ready to be fired the system will either shoot again at the same target or switch targets. This depends on the firing policy it is using and whether it has satisfied that policy. Under tactic 3, the firer fires a fixed number of rounds at the target and then switches targets. If the firer is using this policy and has satisfied it, switching occurs; otherwise the next round is fired in 1/10 sec.

Finally, if the target ID is not zero, clear the record that the firer is on target. It is possible that 'it=-1'; this implies the target is false; however, there is no place in the fot matrix to store data for false targets.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer</td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
</tr>
<tr>
<td>m</td>
<td>1 if target is Blue, 2 if Red</td>
</tr>
<tr>
<td>nrd(I)</td>
<td>Number of rounds fired by ith tank</td>
</tr>
<tr>
<td>nrds(m)</td>
<td>Magazine capacity of systems on mth side.</td>
</tr>
<tr>
<td>nipods(m)</td>
<td>Number of rounds in pod for mth side.</td>
</tr>
<tr>
<td>tactic(m)</td>
<td>Side m fires a fixed number of rounds at tgt</td>
</tr>
<tr>
<td>nrot(I)</td>
<td>Number of rounds I fired at current target</td>
</tr>
<tr>
<td>nrpt(m)</td>
<td>Number of rounds per target before switching targets</td>
</tr>
<tr>
<td>empty</td>
<td>True missile pod is empty</td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>True IFF firer i on target j.</td>
</tr>
</tbody>
</table>

SUBROUTINE FRDMSL (t, l, it, m)

if (trace) print *, '>frdmsl!

IF (nrd(I).eq.nrds(m)) THEN
    empty(I) = true
    call cancel(I; 'fire ',it)
    call cancel(I; 'select',NULL)
    nrot(I) = 0
    subh ft that is slowing to engage speed up now
    call skedul (t+treold(m).l; 'reload',NULL)
    if (history) print 2.t,color(m).I
    ENDIF

ELSE
    IF (nrot(I).eq.nrpt(m)) THEN
        Schedule next round fired
        call skedul (t+1.I; 'fire ',it)
        ENDIF
    ELSE
        Treat empty missile pod

47
5.6 **Reload: Bring up Another Pod of Missiles.** The subroutine **Reload** simulates completion of reloading when a pod of missiles is empty. The primary consideration is whether the firer is a defender who has popped down to reload or is fully exposed while reloading. If it's a defender, it'll pop-up and begin searching. Otherwise, it's already fully exposed and attempts to select a target right away.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td>Time (sec)</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>ID of tank</td>
</tr>
<tr>
<td>firer</td>
<td></td>
<td>Number of the firer</td>
</tr>
<tr>
<td>nrtgt</td>
<td>0.5</td>
<td>Number of current target of tank</td>
</tr>
<tr>
<td>thuman</td>
<td></td>
<td>Human reaction time</td>
</tr>
</tbody>
</table>

```fortran
SUBROUTINE Reload (t,l)
  c V7.2
  c 6 Reload: simulates completion of reloading
  c 30 Oct 85 Fixed statement printing error message
  c  include 'common.h'
  logical defndr
  1 format(f8.2,lxa4,i3,' finishes reloading')
  2 format(f8.2,lx,a4,i3,' pops-up')
  c
  if (trace) print *,'>reload'
  m = army(l)
  if (history) print 1,t,color(m),l
  nrtgt(l) = 0
  empty(l) = .false.
  defndr = (scene.eq.BATTAK .and. m.eq.RED) .or.
            (scene.eq.BATTAK .and. m.eq.BLUE)
  IF (defndr) THEN
    c Defender pops back up and will start searching.
    if (history) print 2,t,color(m),l
    call appter(t,l,tgt,HD)
  ELSE
    c Attacker or 'meeter' never popped down.
    thuman = 2.0*exp(rann(0.5))
    call selecs(t,l,thuman)
  ENDIF
  if (trace) print '.'<reload''
END
```
6. HIT PROBABILITY AND IMPACT ROUTINES

These routines are called when the round passes through the target plane. If the target is a false target, the gunner seeks a new target. They find whether the round hits the target or not and whether it was a dud or not. Missile guidance channels are cleared and pop-down to reload is sometimes initiated. Under certain policies, the firer switches to a new target. If a hit is not a dud, the damage event is scheduled.

The figure below shows the calling hierarchy of the routines discussed in this section. The `accerr` routine and its subroutines are discussed in the next section.
6.1 Impact: Find What Bullet and Firer Do at Impact. The impact event simulates what occurs when the round passes through the target plane. The target may be a false target, in which case the gunner realizes it is a false target and switches to a new target. If the round is a direct fire round and the target is in full defilade no hit occurs. If the round is top attack or the target is exposed, impact finds if a hit occurs.

Finally, impact finds what the firer does. If the firer fires simultaneous missiles, the guidance channel is cleared. This firer disengages and if pods are empty, he may pop down to reload. Other types of firers may simply disengage the target.

First impact recovers some useful information about the round, then it figures out the effect of the round on the target, and finally, it decides what the firer does next. It recovers the target ID, and firer ID as well as finding which side the firer is on, what type of round it is firing and the exposure of the target.

Next it finds whether the round hit or not. The target may be a false target or a 'real' one. If the target is a false target (a natural object that was mistaken for a target), the code simply tallies the result for the summary statistics. If the target has vanished and the round is a typical ballistic round, the code tallies that the round hit the 'berm' (the intervening terrain). In this case, missiles have already been aborted and impact doesn't occur. Top attack rounds, however, still have a chance of hitting the target. If the target is 'real' and intervening terrain is no problem, impact calls the may hit routine to see if the round actually hits the target or not.

The rest of impact treats the future activities of the firer. If the round was a missile, impact clears the guidance channel, disengages the target, and attempts to select a new target. If the missile firer is in hull defilade, all guidance channels are free, and the missile pod is empty then the firer will pop down while it brings up another pod of missiles.

If the round was not a missile, the firer may switch targets. It does this if the current target is a false target, or is out of range (beyond 4km), or the firer's policy is to switch targets after each hit.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>bullet</td>
<td>ID of bullet</td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer</td>
</tr>
<tr>
<td>n</td>
<td>Side of firer (1=Blue, 2=Red)</td>
</tr>
<tr>
<td>k</td>
<td>Kind of round (1=KE, 2=HEAT, 4=msl, 5=TOP ATK)</td>
</tr>
<tr>
<td>rgx</td>
<td>Range to target (m).</td>
</tr>
<tr>
<td>mot(I,it)</td>
<td>True iff missile on target.</td>
</tr>
<tr>
<td>fot(l,it)</td>
<td>True iff firer on target.</td>
</tr>
<tr>
<td>nchan(l)</td>
<td>Number of busy guidance channels for firer</td>
</tr>
<tr>
<td>empty(m)</td>
<td>Number of guidance channels for side m tanks</td>
</tr>
<tr>
<td>hit</td>
<td>True iff current missile pod or system is out of ammo</td>
</tr>
<tr>
<td>tactic(m)</td>
<td>Target switching policy for side m.</td>
</tr>
<tr>
<td>ndet(l)</td>
<td>Number of detections lth tank has.</td>
</tr>
<tr>
<td>nrtgt(l)</td>
<td>ID of firer's latest target.</td>
</tr>
</tbody>
</table>

c V7.5
SUBROUTINE IMPACT (t, bullet)
c 0 Impact: find what bullet does & what firer does.
include 'common.h'
logical loaded, hit
integer bullet, expose
atr(i) = a(bullet+i)
if (trace) print *, '>impact'
c Find useful variables.
it = atr(1)
l = atr(2)
n = army(l)
k = kindr(n)
expose = kncal(it)
rgx = 0.0
c Find what bullet does.
IF (it eq FLS TGT) THEN
Round does nothing.

kshot(n,4) = kshot(n,4)+1
ELSEIF (expose.eq.FD .and. k.le.4) THEN
  Count round hitting berm.
  kshot(n,5) = kshot(n,5)+1
  if (history) print *, 'Tgt in full deflade.'
ELSE
  See if round hits.
  call mayhit(t,1,it,n,k,atr(9),atr(10),expose,hit,rgx)
ENDIF

if (bullet) = -a(bullet)

Find what firer does.
IF (k.eq.4) THEN
  Missile
  Clear guidance channel.
  DO 20 j=1,5
      IF (chanel(n,1),eq.bullet) GOTO 30
  20 CONTINUE
      print *, 'IMPACT: Msl not assigned a channel.'
      print *, 'Channels assigned to',(chanel(n,1),j=1,5)
      print *, 'Msl #=', bullet,' Contact Fred Bunn'
      STOP

30 CONTINUE
      chanel(n,1,j) = NULL
      loaded = nchan(j).ge.nchans(n)
      call diseng (t,1,it,.true., loaded)
      mot1(it) = .false.
      fot(it) = .false.
      if ((kncrl(j),eq.HD .and. nchan(j),eq.0 .and. empty(id))
        call skedul(t,1,'popdn','NULL')
      ELSE
        KE, HEAT, or STAFF [rethink this for STAFF]
        IF (it.eq.FLS TGT .or. hit .and. tactic(n),eq.2 .or.
          rgx.gt.4000.0) THEN
            Switch targets if false target or rd hit & I switch on a hit.
            Won't go here if I hit the berm; ftg ts don't go behind the
            berm, and if true tgt s do, the rd won't hit.
            nrtg1 = nrtg1-1
            nrtg1 = 0
            call diseng(t,1,it,.true.,.true.)
          ENDIF
        ENDIF
      ENDIF
      if (trace) print *, '<impact'
END
6.2 Mayhit: Find Whether the Round Hits. **Mayhit** finds whether the round hits, handles results of a hit or miss and tallies results. First, it finds the position of the round with respect to the aim point. If it is above the turret ring, **mayhit** finds if it hit the turret. If below and the target is fully exposed, **mayhit** finds if it hit the hull. If the round hits the target, **mayhit** finds if the round was a dud or not. If a hit is not a dud, the routine schedules **damage**. It also tallies the round results as a) sensed miss, b) lost miss c) hit, or d) hit but dud.

Does the round hit? The routine tentatively sets hit=.false., finds the relative positions of firer and target, and from this information finds the crossing angle. The crossing angle is the angle between the target velocity and the target position (relative to the firer). It then calls **accerr** to find the error of the round relative to the aim point.

The next step is to find the position of the round relative to the center of the turret ring. If the target is fully exposed, the aim point is .3 meters below the center of the turret ring. If the target is hull defilade, the aim point is .5 meters times the height of the turret above the bottom of the visible turret. If the height of the incoming round is greater than 0, then it may have hit the turret; otherwise it may have hit the hull (if the target was fully exposed.) The routine then calls **izhit** to find if the round passes through the hull or turret box.

Treatng a hit. When a hit occurs, the code tallies a hit for the appropriate side and tallies a hit on the target. If the target has received enough hits to satisfy the target switching policy of the firer's side, the code schedules a 'late kill.' Missed(l, it) is set to false. This information will be used to select or reject this target later. Prevrd(l) is set to 2. This information will be used by the accuracy routines to inhibit redrawing variable biases because the next shot at the target will be a subsequent round.

Duds. Next, the code finds if the round was a dud. If so, the dud is tallyed for the side. Otherwise, the code schedules **damage** which will determine what if any damage results.

Treatng a miss. The routine tallies a miss for the side and for the firer. Then it determines whether the firer sensed the miss or not.

Finally, whether the round hit or not, the code checks to see if the target or firer was moving at fire time or at impact time. In either case, prevrd is set to 1, to force the next round to be treated like a first round on the target by the accuracy routines. (The drawing of variable biases is inhibited only for subsequent rounds from a stationary firer on a stationary target.)

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>l</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>n</td>
<td>Side of firer (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>k</td>
<td>Kind of round.</td>
</tr>
<tr>
<td>v1</td>
<td>Velocity of target when round was fired (m/s).</td>
</tr>
<tr>
<td>v2</td>
<td>Velocity of firer when round was fired (m/s).</td>
</tr>
<tr>
<td>expose</td>
<td>Exposure of target (FD, HD, or FE).</td>
</tr>
<tr>
<td>hit</td>
<td>True iff round hits target.</td>
</tr>
<tr>
<td>rgx</td>
<td>Range to target (m).</td>
</tr>
<tr>
<td>crs ang</td>
<td>Crossing angle (rad).</td>
</tr>
<tr>
<td>vx0(it)</td>
<td>Last computed speed of target (m/s).</td>
</tr>
</tbody>
</table>
SUBROUTINE MAYHIT (t,1,1,1,1,k,v1,v2,expose,hit,rgx)

C 0 May hit: Find what the round does.
include 'common.h'
common /cimpact/ x,y,theta, disp
logical missed
common /MayPri/ missed(NN,NN)
save /cimpact/, /MayPri/
integer expose
logical hit, izhit
1 format(98,2,i,x,s,i3,' Hits berm')
if (trace) print *, ' > mayhit'
kshot(n,6) = kshot(n,6)+1
C Find whether a hit occurs.
hit = .false.
C Find position of round w.r.t. the aim point.
rgx = rgf(t,1,1,1)
crs ang = 0.0
if (vx0(it).ne.0.0) crs ang = anglef(s,vt)
call accerr(n,rgx,1,crsang,1,v1,v2,x,y,disp)
C Find position of round w.r.t. center of turret ring.
if (expose.eq.FE) y=y-y-O.3
if (exposeeq.HD) y-y+0.5*sysdim(n,TURRET)
C Find whether round hits.
IF (y.gt.0.0) THEN
hit = izhit(TURRET,1,n,x,y,theta)
ELSE
IF (expose.eq.FE) THEN
hit = izhit(HULL5,n,x,y,theta)
ELSE
if (histry) print 1, t, color(n), 1
ENDIF
ENDIF
C IF (hit) THEN
C Treat hit.
kshot(n,8) = kshot(n,8)+1
if (life(it).eq.MFKILL) nhot(it)=nhot(it)+1
if (nhot(it).gt.nbump(n)) call skedul(t,1,1,1,'ikill ',1,3,NULL)
missed(l,1) = .true.
prevrd(l) = 2
IF (reliab(n).ge. ranu(0.0)) THEN
call skedul (t,1,1,1,.,'damage ',it)
ELSE
ENDIF
ENDIF
C ELSE
C Round is a dud.
kshot(n,9) = kshot(n,9)+1
ENDIF
C ELSE
C Treat miss.
kshot(n,7) = kshot(n,7)+1
missed(l,1) = .true.
nrg = max(0,1,int(0.5+rgx/rwinr))
IF (psense(n,nrg)-gt.ranu(0.0)) THEN
prevrd(l) = 4
if (histry) print *, ' Miss is sensed.'
ELSE
prevrd(l) = 3
if (histry) print *, ' Miss is not sensed.'
ENDIF
ENDIF
C Careful. If either moving, make sure nx rd is treated as 1st
C round if SS case occurs.
if (vx0(l).ne.0.0 .or. vx0(it).ne.0.0) prevrd(l)=1
8.3 Ishit: Find if the Target Is Hit. Ishit discards rounds that are too high or too low. For other rounds, it finds the orientation of the hull or turret and its horizontal boundaries. If the round is within the horizontal boundaries, ishit reports a hit.

The array sysdim contains the distance from the center of the turret ring to the ith edge of the target. For example, sysdim(1,5) is the distance from the center of the turret ring to the bottom of the hull. These dimensions help determine if the round was too high or too low.

<table>
<thead>
<tr>
<th>Distance from Center of Turret Ring to</th>
<th>turret</th>
<th>hull</th>
</tr>
</thead>
<tbody>
<tr>
<td>i</td>
<td>sysdim(i)</td>
<td>i</td>
</tr>
<tr>
<td>1</td>
<td>top</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>side</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>front</td>
<td>7</td>
</tr>
<tr>
<td>4</td>
<td>rear</td>
<td>8</td>
</tr>
</tbody>
</table>

The code tentatively sets ishit=.false. It then checks to see if the vertical error of the round is greater than the vertical dimensions of the target box. If so, it is a miss and the code reports the miss if the print flag is set. If not, the code checks to see if the round is within the horizontal dimensions of the box. To do this, it uses the ranang routine which draws a random angle from the cardioid or frontal distribution. This angle will be used as the orientation of the target relative to the incoming round. Next the code calls the bounds routine to find the left and right edges of the target box. If the horizontal error is within the horizontal boundaries of the box a hit has occurred.

\[ x_l < x < x_r \]

**CODE**

```fortran
c V7.2

LOGICAL FUNCTION IZHIT (nbox, ndim, n, x, y, theta)

include 'common.h'

LOGICAL izhit, printhit, isgo, trace
INTEGER nbox, ndim, n
REAL x, y, theta
REAL xleft, xright, ylimit

izhit = .false.

ylimit = sysdim(n, ndim)

IF (ylimit.le.abs(y)) THEN
  IF (isgo) THEN
    IF (ynbox(6).gt.0) THEN
      IF (izhit) THEN
        format ('IZHIT: the round is high. ylimit =', 3F17.3)
      ELSE
        format ('IZHIT: the round is low. ylimit =', 3F17.3)
      ENDIF
    ENDIF
  ELSE
    theta = rndang(iangd)
    call bounds(n, nbox, theta, xleft, xright)
    izhit = xleft.lt.x and. x.lt.xright
  ENDIF
ELSE
  IF (trace) THEN
    print *, '<', izhit
  ENDIF
  izhit = true
  if (trace) THEN
    format ('<', izhit)
  ENDIF
  izhit = false.
ENDIF
END
```

**COMMENT**

- `izhit`: True iff the round hit the target. Logical
- `ylimit`: Height of hull or turret (m).
- `iangd`: 1) if cardioid distribution, 2) if frontal distribution
- `xleft`: X coordinate of left side of tank (m).
- `xright`: X coordinate of right side of tank (m).
- `x`, `xleft`, `xright`: X coordinate of bullet on target (m).
- `y`, `ylimit`: Y coordinate of bullet on target (m).
- `theta`: Angle at which round struck target (rad).
- `nbox`: 1=turret, 2=hull
- `ndim`: Index of box height (1 for turret, 5 for hull).
- `n`: Side of target (1=Blue, 2=Red)
- `x`, `y`: X, Y coordinate of bullet on target (m).
- `izhit`: Print 3, 4, 5, 6, 7, 8, 9, 10
- `trace`: Print 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- `isgo`: Print 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
- `ynbox(6)`: Print 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
6.4 Bounds: Find the Horizontal Bounds of the Hull or Turret.

**Bounds** finds the distances from the center of the turret ring to the left and right edges of the target box.

Theta is the angle from the nose of the box to the bullet hitting the turret center. Calculations with theta are done to assure that the angle is between 0 and 360 degrees.

The array sysdim contains the distance from the center of the turret ring to the ith position. R1 is the left boundary and r2 is the right boundary. C is the portion of r1 and r2 due to the width of the target. S2 and s3 are the portions due to the 'depth' of the target.

The figure below shows the 4 corners of the turret and the angle from the nose to the bullet. The table below lists the left and right horizontal boundaries of the box when the bullet enters it at a certain angle. The variables c, s2, and s3 are used to find these boundaries.

<table>
<thead>
<tr>
<th>Boundary</th>
<th>Quadrants</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0&lt;θ&lt;90</td>
</tr>
<tr>
<td>left</td>
<td>c</td>
</tr>
<tr>
<td>right</td>
<td>b</td>
</tr>
</tbody>
</table>

The figure below shows how to find the horizontal boundaries of a hull or turret using c,s2,and s3. This case is where the bullet enters the box at an angle between 0 and 90 degrees. The horizontal boundaries of the other 3 cases can be found in a similar manner.

d2= distance from center of turret ring to turret side
d3 = distance from center of turret ring to turret rear
\[ d_4 = \text{distance from center of turret ring to turret front} \]
\[ c = d_2 \cos \theta \]
\[ s_2 = d_3 \sin \theta \]
\[ s_3 = d_4 \sin \theta \]

SUBROUTINE BOUNDS (narmy, box, angll, r1, r2)

Bounds: find the horizontal bounds of hull or turret.

Definitions:
\( \text{angll} \) - angle off the nose of the box (rad).
\( \text{box} = 1 \) means turret box, 2 means hull box.
\( \text{narmy} = 1 \) means blue firers, 2 means red firers.
\( c, s_2, s_3 \) - temporary variables.
\( r_1, r_2 \) - left and right boundaries of boxes (m).

include 'common-h'
integer box

if (trace) print *,'> bounds'
initialize
\[ \text{temp} = \frac{(\text{angll} + \text{twopi})}{\text{twopi}} \]
\[ \text{theta} = \text{amod} \left( \text{temp} - \text{aint} \left( \text{temp} \right) \right) \cdot \text{twopi} \]
\[ c = \text{sysdim}(\text{narmy}, 4^* (\text{box} - 1) + 2) \cdot \cos(\text{theta}) \]
\[ s_2 = \text{sysdim}(\text{narmy}, 4^* (\text{box} - 1) + 3) \cdot \sin(\text{theta}) \]
\[ s_3 = \text{sysdim}(\text{narmy}, 4^* (\text{box} - 1) + 4) \cdot \sin(\text{theta}) \]

IF (theta.le.0.25*twopi) THEN
E N DIF
else if (trace) print *,'< bounds'
END
7. ACCURACY ROUTINES

The accuracy routines read and interpolate in accuracy tables. These tables contain data for stationary firer vs stationary target, stationary firer vs moving target, and moving firer vs stationary target.

The diagram below shows the relationship between the accuracy routines. The dashed line between boxes shows which routines share data via common. The impact routine may hit calls accerr.
7.1 Rderor: Read Accuracy Data for One Side. This routine reads accuracy data for stationary firers versus stationary targets, stationary firers versus moving targets, and moving firers versus stationary targets. If desired, this data is printed to standard output with appropriate labels. The routine then prints the name of the accuracy data file used.

Rderor reads in data for 1) stationary firer vs stationary target, 2) stationary firer vs moving target, and 3) moving firer vs stationary target. For each of these sets, it reads in one or two header lines and then the data proper. If iecho is set to zero, only the name of the accuracy file is echoed. If it is set to one, the headers for each set of data is echoed. And if it is set to two or greater, all the data is echoed.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbname</td>
<td>Name of data file</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer</td>
</tr>
<tr>
<td>iecho</td>
<td>Echo control</td>
</tr>
<tr>
<td>nrows</td>
<td># of rows to read</td>
</tr>
<tr>
<td>ncols</td>
<td># of columns of data</td>
</tr>
<tr>
<td>descr</td>
<td>One line description of table</td>
</tr>
<tr>
<td>nss(m)</td>
<td># of columns of stationary-stationary data for mth side</td>
</tr>
<tr>
<td>q</td>
<td>Description of row</td>
</tr>
<tr>
<td>sstbl</td>
<td>Stationary-stationary table (mils)</td>
</tr>
<tr>
<td>nsm(m)</td>
<td># of columns of stationary firer - moving target data</td>
</tr>
<tr>
<td>smtbl</td>
<td>Stationary-moving data table (mils)</td>
</tr>
<tr>
<td>nms(m)</td>
<td># columns in moving firer - stationary target table</td>
</tr>
<tr>
<td>kindms(m)</td>
<td>3 implies MS data is a function of firer speed.</td>
</tr>
<tr>
<td></td>
<td>5 implies MS data is a function of target range.</td>
</tr>
<tr>
<td>mstbl</td>
<td>Moving-stationary data table (mils)</td>
</tr>
</tbody>
</table>

V7.4

SUBROUTINE RDEROR (dbname, m, iecho)

C Read error: read accuracy data for a side.
C dbname - name of file containing error (accuracy) data.
C m - 1 if for Blue, 2 if for Red.
C iecho - print input echo iff true.
include 'common.h'
c character*72 descr
character*32 dbname
character*8 q
real mstbl
common /comms/ nss(2), sstbl(10,7,2)
common /comms/ nsm(2), smtbl(10,17,2)
common /comms/ kindms(2), nms(2), mstbl(10,5,2)
save /comms/ /comms/
1 format (2I3, A72)
2 format (1A8, 10F8.0)
3 format (' ', 1A8, 10F8.2)
c if (trace) print *, '<rderor'
open (4, file=dbname, status='old')
rewind 4
c Read stationary-stationary errors
read (4,1) nrows, ncols, descr
if (iecho.ge.1) print 1, nrows, ncols, descr
nss(m)=ncols
DO 10 nrows=1, ncols
read (4,2)q, (sstbl(ncol, nrow, m), ncol=1, ncols)
inffic(echo.ge.2) print 3, q, sstbl(ncol, nrow, m),
ncol=1, ncols)
1 continued
10 continue
C Read stationary-moving errors
read (4,1) nrows, ncols, descr
if (iecho.ge.1) print 1, nrows, ncols, descr
nsm(m)=ncols
read (4,1)
DO 20 nrow=1, nrows
read (4,2)q, smtbl(ncol, nrow, m), ncol=1, ncols)
inffic(echo.ge.2) print 3, q, smtbl(ncol, nrow, m),
ncol=1, ncols)
20 continued

END
7.2 Accerr: Find the Linear Error for a Single Round. **Accerr** finds which table of accuracy data is appropriate, calls the associated routine to generate angular errors, and converts them to linear errors. If the round fired is a ballistic round, **accerr** checks the motion of the firer and target when the round was fired and chooses one of three tables. If the round is guided, **accerr** simply uses a 'stationary-stationary' table.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td></td>
<td>Firer's side</td>
</tr>
<tr>
<td>r</td>
<td></td>
<td>Range to target (m)</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>Firer ID</td>
</tr>
<tr>
<td>theta</td>
<td></td>
<td>Crossing angle (rad)</td>
</tr>
<tr>
<td>vtgt</td>
<td></td>
<td>Speed of target (m/s)</td>
</tr>
<tr>
<td>vfirer</td>
<td></td>
<td>Speed of firer (m/s)</td>
</tr>
<tr>
<td>x</td>
<td>$x = .001 z \theta$</td>
<td>Linear horizontal error (m)</td>
</tr>
<tr>
<td>y</td>
<td>$y = .001 y_{tgt}$</td>
<td>Linear vertical error (m)</td>
</tr>
<tr>
<td>rex</td>
<td>$r_x = .001 r_y$</td>
<td>Linear horizontal dispersion (m)</td>
</tr>
<tr>
<td>rey</td>
<td>$r_y = .001 r_x$</td>
<td>Linear vertical dispersion (m)</td>
</tr>
<tr>
<td>disp</td>
<td>$d = 3.28 \sqrt{0.5(r_x^2 + r_y^2)}$</td>
<td>RMS dispersion (ft)</td>
</tr>
</tbody>
</table>

```fortran
7.4 SUBROUTINE ACCERR(m, r, I, theta, vtgt, vfirer, x, y, disp)
  c V7.4
  Acc err: find the linear error for a single round.
  include 'common.h'
  logical fmove, tstat, tmove, tstat, burst
  c
  IF (trace) print *, 'acc err'
  fmove = vfirer.gt.0.0
  fstat = .not.fmove
  tmove = vtgt.gt.0.0
  tstat = .not.tmove
  burst = nrpb(m).gt.1
  IF (burst) THEN
    print*, 'ACC ERR: Burst fire not modelled.'
    STOP
  ELSEIF (fmove.and.tmove) THEN
    print*, 'ACC ERR: No moving-moving data.'
    STOP
  ELSEIF (kindrd(m).le.2) THEN
    Either KE or HEAT gun system. (kindrd=1,2)
    IF (fstat.and.tstat) call accsn (m,r,theta,x,y,rex,rey)
    IF (fmove.and.tstat) call accms (m,r,vfirer,x,y,rex,rey)
    ELSE
      Direct fire or top attack missile. (kindrd=4,5)
      call accas (m,r,theta,x,y)
  ENDIF
  c Convert from angular to linear errors.
  x = x*r*0.001
  y = y*r*0.001
  rex = rex*r*0.001
  rey = rey*r*0.001
  disp = 3.28*sqrt(0.5*(rex**2 + rey**2))
  IF (trace) print *, '<acc err'
END
```

73
7.3 Access: Find Angular Accuracy for Stationary Firer vs Stationary Target. Access finds the angular errors when a stationary firer shoots at a stationary target.

Table 6 shows the format of the data as AMSAA generates it.

Table 6. First Round Accuracy
Stationary Firer vs Stationary Target

<table>
<thead>
<tr>
<th>Range (Meters)</th>
<th>Fixed Biases (mils)</th>
<th>Random Error</th>
<th>Variable Error</th>
<th>Total Error</th>
<th>Horizontal (mils)</th>
<th>Vertical (mils)</th>
<th>Random Variable</th>
<th>Total Variable</th>
<th>Probability of Hit</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td><strong>Horizontal</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>250</td>
<td>1.072</td>
<td>1.3702</td>
<td>.5728</td>
<td>1.4272</td>
<td>1.3702</td>
<td>.6284</td>
<td>1.4504</td>
<td>9627</td>
<td></td>
</tr>
<tr>
<td>500</td>
<td>.357</td>
<td>.7290</td>
<td>.8940</td>
<td>1.0043</td>
<td>.7250</td>
<td>.8572</td>
<td>1.6233</td>
<td>9343</td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td>.000</td>
<td>.4652</td>
<td>1.1845</td>
<td>1.2292</td>
<td>.4652</td>
<td>1.1845</td>
<td>1.9015</td>
<td>9016</td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td>-.119</td>
<td>.3929</td>
<td>1.7880</td>
<td>1.8867</td>
<td>.3929</td>
<td>3.4496</td>
<td>5.4719</td>
<td>9580</td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td>-.178</td>
<td>.3621</td>
<td>2.6669</td>
<td>2.8314</td>
<td>.3621</td>
<td>6.2610</td>
<td>6.2715</td>
<td>9126</td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td>-.214</td>
<td>.3459</td>
<td>3.8062</td>
<td>3.8757</td>
<td>.3459</td>
<td>11.1232</td>
<td>11.1286</td>
<td>9032</td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td>-.238</td>
<td>.3362</td>
<td>5.4729</td>
<td>5.6122</td>
<td>.3362</td>
<td>10.1972</td>
<td>19.2061</td>
<td>9009</td>
<td></td>
</tr>
</tbody>
</table>

The user must re-arrange the data in the above table into the format shown below.

Figure 5 shows the relationship of the variables. The aim point is at the origin of the coordinate system. Each of the solid arrows represents the fixed parameters of a distribution. The dashed arrows represent random draws from those distributions. The solid arrow from the origin to the center of the large ellipse illustrates the fixed bias. Older fire controls have fixed biases due to factors like parallax.
The large ellipse illustrates the one sigma limits on the variable bias. The variable bias changes from occasion to occasion due to factors like vehicle cant or gunner lay error. The dashed arrows from the center of the large ellipse represent random draws from the variable bias distribution.

The smaller ellipses represent the one sigma limits on the dispersion of the round. The dashed arrows from the centers of the smaller ellipses represent draws from the dispersion distributions.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Side of firer</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Range to target (m)</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>ID of firer.</td>
<td></td>
</tr>
<tr>
<td>sstbl(n,k,m)</td>
<td>S_{n,k,m}</td>
<td>Errors for nth range, kth type, and mth side (mils).</td>
</tr>
<tr>
<td>k</td>
<td>k</td>
<td>Interpolate for range in columns k, k+1</td>
</tr>
<tr>
<td>frac</td>
<td>f = (r-r_k)/(r_{k+1}-r_k)</td>
<td>Fraction of distance in interval</td>
</tr>
<tr>
<td>mux, may</td>
<td>\mu_x, \mu_y</td>
<td>Horizontal, vertical fixed bias (mils)</td>
</tr>
<tr>
<td>sgx, sgy</td>
<td>\sigma_x, \sigma_y</td>
<td>Horizontal, vertical dispersion (mils)</td>
</tr>
<tr>
<td>sx, sy</td>
<td>\delta_x, \delta_y</td>
<td>Random draw for horizontal, vertical random error (mils)</td>
</tr>
<tr>
<td>nux, nuy</td>
<td>\nu_x, \nu_y</td>
<td>Horizontal, vertical variable bias (mils)</td>
</tr>
<tr>
<td>vx, vy</td>
<td>\nu_x = \nu_y[N]</td>
<td>Random draw for variable bias (mils)</td>
</tr>
<tr>
<td>tx, ty</td>
<td>t_x = \sqrt{\sigma_x^2+\nu_x^2}</td>
<td>Total dispersion (mils)</td>
</tr>
<tr>
<td>ax, ay</td>
<td>\sigma_z = \mu_x+\nu_x+\delta_x</td>
<td>Angular error (mils)</td>
</tr>
</tbody>
</table>

The array sstbl(10,7,2) stores the input data. The diagram below illustrates the first plane sstbl_{n,k,1} for a gun system. Since missile systems have no variable biases, the 6th and 7th lines become the 4th and 5th lines in the table for missiles.

<table>
<thead>
<tr>
<th>n</th>
<th>k=</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>r</td>
<td>s_{1,1,1}</td>
<td>s_{2,1,1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>\mu_x</td>
<td>\delta_{1,1,1}</td>
<td>\delta_{2,1,1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>\mu_y</td>
<td>\delta_{1,2,1}</td>
<td>\delta_{2,2,1}</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>\nu_x</td>
<td>\sigma_x</td>
<td>\sigma_y</td>
<td>\sigma_z</td>
<td>\nu_z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>\nu_y</td>
<td>\sigma_z</td>
<td>\sigma_y</td>
<td>\sigma_z</td>
<td>\nu_z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>\sigma_x</td>
<td>\sigma_z</td>
<td>\sigma_y</td>
<td>\sigma_z</td>
<td>\nu_z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>\sigma_y</td>
<td>\sigma_z</td>
<td>\sigma_y</td>
<td>\sigma_z</td>
<td>\nu_z</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

The following pseudo code shows how the various errors are aggregated to produce the random angular error and the total error. It shows the calculation of the x values; y values are found similarly. Primed values are values that will not change from shot to shot and are saved for subsequent rounds fired at the same target until it is disengaged (either temporarily or permanently.)
Before the first executable statement is a statement function which does a linear interpolation between the \(k\)th and \(k+1\)st columns of row \(n\) in plane \(m\) of the data table.

Note that for missiles or guns, the bias is stored in the 2nd row of the table. For guns, the variable bias is stored in the 4th row and the random error is stored in the 6th row. For missiles, there is no variable bias, so the random error is stored in the 4th row.
7.4 Accsm: Find Angular Accuracy for Stationary Firer vs Moving Target. Accsm interpolates in the stationary-firer moving-target accuracy to select the errors for each shot. For each side, these errors are a function of the range and crossing angle.

Table 7 shows the format of the data AMSAA generates.

<table>
<thead>
<tr>
<th>Target Speed (KPH)</th>
<th>Stationary Firer vs Moving Target</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Evasive Factor = .25</td>
</tr>
<tr>
<td></td>
<td>Target Crossing Direction = counterclockwise</td>
</tr>
<tr>
<td></td>
<td>Target Crossing Angle = 0 degrees</td>
</tr>
<tr>
<td></td>
<td>Bias and Dispersion in mils</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Target Speed (KPH)</th>
<th>ACCURACY DATA AS A FUNCTION OF RANGE (METERS)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>250</td>
</tr>
<tr>
<td>2</td>
<td>H BIAS</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
</tr>
<tr>
<td></td>
<td>P(H)</td>
</tr>
<tr>
<td>10</td>
<td>H BIAS</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
</tr>
<tr>
<td></td>
<td>P(H)</td>
</tr>
<tr>
<td>20</td>
<td>H BIAS</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
</tr>
<tr>
<td></td>
<td>P(H)</td>
</tr>
<tr>
<td>30</td>
<td>H BIAS</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
</tr>
<tr>
<td></td>
<td>P(H)</td>
</tr>
<tr>
<td>40</td>
<td>H BIAS</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
</tr>
<tr>
<td></td>
<td>P(H)</td>
</tr>
</tbody>
</table>

H - HORIZONTAL
V - VERTICAL
P(H) - Probability of hit against a 2.3m x 2.3m vertical moving target

Tables similar to the one above are produced for crossing angles of 0, 30, 60, and 90 degrees. The user must extract data from these tables for the combat cruise speed he wants. He then puts it in the accuracy data file. Table 8 shows the format of the stationary-moving data included in the accuracy data file.

First, accsm finds the column k in the data so that it can linearly interpolate between columns k and k+1. Then it finds frac, the fraction of the distance into the range interval. This value will be used to linearly interpolate. Next it finds the sub-table for the appropriate crossing angle.

The crossing angle $-\pi < \theta < \pi$ is an input to accsm. Since data is only available for crossing angles between 0 and 90 degrees and there is a certain amount of symmetry, accsm converts $\theta$ into a $\theta'$ as follows:

$$\theta' = \pi/2 - |\pi/2 - |\theta||$$

The result is $-\pi/2 \leq \theta' < \pi/2$. Then accsm finds the appropriate rows i, i+1 bounding $\theta'$. After interpolating for range, accsm interpolates for crossing angle.
Table 8. Stationary Firer vs Moving Target

<table>
<thead>
<tr>
<th>(KPH)</th>
<th>0 deg</th>
<th>250</th>
<th>500</th>
<th>1000</th>
<th>1500</th>
<th>2000</th>
<th>2500</th>
<th>3000</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>V BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
<td>.6472</td>
<td>.7586</td>
<td>1.1739</td>
<td>1.8114</td>
<td>2.6840</td>
<td>3.8721</td>
<td>5.4983</td>
</tr>
<tr>
<td>30 deg</td>
<td>H BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
<td>0.6472</td>
<td>0.7586</td>
<td>1.1739</td>
<td>1.8114</td>
<td>2.6840</td>
<td>3.8721</td>
<td>5.4983</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
<td>0.7693</td>
<td>0.9702</td>
<td>1.9082</td>
<td>3.4875</td>
<td>6.2855</td>
<td>11.1399</td>
<td>19.2091</td>
</tr>
<tr>
<td>60 deg</td>
<td>H BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
<td>0.6472</td>
<td>0.7586</td>
<td>1.1739</td>
<td>1.8114</td>
<td>2.6840</td>
<td>3.8721</td>
<td>5.4983</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
<td>0.7693</td>
<td>0.9702</td>
<td>1.9082</td>
<td>3.4875</td>
<td>6.2855</td>
<td>11.1399</td>
<td>19.2091</td>
</tr>
<tr>
<td>90 deg</td>
<td>H BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>V BIAS</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
<td>0.000</td>
</tr>
<tr>
<td></td>
<td>H DISP</td>
<td>0.6472</td>
<td>0.7586</td>
<td>1.1739</td>
<td>1.8114</td>
<td>2.6840</td>
<td>3.8721</td>
<td>5.4983</td>
</tr>
<tr>
<td></td>
<td>V DISP</td>
<td>0.7693</td>
<td>0.9702</td>
<td>1.9082</td>
<td>3.4875</td>
<td>6.2855</td>
<td>11.1399</td>
<td>19.2091</td>
</tr>
</tbody>
</table>

**CODE** | **MATH** | **COMMENT**
--- | --- | ---
**m** | Side of firer. Identifies plane of data array to use. |
**r** | Range from firer to target (m). |
**theta** | Crossing angle (rad). |
**k** | k, k+1 are columns between which to interpolate. |
**frac** | Fraction of distance into range interval. |
**i** | Index selecting rows in table. |
**mu**, **mu** | Horizontal and vertical biases (mils). |
**ax**, **ay** | Horizontal and vertical angular errors (mils). |
**tx**, **ty** | Horizontal and vertical total errors (mils). |
7.5 Accms: Find Angular Accuracy for Moving Firer vs Stationary Target. Acc ms interpolates in the MS table to find the additional errors due to motion of the firer. It combines these errors with those for a stationary firer against a stationary target to generate the total errors for the moving firer firing at a stationary target.

Input/Output. Table 9 illustrates the data produced by AMSAA for moving firer add-on errors. It contains horizontal and vertical data for six types of terrain:

I - Level farmland meadows
II - Field with overpass road
III - Frozen plowed fields with crossings
IV - Rolling meadows
V - Stony farmland with crossings
VI - Heavily used tank road

Usually, Tank Wars runs use type 4 terrain. The user converts velocity to meters per second and reformats the data as shown in figure 7.5b.

Table 9. Add-on Dispersions for Moving Firers

<table>
<thead>
<tr>
<th>Velocity [KPH]</th>
<th>TT I H</th>
<th>TT I V</th>
<th>TT II H</th>
<th>TT II V</th>
<th>TT III H</th>
<th>TT III V</th>
<th>TT IV H</th>
<th>TT IV V</th>
<th>TT V H</th>
<th>TT V V</th>
<th>TT VI H</th>
<th>TT VI V</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>.40</td>
<td>.40</td>
<td>.49</td>
<td>.51</td>
<td>.48</td>
<td>.60</td>
<td>.40</td>
<td>.40</td>
<td>.85</td>
<td>.10</td>
<td>.40</td>
<td>.49</td>
</tr>
<tr>
<td>8</td>
<td>.40</td>
<td>.40</td>
<td>.49</td>
<td>.51</td>
<td>.48</td>
<td>.60</td>
<td>.40</td>
<td>.40</td>
<td>.84</td>
<td>.10</td>
<td>.83</td>
<td>1.04</td>
</tr>
<tr>
<td>12</td>
<td>.40</td>
<td>.49</td>
<td>.78</td>
<td>.97</td>
<td>-.</td>
<td>.40</td>
<td>.40</td>
<td>.83</td>
<td>1.04</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>.45</td>
<td>.58</td>
<td>1.15</td>
<td>1.44</td>
<td>-.</td>
<td>.43</td>
<td>.66</td>
<td>.175</td>
<td>2.20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>.54</td>
<td>.67</td>
<td>4.30</td>
<td>5.50</td>
<td>-.</td>
<td>.91</td>
<td>1.14</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
</tr>
<tr>
<td>24</td>
<td>.76</td>
<td>.95</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>1.45</td>
<td>1.80</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
</tr>
<tr>
<td>32</td>
<td>1.70</td>
<td>2.10</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>10.9</td>
<td>13.7</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
</tr>
<tr>
<td>40</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
<td>-.</td>
</tr>
</tbody>
</table>

Table 10. Moving Stationary Input Data (as a function of velocity)

<table>
<thead>
<tr>
<th>vel(m/s)</th>
<th>1.11</th>
<th>2.22</th>
<th>3.33</th>
<th>4.44</th>
<th>5.55</th>
<th>6.66</th>
<th>7.77</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adx</td>
<td>.40</td>
<td>.40</td>
<td>.40</td>
<td>.43</td>
<td>.91</td>
<td>1.45</td>
<td>10.9</td>
</tr>
<tr>
<td>Ady</td>
<td>.40</td>
<td>.40</td>
<td>.40</td>
<td>.66</td>
<td>1.14</td>
<td>1.80</td>
<td>13.7</td>
</tr>
</tbody>
</table>

Table 11. Moving Stationary Input Data (as a function of range)

<table>
<thead>
<tr>
<th>rg(m)</th>
<th>500.</th>
<th>1000.</th>
<th>1500.</th>
<th>2000.</th>
<th>2500.</th>
<th>3000.</th>
<th>3500.</th>
<th>4000.</th>
</tr>
</thead>
<tbody>
<tr>
<td>FxBh</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
<td>.1000</td>
</tr>
<tr>
<td>v</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
<td>.0000</td>
</tr>
<tr>
<td>Te h</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
</tr>
<tr>
<td>v</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
<td>.8000</td>
</tr>
</tbody>
</table>

MS error as a function of velocity. In this case, the routine finds the errors for the SS case and the add-on errors for the MS case and root sum squares them. See section 7.1 for a discussion of the SS data. Acc ms performs linear interpolation to find the horizontal and vertical add-on dispersions for the appropriate firer speed. For example, at 5 m/s, the calculations are:

\[
\frac{5-4.44}{5.55-4.44} = 0.504
\]

\[
Adx = interp(.43,.91) = .43+.504(.91-.43) = .672
\]

This add-on error is combined with the SS error to give the dispersion (rex), and a randomly drawn angular error as shown below.
### Table

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>m</td>
<td>Side of firer (1=Blue, 2=Red).</td>
<td></td>
</tr>
<tr>
<td>r</td>
<td>Range from firer to target (m).</td>
<td></td>
</tr>
<tr>
<td>v</td>
<td>Speed of firer at fire time (m/s).</td>
<td></td>
</tr>
<tr>
<td>ax,ay</td>
<td>Angular error of round (mils).</td>
<td></td>
</tr>
<tr>
<td>rex,rey</td>
<td>Dispersion of round (mils).</td>
<td></td>
</tr>
<tr>
<td>kindms</td>
<td>MS data table (mils).</td>
<td></td>
</tr>
<tr>
<td>brgs</td>
<td>Number of lines of MS data.</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>3 if data is a function of velocity</td>
<td></td>
</tr>
<tr>
<td></td>
<td>5 if data is a function of range</td>
<td></td>
</tr>
<tr>
<td>frac</td>
<td>Blue ranges in data table (m).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Index such that ( r_k &lt; r &lt; r_{k+1} ).</td>
<td></td>
</tr>
<tr>
<td>mux</td>
<td>Fraction of distance into interval.</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Horizontal bias (mils).</td>
<td></td>
</tr>
<tr>
<td>rex</td>
<td>Random error horizontally (mils).</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Random draw from the standard normal distribution.</td>
<td></td>
</tr>
<tr>
<td>ax</td>
<td>Horizontal error of round (mils).</td>
<td></td>
</tr>
</tbody>
</table>

### MS error as a function of range

When the input data is in this format, there is no need to access the SS data. Again the routine performs linear interpolation. In this case, it directly finds \( \mu_x, \mu_y \) and \( s_x, s_y \) as well as the corresponding \( y \)-values.

```
SUBROUTINE ACC MS(m, r, v, ax, ay, rex, rey)
  c MS data is a function of range.
  v7.4
  c MS data is a function of velocity.
  IF (kindms(m) .eq. 3) THEN
    c Find SS errors.
    IF(m.eq.1) K = indexx(brgs, nss(m), r)
    IF(m.eq.2) K = indexx(rbrms(m), r)
    ELSEIF (kindms(m).eq.5) THEN
      IF (trace) print *, 'MS data is a function of range.
      c Find SS errors.
      IF(m.eq.1) K = indexx(bbrms, nss(m), r)
      IF(m.eq.2) K = indexx(rbrms(m), r)
      ENDIF
      IF (trace) print *, 'MS data is a function of velocity.
    ENDIF
    c Find SS errors.
    IF(m.eq.1) K = indexx(brgs, nss(m), r)
    IF(m.eq.2) K = indexx(rbrms(m), r)
    c Find SS add-ons.
    IF(m.eq.1) K = indexx(bbrms, nss(m), r)
    IF(m.eq.2) K = indexx(rbrms(m), r)
    ENDIF
    c Combine SS data and MS add-ons.
    c MS data is a function of range.
    c Find SS errors.
    IF(m.eq.1) K = indexx(brgs, nss(m), r)
    IF(m.eq.2) K = indexx(rbrms(m), r)
    c Find SS add-ons.
    IF(m.eq.1) K = indexx(bbrms, nss(m), r)
    IF(m.eq.2) K = indexx(rbrms(m), r)
    ENDIF
```
8. DAMAGE ROUTINES

The Damage Routines: **Damage**, **Damagf**, **Damagm**, and **LateKL**, find the degree of damage done to the target by incoming rounds and schedule appropriate events. The target's damage may be categorized into these types: M-Killed, F-Killed, M&F-Killed, I-Killed, and K-Killed. The effects on the target's foes include: discard target and switch to another. **Deaths** is one of the five damage routines. It tallies the tanks that are known to be dead on each side.

The diagram below shows the relationship of the routines called by **damage**. The routines in dashed boxes find the level of damage. They will be discussed in the following section. The routines shown in solid boxes deal with the results of damage at various levels. They are discussed in this section.
8.1 Damage: Simulate Damage to the Target. **Damage** finds the amount of damage caused by the current round, finds the cumulative damage, and schedules effects of any new damage. It finds the aspect angle and dispersion of the round, and then calls the appropriate kill routine to find the type of damage caused. It then scores new damage, if any. Finally, it schedules the effects of any new M, F, M&F, or K kills. These may include abort missile, cease movement, cease fire, and seek cover. The effects on the target's foes include: discard target and switch to another target.

First, **damage** finds a few useful variables: n, m, and k. Then it determines nang, the index of the 30 degree sector in which the incoming round strikes the target. These 30 degree bands are shown in figure 6 below. Then it finds disp, the one foot dispersion band associated with the dispersion of the round.

![Figure 6: Attack Angle Bands Around the Target](image)

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>ID of firer.</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>1 if firer is Blue, 2 if Red</td>
<td></td>
</tr>
<tr>
<td>m</td>
<td>1 if target is Blue, 2 if Red</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>1 for KE, 2 for HEAT, 4 for missile, 5 for STAFF</td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>$\theta$</td>
<td>Aspect angle of incoming round (rad) ($-\pi \leq \theta \leq \pi$)</td>
</tr>
<tr>
<td>theta2</td>
<td>$\theta_2 = 180 \theta/\pi$</td>
<td>Aspect angle of incoming round (deg) ($-180 \leq \theta_2 \leq 180$)</td>
</tr>
<tr>
<td>nangls(n)</td>
<td>$N_n$</td>
<td>Number of angles tabled for side n.</td>
</tr>
<tr>
<td>nang</td>
<td>$N = \lfloor \theta_2/30+1.5 \rfloor$</td>
<td>1-12 for 30 degree bands from head-on to tail-on</td>
</tr>
<tr>
<td>rex, rey</td>
<td>$x, y$</td>
<td>Horizontal, vertical dispersion of rounds (m)</td>
</tr>
<tr>
<td>disp</td>
<td>$3.28 \sqrt{\left(\frac{x^2 + y^2}{2}\right)}$</td>
<td>Root mean square of dispersions (ft)</td>
</tr>
<tr>
<td>ndisp</td>
<td>Dispersion bands (1 - 10 ft)</td>
<td></td>
</tr>
<tr>
<td>k</td>
<td>Kind of round (1=KE, 2=HEAT, 4=missile, 5=top attack)</td>
<td></td>
</tr>
<tr>
<td>life(it)</td>
<td>Status of target (1=ALIVE, 2=MKILL, 3=FKILL, 4=M&amp;FKILL, 5=IKILL, 6=KKILL)</td>
<td></td>
</tr>
<tr>
<td>injury</td>
<td>Damage caused by latest round.</td>
<td></td>
</tr>
<tr>
<td>injold</td>
<td>Damage incurred before latest round hit.</td>
<td></td>
</tr>
</tbody>
</table>

After damage finds the damage caused by the new round, it finds the cumulative damage as shown in table 12. Then damage calls damagm to schedule mobility damage effects, and damagf to schedule firepower damage effects. For catastrophic kills, it calls newtgt to switch the target's foes to new targets, and calls deaths to count up the score and see if we have a winner. For an M&F kill, it schedules an
inactivity kill a short time in the future, which simulates foes recognizing that an inactive target is no longer a threat.

Table 12. Cumulative Damage

<table>
<thead>
<tr>
<th>Previous Damage</th>
<th>Damage Caused by Current Round</th>
</tr>
</thead>
<tbody>
<tr>
<td>none</td>
<td>none M F M&amp;F K</td>
</tr>
<tr>
<td>M</td>
<td>M M&amp;F M&amp;F K</td>
</tr>
<tr>
<td>F</td>
<td>M&amp;F M&amp;F K</td>
</tr>
<tr>
<td>M&amp;F</td>
<td>M&amp;F M&amp;F K</td>
</tr>
<tr>
<td>I</td>
<td>- K K</td>
</tr>
<tr>
<td>K</td>
<td>- K K</td>
</tr>
</tbody>
</table>

- indicates no change

ENDIF

SUBROUTINE DAMAGE (t, i, it)
C 0 Damage: find if is hit & schedule effects.
include 'common.h'
character*2 kt(6)
common /cphk2/ nangls(2), pkh(2,7,12,2,4,11)
common /kimpct/ x,y,theta, disp
save /kimpct/, /cphk2/
data kt /'no','M-','F-','MF','I-','K-'/
format(8,2,1x,a4,i3,1x,'Hits',a4,i3,'(no damage).')
format(8,2,1x,a4,i3,1x,a4,i3,'-kills',a4,i3)
c if (trace) print '*','>damage'
array(l)
m = 3-n
k = kindrd(n)
theta2 = theta*DEG
if (nangls(n).eq.7) theta2=abs(theta2)
if (theta2.1,15.0) theta2=theta2+360.0
nang = int(theta2/30.0+1.5)
if (nang.lt.1) nang=1
if (nang.gt.12) nang=12
disp = 2.5*sqrt(6.0*(rex**2+rey**2))
nDisp = max(0,1,min(10,int(0.5+disp)))
c Find kill type (if any) and branch accordingly
if (k.ne.5) injury = kill(i, y, ndisp, nang)
if (k.ne.5) injury = kill(i, x, y, nang)
IF(history) THEN
if (injury.eq.1) print 1,t,color(n),color(m),it
if (injury.gt.1) print 2,t,color(n),1,kt(injury),color(m),it
ENDIF
injold = life(it)
IF (injury.eq.MFKILL and injold.it.MFKILL) THEN
else IF (injury.eq.FKILL) THEN
1 call skedul(t+tbump(n),it,'kill',NULL)
ENDIF
if (trace) print '*','<damage'
END

V7.5

c Treat first catastrophic kill.
life(it) = MKILL
call damagf(t, it)
call damagm(t, it)
call cancel(t, itkill ',NULL)
call newtgt(t, it)
call death(t)
ELSEIF (injury.ne.injold and injold.it.MFKILL) THEN
else Treat new damage (less than catastrophic).
IF (injury.eq.MFKILL) THEN
IF (injold.eq.MFKILL) THEN
IF (injold.eq.ALIVE) THEN
life(it) = MKILL
ELSE IF (injury.eq.FKILL) THEN
IF (injold.eq.MFKILL) THEN
call damagf(t, it)
ELSE IF (injury.eq.MFKILL) THEN
IF (injold.eq.FKILL) THEN
IF (injold.eq.MFKILL) THEN
ELSE IF (injury.eq.FKILL) THEN
IF (injold.eq.FKILL) THEN
IF (injold.eq.MFKILL) THEN
end
8.2 **DamagF:** Simulate Firepower Damage. Damagf cancels all firepower activities. It aborts any missiles fired by the target and cancels fire, reload, and select events scheduled for the target. If the target is still mobile, it seeks cover. If the target is only F-Killed and its top speed is greater than zero, damagf schedules an immediate accel event and a hide event a short time in the future.

The damaged tank can no longer fire. Its target is set to zero so the death of its target won’t trigger switching to a new target. Then any missiles it may have been guiding are aborted and the record of it being in the process of firing on any target is cleared (fot(it,j) = .false.). Finally, any firepower related events are canceled.

If the target is only F-killed and its top speed is greater than zero, damagf cancels all pending motion events and has it seek cover. To do this, damagf schedules an immediate accel event followed by a hide event. If it is in a defensive position, it hides in 5 seconds. Otherwise it hides after a delay determined by input values.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
</tr>
<tr>
<td>n</td>
<td>1 if target is Blue, 2 if Red.</td>
</tr>
<tr>
<td>nrtgt(it)</td>
<td>ID of target’s target.</td>
</tr>
<tr>
<td>nchan(it)</td>
<td>Number of guidance channels target has active.</td>
</tr>
<tr>
<td>kindrd(n)</td>
<td>Kind of round fired by firer.</td>
</tr>
<tr>
<td>fot(it,j)</td>
<td>Record that ‘it’ is firing at foe j.</td>
</tr>
<tr>
<td>life(it)</td>
<td>Status of ‘it’.</td>
</tr>
<tr>
<td>speed(n)</td>
<td>Combat cruise speed of target (m/s).</td>
</tr>
<tr>
<td>dt</td>
<td>Time required for side n to hide.</td>
</tr>
</tbody>
</table>

c V7.1

SUBROUTINE DAMAGF (t, it, n)

include 'common.h'

nrtgt(it) = 0
nchan(it) = 0

c Clear any guidance channels in use by target.
   if (kindrd(n).eq.4) call abort(t, it, 0)
   DO 40 j = 1, nblue+nred
   rot(it,j) = .false.
   40 CONTINUE
   call cancel(it,'fire ',NULL)
   call cancel(it,'reload',NULL)
   call cancel(it,'select',NULL)
   IF (life(it).eq.FKILL and. speed(n).gt.0.0) THEN
   call cancel (it,'slowup',NULL)
   call cancel (it,'halt ',NULL)
   call cancel (it,'accel ',NULL)
   call skedul (t, it, 'accel ',NULL)
   dt = thide(n)
   if (n.eq.BLUE .and. scene.eq.RATTAK) dt = 5.0
   if (n.eq.RED .and. scene.eq.BATTAK) dt = 5.0
   call skedul (t+dt, it, 'hide ',NULL)
   ENDIF
END
8.3 DamagM: Simulate Mobility Damage. **Damagm** discards all activities due to mobility kill including `maxvel`, `accel`, `hide`, and `vanish`. If the target M-Killed is not stopped or slowing, **damagm** schedules an immediate **slowup**.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
<td></td>
</tr>
<tr>
<td>vx0</td>
<td>v</td>
<td>Current speed of target</td>
</tr>
<tr>
<td>motion</td>
<td>m</td>
<td>1 = slowing, 2 = stopped, 3 = accelerating, 4 = cruising</td>
</tr>
<tr>
<td>SLOWNG</td>
<td>s</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>( v_{x_i} = 0 )</td>
<td>Implies it is stopped</td>
</tr>
<tr>
<td></td>
<td>( m_x = s )</td>
<td>Implies it is slowing down</td>
</tr>
</tbody>
</table>

```c
V7.4
SUBROUTINE DAMAGM (t, it)
include 'common.h'
logical sos
sos = stopped or slowing
if (trace) print '*', '>', 'damagm'
call cancel (it, 'maxvel', NULL)
call cancel (it, 'accel', NULL)
call cancel (it, 'hide', NULL)
call cancel (it, 'vanish', NULL)
sos = vx0(it).eq.0.0 .or. motion(it).eq.SLOWNG
if (.not.sos) call sched (t, it, 'slowup', NULL)
if (trace) print '*', '<', 'damagm'
END
```
8.4 Deaths: Tally Deaths. Deaths tallies the tanks that are known to be dead on each side. A tank is known to be dead if it is I-Killed, K-killed, or F-Killed and hidden. When the number of tanks on one side equals the number of dead tanks on that side, the engagement is considered finished and the opposing army wins the engagement if it can still shoot.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>t</code></td>
<td>Time (sec).</td>
</tr>
<tr>
<td><code>dead(n)</code></td>
<td>Count of dead on side n.</td>
</tr>
<tr>
<td><code>life(i)</code></td>
<td>Status of tank i.</td>
</tr>
<tr>
<td><code>knceal(i)</code></td>
<td>Concealment of tank i. 1=FD, 2=HD, 3=FE.</td>
</tr>
<tr>
<td><code>dead1</code></td>
<td>True IFF tank is known to be dead.</td>
</tr>
<tr>
<td><code>dead2</code></td>
<td>True IFF tanks is concealed and Fkilled (or worse).</td>
</tr>
</tbody>
</table>

If all the Blue tanks are dead or all the Red tanks are dead, the code schedules a finish event. For this purpose, a tank is considered dead if it is in view but I-killed or K-killed, or if it has been F-killed and has hidden.

c V7.2
SUBROUTINE DEATHS (t)
    do 1 Format (i3,' Blud dead,','i3',' Red dead.'
      1 Format(i3,'Blud dead,','i3',' Red dead.'
      if (trace) print *,>'deaths'
      dead(BLU) = 0
      dead(RED) = 0
      DO 20 i=1,nblu+nred
      dead1 = life(i).ge.IKILL
      dead2 = knceal(i).eq.FD .and. life(i).ge.FKILL
      if (dead1 .or. dead2) dead(army(i))=dead(army(i))+1
      20 CONTINUE
      if (histry) print 1,dead
      if (nblu.eq.dead(BLU) .or. nred.eq.dead(RED))
      call skedul(t+5,NULL,'finish',NULL)
      if (trace) print *,'<deaths'
    END
8.5 LateKI: Simulate Discard of Inactive M&F-Killed Target. After a target suffers an M&F-Kill and \( t \) seconds elapse or \( n \) hits are scored, the foes of the target will recognize that it is dead due to its inactivity. Those engaging the target will then seek a new target to engage. It will be marked as I-Killed (Inactivity killed) so that no foes will engage it again.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>tgt</td>
<td>ID of target (integer)</td>
</tr>
<tr>
<td>( j )</td>
<td>Unused dummy variable</td>
</tr>
<tr>
<td>firer</td>
<td>ID of target's first foe. (Any will do.)</td>
</tr>
<tr>
<td>life(tgt)</td>
<td>Status of target is bumped to I-killed.</td>
</tr>
</tbody>
</table>

V7.2
SUBROUTINE LATE KL (t, tgt, jj)
include 'common.h'
integer firer, tgt
format(8.2,1x,a4,i3,' I-killed. ')
if (trace) print *,'>latekl'
if (history) print 1, t, color(army(tgt)), tgt, firer - 1
if (tgt.le.nblu) firer=nblu+1
life(tgt) = IKILL
call cancel (tgt, 'ikill ',NULL)
call newtgt (t, firer,tgt)
call deaths(t)
if (trace) print *,'<latekl'
END
9. LETHALITY ROUTINES

This set of six routines handles the lethality data. The program calls the \texttt{rd pkh1}, \texttt{rd pkh2}, or \texttt{rd pkh5} routine once per side to read in the appropriate lethality data. \texttt{Rd pkh2} is used if lethality is known only for 7 ranges at the catastrophic kill level. \texttt{Rd pkh5} is used for top attack rounds like \texttt{STAFF}. If the user wants sample lethality data printed, \texttt{Rd pkh1} calls \texttt{mk tbl} to generate raw and processed data for head-on cases which will be echoed. The program calls the \texttt{kill} or \texttt{kill5} routine every time a round hits to find the amount of damage caused.

The diagram below shows the relation of the lethality routines. This section discusses those in solid boxes. The dashed lines indicate routines that share lethality data via common.

\begin{center}
\begin{tikzpicture}
  \node[draw] {input} child {node[draw] {rdpkh1} \node[draw] {mk tbl} \node[draw] {rdpkh2} \node[draw] {rdpkh5}} child {node[draw] {kill} \node[draw] {kill5} \node[draw] {damage}};
\end{tikzpicture}
\end{center}

\textbf{Data}. Data stored in the labeled common blocks /cpkh/, /cpkh2/, and /cpkh5/ are known only to the lethality routines. Normally, targets are considered symmetrical so data is read for aspect angles from zero to 180 degrees at 30 degree increments, for a total of 7 individual aspect angles. When a target is asymmetrical the program reads data for 12 aspect angles from -180 to 180 degrees.

The block /cpkh/ is known only to \texttt{rd pkh1} and \texttt{mk tbl} and the variables are:
- \texttt{table(4,12)} - \texttt{table(i,j)} is the lethality of \texttt{i}th type at \texttt{j}th aspect angle.
- \texttt{echo(2,7,7)} - \texttt{echo(i,j,k)} lethality data for \texttt{i}th side, \texttt{j}th range, and \texttt{k}th type.
- \texttt{jrg(2,7)} - \texttt{jrg(i,j)} \texttt{i}th side, \texttt{j}th range (meters).
- \texttt{jdisp(2,7)} - \texttt{jdisp(i,j)} \texttt{i}th side’s \texttt{j}th dispersion (ft).

The block /cpkh2/ is known only to \texttt{rd pkh1}, \texttt{rd pkh2}, and \texttt{kill}. The read routines store the processed lethality data in its variables. The arrays are:
- \texttt{nangls(2)} - \texttt{nangls(i)} contains the number of angles of data stored for the \texttt{i}th side.
pkh(2,7,12,2,4,11) - the processed lethality data.

The lethality data is a function of the:
- 2 sides,
- 7 ranges,
- 12 aspect angles,
- 2 exposures,
- 4 kill levels, and
- 11 dispersions

The block /cpkh5/ is known only to rd pkh5 and kill5. The arrays are:
- anglim(4) - Lower angular limit of fan sectors (deg).
- pkh5(2,7,4,4,12) - the processed lethality data.
- y1 - Distance from center of turret ring to lower edge of fan (m).
- y2 - Distance from center of turret ring to base of fan (m).
- y3 - Distance from center of turret ring to center of shot pattern (m).
- fans - Number of fans of data over target (integer).

Section 9.6 discusses the variables in block /cpkh5/ further.
9.1 RdPkh1: Read Standard Lethality Data. This routine reads standard IUA lethality data produced by VLD/BRL converts it, does some checking of the data and if desired, echos sample lethality data for a head on hit on the target. The IUA data consists of a header line plus 88 lines of data for HEAT rounds. Since the lethality of a KE round is range dependent, IUA data for KE consists of a header line plus 88 lines of data for each of seven ranges from zero meters to 3,000 meters in 500 meter increments.

The data is treated as probabilities of kill given a hit, where the first line in each set of four lines contains the probability of a mobility kill, the second has firepower kill, the third has mobility and firepower kill, and the fourth has catastrophic kill. Lethality data for a single dispersion is initially read into an array table. If the target is symmetrical, the DO 30 loop will read 7 values for aspect angles from 0 to 180 degrees. For asymmetrical targets, it will read 12 values. Table 13 illustrates this.

Table 13. Lethality Data Stored in Table(4,12)

<table>
<thead>
<tr>
<th>Kill Level</th>
<th>Aspect Angle</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
</tr>
<tr>
<td>M</td>
<td></td>
</tr>
<tr>
<td>F</td>
<td></td>
</tr>
<tr>
<td>M/F</td>
<td></td>
</tr>
<tr>
<td>K</td>
<td></td>
</tr>
</tbody>
</table>

The DO 40 loop checks to see if any values are less than zero or greater than 1 and tallies an error if so. If the routine finds bad data, it prints the line number. This occurs for up to 20 bad lines. The routine stops when 20 bad lines are found or before returning to the calling routine if any bad lines were found. The most likely cause of an error is a change in the format of the lethality data.

The DO 10 loop converts the data just read in into a usable form. It finds the probabilities of a mobility kill only, of a firepower kill only, and of a mobility and firepower kill only. These last three and the catastrophic kill are the values actually used by the program. There are several ways to calculate these values; here is one:

Let

\[ M = \text{probability of a mobility kill given a hit} \]
\[ F = \text{probability of a firepower kill given a hit} \]
\[ E = \text{probability of either mobility or firepower kill as above} \]
\[ K = \text{probability of a catastrophic kill given a hit} \]
\[ M' = \text{probability of a mobility kill only given a hit} \]
\[ F' = \text{probability of a firepower kill only given a hit} \]
\[ B = \text{probability of a both a mobility and firepower kill (but not K-kill)} \]

Then given the raw inputs M, F, E, K, the specific kill probabilities are:

\[ M' = E - F \]
\[ F' = E - M \]
\[ B = F - F' - K \]
\[ K = K \]

The Venn diagrams in Figure 7 illustrate how the raw kills are separated into specific kills.
If desired, the mk tbl routine is called to generate sample output, and in any case, the name of the lethality file is printed.

Arguments and Local Variables.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>dbname</td>
<td>Character name of lethality data file</td>
</tr>
<tr>
<td>narmy</td>
<td>1 for Blue, 2 for Red</td>
</tr>
<tr>
<td>iecho</td>
<td>Echos sample lethality data if &gt; 2</td>
</tr>
<tr>
<td>fandm</td>
<td>Probability of M and F kill but not K-kill.</td>
</tr>
<tr>
<td>header</td>
<td>Character header line for sample data</td>
</tr>
<tr>
<td>irg</td>
<td>1 to 7 for the appropriate range bands. Normally, they are 0, 500, 1000, ..., 3000 meters.</td>
</tr>
<tr>
<td>line</td>
<td>Line number (for reporting bad data lines)</td>
</tr>
<tr>
<td>mang</td>
<td>Number of column containing aspect angle data.</td>
</tr>
<tr>
<td>maxexp</td>
<td>2 (Hull defilade and fully exposed).</td>
</tr>
<tr>
<td>mdisp</td>
<td>1 to 10 for 1 to 10 foot dispersions. 11 for uniform dispersion.</td>
</tr>
<tr>
<td>nerr</td>
<td>Number of errors found</td>
</tr>
<tr>
<td>nhfde</td>
<td>1 for hull defilade, 2 for fully exposed data</td>
</tr>
<tr>
<td>nrrgs</td>
<td>KE lethality is a function of range, so read 7 sets of data for KE. Only one is required for HEAT rounds.</td>
</tr>
<tr>
<td>nrow</td>
<td>Row of sample data to fill in table.</td>
</tr>
<tr>
<td>onlym</td>
<td>Probability of M-kill only.</td>
</tr>
<tr>
<td>onlyf</td>
<td>Probability of F-kill only.</td>
</tr>
<tr>
<td>onlyf</td>
<td>Probability of F-kill only.</td>
</tr>
</tbody>
</table>

SUBROUTINE RDPKHI (dbname, narmy, iecho)

format(' Rdpkh: line1,i4,i4,i4,i5,i13,i8 give m,f,m&f=:', 1 305.3)

if (trace) write(*,*)'*** rdpkh'

include 'common.h'

if (iecho.gt.1) write(*,3) header

format(2i4,i5,i2,32,x,12f6.3) line

format(i7,7m.3,iA) nerrs

format(' 'Rg',7('H'),Sample Head-on',)

if (kind rd(narmy).gt.1) write(*,3) header

if (iecho.gt.1) write(*,3) header

format(2i4,i5,i2,32,x,12f6.3) line

format(i7,7m.3,iA) nerrs

format(' Rdpkh: line1,i4,i4,i4,i5,i13,i8 give m,f,m&f=:', 1 305.3)
DO 60 mdisp=1,11
   DO 30 i=1,4
      read (4,1) kgtl,kproj,krg,kexp,kdisp,ktype,
      *(table(i,j)=1,nangl)
   CONTINUE

   CHECK table for pkh's < 0 or > 1.
   DO 40 i=1,4
      line = line+1
      IF (table(i,j).lt.0.0 or. table(i,j).gt.1.) THEN
         write(*,8) line,j,table(i,j)
         nerrs = nerrs+1
         if (nerrs.gt.20) write(*,'RD PKH: too many errors')
         if (nerrs.gt.20) STOP
      ENDIF
   CONTINUE

   35 CONTINUE
   40 CONTINUE

   Convert to m, f, mf, k only and move to pkh array
   DO 10 mang=nangl,1,-1
      onlym = table(3,mang)-table(2,mang)
      onlyf = table(3,mang)-table(1,mang)
      fandm = table(2,nangl)-table(4,nangl)
      pkh(narmy,irg,mang,nndfe,1,mdisp) = onlym
      pkh(narmy,irg,mang,nndfe,2,mdisp) = onlyf
      pkh(narmy,irg,mang,nndfe,3,mdisp) = fandm
      pkh(narmy,irg,mang,nndfe,4,mdisp) = table(4,mang)
      IF (onlym.lt.0.0 or. onlyf.lt.0.0 or. fandm.gt.0.0) THEN
         write(*,9) line-3, line, mang, onlym, onlyf, fandm
         nerr = nerr+1
      ENDIF
   10 CONTINUE
   IF (iecho.ge.2) THEN
      nrow = 0
      if (irg.eq.1 and. mdisp.eq.1) nrow = 1
      if (irg-1.eq.mdisp and. nrrgs.eq.1) nrow = irg
      if (mdisp.eq.1 and. nrrgs.eq.1) call mk tbl(1)
      if (mdisp.le.6 .and. nrrgs.eq.1) nrow = mdisp+1
      if (nrow.gt.0) call mk tbl(1)
   ENDIF

   60 CONTINUE
   70 CONTINUE
   80 CONTINUE
   close(4)
   IF (iecho.ge.2) THEN
      'Hull Defilade'
      write(*,4)
      'Fully Exposed'
      write(*,4)
      ENDIF
   IF (nerrs.gt.0) THEN
      write(*,'RD PKH: errors found in data so program stops.')
      STOP
   ENDIF

   print*.'Pkh file is: dbname'
   if (trace) write(*,'< rdppkh'
   END
9.2 MkTbl: Make Table of Head-on Lethality Data. The subroutine **mk tbl** makes a head-on pkh table for echo consisting of raw pk data and disaggregated data. If the user desires an echo of sample lethality data, **rd pkh1** calls **mk tbl** to generate the sample data. **Mk tbl** inserts data for 1 foot dispersion at zero range and 1 foot dispersion per 500 meters at greater ranges.

**Mk tbl** copies the raw data read by **rd pkh** into the first four columns of the echo table and the processed data generated by **rd pkh** into the fifth through seventh columns.

### Table 14. Table Generated by MkTbl

<table>
<thead>
<tr>
<th>Range (m)</th>
<th>Kill Probability by Category</th>
<th>Dispersion (ft)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>M-kill</td>
<td>F-kill</td>
</tr>
<tr>
<td>500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2000</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2500</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3000</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**SUBROUTINE** MK_TBL (onlym, onlyf, fandm, mdisp, k, nrow)
**include** 'common.h'
**common** /epkh/ table (4,12), echo(2,7,7), jrg(2,7), jdisp(2,7)

if (trace) print *, '>mktbl'
jrg(k,nrow) = (nrow-1)*irginc
DO 11 j=1,4
      echo(k,nrow,j) = table(j,1)
11 CONTINUE
echo(k,nrow,5) = onlym
echo(k,nrow,6) = onlyf
echo(k,nrow,7) = fandm
jdisp(k,nrow) = mdisp
if (trace) print *, '<mktbl'
END
9.3 RdPkh2: Read HEAT and Missile Lethality Data. Rd Pkh2 reads only 7 lethality values for 7 ranges. These are all catastrophic kill values. It sets the probability of lesser damage levels to zero and stores the K-kill values in appropriate places in the pkh array. This fakes out the kill routine which assumes lethality is a function of other variables besides range.

c V7.3
SUBROUTINE RDPKH2 (dbname, narmy, iecho)
  c
  idh... Jan 1986
  c from grc.oct 1981(tankwars)
  c this replaces rdpkh when the following pk values are read
  c instead. below puts pk values (1 for each range into
  c the k-kill spot in the pkh tables, 0 for m,m+f, f kill spots
  include 'common.h'
  integer dbname
  common /cpkh2/ nangis(2), pkh(2,7,12,2,4,11)
  1 format(6rS.2)
if (trace) print *, '<rdpkh2'
read(dbname,1)pk1,pk2,pk3,pk4,pk6,pk6,pk7
DO 40 irg=1,7
DO 30 nhdfe=1,2
DO 20 mdisp=1,10
  DO 10 mang=1,7
  pkh(narmy,irg,mang,nhdfe,1,mdisp)=0.0
  pkh(narmy,irg,mang,nhdfe,2,mdisp)=0.0
  pkh(narmy,irg,mang,nhdfe,3,mdisp)=0.0
  pkh(narmy,irg,mang,nhdfe,4,mdisp)=pk1
  pkh(narmy,2,mang,nhdfe,4,mdisp)=pk2
  pkh(narmy,3,mang,nhdfe,4,mdisp)=pk3
  pkh(narmy,4,mang,nhdfe,4,mdisp)=pk4
  pkh(narmy,5,mang,nhdfe,4,mdisp)=pk5
  pkh(narmy,6,mang,nhdfe,4,mdisp)=pk6
  pkh(narmy,7,mang,nhdfe,4,mdisp)=pk7
10 CONTINUE
20 CONTINUE
30 CONTINUE
40 CONTINUE
print *, 'Pkh file is:', dbname
if (trace) print *, '<rdpkh2'
END
9.4 RdPkh5: Read Top Attack Lethality Data. Rd pkh5 reads STAFF’s (level 5 round type) lethality data which includes the distance to the lower bound of the fans \( (y_1) \), ‘base’ of the fans \( (y_2) \), and aim point \( (y_3) \), also the number of bands in a fan, sectors in a band, and fans. The lethality of the round depends on its distance above the target and its elevation angle. Each of the 4 kill categories: only mobility, only firepower, only M&F, K-Kill, are read in and separated in each fan.

**Input data**

The game file contains a line specifying where the lethality data is stored. The first column of this line contains an integer which should be 5 for STAFF data. The rest of that line contains the name of the lethality data file.

The lethality file describes the lethality fans. The data includes:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>y1</td>
<td>The distance from the target centroid to the lower bound of the fans.</td>
</tr>
<tr>
<td>y2</td>
<td>The distance from the target centroid to the ‘base’ of the fans.</td>
</tr>
<tr>
<td>y3</td>
<td>The distance from the target centroid to the aim point.</td>
</tr>
<tr>
<td>bands</td>
<td>The number of bands in a fan (1-7).</td>
</tr>
<tr>
<td>sectrs</td>
<td>The number of sectors in a band (1-4).</td>
</tr>
<tr>
<td>fans</td>
<td>The number of fans (7 or 12).</td>
</tr>
</tbody>
</table>

The lethality data is stored in the array pkh5(2,7,4,4,12), where pkh5(i,j,k,l,m) is the lethality data for the ith army, in the jth distance band above the target, in the kth angular sector, for the lth kill level, and for the mth fan. The four levels of kill are: mobility kill, firepower kill, mobility & firepower kill, and catastrophic kill. Table 15 contains dummy data illustrating the format of the STAFF lethality file.

**Table 15. Sample Lethality File**

<table>
<thead>
<tr>
<th>band/sector/kill</th>
<th>FAN DATA</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0 deg</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>2.8, 20,</td>
<td></td>
</tr>
<tr>
<td>7.4,7.</td>
<td>distances to fan, ‘base’, aim point</td>
</tr>
<tr>
<td>45, 52.5, 67.5, 82.5,</td>
<td>#bands, sectors, fans</td>
</tr>
<tr>
<td>sector lower bounds</td>
<td></td>
</tr>
<tr>
<td>60 45.0 m</td>
<td>0.35</td>
</tr>
<tr>
<td>60 45.0 f</td>
<td>0.92</td>
</tr>
<tr>
<td>60 45.0 e</td>
<td>0.94</td>
</tr>
<tr>
<td>60 45.0 k</td>
<td>0.30</td>
</tr>
<tr>
<td>60 52.5 m</td>
<td>0.45</td>
</tr>
<tr>
<td>60 52.5 f</td>
<td>0.88</td>
</tr>
<tr>
<td>60 52.5 e</td>
<td>0.90</td>
</tr>
<tr>
<td>60 52.5 k</td>
<td>0.25</td>
</tr>
<tr>
<td>60 67.5 m</td>
<td>0.52</td>
</tr>
<tr>
<td>60 67.5 f</td>
<td>0.74</td>
</tr>
<tr>
<td>60 67.5 e</td>
<td>0.78</td>
</tr>
<tr>
<td>60 67.5 k</td>
<td>0.20</td>
</tr>
<tr>
<td>60 82.5 m</td>
<td>0.62</td>
</tr>
<tr>
<td>60 82.5 f</td>
<td>0.62</td>
</tr>
<tr>
<td>60 82.5 e</td>
<td>0.70</td>
</tr>
<tr>
<td>60 82.5 k</td>
<td>0.15</td>
</tr>
<tr>
<td>50 45.0 m</td>
<td>0.35</td>
</tr>
<tr>
<td>50 45.0 f</td>
<td>0.35</td>
</tr>
<tr>
<td>50 45.0 e</td>
<td>0.35</td>
</tr>
<tr>
<td>50 45.0 k</td>
<td>0.35</td>
</tr>
<tr>
<td>0 45.0 m</td>
<td>0.35</td>
</tr>
<tr>
<td>0 45.0 f</td>
<td>0.35</td>
</tr>
<tr>
<td>0 45.0 e</td>
<td>0.35</td>
</tr>
<tr>
<td>0 45.0 k</td>
<td>0.35</td>
</tr>
</tbody>
</table>
SUBROUTINE RDPKH5 (dbname,narmy,iecho)

Rdpkh5: read STAFF lethality data.

integer fan, fans, band, bands, sector, sectrs
logical trace, history
real tbl(4,12)
character*32 dbname
common /cpkh5/ anglim(4), pkh5(2,7,4,4,12), y1, y2, y3, fans
common /ctrace/ trace, history
save /cpkh5/

format(12x,120.2)
format(S16.2)
format('LETHALITY: head-on fan, middle band;',/,
  ' angle onlym onlyf onlymf k-kill')

if (trace) print *, '>rdpkh5'

Read pkh in fan.
open (4, file=dbname, status='old')
rewind 4
read(4,*) y1, y2, y3
read(4,*) bands, sectrs, fans
read(4,*) (anglim(n),n=1,sectrs)
DO 30 band=1,bands
  DO 20 sector=1,sectrs
    DO 5 i=1,4
      read(4,1)(tbl(i,fan),fan=1,fans)
  5 CONTINUE
  DO 10 fan-l,fans
    onlym = tbl(3,fan)-tbl(2,fan)
    onlyf = tbl(3,fan)-tbl(1,fan)
    pkh6(narmy,band,sector,1,fan) = onlym
    pkh6(narmy,band,sector,2,fan) = onlyf
    pkh5(narmy,band,sector,3,fan) =
      tbl(3,fan)-onlym-onlyf+tbl(4,fan)
  10 CONTINUE
  DO 20 sector=1,4
    print 2, anglim(sector),(pkh5(narmy,4,sector,i,1),i=1,4)
  20 CONTINUE
  CONTINUE
30 CONTINUE
close(4)

Print sample data in 4th band if desired.
if (iecho.gt.1) THEN
  DO 60 sector=1,4
    print 3, anglim(sector),(pkh5(narmy,4,sector,i,1),i=1,4)
  60 CONTINUE
  CONTINUE
ELSE
  print *, 'Pkh file is:', dbname
  if (trace) print *, '<rdpkh5'
END
9.5 Kill: Find Type of Damage Caused. Kill finds the kill type and probabilities for a hit on a target including: No damage, M-Kill, F-Kill, M&F-Kill, and K-Kill. The results of each shot by kill type and the army firing the shot are tallied in the array kshot. Arguments and local variables.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>I</td>
<td>ID of the firer (0 &lt; I &lt; NN).</td>
</tr>
<tr>
<td>it</td>
<td>ID of the target.</td>
</tr>
<tr>
<td>y</td>
<td>Variable no longer used</td>
</tr>
<tr>
<td>ndisp</td>
<td>Dispersion (1 to 10 ft). 11 for uniform dispersion.</td>
</tr>
<tr>
<td>nang</td>
<td>Aspect angle band. 1 is head on, 2 is 30 degrees, etc.</td>
</tr>
<tr>
<td>pk</td>
<td>Random draw from the uniform distribution</td>
</tr>
<tr>
<td>nhdf</td>
<td>1 for hull defilade, 2 for fully exposed.</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer. 1 for Blue, 2 for Red.</td>
</tr>
<tr>
<td>jrg</td>
<td>Range band. Normally 1 for 0 range, 2 for 500 meters, etc.</td>
</tr>
<tr>
<td>pk_save(i)</td>
<td>Appropriate kill probability for M'-kill, F'-kill, M&amp;F'-kill, and K-kill.</td>
</tr>
</tbody>
</table>

FUNCTION KILL (I, it, y, ndisp, nang)

if (trace) print *, '>kill'
pk = ranu(0.0)
nhdf = kncalc(it)-1
Find probabilities for a production run
m = army(I)
jrg = min(0, nrg-1)
if (kind r(m) > 1) jrg = 1
pk_save(1) = pkh(m, jrg, nang, nhdf, 1, ndisp)
pk_save(2) = pk_Save(1) + pkh(m, jrg, nang, nhdf, 2, ndisp)
pk_save(3) = pk_Save(2) + pkh(m, jrg, nang, nhdf, 3, ndisp)
pk_save(4) = pk_Save(3) + pkh(m, jrg, nang, nhdf, 4, ndisp)

Find which kill type occurs.
if (pk.lt.pk_save(4)) kill = KILL
if (pk.lt.pk_save(3)) kill = MKILL
if (pk.lt.pk_save(2)) kill = FKILL
if (pk.lt.pk_save(1)) kill = MKILL
if (pk.gt.pk_save(4)) kill = ALIVE
if (kill.eq.ALIVE) kshot(m, 10) = kshot(m, 10) + 1
if (kill.eq.MKILL) kshot(m, 11) = kshot(m, 11) + 1
if (kill.eq.FKILL) kshot(m, 12) = kshot(m, 12) + 1
if (kill.eq.MKILL) kshot(m, 13) = kshot(m, 13) + 1
if (kill.eq.FKILL) kshot(m, 14) = kshot(m, 14) + 1
if (trace) print *, '<kill'
END
9.6 Kill5: Find Type of Damage Caused by Top Attack Round. Kill5 randomly chooses the type of damage caused by a single top attack round. It was designed to simulate STAFF rounds but may be appropriate for other top attack rounds. The probability of choosing a mobility, firepower, or catastrophic kill is a function of: height of the round over the target, elevation angle of round, and aspect angle (path of round w.r.t. target front).

Unlike other rounds simulated in the Tank Wars model, STAFF rounds are not fired directly at a target. They are fired over the target, sense the target below them, and fire an explosively formed projectile (EFP) down through the top of the target.

The lethality data is represented as 7 or 12 fans above the target. If the target is symmetrical, only 7 fans are needed; one at each aspect angle in the series 0, 30, ..., 180 degrees. If the target is asymmetrical, 12 fans are needed from 0 to 330 degrees. Figure 8 illustrates a single fan.

The STAFF round must hit a fan above the target as shown in Figure 8. The lethality of the round depends on its distance above the target and its elevation angle. The left figure illustrates cells in the fan. Each cell has kill probabilities at all 4 levels. The right figure illustrates the bivariate normal shot pattern, several defining measurements, and the $x$, $y$ position of the actual round.

Each fan has up to 7 circular bands and 4 wedge shaped sectors on each side of the vertical. Kill5 finds which band and sector the round passes through. It then consults the pkh5 table to find the M, F, M&F, and K-kill probabilities. The program then lays out the probabilities on the unit interval as shown below. Next, it draws a random number to choose an interval. The kill level associated with the interval becomes the damage level.

<table>
<thead>
<tr>
<th>no dmg</th>
<th>M'-kill</th>
<th>F'-kill</th>
<th>M&amp;F'-kill</th>
<th>K-kill</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$p_1$</td>
<td>$p_2$</td>
<td>$p_3$</td>
<td>$p_4$</td>
</tr>
</tbody>
</table>

The program uses the following symbols and relations:
<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>x, y</td>
<td>$x, y$</td>
<td>Coordinates of round from center of impact (m).</td>
</tr>
<tr>
<td>nang</td>
<td># of angular sector in which round approaches target.</td>
<td></td>
</tr>
<tr>
<td>$y_1$</td>
<td>Distance from target to lower edge of fan.</td>
<td></td>
</tr>
<tr>
<td>$y_2$</td>
<td>Distance from target to apex of fan.</td>
<td></td>
</tr>
<tr>
<td>$y_3$</td>
<td>Distance round is aimed above target.</td>
<td></td>
</tr>
<tr>
<td>h</td>
<td>$h = y_1 + y_2 + y_3$</td>
<td>Height of round above fan vertex (m).</td>
</tr>
<tr>
<td>$d$</td>
<td>$d = \sqrt{x^2 + h^2}$</td>
<td>Distance of the round from the base of the fan (m).</td>
</tr>
<tr>
<td>$d &gt; y_1 + y_2$</td>
<td>Implies round above fan.</td>
<td></td>
</tr>
<tr>
<td>$d = y_1 + y_2$</td>
<td>Implies round is above fan.</td>
<td></td>
</tr>
<tr>
<td>$\epsilon = \arctan(h/x)$</td>
<td>Elevation angle of the round.</td>
<td></td>
</tr>
<tr>
<td>$\epsilon &gt; 45$</td>
<td>Implies round is left or right of fan.</td>
<td></td>
</tr>
<tr>
<td>fan</td>
<td>ID of fan perpendicular to approach angle.</td>
<td></td>
</tr>
<tr>
<td>fans</td>
<td># of fans defined by data.</td>
<td></td>
</tr>
<tr>
<td>sector</td>
<td>ID of sector.</td>
<td></td>
</tr>
<tr>
<td>anglim(n)</td>
<td>Lower angular limit of the nth cell in the fan.</td>
<td></td>
</tr>
<tr>
<td>band</td>
<td>ID of band.</td>
<td></td>
</tr>
<tr>
<td>pk</td>
<td>Random draw from uniform distribution</td>
<td></td>
</tr>
<tr>
<td>$m$</td>
<td>1 if Blue firer, 2 if Red firer</td>
<td></td>
</tr>
<tr>
<td>pkh5()</td>
<td>Probability of kill when the round passes through the i;j cell. (For side m, kill level k, and fan l.)</td>
<td></td>
</tr>
<tr>
<td>pksave(4)</td>
<td>$p_i$ Sums of kill probabilities.</td>
<td></td>
</tr>
<tr>
<td>kill5</td>
<td>Damage caused by round. 1 if none, 2 if mobility kill, 3 if firepower kill, 4 if mobility &amp; firepower, 6 if catastrophic.</td>
<td></td>
</tr>
<tr>
<td>kshot()</td>
<td>Tallies statistical results.</td>
<td></td>
</tr>
</tbody>
</table>

FUNCTION KILLS (1, x, y, nang)

if (trace) print '*','>kill5'

h = $y_1 + y_2 + y_3$
d = $\sqrt{x^2 + h^2}$

kill5 = ALIVE

IF (d.gt.y1+y2 and d.lt.y1+y2+70.0) THEN

Round is within distance limits, proceed.

$e = \text{atan2} (h,\text{abs}(x)) \times \text{DEG}$

IF (e.gt.45.) THEN

Round is within angular limits, proceed.

fan = nang

if (nang.gt.fans) fan=14-nang

sector = index[e,4,anglim]

band = 7-int((d-y1-y2)/10.0)

pk = ranu(0.0)

m = army(1)

pksave(4) = 1. - pkh5(m,band,sector,4,fan)

pksave(3) = pksave(4) - pkh5(m,band,sector,3,fan)

pksave(2) = pksave(3) - pkh5(m,band,sector,2,fan)

pksave(1) = pksave(2) - pkh5(m,band,sector,1,fan)

if (pk.gt.pksave(4)) kill5 = KKILL

if (kill5.eq.ALIVE) kshot(m,10) = kshot(m,10)+1

if (kill5.eq.MKILL) kshot(m,11) = kshot(m,11)+1

if (kill5.eq.FKILL) kshot(m,12) = kshot(m,12)+1

if (kill5.eq.MFKILL) kshot(m,13) = kshot(m,13)+1

if (kill5.eq.KKILL) kshot(m,14) = kshot(m,14)+1

ENDIF

if (trace) print '*','>kill5'

END
10. DISENGAGEMENT ROUTINES

There are four categories of disengagement in tank vs tank combat. Each has different causes, consequences, and must be handled separately. A single tank disengages a single target when it has partially or completely serviced it. A single tank may disengage several targets when the tank dies or goes behind terrain. Multiple tanks will disengage a target when it dies or goes behind terrain. And finally, there is general disengagement at the end of combat.

Experience over a dozen years shows that the logic for target disengagement is extremely complex and error prone. For that reason, this section discusses the subject in detail.

One tank disengages a single target. A single tank will disengage a single target if it runs out of ammo, meets its target switching criterion, smoke blocks the line of sight, or the target goes out of range. The target switching criterion are 1) the firer hits the target or 2) the firer fires $n_{temp}$ rounds at the target.

One tank disengages several targets. When a target is F-killed or worse, it disengages any targets it has and takes no new ones. When a target vanishes due to terrain, it disengages any targets it has and again takes no new ones.

Many tanks disengage a single target. All foe will disengage a target when it is visibly killed or vanishes due to terrain. The target is visibly killed when it is K-killed or I-killed.

The I-kill (inactivity kill) is a concept unique to Tank Wars. It occurs when a tank is M&F killed and one or both of the following occur: the target suffers $n_{temp}$ hits or $t_{temp}$ seconds elapse. This is an attempt to simulate gunners discarding a tank that is inactive for some time. The foe will disengage a K-killed target when damage calls newtgt and an I-killed target when itkill calls newtgt.

The target vanishes due to terrain if it's path simply takes it behind terrain, it hides because it is firepower killed or out of ammo, or it is a defender that pops down to 'reload' by replacing an empty missile pod with a full one. The foe will disengage such a vanished target when vanter calls newtgt.
10.1 Disengagement Tactics. Tank Wars contains logic for three disengagement policies so that you can evaluate weapon systems under each policy. Under the first policy, a firer continues to engage its target until the target is K-killed. A variant of this policy allows the firer to disengage if the target is M&F killed and certain other conditions are met. Under the second policy, a firer disengages the target after firing a fixed number of shots at it, but may re-engage at a later time. Under the third policy, a firer disengages the target after hitting it, but may re-engage it at a later time.

Tactic 1: Standard U.S. Armor doctrine says the gunner will shoot at a target until it is known to be dead (K-killed). This ignores the very real possibility that the gunner would do better by switching targets on evidence that the current target is no longer a threat. If the target ceases all activity and the gunner pumps further rounds at the target or a reasonable period of time passes, the target is very likely M&F-killed and should be disengaged. In the model if the target has been M&F-killed and the gunner sees no activity for $t_{bump}$ seconds or pumps another $n_{bump}$ rounds on the target he then disengages the target permanently. If you want to force the model to keep firing at an M&F killed target, just make $t_{bump}$, and $n_{bump}$ very large values.

Tactic 2: If the probability of an F-kill given a shot is high, it may be reasonable to fire a fixed number of rounds at the target and then switch to a more threatening target. Later, at leisure the gunner may return to these partially serviced targets to make sure they are dead. Tactic 2 plays this early switching.

Tactic 3: Similarly, if the probability of an F-kill given a hit is high, it may be reasonable to fire until the target is hit and then switch to another target. Again with the possibility of returning to a previous, partially serviced target. Tactic 3 plays this early switching.

Naturally a firer disengages a target if it vanishes behind terrain and he disengages all his targets if he himself vanishes. Similarly, a firer disengages all targets if he is F-killed or worse.

Actual switching varies a great deal depending on the types of rounds.
10.2 Single-Shot Ballistic Projectiles. For guns firing single-shot KE or HEAT ballistic projectiles, the logic is straightforward to implement. This is discussed below and summarized in Table 16.

The first disengagement policy causes the firer to permanently disengage a target because it is known to be dead. Policies 2 and 3 cause a temporary disengagement because there is reason to believe the target is dead. Even if policy 2 or 3 is used, the disengagement criteria of policy 1 causes a permanent disengagement.

With tactic set to 1, 2, or 3, when a K-kill occurs, the program immediately and permanently disengages the firer from the target. At this same time, if the target is just M&F-killed, the program schedules a "late kill event" and sets a flag. A K-kill causes the M&F-kill to be bumped up to a K-kill and the target to be permanently disengaged. A Late-kill causes bumping to a K-kill by the fire event if \( n_{\text{temp}} \) additional rounds have been fired at the target, and again permanent disengagement occurs. Note that \( t_{\text{temp}} \) is currently a constant but should be replaced with a draw from a random distribution when it becomes known.

Table 16: Switching Logic for Bullets

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Condition</th>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K-kill</td>
<td>Impact</td>
<td>Disengage target forever and select new target.</td>
</tr>
<tr>
<td></td>
<td>M&amp;F-kill</td>
<td>Impact</td>
<td>Schedule a late kill in ( t ) seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Also set a flag to switch after ( n_{\text{a}} ) more shots.</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>F-kill +</td>
<td>LateKill</td>
</tr>
<tr>
<td></td>
<td>( t_{\text{temp}} ) sec</td>
<td>Fire</td>
<td>select new target.</td>
</tr>
<tr>
<td></td>
<td>M&amp;F-kill. and ( n_{\text{temp}} ) shots</td>
<td></td>
<td>Disengage target forever and select new target.</td>
</tr>
<tr>
<td>2*</td>
<td>Hit</td>
<td>Impact</td>
<td>Disengage target for now and select new target.</td>
</tr>
<tr>
<td>3*</td>
<td>Fired</td>
<td>Fire</td>
<td>Disengage target for now and select a new target.</td>
</tr>
<tr>
<td></td>
<td>( n_{\text{rds}} )</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* The logic of tactic 1 is used for permanently disengaging a target.

In the second type of policy the gunner fires a fixed number of rounds at a target and then attempts to switch targets. The fire routine triggers this switch as soon as say 3 rounds are fired. This is a temporary disengagement; the firer may re-engage the target later.

In the third type of policy the gunner fires until he gets a hit and then attempts to switch. The Impact event triggers this switch as soon as a hit occurs. This is a temporary disengagement; the firer may re-engage the target later.
10.3 Missile Systems. Tank Wars simulates current 'simple' missile systems as well as proposed systems that can guide N missiles to N targets simultaneously. Appropriate disengagement policies are discussed below.

Simple missile systems. A simple missile system is one that can only handle one target at a time and one whose missile must be guided by the gunner until impact. Target disengagement for this kind of weapon is discussed below and summarized in table 17.

Tactic 1: The standard missile uses the same logic as for single shot bullets except that the missile must be guided to impact. As before, target disengagement on a K-kill occurs at impact time. If the target is M&F-killed and disengagement is to occur after a delay, the firer may have already launched another missile at the target. Disengagement should not take place until this missile impacts.

Tactic 2: Don't use this tactic. Switching after firing doesn't make much sense for the simple missile system. The gunner will surely guide the missile until impact and if it is a miss he will fire again. If it is a hit and the gunner wishes to switch after a hit, tactic 3 should be used.

Tactic 3: The gunner temporarily disengages the target and selects a more threatening target if one is available.

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Condition</th>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>K-kill</td>
<td>Impact</td>
<td>Disengage target forever and select new target.</td>
</tr>
<tr>
<td></td>
<td>M&amp;F-kill</td>
<td>Impact</td>
<td>Schedule a late kill in t seconds.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Also set a flag to switch after n more shots.</td>
</tr>
<tr>
<td></td>
<td>M</td>
<td>F-kill +</td>
<td>LateKL select new target.</td>
</tr>
<tr>
<td></td>
<td>$t_{\text{ samp }}$ sec</td>
<td>Fire</td>
<td>Disengage target forever and select new target.</td>
</tr>
<tr>
<td></td>
<td>M&amp;F-kill, and $n_{\text{ same }}$ shots</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2*</td>
<td>(Don't use)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3*</td>
<td>Fired</td>
<td>Impact</td>
<td>Disengage target for now and select a new target.</td>
</tr>
<tr>
<td></td>
<td>$n_{1}$ rd</td>
<td></td>
<td>(Since msl is guided to impact switching cannot occur until then)</td>
</tr>
</tbody>
</table>

* The logic of tactic 1 is used for permanently disengaging a target.

Multi-Target Missile Systems. Suppose a multi-missile can handle 4 targets at a time. It selects the first target and fires at it, then it selects a second target and fires at it while it automatically guides the first missile to impact. It fires at the second target, then selects a third target meanwhile guiding perhaps two missiles. The fourth target is selected and fired upon and now if it is guiding four missiles, it cannot switch to a fifth target after firing because all the guidance 'circuits' are busy and it must wait until one of the missiles impact and frees up guidance 'circuits'. Thus the first select is triggered by a detection, the second, third, and fourth by a fire event, and the fifth by an impact event. This is discussed below and summarized in table 18.
Tactic 1: Do not use this tactic. This kind of system selects a new target for every missile. If the target is known to be dead, it will not be reselected.

Tactic 2: Don’t use this tactic. Switching after firing doesn’t make much sense for the simple missile system. The gunner will surely guide the missile until impact and if it is a miss he will fire again. If it is a hit and the gunner wishes to switch after a hit, tactic 3 should be used.

Tactic 3: This is the appropriate tactic; select a new target after each fire event if not loaded with targets, otherwise switch at impact. Disengage each target at impact.

Table 18 Switching Logic for Multi-Target Missile Systems

<table>
<thead>
<tr>
<th>Tactic</th>
<th>Condition</th>
<th>Event</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Don’t use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>2*</td>
<td>Don’t use</td>
<td></td>
<td></td>
</tr>
<tr>
<td>3*</td>
<td>Fired</td>
<td>Impact</td>
<td>Disengage target for now and select a new target.</td>
</tr>
<tr>
<td></td>
<td>n_{temp} rd</td>
<td></td>
<td>(Since msl is guided to impact switching cannot occur until then)</td>
</tr>
</tbody>
</table>

* n_{temp} should be set to 1. A target that is known to be dead (K-killed) will never be reselected. Suppose the system can handle 4 targets at a time. If it is currently engaging less than 4 it is not loaded: it can handle more targets. At fire time, if it is not loaded with targets, it then selects another target. If it is loaded at fire time, it waits until one of the missiles impacts before selecting another target.
10.4 Key Disengagement Variables. At various places in the program the code must find the answer to certain key questions. The variables containing the answers and the key questions are:

<table>
<thead>
<tr>
<th>VARIABLE</th>
<th>QUESTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>busy(i)</td>
<td>Is i too busy to select a new target?</td>
</tr>
<tr>
<td>nr tgt(i)</td>
<td>What’s firer i’s latest target?</td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>Is firer i engaging target j?</td>
</tr>
<tr>
<td>mot(i,j)</td>
<td>Does firer i have a missile enroute to target j?</td>
</tr>
<tr>
<td>loaded</td>
<td>Are firer i’s guidance channels full? (missile firers)</td>
</tr>
<tr>
<td>empty(i)</td>
<td>Is firer i engaging a target now? (gun systems)</td>
</tr>
<tr>
<td></td>
<td>Is firer i’s upraised missile pod empty? (missiles)</td>
</tr>
<tr>
<td></td>
<td>Is firer i out of ammo? (guns or missiles)</td>
</tr>
</tbody>
</table>

The table below tells which routines set, unset, or use certain key variables. A dash means the variable is not mentioned. Roman font means this is the normal place to set or unset the variable. Such settings are clear and well understood. Italic indicates an abnormal setting. These settings are a mystery.

The routines in the first column are divided into three groups. The first routine init unsets the variables at the beginning of each engagement. The second set of routines, selects through reload are the normal firing sequence routines. The last set of routines, newtgt through vansmk handle the firer's activities when the target dies or vanishes.

```
<table>
<thead>
<tr>
<th></th>
<th>busy</th>
<th>nr tgt</th>
<th>fot</th>
<th>mot</th>
<th>loaded (local)</th>
<th>empty</th>
</tr>
</thead>
<tbody>
<tr>
<td>Init</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
</tr>
<tr>
<td>Select</td>
<td>set</td>
<td>-</td>
<td>set</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Select</td>
<td>unset</td>
<td>set</td>
<td>set</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Fire</td>
<td>unset</td>
<td>-</td>
<td>set</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-Frdsml</td>
<td>unset</td>
<td>-</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>set</td>
</tr>
<tr>
<td>-Frdsng</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>set</td>
</tr>
<tr>
<td>Impact</td>
<td>-</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>-Diseng</td>
<td>-</td>
<td>unset</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>--Frdsml</td>
<td>-</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Reload</td>
<td>-</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>unset</td>
</tr>
<tr>
<td>Newtgt</td>
<td>unset</td>
<td>unset</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>set</td>
</tr>
<tr>
<td>Abort</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>unset</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Damagf</td>
<td>-</td>
<td>unset</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Vanter</td>
<td>-</td>
<td>unset</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Vansmk</td>
<td>unset</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
```

Below is a discussion of each variable and why each routine gives it the value it does. In the discussion, the subscript i is the ID of a firer:searcher, j is the ID of the target, and k is the ID of the target’s target.

**Busy**

 Tells whether the firer is too busy to select a new target. It's purpose is to inhibit selection of a second target from the time it starts to select a first target until it fires on that first target. (The variables loaded and empty inhibit selection for other reasons.)

**Init**

 When the game begins, i is not busy with a target, so busy(i)=false.
Selecs When i begins to select a target, i is busy selecting, so busy(i)=.true.
Select At the end of selection, if i has no targets, i is no longer busy with a target, so busy(i)=.false.
Fire If i did select a target, then when i fires at it, i is no longer busy, so busy(i)=.false.
Frdssg Resetting busy is redundant since it was just reset in fire. Anyway, here's the scenario. The firer is a gun system that still has ammo. It disengages after firing nrpt rounds at the target and it's done just that.
Newtgt Firer i has selected target j but hasn't yet fired on it. Either target j dies or goes behind terrain, so newtgt is called.
Vansmk Firer i has selected target j but hasn't yet fired on it. Now smoke blocks the line of sight between i and j. Busy must be reset so firer i can select a new target. The code is: if (busy(i).and.nrtgt(i).eq.j) busy(i)=.false.
Nrtgt It is a vector containing the ID of the latest target for each firer. It contains information similar to that in fot.
Init At the beginning of the engagement, firer i has no target, so nrtgt(i)=0.
Select When i selects target j, then nrtgt(i)=j.
Impact Firer i is a gun system. Target j is a false target or firer i switches targets on a hit or target j is beyond 4km. If any of these are true, firer i will disengage.
Diseng This is where a single firer normally disengages a single target.
Reload When reload is complete, the firer selects a target. Any prior target is discarded by resetting nrtgt(i)=0. Note that reload is scheduled by frdmsl and by newtgt. After the latter schedules reload it also resets nrtgt(i). This is redundant.
Newtgt Firer i's target has died or vanished. The code discards i's target by resetting nrtgt(i)=0.
Damagf System j is a target that has just been F-killed. It no longer has a target of its own, so the routine sets nrtgt(j)=0.
Vanter System j is an attacker that has moved behind terrain or a system that has popped down to reload. It discards its target, so nrtgt(j)=0. Note: When systems are F-killed and then hide, vanter is also called. Nrtgt is already reset, so resetting it here is redundant, but that's ok.
Fot A matrix telling whether a firer is engaging a target. It contains information similar to that in nrtgt.
Init At the beginning of the engagement, no firer is engaging any target, so fot(i,j)=.false.
Select When i selects a target j, the fot(i,j)=.true.
Frdmsl Missile system i is ready to fire another missile system. It's firing policy is to fire a hxed number of rounds at the target and then switch targets. It's done that and is ready to switch targets. As part of dropping target j, the code resets fot(i,j)=.false.
Impact Here's the scenario. Firer i is a gun system. Target j is a false target or firer i switches targets on a hit or target j is beyond 1km. If any of these are true, firer i will disengage.
Diseng Firer i is disengaging target j, so it resets fot(i,j)=.false.
Newtgt Firer i is disengaging target j, so it resets fot(i,j)=.false.
Damagf When a system j is F-killed, it no longer has a target. That's why fot(j,k) is set to 0 for all foes k.
Vanter System j is an attacker that has moved behind terrain or a system that has popped down to reload. It discards all its targets, so fot(j,k)=true. for all foes k.
Note When systems are F-killed and then hide, vanter is also called. fot is already reset, so resetting it here is redundant, but that's ok.
Mot An array that tells which firers have a missile in flight to which targets.
Init When the game begins, firer i has no missiles enroute to it, so mot(i,j)=.false.
Fire When i fires a missile at j, mot(i,j)=.true.
Impact The missile is no longer enroute, so mot(i,j)=.false.
Abort If j dies or disappears enroute, the missile is aborted, so mot(i,j)=.false.
Loaded  In impact, newtgt, and selecs it tells whether all of a missile system's guidance channels are loaded. In selecs it tells whether a gun system has a target or not. Loaded is similar to busy in that it inhibits the selection of a new target.

Empty  Tells whether the current missile pod is empty or not. Sometimes tells when the system is out of ammo.

Init  When the game begins, the pod is not empty, so empty(I) = .false.

Frdmsl  During the course of firing, it may become empty.

Frdssg  Gun system i is out of ammo. Reset empty(i) = .true.

Reload  At the end of reload, the pod is not empty, so empty(I) = .false.

Newtgt  The target died or vanished. Firer i was engaging it, so the code checks to see if the current missile pod is empty and records whether it is by setting empty(i) = .true.
10.5 Diseng: Attempt to Disengage 1 Firer from 1 Target. There are two circumstances in the model where a single firer disengages a single target. When a round impacts the firer may choose to disengage. When smoke breaks line of sight, either or both of the parties involved will disengage if they were engaged. **Diseng** handles these two cases.

The first step is to find whether the firer is actually engaging the target. If not, **diseng** does nothing. If the firer is engaging the target **diseng** branches depending on whether it is firing a gun or missile.

If it’s a gun system, the following steps are taken:

1. If the firer is about to fire at a real target, the fire event is canceled and it’s disengaged from it.
2. If the firer has ammo it begins to select a new target.
3. If it has no ammo but has mobility and is a halt-to-fire system it starts to accelerate.

In any case, the number of rounds on target is reset to zero and the ID of the firer’s target is reset to zero.

If it’s a missile system, the following occurs:

1. The code may decrement the number of guidance channels the firer has busy.
2. If the target is a real target (not a false one), and the firer is about to fire on the target, the fire event is canceled and the firer is disengaged from the target. If the calling event is impact (on false target) the firer reselects.
3. The code may call **frdmsl** to start on a new target.

Finally, whether it’s a gun or missile, if the firer was engaging this target, the code checks to see if search is off. If it’s off, it gets turned back on.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>drop</td>
<td>True IFF target should be dropped (logical).</td>
</tr>
<tr>
<td>take</td>
<td>True IFF new target should be selected (logical).</td>
</tr>
<tr>
<td>mytgt</td>
<td>ID of firer’s actual target. ??</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>n</td>
<td>Side of target (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>havamo</td>
<td>True IFF firer has ammo. (logical)</td>
</tr>
<tr>
<td>nrd(I)</td>
<td>Number of rounds fired by firer.</td>
</tr>
<tr>
<td>nrd(m)</td>
<td>Number of rounds firers on side m start with.</td>
</tr>
<tr>
<td>inbrst</td>
<td>True IFF a burst firer is in the middle of a burst. (logical)</td>
</tr>
<tr>
<td>nrpb(m)</td>
<td>Number of rounds in a burst. If &lt; 2 it is a single shot gun.</td>
</tr>
<tr>
<td>nrib(I)</td>
<td>Number of rounds fired in the burst so far.</td>
</tr>
<tr>
<td>on tgt</td>
<td>True IFF firer is engaging this target. (logical)</td>
</tr>
<tr>
<td>kind</td>
<td>Kind of round. (1=KE, 2=HEAT, 3 unused, 4=missile, 5=top attack)</td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>True IFF firer is about to fire at this target. (logical)</td>
</tr>
<tr>
<td>thuman</td>
<td>Random time for firer to select a target (sec).</td>
</tr>
<tr>
<td>nrot(I)</td>
<td>Number of rounds firer has fired at target since engaging it.</td>
</tr>
</tbody>
</table>

C 7.3

SUBROUTINE DIS ENG (t, I, it, drop, take)

c Diseng: attempt to disengage 1 firer from 1 target.
c Diseng is called by impact if firer condition warrants.
c When I include guns, other routines may call it.
c include 'common.h'

c format (i8.2,1x,a4,13,'dis-eng',14,3.20x,'#tgts=',i2)
c if (ftrav) print *, ' dis-eng'

c Set useful local variables
my tgt = nrtgt(I)
m = army(I)
n = 3-m
hav amo = nrd(I)lt.nrdls(m)|
inbrst = nrpb(m)gt.1 and. (0.ne.mod[nrib(I),1])
1 nrpb(m11)
if (it.eq.FLS.T(I)) on tgt = true.
if (it.ne.FLS.TGT) on tgt = fot(I.Lt) or.
1 (kind(I),m1)-eq.I and. mo(I.lt.i))
IF (on tgt) THEN

c  Firer on this target
  kind = kindrd(m)
  IF (kind .le. 2 .or. kind .eq. 5) THEN
    IF (nrpb(m) .le. 1) THEN
      Single shot gun system or STAFF fire & forget system.
      IF (it .ne. FLS TGT) THEN
        IF (nrpb(m) .le. 1) THEN
          Firer attempts to select a new target.
          IF (take) THEN
            The firer begins to select a new target right now and
            finishes the selection in a few seconds.
          END IF
          END IF
        ELSE
          Burst fire gun system.
          print *. 'DISENG: Not implemented for burst fire guns.'
          STOP
          END IF
        ELSEIF (kind .eq. 4) THEN
          Guided missile system.
          IF (drop) nchan(l) = nchan(l)-1
          IF (it .ne. FLS TGT) THEN
            IF (rot(l, it)) THEN
              call cancel(l, 'fire ', it)
              rot(l, it) = false.
            END IF
          IF (histr) print 3, t, color(m), I,
          color(n), it, nchan(I)
          call skedul(t, I, 'accel ', NULL)
          END IF
          nrot(I) = 0
          nrg(I) = 0
        ELSE
          Burst fire gun system.
          print *. 'DISENG: Not implemented for burst fire guns.'
          STOP
          END IF
        ELSEIF (kind .eq. 4) THEN
          Guided missile system.
          IF (drop) nchan(l) = nchan(l)-1
          IF (it .ne. FLS TGT) THEN
            IF (rot(l, it)) THEN
              call cancel(l, 'fire ', it)
              rot(l, it) = false.
            END IF
          END IF
          END IF
          IF (.not. repeat) THEN
            repeat = true.
            call skedul(t+.01, 0, 'search', NULL)
          END IF
          ELSEIF (kind .eq. 4) THEN
          Guided missile system.
          IF (drop) nchan(l) = nchan(l)-1
          IF (it .ne. FLS TGT) THEN
            IF (rot(l, it)) THEN
              call cancel(l, 'fire ', it)
              rot(l, it) = false.
            END IF
          END IF
          END IF
        END IF
      END IF
    END IF
  END IF
END IF

10.6 Newtgt: Redirect All Foe to a New Target. When a target is obviously killed or disappears every foe engaging the target disengages it and attempts to engage a new target. In Tank Wars this occurs when the target becomes K-killed or I-killed. It also occurs on a hide or vanish due to terrain.

First the code finds the first and last foe of this target. Then it sets a few variables for later use. Then it checks all the foe to see if they are engaging the target. If not, nothing happens. If so, the code branches for gun & missiles.

**Gun systems.** If the gun fires bursts, the program stops because this branch is not developed. If it’s single shot, the code does the following:

1. Cancel foe’s firing at target.
2. Permit foe to select new target (if this was current one).
3. Delete foe’s current target (if this was current one).
4. Delete current target of foe.
5. Reset # rds fired at current target to zero.
6. IF (foe has ammo) THEN
   foe starts selection
ELSEIF (foe can go and is halt-to-fire) THEN
   schedule acceleration
ENDIF

**Missile systems.** The code does the following:

1. Cancel foe’s firing at target (if fot).
2. If (foe has msl enroute to target) abort it
3. Some other stuff under special conditions.

```plaintext
CODE          COMMENT

 t         Time (sec).
 I         ID of firer.
 it        ID of target.
 m         Side of firer (1=Blue, 2=Red).
 n         Side of target (1=Blue, 2=Red).
 havamo    True IFF firer has ammo. (logical)
 nrd(I)   Number of rounds fired by firer.
 nrdso(m) Number of rounds firers on side m start with.
 nrpb(m)   Number of rounds in a burst. If < 2 it is a single shot gun.
 kind      Kind of round. (1=KE, 2=HEAT, 3=un:used, 4=missile, 5=top attack)
 fot(i,j)  True IFF firer is about to fire at this target. (logical)
 thuman    Random time for firer to select a target (sec).
 nrot(l)   Number of rounds firer has fired at target, since engaging it.

V7.5
SUBROUTINE NEWTGT (t, I, it)
3 Newtgt: redirect all ’attackers’ of a new target.
 c Newtgt called for non-false tgts only and only if it warrants it. It should only be called if tgt is V-killed.
 c v7.5 if (I.gt.ITU) first = I
        last = last + first
        integer loaded, hav amo, range
1 format(82, 1x, 34, (3, 'begins to reload.')
2 format(82, 1x, 34, (3, 'begins to reload.'))
      if (trace) print *, 'newtgt'
      Find first and last ’attacker’
      first = 1
      if (I.gt.ITU) first = nblu + 1
      last = nblu
      if (I.gt.ITU) last = nblu + pred
      m = army(first)
      n = 3. m
      kind = kind + m
```
nrpb2 = nrpb(m)
DO 20 j=first, last
  IF ((mot(j, it) .or. fot(j, it)) .and. life(j, it) .eq. FKILL) THEN
    IF (kind.eq.2 .or. kind.eq.5) THEN
      c Single shot gun system or other fire & forget system.
      IF (nrpb(m).le.1) THEN
        Single shot gun system.
        call cancel(j, 'fire', it)
      ELSE
        Single shot gun system.
        call cancel(j, 'fire', it)
        if (nrtgt(j).eq.i) busy(j) = .false.
        if (nrtgt(j).eq.i) nrpt(j) = 0
        hav amo = nrd(j).lt. nrd(m)
        IF (hav amo) THEN
          thuman = 2. * exp(rann(0.5))
          call selec(t, j, thuman)
        ELSEIF (can got(i, j) and. ishtfs(m).gt.0) THEN
          Move out
          call sched(j, 'accel', NULL)
          ENDIF
          nrot(j) = 0
          fot(j, it) = false.
          if (hist) print 1, t, color(m), j, 1
        ELSE
          Burst fire gun.
          print *, 'NEWTGT: Not implemented for burst fire.
          STOP
        ENDIF
      ELSEIF (kind.eq.4) THEN
        Guided missile branch.
        if (fot(j, it)) call cancel(j, 'fire', it)
        if (mot(j, it)) call abort(t, j, it)
        loaded = nchan(j).eq. nchan(m)
        IF ((.not. empty(j) and. mot(j, it) and. loaded) .or.
            (.not. empty(j) and. fot(j, it))) THEN
          IF ((mod(nrd(j), npsd(m)).gt.0) .or. 1
            fot(j, it)) THEN
            More rds in pod
            call cancel(j, 'select', NULL)
            busy(j) = .false.
            fot(j, it) = .false.
            ENDIF
          ENDIF
        ENDIF
        c if (it.ne.FLS TGT) fot(j, it) = .false.
        thuman = 2. * exp(rann(0.5))
        call selec(t, j, thuman)
        ELSE
          Treat empty missile pod
          empty(j) = .true.
          call cancel(j, 'fire', it)
          call cancel(j, 'select', NULL)
          busy(j) = .false.
          nrd(j) = 0
          ENDIF
        c shud btf that is slowing to engage speed up now?
        call sched (t, trelod(m), j, 'reload', NULL)
        if (hist) print 2, t, color(m), j
        ENDIF
      ENDIF
      ENDIF
    ELSEIF (can got(i, j) and. ishtfs(m).gt.0) THEN
      ELSE
10.7 Abort: Abort a Missile in Flight. This routine simulates the abortion of a missile in flight. This happens when the target or firer disappears behind terrain or when either is killed. In any of these cases, the code cancels the impact event for the missile. If the launch vehicle is in a defensive position, alive, and has an empty missile pod, it pops down to bring up another missile pod.

Relevant variables are:

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of launch vehicle.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>m</td>
<td>Side of launch vehicle (1 for Blue, 2 for Red).</td>
</tr>
<tr>
<td>n</td>
<td>Side of target.</td>
</tr>
<tr>
<td>k</td>
<td>Guidance channel number.</td>
</tr>
<tr>
<td>chanel(m,1,k)</td>
<td>ID of missile in kth guidance channel.</td>
</tr>
<tr>
<td>msl</td>
<td>ID of missile and location of its attributes in the a-array.</td>
</tr>
<tr>
<td>msltgt</td>
<td>ID of missile's target.</td>
</tr>
<tr>
<td>kshot(m,3)</td>
<td>Count of missiles aborted for side m.</td>
</tr>
<tr>
<td>mot(I,it)</td>
<td>Set 'I have a missile on (target) it' to .false.</td>
</tr>
<tr>
<td>a(msl)</td>
<td>Storage area for attributes of missile. Release it.</td>
</tr>
<tr>
<td>empty(I)</td>
<td>True IFF I (firer) am in a defensive posture.</td>
</tr>
</tbody>
</table>

The code loops through all guidance channels and finds the ID of the missile assigned to the channel. From the ID, it finds which target the missile is approaching. If the target is the right one, the code aborts that missile.

In aborting the missile, the code tallies another abort for the side the launch vehicle is on and cancels missile impact. Then it frees the guidance channel. If the target is not a false target, it clears the 'missile on target' record. Next, it releases the storage area containing the attributes of the missile. Finally, it determines whether the firing vehicle is in a defensive posture and needs to pop down behind cover while it brings up another missile pod.
11. MOTION ROUTINES

The motion routines handle initial deployment of the tanks on each side, the significant motion events, and provide motion information to other routines.
11.1 Deploy: Place Combatants at Start of Engagement. Deploy sets the exposure, position, and motion of the tanks at the beginning of each engagement.

Opening Range. The initial separation between forces may be thought of as the distance between the forces when they are first able to see each other because one of the forces just came from behind terrain or the fog just lifted. Naturally, the initial separation can vary greatly due to the terrain, weather, or tactical obscurants.

Peterson\textsuperscript{1,2} analyzed World War II data to find the distribution of ranges at which tanks were killed in Northern Europe and a later study (The NATO Range Study) in the sixties also analyzed the ranges at which tanks might be killed in future combat. (This latter study is apparently unpublished and is classified.) While these studies indicate the ranges at which tanks are killed, they do not give the analyst or simulation builder what he needs; the distributions of opening ranges and the motion of the tanks during combat. The user should consult these references or use good engineering judgement and give the model the appropriate opening ranges.

The code breaks into three divisions. First, it initializes the Blue tanks, then the Red, and finally, it calculates some values for sinusoidal motion.

To initialize the Blue tanks, the code sets the spacing between the tanks to 100 meters. Realistic values are 70 to 150 meters. Then it finds a point South of the East-West axis as a base for placing the tanks.

Then the DO 20 loop assigns values to each of the Blue tanks. The following chart explains these assigned values.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>army(I) = BLU</td>
<td>Tank I is associated with the Blue side.</td>
</tr>
<tr>
<td>kneal(I) = FE</td>
<td>Tentatively made Fully Exposed.</td>
</tr>
<tr>
<td>scene.eq.RATTAK</td>
<td>True if Blue is in a defensive posture.</td>
</tr>
<tr>
<td>kneal(I) = HD</td>
<td>Exposure changed to Hull Defilade.</td>
</tr>
<tr>
<td>t0(I) = 0.0</td>
<td>Time position of tank I last updated.</td>
</tr>
<tr>
<td>x0(I) = 0.0</td>
<td>Tank I placed on the y axis.</td>
</tr>
<tr>
<td>y0(i) = sp*i+c</td>
<td>Tank I offset along the y axis from previous tank.</td>
</tr>
<tr>
<td>vx0(i) = 0</td>
<td>Tentatively set to zero speed.</td>
</tr>
<tr>
<td>vx0(I) = speed(BLU)</td>
<td>Reset to combat speed if Blue is attacking.</td>
</tr>
<tr>
<td>motion(I)</td>
<td>Tentatively set to stationary.</td>
</tr>
<tr>
<td>motion(I) = MAXVEL</td>
<td>Reset to moving at maximum combat speed if attacking.</td>
</tr>
<tr>
<td>nwaves.gt.1</td>
<td>Test to see if Blue fights multiple waves of Reds.</td>
</tr>
<tr>
<td>nrd(i) = nused(neval+1)</td>
<td>Reset number of rounds fired by ith tank.</td>
</tr>
<tr>
<td>neval = neval+nblu</td>
<td>Tallies # of Blues taken from the regrouping pool.</td>
</tr>
</tbody>
</table>

The DO 30 loop performs similar assignments for the Red tanks. The major difference is that Red survivors are not regrouped to fight subsequent waves of Blue tanks, so the last two lines of the DO 20 loop have no equivalent in the Red DO 30 loop.

Lateral motion values found at the end may be ignored in later routines. If so this portion can be deleted. Nobody's ever used it anyway.
y0(i) = sp*i+c
vx0(i) = 0.0
if (scene.eq.BATTAK) vx0(i) = speed(BLU)
motion(i) = STATNY
if (scene.eq.BATTAK) motion(i) = MAXVL
if (nwaves.gt.l1) nrd(i) = nused(neval+1)
if (nwaves.gt.l1) neval = neval+nblu
20 CONTINUE

c Initialize Red tanks.
sp = 100.0
k = -sp*(1+nred)/2.0
DO 30 j=1,nred
i = nblu+j
army(i) = RED
kneal(i) = FE
if (scene.eq.BATTAK) kneal(i) = HD
t0(i) = 0.0
x0(i) = 0.0
y0(i) = sp*i+c
vx0(i) = 0.0
if (scene.eq.RATTAK) vx0(i) = -speed(RED)
motion(i) = STATNY
if (scene.eq.RATTAK) motion(i) = MAXVL
30 CONTINUE

c Hardwired lateral motion values.
accmax(BLU) = 0.0
accmax(RED) = 0.0
width(BLU) = 50.
width(RED) = 50.
amp(BLU) = accmax(BLU)/(TWOPISPEED(BLU)/WIDTH(BLU))**2
amp(RED) = accmax(RED)/(TWOPISPEED(RED)/WIDTH(RED))**2
if (trace) print*,'<deploy'
END
11.2 SlowUp: Begin Deceleration. Slow up simulates the moment when a moving tank begins to slow up. If the tank was previously moving, the code records the tank as slowing and schedules a halt event.

Slow up branches to one of 4 branches depending on the previous motion of the tank. Each of the branches prints a line appropriate to that branch if the program is printing an event history. If the tank was already slowing or stationary, the code takes the first or second branch respectively and does nothing beyond possibly printing a line for the event history. If the tank was accelerating or at cruise speed, the code takes the third or fourth branch respectively.

If the tank was accelerating, the third branch finds the position and velocity of the tank at the time it begins to slow down. It calls path to generate these values. Then it finds the time required to stop using the following equation:

\[ dt = \frac{|v_y|}{decel} \]

Finally, it sets motion to SLOWNG (=2) and schedules a halt in \( dt \) seconds.

If the tank was at cruise speed, the code executes the fourth branch which is similar to the third branch.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATHEMATICS</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( t )</td>
<td>( t )</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>( I )</td>
<td>ID of tank.</td>
<td></td>
</tr>
<tr>
<td>kindmv</td>
<td>Motion of tank (1=slowing, 2=stationary, 3=accelerating, 4=cruising).</td>
<td></td>
</tr>
<tr>
<td>( n )</td>
<td>Side of tank (1=Blue, 2=Red).</td>
<td></td>
</tr>
<tr>
<td>( decel(n) )</td>
<td>( d )</td>
<td>Deceleration (( m/s^2 )).</td>
</tr>
<tr>
<td>( v_y )</td>
<td>Current speed (( m/s )).</td>
<td></td>
</tr>
<tr>
<td>( \Delta t = \frac{</td>
<td>v_y</td>
<td>}{d} )</td>
</tr>
<tr>
<td>( \Delta t = \frac{s}{d} )</td>
<td>Time to halt from cruise speed (sec).</td>
<td></td>
</tr>
</tbody>
</table>

```c
SUBROUTINE SLOWUP (t, I)  
  1 IF (kindmv.eq.SLOWNG) THEN  
    IF (trace) print *, ' -slowup'  
    kindmv = motion(I)  
    n = army(I)  
    IF (kindmv.eq.SLOWNG) THEN  
      initialize print 1, t, color(n), 1  
      ELSEIF (kindmv.eq.STATNY) THEN  
      ELSEIF (kindmv.eq.ACCELG) THEN  
      ENDIF  
      call path (I, motion(I), 0, x, y, vx, vy)  
      dt = abs(vy) / decel(n)  
      motion(I) = SLOWNG  
      call skedull(+dt, I, halt = NULL)  
      ELSEIF (kindmv.eq.MAXVL) THEN  
      ENDIF  
      print 1, t, color(n), 1  
      ELSE  
      ENDIF  
      ENDIF  
    ENDIF  
  ELSEIF (kindmv.eq.MAXVL) THEN  
  ELSEIF (kindmv.eq.MAXVL) THEN  
    call path (I, motion(I), 0, x, y, vx, vy)  
    schedule halt time  
    dt = speed(n) / decel(n)  
    call skedull(+dt, I, halt = NULL)  
    motion(I) = SLOWNG  
  ENDIF  
  ENDIF  
END  
```

118
11.3 Halt: Simulate a Tank Halting. Halt simulates a halt event. If the tank is a halt-to-fire system, halt may trigger the engagement of a target.

The simulation of a halt is simple, but the consequences of the event are more complicated than in other motion routines. First, if the model is simulating terrain intervisibility (invisb.eq.1), any vanish events associated with the tank must be canceled. Then halt calls path to update the tank’s position and set the velocity to zero. It then marks the tank as stationary.

The bulk of the code treats the halt-to-fire system. If the system is halt to fire (ishfts(m)=1), and it can shoot (life(I)<FKILL), and it has ammo (nrd(l).lt.nrds(m)), then the branch is executed.

Inside the branch, the code finds whether the firer still has a target. (A firer ‘has a target’ if it has selected a target from among 1 or more it is aware of. This continues until it disengages the target.) If the target is a false target, the firer still has a target if it's not in full defilade. If the target is not a false target, the firer still has a target if line of sight exists between the firer and the target.

If the firer has a target the code schedules a fire event, otherwise the code attempts to schedule the firer to accelerate. The fire event occurs after a delay dt, where:

\[ dt = t_f e^{N[0.5]} \]

Where,
- dt is the delay time (sec),
- \( t_f \) is the median time to fire t-th first round (sec),
- \( N[0.5] \) is a draw from the normal distribution with \( \sigma = 0.5 \).

But wait, \( dt = \max(0.1, dt-3) \) 'hat’s that for? The code indicates that this is a first round being fired at the target, sets the number of rounds fired in a burst to zero, and schedules a fire event.

If the target has disappeared, the code schedules an immediate acceleration for the firer.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>I</td>
<td>ID of tank.</td>
</tr>
<tr>
<td>m</td>
<td>Side of tank (1=Blue, 2=Red).</td>
<td></td>
</tr>
<tr>
<td>motion(i)</td>
<td>Current motion of tank.</td>
<td></td>
</tr>
<tr>
<td>x,y</td>
<td>Coordinates of tank (m).</td>
<td></td>
</tr>
<tr>
<td>vx,vy</td>
<td>Velocity of tank (m/s).</td>
<td></td>
</tr>
<tr>
<td>ishfts(m)</td>
<td>1=halt to fire system. 2=fire on the move.</td>
<td></td>
</tr>
<tr>
<td>life(m)</td>
<td>Status of tank (1=ALIVE, 2=M-killed, etc).</td>
<td></td>
</tr>
<tr>
<td>nrd(l)</td>
<td>Number of rounds fired by tank.</td>
<td></td>
</tr>
<tr>
<td>nrds(m)</td>
<td>Number of rounds at start of engagement.</td>
<td></td>
</tr>
<tr>
<td>nrtgt(l)</td>
<td>ID of tank’s target.</td>
<td></td>
</tr>
<tr>
<td>knceal(l)</td>
<td>Concealment of tank (1=FD, 2=HD, 3=FE).</td>
<td></td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>True if firer i is on target j.</td>
<td></td>
</tr>
<tr>
<td>tfirst(m,nrg)</td>
<td>( t_{m,n} )</td>
<td>Median time to fire first round for side m an nth range (sec).</td>
</tr>
<tr>
<td>rann(0.5)</td>
<td>( N ) Draw from the standard normal distribution with ( \sigma = 0.5 ).</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>( \Delta t = t_f e^{N} )</td>
<td>Time to fire first round (s).</td>
</tr>
<tr>
<td>prevrd</td>
<td>1=1st shot, 2=prev rd hit, 3=sensed miss, 4=lost miss.</td>
<td></td>
</tr>
<tr>
<td>nrib(l)</td>
<td>Number of rounds fired in burst.</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>ID of tank’s target.</td>
<td></td>
</tr>
</tbody>
</table>
SUBROUTINE HALT (t, I)

Halt: simulate tank halting.
include 'common.h'
logical cango, threat

format (f8.2,1x,a4,3, ' halts',12x,'(x=',f8.1, ' y=',f8.1,')')

if (trace) print *, >halt
if(inviab.eq.1)call cancel (I,'vanish',NULL)
m = army(I)
call path (I,motion(I),0.0,x,y,vx,vy)
if (history) print 1, t, color(I), I, x, y
motion(I) = STATNY

See if fire is a halt-to-fire-system and can still shoot
IF (ishtfs(I).eq.1 .and.
life(I).lt.FKILL .and. nr(I).lt.nrds(I)) THEN
This is a halt-to-fire system. schedule firing if target is
still available.
if (nrtd(I).eq.FLSTGT) threat = knceal(I).ne.FD
if (nrtd(I).gt.0) threat = fot(I,nrtd(I))
IF (threat) THEN
  dt = tfire(I,marg)*exp(rann(0.5))
  dt = amax(1.01,dt/3.0)
  prev rd(I) = 1
  nrib(I) = 0
  it-nrtd(I)
call skedul (t+dt, I, 'fire ', it)
ELSE
Move firer if target has vanished.
if(cango(I,t))call skedul (t, I, 'accel ', NULL)
ENDIF
ENDIF

Move firer because it's target has vanished.

END

Move firer if tgt vanished may be redundant.
END
11.4 Accel: Begin Acceleration. Accel simulates acceleration of the tank. Now that the tank can move, it can vanish. So the code first schedules a vanish event. Then it treats acceleration, depending on the previous motion of the tank. If the tank was slowing up the code takes the first branch. In this branch it may print a line for the event history. Then it calls path to find the current position and velocity. It next finds the time to reach combat cruise speed (dt):

\[ dt = \frac{(\text{speed} - |v_y|)}{\text{accel}} \]

Where,
- speed is the combat cruise speed,
- \( v_y \) is the current velocity, and
- accel is the acceleration of the tank.

Finally, motion of the tank is set to accelerating.

If the tank was halted the code takes the second branch. This branch is identical to the first branch, except that the current speed is zero.

If the tank was already accelerating or at combat cruise velocity, the code takes the third or fourth branch and does nothing except perhaps print a line for the event history.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td></td>
<td>ID of tank.</td>
</tr>
<tr>
<td>life(I)</td>
<td>1 if ALIVE, 2 if M-killed, etc</td>
<td></td>
</tr>
<tr>
<td>invisb</td>
<td>1 if terrain, 2 if smoke causes NLOS.</td>
<td></td>
</tr>
<tr>
<td>kneal(I)</td>
<td>1 if FD, 2 if HD, 3 if FE.</td>
<td></td>
</tr>
<tr>
<td>n</td>
<td>Side of tank (1=Blue, 2=Red).</td>
<td></td>
</tr>
<tr>
<td>motion(I)</td>
<td>1=Slowing, 2=stationary, 3=accelerating, 4=cruising.</td>
<td></td>
</tr>
<tr>
<td>x,y</td>
<td>Coordinates of tank (m).</td>
<td></td>
</tr>
<tr>
<td>vx,vy</td>
<td>Velocity of tank (m/s).</td>
<td></td>
</tr>
<tr>
<td>speed</td>
<td>Cruise speed (m/s).</td>
<td></td>
</tr>
<tr>
<td>accel(i)</td>
<td>Acceleration for side mn (rn/C).</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>( \Delta t = \frac{(s-v)}{a} )</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>Time to reach cruise speed from current velocity (sec).</td>
<td></td>
</tr>
<tr>
<td>dt</td>
<td>Time to reach cruise speed from halt (sec).</td>
<td></td>
</tr>
</tbody>
</table>

SUBROUTINE ACCELF (t, I)
if (trace) print *',<accel'
SUBROUTINE ACCELF (t, I)
if (trace) print *',<accel'
if (life(I).eq.FKILL and invisb.eq.1 and kneal(I).eq.FD)
call schedul(t.l 'vanish' NULL)
if (motion(I).eq.SLOWNG) THEN
if (history) print 1, t, color(n), 1, 'was slowing'
call path(t,motion(I),0.0,x,y,vx,vy)
dt = (speed(n)-abs(vy))/accel(n)
call schedul(t+dt,l,'maxvel'NULL)
motion(l) = ACCELG
ELSEIF (motion(I).eq.SLOWNG) THEN
if (history) print 1, t, color(n), 1, 'was slowed'
call schedul(t+dt,l,'maxvel'NULL)
motion(l) = ACCELG
ELSEIF (motion(I).eq.STATNY) THEN
if (history) print 1, t, color(n), 1, 'was halted'
call path(t,motion(I),0.0,x,y,vx,vy)
c schedule time full velocity reached (maxvel)
dt = speed(n)/accel-army(I)
call schedul(t+dt,l,'maxvel'NULL)
motion(l) = ACCELG
ELSEIF (motion(I).eq.ACCELG) THEN
if (history) print 1, t, color(n), 1, 'was speeding up'
ELSEIF (motion(I).eq.MAXVL) THEN
if (history) print 1, t, color(n), 1, 'is cruising'
ENDIF
11.5 MaxVel: Simulate Tank Reaching Combat Cruise Speed. Max vel simulates the tank reaching combat cruise speed. The routine may print a line for the event history. Then it calls path to update the position and velocity of the tank. It sets motion to MAXVEL, and checks to see if the firer has a target. Then the code calls engage.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of tank.</td>
</tr>
<tr>
<td>motion(I)</td>
<td>1=slowing, 2=halted, 3=accelerating, 4=cruising</td>
</tr>
<tr>
<td>it</td>
<td>ID of tank's target.</td>
</tr>
</tbody>
</table>

```fortran
SUBROUTINE MAX VEL(t, I)
  include 'common.h'
  format (f8.2,1x,a4,i3,' at full speed. ')
  if (trace) print *, '>maxvel'
  if (histry) print 1, t, color(army(I)), 1
  call path(I,motion(I),0.0,x,y,vx,vy)
  motion(I) = MAXVL
  it = ntgt(I)
  IF (it.eq.FLSTGT) THEN
    call engage(t,t,I,it)
  ELSEIF (life(it).lt.(KILL) THEN
    call engage(t,t,I,it)
  ENDIF
  if (trace) print *, '<maxvel'
END
```

V7.2
11.6 Path: Find Position and Velocity of Combatant. Path returns the position and velocity of a single tank. Path uses a table that looks like this:

<table>
<thead>
<tr>
<th>Tank</th>
<th>( t_0 )</th>
<th>( x_0 )</th>
<th>( y_0 )</th>
<th>( v_{x_0} )</th>
<th>( v_{y_0} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NN</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

When it is called to return the position and velocity of a single tank, it checks to see if the data in the table is sufficiently recent. If the data is obsolete, path updates it. In either case, Path copies the data to the output arguments: \( x \), \( y \), \( vx \), and \( vy \).

The data in the table is obsolete if a) the tank is on the attacking side, b) the tank still has mobility, and c) the time since the data was last updated is greater than \( \delta t \). For purposes of this routine, defenders and tanks in a meeting engagement don't move, so their position and velocity never needs to be updated. When attackers are mobility like, of course, so new data is updated, so it doesn't have to be updated later for those tanks. This means path must be called just before the program changes life(I) to indicate a mobility kill or worse. The last update time for the \( i \)th tank is \( t_0(I) \) and the current time is \( t \), so the time since last update is: \( \Delta t = t - t_0(I) \). If this is greater than \( \delta t \) then the data is old. If the data needs to be updated, the code stores the current time \( t \) as the update time \( t_0(I) \) and then branches depending on the current motion of the tank.

If the tank is slowing the code finds the velocity change, the new position, and the new velocity as shown in the chart below.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>decel(n)</td>
<td>( a )</td>
<td>Deceleration of tank (m/s**2).</td>
</tr>
<tr>
<td>( dt )</td>
<td>( \Delta t )</td>
<td>Time since last update (s).</td>
</tr>
<tr>
<td>( dv )</td>
<td>( \Delta v = a \Delta t )</td>
<td>Velocity change (m/s).</td>
</tr>
<tr>
<td>( dv )</td>
<td>( \Delta v = -\Delta v )</td>
<td>Change sign because Red moves in negative direction.</td>
</tr>
<tr>
<td>( x0(I) )</td>
<td>( x )</td>
<td>Old x coordinate (m).</td>
</tr>
<tr>
<td>( vx0(I) )</td>
<td>( v )</td>
<td>Old velocity (m/s).</td>
</tr>
<tr>
<td>( x0(I) )</td>
<td>( x' = \Delta t(v - \Delta v/2) )</td>
<td>New x coordinate (m).</td>
</tr>
<tr>
<td>( v )</td>
<td>( v' = v - \Delta v )</td>
<td>New velocity (m/s).</td>
</tr>
<tr>
<td>(</td>
<td>v</td>
<td>&lt; 0 )</td>
</tr>
<tr>
<td>( v'' = v' )</td>
<td></td>
<td>Save new velocity in table.</td>
</tr>
</tbody>
</table>

If the tank is stationary, the speed is updated to zero.

If the tank is accelerating the code finds the velocity change, the new position, and the new velocity as shown in the chart below.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( dv )</td>
<td>( \Delta v = -\Delta v )</td>
<td>Change sign because Red moves in negative direction.</td>
</tr>
<tr>
<td>( vx0(I) )</td>
<td>( v )</td>
<td>Old velocity (m/s).</td>
</tr>
<tr>
<td>( x0(I) )</td>
<td>( x' = \Delta t(v - \Delta v/2) )</td>
<td>New x coordinate (m).</td>
</tr>
<tr>
<td>( v )</td>
<td>( v' = v - \Delta v )</td>
<td>New velocity (m/s).</td>
</tr>
</tbody>
</table>
If the tank is at cruise speed the code finds the new position and velocity as shown in the chart below.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>( x_0(I) )</td>
<td>( x' = x + v \Delta t )</td>
<td>New x coordinate (m).</td>
</tr>
<tr>
<td>( v_0(I) )</td>
<td>( v' = s )</td>
<td>Current velocity is combat cruise speed (m/s).</td>
</tr>
</tbody>
</table>

```c
V7.3
SUBROUTINE PATH (t, motio2, delt, x, y, vx, vy)

Path: search path table for position and vel at time t.
include 'common.h'
logical is atkr, kan go, old

if (trace) print *, '>path'

n = army(I)
is atkr = (scene.eq.RATTAK .and. n.eq.RED).
1 (scene.eq.BATTAK .and. n.eq.BLU)
kan go = (motio2.ne.STATNY .or.
life(I), eq.ALIVE .or. life(I), eq.FKILL)
dt = t-t0(I)
old = dt .gt. delt
IF (is atkr .and. kan go .and. old) THEN

Update positions and velocity.
t0(I) = t
IF (motio2.eq.SLOWNG) THEN
dv = decel(n)*dt
if (n.eq.RED) dv=dv
x0(I) = x0(I)+dt*(v0x(I)-0.5*dv)
v = vx0(I)-dv
if (abs(v).lt.0.001) v = 0.0
v0x(I) = v
ELSEIF (motio2.eq.STATNY) THEN
v0x(I) = 0.0
ELSEIF (motio2.eq.ACCELG) THEN
dv = accel(n)*dt
if (n.eq.RED) dv=dv
x0(I) = x0(I)+dt*(v0x(I)+0.5*dv)
v0x(I) = vx0(I)+dv
ELSEIF(motio2.eq.MAXVL) THEN
x0(I) = x0(I)+vx0(I)*dt
v0x(I) = speed(n)
if (n.eq.RED) v0x(I)=vx0(I)
ELSE
print *, 'PATH: no such motion. motio2=', motio2
STOP
ENDIF
ENDIF
y=y0(I)
vx=vx0(I)
vy=0.0
if (trace) print *, '<path'
END
```
11.7 Rgf: Find Range to Target, Relative Position, and Velocities

Rgf finds the position and velocity of the firer and target. It then finds the position of the firer with respect to the target, the range, the range band number, and the rounded range.

The code calls path to get the position and velocity of the firer and its target. Then it stores the difference in positions in the s array and finds the range between them (temp). Next it finds the range band (nrg) and the range corresponding to that band (rg). Finally, it may print the positions and velocities if the appropriate debug flag is set.

Why find the range band? The program uses numerous tables. Some of them give values as a function of range, for example, accuracy as a function of range. If the program uses the nearest value in a table, it requires the index of the value. Table 19 below shows the relationship between target range and the range band index (nrg). If the target range is in the range bands shown in row 1, the nrg takes the values shown in row 2, and the range is assumed to be the values shown in row 3.

Table 19. Range Bands

<table>
<thead>
<tr>
<th>Range band</th>
<th>0-750</th>
<th>750-1250</th>
<th>1250-1750</th>
<th>1750-2250</th>
<th>2250-2750</th>
<th>2750-3250</th>
<th>3250-3750</th>
<th>3750-4250</th>
</tr>
</thead>
<tbody>
<tr>
<td>nrg</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>Range used</td>
<td>500</td>
<td>1000</td>
<td>1500</td>
<td>2000</td>
<td>2500</td>
<td>3000</td>
<td>3500</td>
<td>4000</td>
</tr>
</tbody>
</table>

CODE

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
</tr>
<tr>
<td>xf,yf</td>
<td>Coordinates of firer (m).</td>
</tr>
<tr>
<td>xt,yt</td>
<td>Coordinates of target (m).</td>
</tr>
<tr>
<td>vfx,vfy</td>
<td>Velocity of firer (m/s).</td>
</tr>
<tr>
<td>vtx,vty</td>
<td>Velocity of target (m/s).</td>
</tr>
<tr>
<td>s(3)</td>
<td>Relative position of target w.r.t to the firer (m).</td>
</tr>
<tr>
<td>nrg</td>
<td>Range band</td>
</tr>
<tr>
<td>rgf</td>
<td>Range to target from firer (m).</td>
</tr>
<tr>
<td>rg</td>
<td>Rounded range to target from firer (m).</td>
</tr>
</tbody>
</table>

V7.4

FUNCTION RGF (t, I, it)

Rgf: find the position of the firer w.r.t. the tgt.

include 'common.h'
commom /pathc /
save /pathc /
1 format (9x,'Firer x, y, vx, vy =',4f10.1/)
   * 9x,'Target x, y, vx, vy =',4f10.1)

   if (trace) print *:':'>rgf'
call path (It,motion(I),0.0.xf,yf,vfx,vy)
call path (it,t,motion(it),0.0.xt,yt,vtx,vty)
s(1) = xf-xt
s(2) = yf-yt
s(3) = 0.0
temp = sqrt(s(1)**2+s(2)**2)
nrg = max0(1,int(0.5+temp/rgincr))
rgf = temp
rg = irginc*nrg
if (keym(20).gt.0) print 1.
   * xf,yf,vfx,vy,xt,yt,vtx,vty
if (trace) print *:':'<rgf'
END
11.8 CanGo: Find if Tank is Stopped but Mobile. Can go finds whether the tank 'can go'. If the tank is an attacker and is mobile and is either stationary or slowing down, then it 'can go'.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>scene</td>
<td>1=Meeting, 2=Blue attack, 3=Red attack.</td>
</tr>
<tr>
<td>isatk</td>
<td>True iff firer is moving toward foes.</td>
</tr>
<tr>
<td>malive</td>
<td>True iff firer mobile.</td>
</tr>
<tr>
<td>faster</td>
<td>True iff firer is able to go faster.</td>
</tr>
<tr>
<td>cango</td>
<td>True iff firer can go faster than it is going now.</td>
</tr>
</tbody>
</table>

LOGICAL FUNCTION CAN GO (I, t)

Can go: True iff is stationary and can move.
include 'common.h'
logical is atkr, malive, faster

m = army(I)
is atkr = (m.eq.BLU and scene.eq.BATTAK) .or. (m.eq.RED .and. scene.eq.RATTAK)
m alive = life(I).eq.ALIVE .or.
1 life(I).eq.FKILL
faster = (motion(I).eq.STATNY .or. motion(I).eq.SLOWNG)
cango = is atkr .and. malive .and. faster
END
12. OBSCURATION ROUTINES

The obscuration routines model the effects of terrain or smoke on line-of-sight (LOS). The model does not handle the combined effects; only one or the other. When terrain effects are modeled, breaks in and restoration of LOS depends on the distance traveled by the attackers. When smoke effects are modeled, breaks in and restoration of LOS depends on time elapsed. In either case, targets appear then vanish, then appear again in a cycle that ends only when the target is mobility killed or moves beyond viewing range.

The diagram below shows the relationship between the obscuration routines. Initially, targets are assumed to be masked if smoke is used and are assumed to be in view if terrain is used. The important relationship is the cycling between the vanish and appear events. Appear calls AprSmk or AprTer as appropriate and these routines turn on the search cycle IFF it has been turned off. (The code models search only if detection is possible.)
12.1 RdSmk: Read Intervisibility Data for Smoke. Rd smk reads smoke intervisibility data. This data consists of cumulative distributions of in-view and out-of-view segment lengths and their probabilities.

Tank Wars simulates breaks in line-of-sight (LOS) caused by intermittent terrain or smoke but not both. If smoke is used, a special file must be created which contains 6 tables. They consist of the following data:

1. First out-of-view for IR band sensors (Thermal viewers).
2. First out-of-view for visual band sensors (eyes, binoculars, periscopes).
4. Subsequent out-of-view for visual band sensors.
5. In-view for IR band sensors.
6. In-view for visual band sensors.

Each table has 21 rows with 5 entries in each row as shown in Table 20. The program draws a random number and finds the range from the sensor to the target and does a 2-way linear interpolation in the table to find the time the target will be in-view. For in-view data, the time in view increases as range decreases. The opposite occurs for out-of-view data; time out of view increases as range increases.

<table>
<thead>
<tr>
<th>Table 20. Time In-View for a Visual Band Sensor</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prob</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>0.00</td>
</tr>
<tr>
<td>0.05</td>
</tr>
<tr>
<td>0.10</td>
</tr>
<tr>
<td>0.15</td>
</tr>
<tr>
<td>0.20</td>
</tr>
<tr>
<td>0.25</td>
</tr>
<tr>
<td>0.30</td>
</tr>
<tr>
<td>0.35</td>
</tr>
<tr>
<td>0.40</td>
</tr>
<tr>
<td>0.45</td>
</tr>
<tr>
<td>0.50</td>
</tr>
<tr>
<td>0.55</td>
</tr>
<tr>
<td>0.60</td>
</tr>
<tr>
<td>0.65</td>
</tr>
<tr>
<td>0.70</td>
</tr>
<tr>
<td>0.75</td>
</tr>
<tr>
<td>0.80</td>
</tr>
<tr>
<td>0.85</td>
</tr>
<tr>
<td>0.90</td>
</tr>
<tr>
<td>0.95</td>
</tr>
<tr>
<td>1.00</td>
</tr>
</tbody>
</table>

The actual data file is free format, with numbers separated by blanks or commas.

Figure 9 below shows a plot of the data. When the tanks are 1000 meters apart, 20% of the time LOS exists only momentarily seconds. Only rarely does it exist for 200 seconds or more.
Figure 9. Probability LOS exists for t seconds or more.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>toutv1(21,5)</td>
<td>Table of out-of-view segment lengths for infrared</td>
</tr>
<tr>
<td>toutl(21,5)</td>
<td>Table of out-of-view segment lengths for visible band</td>
</tr>
<tr>
<td>toutv(21,5)</td>
<td>Table of out-of-view segment lengths for visible band</td>
</tr>
<tr>
<td>tini(21,5)</td>
<td>Table of in-view segment lengths for infrared band</td>
</tr>
<tr>
<td>tinv(21,5)</td>
<td>Table of in-view segment lengths for visible band</td>
</tr>
<tr>
<td>ptbl(21)</td>
<td>Vector of probabilities.</td>
</tr>
<tr>
<td>rtbl(5)</td>
<td>Vector of ranges (m)</td>
</tr>
</tbody>
</table>

c V1.3
SUBROUTINE RDSM (fname)
c
\%d smk: Read intervisibility data for smoke.
character*32 fname
include 'common.h'
common .smkel: toutl(21,5), toutv1(21,5), toutl(21,5),
toutv(21,5), tini(21,5), tinv(21,5), ptbl(21), rtbl(5)
   open * , file=fname, status=old')
   rewind 4
   print * , 'Smoke causes intervisibility.'
   read * , ((toutl(i,j),j=1,5),i=1,21)
   read * , ((toutv1(i,j),j=1,5),i=1,21)
   read * , ((toutv(i,j),j=1,5),i=1,21)
   read * , ((tini(i,j),j=1,5),i=1,21)
   read * , ((tinv(i,j),j=1,5),i=1,21)
   read * , ((ptbl(i,j),j=1,5),i=1,21)
   close(4)
END
12.2 Smoke: Find When Smoke Will Stop Blocking LOS Between Searchers and Targets.

At the beginning of each engagement init calls smoke. For each Blue/Red pair of tanks, smoke finds when each will appear to the other. It then schedules the target to appear for the searcher at that time.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>For all Blue/Red pairs a random number is drawn</td>
</tr>
<tr>
<td>r</td>
<td>Opening range.</td>
</tr>
<tr>
<td>dt</td>
<td>Time smoke blocks LOS between searchers and target (sec)</td>
</tr>
</tbody>
</table>

For each searcher/target pair, the code draws a random number from the standard uniform distribution. It finds the opening range and viewer type for each side. Then it does a 2-dimensional interpolation in the table for the appropriate viewer. The code uses the range and random number to find the time of appearance. If both sides use the same type of viewer, the searcher and target will appear to each other simultaneously.

For details of the interpolation routine, see the section discussing the utility routine tdintp.
12.3 Appear: Simulate or Reschedule an Appear Event. Appear simulates the re-establishing line-of-sight between a target and one or more searchers.

The overall structure of the routine is as follows:

IF (simulating terrain) THEN
    simulate appearance from behind terrain
    find distance tank has traveled
    IF (tank has traversed entire out-of-view distance) THEN
        treat appearance of the tank
    ELSE
        reschedule appearance
    ENDIF
ELSE
    simulate appearance out of smoke
    IF (both sides use similar viewing devices) THEN
        Treat appearance and vanishing for both sides
    ELSE
        Treat appearance for searcher only
    ENDIF
ENDIF

The first branch treats terrain intervisibility. The code checks to see if the tank is stationary and stops the simulation if it is. (This is a redundant check but can be useful if the code that treats hiding behind terrain is altered.) Then the code finds how far the tank has traveled since it vanished. A call to `path` produces the position of the tank. If the tank has traversed the entire out-of-view segment length it’s ready to appear, otherwise `appear` will be rescheduled.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
<td></td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
<td></td>
</tr>
<tr>
<td>l</td>
<td>ID of firer</td>
<td></td>
</tr>
<tr>
<td>armyt</td>
<td>Side of target. (=1 for Blue, 2 for Red)</td>
<td></td>
</tr>
<tr>
<td>armyf</td>
<td>Side of firer. (=1 for Blue, 2 for Red)</td>
<td></td>
</tr>
<tr>
<td>invisb</td>
<td>=1 if terrain modeled. 2 if smoke modeled</td>
<td></td>
</tr>
<tr>
<td>speed(n)</td>
<td>v,</td>
<td>Combat cruise speed of tanks on side armyt (m/s).</td>
</tr>
<tr>
<td>x,y</td>
<td>Current position of target (m).</td>
<td></td>
</tr>
<tr>
<td>xold, yold</td>
<td>x_o, y_o</td>
<td>Position of target when it vanished (m). If target appears, these will become position of target when it appears.</td>
</tr>
<tr>
<td>travel</td>
<td>d = \sqrt{((x-x_o)^2+(y-y_o)^2)}</td>
<td>Distance traveled while masked (m).</td>
</tr>
<tr>
<td>dist(it)</td>
<td>d_i</td>
<td>Length of out-of-view segment (m).</td>
</tr>
<tr>
<td>iseg(it)</td>
<td># of current segment (indexes vector of alternating in view and out-of-view segments.</td>
<td></td>
</tr>
<tr>
<td>dist(it)</td>
<td>d = d_{seg,i}</td>
<td>In view segment length (m).</td>
</tr>
<tr>
<td>dt</td>
<td>\Delta t = d_{i}/v_n + 0.01</td>
<td>Time to next vanish (sec).</td>
</tr>
<tr>
<td>dt</td>
<td>\Delta T = (d_{i}-d)/v_n + 0.01</td>
<td>Remaining time to reach end of out-of-view segment (sec).</td>
</tr>
</tbody>
</table>

The second branch treats smoke intervisibility. The code checks to see if both sides are using the same kind of viewer. If so, the code restores LOS for both simultaneously. There’s a fine point here. The code schedules `appear` for both but to re-establish LOS simultaneously, it has to discard the `appear` event for one and treat it for both when the other occurs. So when the `appear` occurs for Blue, the code finds the next time in-view and out-of-view for both systems. It the schedules the next `vanish` and next `appear` for both.
If the systems are using different viewers, the code for the less effective viewer must re-establish LOS after and lose it before the more effective viewer. For this reason, the less effective visual band viewer triggers the next in view and out-of-view times for both systems.

Figure 10 shows how in view and out-of-view time segments overlap for two viewers. Neither viewer has line-of-sight through the middle of a smoke cloud where the density is highest. However, nearer the edges of the smoke cloud, where the smoke is less dense, the IR viewer has line-of-sight while the visual band viewer still has no line-of-sight.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>d1</td>
<td>$\Delta_1$</td>
<td>Random in view time for visual (sec).</td>
</tr>
<tr>
<td>d2</td>
<td>$\Delta_2$</td>
<td>Random out-of-view time for visual (sec).</td>
</tr>
<tr>
<td>d4</td>
<td>$\Delta_4$</td>
<td>Random out-of-view time for IR (sec).</td>
</tr>
<tr>
<td>d3</td>
<td>$\Delta_3 = \Delta_1 + 0.5(\Delta_2 - \Delta_4)$</td>
<td>In view time for IR (sec).</td>
</tr>
</tbody>
</table>

IR band

\[
\begin{array}{c|c|c}
\text{in view} & \text{out-of-view} & \text{out-of-view} \\
\hline
t & t + \Delta_3 & t + \Delta_3 + \Delta_4 \\
\end{array}
\]

visual band

\[
\begin{array}{c|c|c}
\text{in view} & \text{out-of-view} & \text{out-of-view} \\
\hline
t & t + \Delta_1 & t + \Delta_1 + \Delta_2 \\
\end{array}
\]

---

Figure 10. Overlap of Visibility Segments for Disparate Viewers
12.4 Aprsmk: Simulate Target Appearing from Behind Smoke. When a target appears from behind smoke, apr smk restores the line-of-sight from the firer to the target (but not from target to firer.) If search has been de-activated, apr smk re-schedules it.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>n</td>
<td>Side of target (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>knceal(it)</td>
<td>Concealment of target (Reset to HD or FE).</td>
</tr>
<tr>
<td>nblu, nred</td>
<td>Number of Blue, Red combatants.</td>
</tr>
<tr>
<td>los(i,j)</td>
<td>True IFF i has line of sight to j.</td>
</tr>
<tr>
<td>repeat</td>
<td>If false, reset to true and restart search.</td>
</tr>
</tbody>
</table>

c V7.1
SUBROUTINE APRSMK(t,it,I)
include 'common.h'
common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
if (trace) print *,'>aprsmk'
army(it)
 Restore line-of-sight from firer to tgt.
los(l,it) = army(l).ne.n
Turn search on if it is .off
IF (.not.repeat) THEN
 repeat = true.
call skedul(t+.01,0,,'search',NULL)
ENDIF
if (trace) print *,'<aprsmk'
END
12.5 Apter: Simulate Target Appearing from Behind Terrain. The target has just re-appeared from behind terrain. If it’s a defender it will pop-up to hull defilade and if it’s an attacker it will be fully exposed. Apter re-establishes line-of-sight to all targets that are not in full defilade. Then, if search was turned off, it is turned back on.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer (UNUSED).</td>
</tr>
<tr>
<td>jexpos</td>
<td>Exposure of target after re-appearing (HD or FE).</td>
</tr>
<tr>
<td>n</td>
<td>Side of target (1=Blue, 2=Red).</td>
</tr>
<tr>
<td>knceal(it)</td>
<td>Exposure of target.</td>
</tr>
<tr>
<td>los(i,j)</td>
<td>True IFF i has line of sight to j.</td>
</tr>
<tr>
<td>repeat</td>
<td>If false, reset to true and restart search process.</td>
</tr>
</tbody>
</table>

**CODE**

```fortran
SUBROUTINE APTER(t,it,jexpos)
  include 'common.h'
  integer it,firer
  common /terrain/ d(40), xold(20), yold(20), dist(20), iseg(20)
  format(8.2,1x,s4,i3,' apter',1x,'(x-',f8.1,' y-',f8.1,')')
  if (trace) print *,' >apter'
  knceal(it) = jexpos
  Restore all lines-of-sight involving it
  DO 20 i=1,nblu+nred
     IF (knceal(i).ne.FD) THEN
        los(i,j) = army(i),ne.n
     END IF
  20 CONTINUE
  Turn search on if it is off
  IF (.not.repeat) THEN
     call schedul(t+01.0,'search',NULL)
  ENDIF
  if (trace) print *,' <apter'
END
```

V7.1
12.6 Terain: Find Path Lengths Where Attacker is Masked by Terrain. Terain finds the portions of the attacker paths where the attackers are hidden from the defenders by terrain.

**Init** calls this routine at the beginning of each engagement if the scenario is a Blue or Red attack. **Terain** then creates a table $d(40)$ and puts randomly chosen in-view segment lengths in the odd elements of $d$ and out-of-view as shown in figure 11.

<table>
<thead>
<tr>
<th>in</th>
<th>out</th>
<th>in</th>
<th>out</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_1$</td>
<td>$d_2$</td>
<td>$d_3$</td>
<td>$d_4$</td>
</tr>
</tbody>
</table>

Distance

**Figure 11. Alternating In View and Out-of-View Segments**

Later, the code will need to know which segment each attacker is in, the length of the segment, and where the segment began. The DO 30 loop stores this information for each attacker and tentatively schedules a **vanish** for each. (The **vanish** is only tentative because the attacker may stop while traversing the in view segment; it may halt to fire or it may be mobility killed.)

The segment lengths are random variates drawn from Wiebull distributions. The in view segment length is:

$$f = \alpha_1 f^\beta_1$$

and the out-of-view segment length is:

$$f = \alpha_2 f^\beta_2,$$

where

$$f = -\log(ran),$$

and

$ran$ is a draw from the standard uniform distribution.

Sometimes these segment lengths are excessively long so that the attackers are out-of-view at all reasonable engagement ranges, with the result that no engagement occurs. For this reason, the segment lengths are truncated to $30\%$ of the opening range.

### CODE

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ifirst</td>
<td>ID of first attacker.</td>
</tr>
<tr>
<td>last</td>
<td>ID of last attacker.</td>
</tr>
<tr>
<td>d(i)</td>
<td>Length of segment (m)</td>
</tr>
<tr>
<td>rg0</td>
<td>Opening range (m).</td>
</tr>
<tr>
<td>x0(i), y0(i)</td>
<td>Position of attacker (m).</td>
</tr>
<tr>
<td>xold(i), yold(i)</td>
<td>Beginning of segment (m).</td>
</tr>
<tr>
<td>dist(i)</td>
<td>Length of segment (m).</td>
</tr>
<tr>
<td>iseg(i)</td>
<td>Number of segment ith attacker is in.</td>
</tr>
</tbody>
</table>

if (trace) print *, '>terain' find segment length at start of each engagement.

DO 20 = 1, 39, 2

Hunfeld terrain constants

- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = -\log(ran(0.0))$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$

if (trace) print *, '>terain' find segment length at start of each engagement.

DO 20 = 1, 39, 2

Hunfeld terrain constants

- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
- $f = \alpha_1 f^\beta_1$
- $f = \alpha_2 f^\beta_2$
12.7 Vanish: Simulate or Reschedule Vanish Event. Vanish models the disappearance of a target due to smoke or terrain blocking the line of sight. Smoke blocks the line of sight at a definite time. Terrain blocks the line of sight only when the attacker traverses the in view segment, so vanish is only scheduled tentatively for terrain blockage. The code checks to see if the attacker has completed the in view segment. If so, it schedules a subsequent vanish, otherwise it reschedules vanish based on the in view distance left to travel and the combat cruise speed of the attackers.

If smoke causes the target to disappear, the code simply calls vansmk.

If terrain causes the target to disappear, vanish is only tentative. If the attacking tank has completed the in view segment, the code sets up the next appear event and calls vanter to complete the vanish event. To set up the next appear event, the code records the beginning of the out-of-view segment, the segment number, and the length of the segment. It then finds the time to complete the out-of-view segment and schedules an appear event at the end of that time.

<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td></td>
<td>Time (sec).</td>
</tr>
<tr>
<td>l</td>
<td></td>
<td>ID of searcher.</td>
</tr>
<tr>
<td>it</td>
<td></td>
<td>ID of target.</td>
</tr>
<tr>
<td>n</td>
<td></td>
<td>Side target is on. (1 if Blue, 2 if Red)</td>
</tr>
<tr>
<td>invis</td>
<td></td>
<td>1 if terrain blocks LOS, 2 if smoke blocks LOS</td>
</tr>
<tr>
<td>speed(n)</td>
<td>v</td>
<td>Combat cruise speed of target (m/s).</td>
</tr>
<tr>
<td>xold, yold</td>
<td>x,y</td>
<td>Position of target (m).</td>
</tr>
<tr>
<td>travel</td>
<td>$d = \sqrt{(x-x_{old})^2 + (y-y_{old})^2}$</td>
<td>Beginning of in view segment (m).</td>
</tr>
<tr>
<td>dist(it)</td>
<td>$d_t$</td>
<td>Distance traversed in segment (m).</td>
</tr>
<tr>
<td>dt</td>
<td>$\Delta t = d_t/v + 0.01$</td>
<td>Time to travel out-of-view segment if target vanished (sec).</td>
</tr>
<tr>
<td>dt</td>
<td>$\Delta t = (d_t/d)/v + 0.01$</td>
<td>Time to finish traveling in view segment if target paused enroute (sec).</td>
</tr>
</tbody>
</table>

SUBROUTINE VANISH(t, it, l)
0 Vanish: if tgt vanishes treat, otherwise reschedule vanish
include 'common.h'
common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
rss(x,y)=sqrt(x*x+y*y)
c if [trace] print *,'>vanish'
    n = army(it)
    IF (invis.eq.1) THEN
        if(speed(n).le.0.) print *,'VANISH: n.speed=',n,
        speed(n) = speed(n)+.1
    ENDIF
1 speed(n) IF (speed(n).le.0.) STOP
    call path(it,t,motion(it),0.0,xy,vx,vy)
c Terrain causes intervisibility
    travel = rss(x-xold(it), y-yold(it))
    IF (travel.gt.dist(it)) THEN
        c Tgt is now masked by terrain
            xold(it) = x
            yold(it) = y
            iseg(it) = iseg(it)+1
            IF (iseg(it).gt.40) iseg(it)=iseg(it)-40
            dist(it) = dist(it)+0.01
            call vanter(it, l, t)
            dt = dist(it)/speed(n) + 0.01
            call skedul (t+dt,it,'appear',NULL)
        ELSE IF (life(it).eq.ALVIVE) THEN
            c Not yet masked by terrain, so reschedule
                dt = (dist(it)-travel)/speed(n) + 0.01
                call skedul (t+dt,it,'vanish',NULL)
            ENDIF
        ELSE
            c Tgt is now masked by smoke
                call vansmk(t, it, l)
        ENDIF
    ENDIF
if [trace] print *,'<vanish'
END
12.8 Vansmk: Simulate Target Vanishing Behind Smoke. Smoke breaks line-of-sight. If the firer had detected the target, the number of targets detected is decremented. The firer loses the target and the time the firer last shot at the target is reset to zero. This means when the target re-appears, it will be treated as a new target for selection priority purposes. If the firer was busy and this was its latest target, the firer is reset to 'unbusy'. If the firer had selected this target (mot=T or fot=T) it now disengages from it. If the target was slowing down to fire and was fully functional and about to engage the firer, the code cancels his halt and schedules an acceleration event.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec).</td>
</tr>
<tr>
<td>I</td>
<td>ID of firer.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>m</td>
<td>Side of firer (1 if Blue, 2 if Red)</td>
</tr>
<tr>
<td>n</td>
<td>Side of target.</td>
</tr>
<tr>
<td>los(I,it)</td>
<td>True IFF firer has line-of-sight to target.</td>
</tr>
<tr>
<td>see(I,it)</td>
<td>True IFF firer has sight of target.</td>
</tr>
<tr>
<td>tfire(I,it)</td>
<td>Time firer last fired at target (sec). If zero, firer treats it as a brand new target.</td>
</tr>
<tr>
<td>busy(I)</td>
<td>If firer was occupied with this target it isn't any more.</td>
</tr>
<tr>
<td>nrtgt(I)</td>
<td>ID of target firer is occupied with.</td>
</tr>
<tr>
<td>mot(I,it)</td>
<td>IF true, firer has a missile assigned to this target.</td>
</tr>
<tr>
<td>fot(I,it)</td>
<td>If true, firer is about to shoot a gun at this target.</td>
</tr>
<tr>
<td>motion(it)</td>
<td>Target is slowing, halted, accelerating, or cruising.</td>
</tr>
<tr>
<td>life(it)</td>
<td>Status of target.</td>
</tr>
</tbody>
</table>

c V7.3
SUBROUTINE VANSMK(t, it, I)
include 'common.h'
1 format([F.2, 1x, a4, i3, ' LOS to ', a4, i3, ' broken by smoke. '])
c if (trace) print *','vansmk'
  m = army(it)
  n = 3-m
  if (history) print 1, t, color(n), I,
 1 color(m),it
c Cancel line-of-sight between tgt and firer.
    los(I,it) = .false.
    if (see(I,it)) ndet(I) = ndet(I)-1
    see(I,it) = .false.
    tfire(I,it) = 0.0
    if (busy(I).and.nrtgt(I).eq.it) busy(I) = .false.
c Abort firer missile on tgt.
    if (mot(I,it) .or. fot(I,it)) THEN
      call diseng(t,I,it,.true.,.true.)
      if (mot(I,it)) call abort(t,I,it)
    ENDIF
c Accelerate tgt that was halting to fire.
    if (motion(it).eq.SLOWNG .and. life(it).eq.1 .and.
1 fot(it,I)) THEN
      call skedul (t,it,'accel',NULL)
      call cancel (it,'halt',NULL)
    ENDIF
    if (trace) print '*' '<'vansmk'
c NOTE: shouldn't halted tgt accelerate too?
END
12.9 Vanter: Treat Target Vanishing Behind Terrain. The target has now definitely vanished behind terrain. It is marked as being in full defilade, as having no targets, and no detections. All lines of sight to and from it are broken. The target sees no foes and they no longer see it. The last fire times are reset to zero so the target and its foes are treated as new threats when the target reappears. Any missiles are aborted. Foes engaging the target disengage it and if they were halting to fire, they begin to accelerate.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>I</td>
<td>ID of any foe.</td>
</tr>
<tr>
<td>it</td>
<td>ID of target.</td>
</tr>
<tr>
<td>n</td>
<td>Side of target (1=Blue, 2=Red)</td>
</tr>
<tr>
<td>kneeal(it)</td>
<td>Set to full defilade.</td>
</tr>
<tr>
<td>nrtgt(it)</td>
<td>ID of target's target.</td>
</tr>
<tr>
<td>ndet(it)</td>
<td>Number of foes target is aware of.</td>
</tr>
<tr>
<td>nblu, nred</td>
<td>Number of Blue, Red combatants.</td>
</tr>
<tr>
<td>los(i,j)</td>
<td>True IFF i has line of sight to j.</td>
</tr>
<tr>
<td>see(i,j)</td>
<td>True IFF i sees j.</td>
</tr>
<tr>
<td>tfire(i,j)</td>
<td>Time i last fired at j. Zero implies j is a new target.</td>
</tr>
<tr>
<td>fot(i,j)</td>
<td>True IFF i is about to fire a gun at j.</td>
</tr>
<tr>
<td>nchan(it)</td>
<td>Number of busy missile guidance channels for target.</td>
</tr>
<tr>
<td>ifirst</td>
<td>ID of target's first foe.</td>
</tr>
<tr>
<td>last</td>
<td>ID of target's last foe.</td>
</tr>
<tr>
<td>motion(it)</td>
<td>Motion of target.</td>
</tr>
<tr>
<td>life(it)</td>
<td>Status of target. (Alive, m-killed, etc)</td>
</tr>
</tbody>
</table>

c V7.5
SUBROUTINE VANTER(t,it,I)
include 'common.h'
DO 20 i=l,nblu+nred
los(it,i) = false.
los(i,it) = false.
if (see(i,it)) ndet(i )=ndet(i )+1
see(it,i) = false.
see(i,it) = false.
tfire(it,i) = 0.0
tfire(i,it) = 0.0
fot(it,i) = false.
20 CONTINUE

c Abort outgoing missiles
  call abort(it,ALL)
nchan(it) = 0

c Abort incoming rounds & disengage tanks firing at tgt
  ifirst=1
  if (n.eq.i) ifirst = nblu+1
  call newtgt(it,ifirst)
call cancel (it,'fire .NULL)
call cancel (it,'select',NULL)
c Accelerate tgt that was halting to fire.
  IF (motion(it).eq.SLOWNG .and. life(it).eq.1) THEN
    call skedul (t,it,'accel ',.NULL)
call cancel (it,'halt '.NULL)
  ENDIF
if (trace) print ":< vanter END
12.10 Hide: Simulate Tank Hiding. If the tank can move and it is firepower killed, it attempts to hide. It goes into full defilade, relevant lines of sight are broken, its foes disengage it, all events associated with the hidden tank are discarded, and it halts. Since it is no longer involved in the engagement, a check is made to see if the engagement is over.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>it</td>
<td>ID of target</td>
</tr>
<tr>
<td>knceal(it)</td>
<td>Concealment is set to full defilade.</td>
</tr>
<tr>
<td>I</td>
<td>ID of first foe</td>
</tr>
<tr>
<td>last</td>
<td>ID of last foe</td>
</tr>
<tr>
<td>los(i,j)</td>
<td>Line-of-sight is broken (set to .false.)</td>
</tr>
</tbody>
</table>

c V7.2
SUBROUTINE HIDE (t, it)
  c 5 Hide: Simulate tank hiding.
  include 'common.b'
  1 format (f8.2,x,a4,i3,' goes into full defilade.')
  c
  if (trace) print *,'>hide '
  if (histry) print 1, t, color(army(it)), it
  knceal(it) = FD
  c Cancel all activities involving this tgt.
  c except discard rounds-in-flight in the impact routine
  ifirst = 1
  if (it.le.nblu) ifirst=nblu+1
  last = nblu
  if (it.le.nblu) last=nblu+nred
  DO 20 i=ifirst,last
       los(i,it) = false.
       los(it,i) = false.
  20 CONTINUE
  call newtgt (t, ifirst, it)
  call cancel (itall ',NULL)
  call skedul (t,it,'slowup',NULL)
  call deaths(t)
  if (trace) print *,'<hide '
END
12.11 PopDn: Simulate Defender Popping Down to Reload Missile Pods. The defender pops down to bring up another missile pod.

Popdn simply calls vanter.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>t</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>I</td>
<td>ID of defender.</td>
</tr>
</tbody>
</table>

c V7.1
SUBROUTINE POP DN (t,I)
   Pop dn: Have defender pop down to reload
   include 'common.h'
   if (trace) print *, 'pop dn'
   call vanter(t,1,NULL)
   if (trace) print *, '<pop dn'
END
13. TIME ADVANCE ROUTINES

The event routines are: reset, skedul, event, and cancel. Reset can be thought of as resetting the clock, clearing the calendar, or initializing the list of pending events. Skedul inserts an event in chronological order while saving the time, ID of the entity performing the action, type of action, and possibly the ID of the entity receiving the action. Event fetches the next pending event, recovering the time, subject entity, action type, and object entity. Cancel removes zero or more events from the list.
13.1 Event Handling Using Linked Lists. The two major ways of handling events are stepping a fixed time interval and stepping to the next significant event. Stepping to the next significant event (the method discussed here) requires routines to reset (initialize) the data structure, schedule an event, fetch an event, and cancel events. This section, touches on various techniques for handling the event data and then discusses the Linked List technique used by the software in the next four sections.

As a minimum, the model must store the time at which an event will occur, the identity of the entity that will perform the event, and the type of event. It may also be desirable to store the entity receiving the action of the event and other information about the event. If, at the current time \( t \), the program finds that after a delay of 5 seconds, tank 4 may fire at tank 6, this would require a Fortran call as follows:

```fortran
  call skedul (t+5.0,4,'fire..',6)
```

Methods of handling event data. A great many methods have been used for storing and retrieving event data. The simplest is to add an event to the end of a list and when the next event is needed, simply search the list for the event with the smallest time. The next simplest is to insert the event just before the next following event. This requires moving the next and all subsequent events down in the list and is slow. The method used here uses linked lists, so that the events are always sorted chronologically, but records of subsequent events need not be moved. (McCormack discusses eight methods for handling event data applied to 12 problems. None of the eight was fastest for all 12 problems, however the method described here was best for 6 of them.)

The search from the front linear linked-list technique was used in the algorithms in the following sections. The key elements are:

- A set of links
- A pointer to the first idle link
- A pointer to the link containing the next event
- Several auxiliary pointers for manipulating the links

Initially, the 'idle' pointer points to the first available link, which points to the subsequent link, and so on until the last available link, which points to the null link \( \Omega \). The 'next event' pointer points to \( \Omega \) also. When an event is inserted in the list, the algorithm removes the first idle link from the chain of idle links, inserts it chronologically in the chain of active links, and inserts the event data into the link.

Retrieving the next event simply involves copying the data from the first link of the chain of active links, removing the link from that chain, and inserting it at the head of the chain of idle links.

 Cancelling an event is similar, but involves links anywhere in the chain of active links. This implementation stores up to 100 events. Each type of information is stored in an array dimensioned to 100. however a given link consists of the ith element of each array. The arrays are:

<table>
<thead>
<tr>
<th></th>
<th>real when(100)</th>
<th>integer who(100)</th>
<th>character*6 what(100)</th>
<th>integer whom(100)</th>
<th>integer next(100)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time of the event</td>
<td>The entity performing the event</td>
<td>The type of event performed</td>
<td>The entity receiving the event</td>
<td>The pointer to the next link</td>
</tr>
</tbody>
</table>

<p>| | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>18.32</td>
<td>4</td>
<td>fire..</td>
<td>6</td>
<td>31</td>
</tr>
</tbody>
</table>

Figure 12. Contents of a Link
13.2 Reset: Re-initialize the Event List. The Reset subroutine 'resets the clock' to time zero. To do this it rebuilds the linked list of idle events and clears the linked list of active events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo Simulations. Although it was designed for use in combat simulations, it has much broader use. Only the waves routine calls it.

If the single argument to reset is true, the event routines will print out each event as it is scheduled or cancelled; if false, this printing is not done. The subroutine then builds a linked list of idle links, as shown at the top of exhibit 13.2. It also makes a null linked list of active links as shown at the bottom of exhibit 13.2; no events are yet scheduled.

![Figure 13. The Initial Linked Lists](image)

Code.

```c
V7.1
file
parameter (NE=200)
character*6 what
integer who, whom
logical prflag
common /event1/ what(NE)
common /event2/ when(NE), who(NE), whom(NE), next(NE), nxevnt, nxidle, prflag
save /event1/, /event2/
V7.1
SUBROUTINE RESET (prflag)
0 Reset: Initialize the clock to time zero.
include 'clock.h'
logical prflag

prflag = prflag
nxevnt = 0
nxidle = 1
DO 10 j=1,NE
   next(j) = j+1
10 CONTINUE
next(NE) = 0
END
```
13.3 Skedul: Schedule an Event. The Skedul subroutine schedules an event in a linked list of events. It is one of four routines, 'Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo simulations. Although it was designed for use in combat simulations, it has much broader use. The event information stored is; event type, entity that will perform the event, time the event will occur, and perhaps the receiver of the action.

The calling statement. The arguments to the Skedul subroutine tell when, who, what, and whom. That is when will a future (tentative) event occur, which entity will perform that event, what event (activity) will be performed and possibly to whom will that activity be directed. If, for example, the second Blue system will fire 12 seconds in the future then the following statement would appear in the program:

call skedul (t+tf,I,'impact',it)

Where:
- t is the current time,
- tf is the time delay after which the event may occur,
- I is the subject or actor causing the event,
- 'impact' is a 6 character string identifying the type of event, and
- it is an integer identifying the object of the event.

Note that the event must always occur in the future, so in the example, \( tf > 0.0 \).

When skedul is called like this, it inserts the when, who, what, and whom data into a linked list in chronological order with other scheduled events. In this case, the time to fire is the current time 't' plus 12 seconds, the actor is tank 2, the event is indicated by an integer stored in the 'eFIRE' variable, and the target (the whom) is indicated by an integer stored in the 'tgt' variable.

Algorithm. On average, Skedul must traverse half the linked list to find the place to insert the event link. It must also check to see if an idle link is available. If so, it then inserts the new event using these 6 steps, as shown in exhibit 13.3.

1. Store the index of the idle link/event in n.
2. Store the index of the new head of the idle chain in idle.
3. Store the index of the immediately preceding link/event in l.
4. Store the index of the succeeding idle link/event in m.
5. Store the index of the now active link/event in next(l).
6. Store the index of the succeeding link/event in the now active link event in next(n).

![Diagram](image-url)

Figure 14. Scheduling an Event
SUBROUTINE SKEDUL (tI,act,it)

Schedule: Schedule an event for later execution.

include 'clock.h'
character*6 act
format(9x,'skedul ','i3,','a6,i3,' at time','f8.2)

if (prflag) print 1, i, act, it, t
IF (nxidle.eq.0) THEN
  IF storage all used stop
     print 'x' Storage overloaded with too many events.'
     STOP
   ELSE
     Store the event
     Cut storage unit from empties
     n = nxidle
     nxidle = next(nxidle)
     Then find where to insert this event in the event list.
     IF (nxevent.le.0) THEN
       New event is only event
       next(n) = 0
       nxevent = n
     ELSE
       Find where to insert it.
       Point to first 2 events
       l = nxevent
       m = next(l)
     Find where to insert them
       IF (l.ge.when(l)) THEN
         See if between 2 scheduled events.
         Loop till found.
         IF (m.ne.0.and. t.ge.when(m)) THEN
           l = m
           m = next(m)
           GOTO 20
         ELSE
           Splice new event into list
           next(n) = m
           next(l) = n
           ENDIF
         ELSE
           Place new event as most imminent
           next(n) = nxevent
           nxevent = n
           ENDIF
       ENDIF
     ENDIF
   END
   Finally store event info
   when(n) = t
   what(n) = act
   whom(n) = 1
   whom(n) = it
   END
END

1 15
13.4 Event: Find Next Event. The Event subroutine finds the next event to be simulated from a linked list of events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo simulations. Although it was designed for use in combat simulations, it has much broader use.

The Event subroutine is only called when an event is completed and the simulation is ready to execute the next event at the top of the list. One of the 'model' routines called Events is the only routine that calls Event. It is called as follows:

```
call event(I, act, it, t)
```

All four arguments are output from Event and contain the time of the most imminent event, who (which tank) is performing the event, what event is being performed, and whom (which target) is receiving the action. If t, I, act have the values 10.5, 4, 'select' then the current becomes 10.5 seconds and at that time tank 4 attempts to select a target. (The variable 'it' is undefined for this particular event.)

The event routine simply extracts the information for the next event from the first link on the linked list of events and then moves that link to the head of the linked list of idle links. The information extracted is:

- I - the entity performing the event
- act - the event or act
- it - the object of the event (or other useful information)
- t - the time the event occurs

Figure 15 shows the arrangement of the idle and active linked lists before and after the most imminent event is fetched.

![Diagram showing before and after of idle and active linked lists](image)

Figure 15. Selecting the Next Event

**Code.**

```c
V7.2
SUBROUTINE EVENT (I, act, it, t)
   include 'clock.h'
   character*6 act
   
   Fill arguments
   I = who(nxevnt)
   act = what(nxevnt)
   it = whom(nxevnt)
   t = when(nxevnt)
   
   Drop storage unit from active storage chain
   n = nxevnt;
   nxevnt = next(nxevnt)
   
   Add storage unit to inactive storage.
   next(n) = nxidle
   nxidle = n
END
```
13.5 Cancel: Cancel an Event. The Cancel Subroutine cancels an event from a linked list of events. It is one of four routines, Reset, Skedul, Cancel, and Event, that cooperate to handle events in Monte Carlo Simulations. Although it was designed for use in combat simulations, it has much broader use.

Cancel removes zero or more links (events) from the list of scheduled events and places them in the linked list of idle links. This removes the record of these events, so they never occur. The cancel routine is called in the four ways illustrated below:

- call cancel (I,'fire ',it)
- call cancel (I,'all ',it)
- call cancel (I,'all ',NULL)
- call cancel (I,'fire ',NULL)

The first call to cancel cancels any fire events associated with entity I and object it. The second version cancels all events associated with entity I and object it. The third version cancels all events associated with entity I, no matter what is the object of the action. The fourth version cancels all fire events associated with entity I.

Figure 16 shows how the active and idle chains look before and after cancelling the second active event; event y.

Before idle: link a → link b → ... nxevnt: link x → link y → link z → ...

After idle: link y → link a → link b → ... nxevnt: link x → link z → ...

Figure 16. Cancelling an Event

Code.

```
c 0 Cancel: cancel 'act' events for 'I' entity.
c (all events if act="")
c
SUBROUTINE CANCEL (I, act, it)
  c Definitions of local variables:
  c m - pointer to previous event
c n - pointer to current event being considered
  include 'clock.h'
  logical is what, is who, is whom
  character*6 act
  format(9x,'cancel ','i3',' ','a6,i3',' at time','f8.2)
  n = next(n)
  m = 0
  n = nxevnt
  IF (n.ne.0) THEN
    c Continue until n=0
    c is who = I .eq.who(n)
    c is what = act.eq.what(n) or. act.eq.'all'
    c is whom = it.eq.whom(n) or. it.eq.0
    IF (is who .and. is what .and. is whom) THEN
      c Then remove event
      if (print) print I, I, act, it, when(n)
      if (m.eq.0) nxevnt = next(n)
      if (m.ne.0) next(m) = next(n)
      next(n) = nxidle
      nxidle = n
      if (m.eq.0) n = nxevnt
      if (m.ne.0) n = next(m)
    ELSE
      c Don't remove event. Shift to next event.
      m = n
  ENDIF
  GO TO 10
END
```

147
14. OTHER UTILITY ROUTINES

The routines in this section are general purpose routines. They are useful for more than just the simulation of combat. They have few if any common statements and generally are stand alone routines. The exception is that several of the random number routines call the uniform random number generating routine.
14.1. Create: Find Space to Store Bullet Data. Create 'creates' temporary entities. So far, it
is only used to create bullets and missiles. They are created by fire and are destroyed by impact. Actually, it
finds an ID for the entity and allocates space in a linked list to store vital parameters for the entity.

Create manipulates the array a(1000) as shown below.

```
| a_1 | a_{21} | a_i | a_j | a_{1000} |
```

The ID of the temporary entity is the index of the first word storing information for that entity. For
equency, the ID of the first entity created is 21. If 9 values are stored for the entity, they will be stored in
locations 22..30. The next entity created would have ID=31. The second through 20th words of the array
are not used so entity ID 1..20 will not be assigned and cause a conflict with the ID's of the permanent
entities.

When the code creates an entity with n attributes, it finds an unused block of words. If the block is
m words long, it is divided into two blocks of n+1 and m-n-1 words. If a_i is the first word of the first
block and a_j is the first word of the second block, the code sets a_i = ± j. When a_i is negative, the first
block is active. After impact of the round, the round is destroyed by setting a_i = abs(a_i). This tells
create that this block of the a-array is available for use.

When create is searching for an unused block of sufficient length, it checks the current (ith) block
and its successor (jth) block. If a_i and a_j are both positive, they are both unused and create joins them
into one block by setting a_i = a_i + a_j.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>The number of attributes to be stored.</td>
</tr>
<tr>
<td>a</td>
<td>The vector used to store attributes in.</td>
</tr>
<tr>
<td>done</td>
<td>True iff the routine is done; if it finds space to store the attributes in.</td>
</tr>
<tr>
<td>i</td>
<td>The index of the storage space we are currently looking at.</td>
</tr>
<tr>
<td>ient</td>
<td>The index of the storage space where the attributes will be stored. It is also the number that will be used to identify the temporary entity created.</td>
</tr>
<tr>
<td>istart</td>
<td>The starting point for the search. If we get back to istart without a find, we have a storage overload and we error off.</td>
</tr>
<tr>
<td>j</td>
<td>The index of the next storage space. We want to look at it with the possibility of catenating it to the storage space beginning at i.</td>
</tr>
<tr>
<td>nreq</td>
<td>The number of spaces required. It equals the number of attributes plus one word. This one word is used for searching purposes. If it is negative (-abs(m)), that indicates that the next m words are being used to store the m attributes of an entity. If it is positive, then the next m words are available for use.</td>
</tr>
</tbody>
</table>

The amount of storage space is 1000 words. If you want to increase this, change all occurrences of 1000 in this routine and in reset.
SUBROUTINE CREATE (n, lent)

Create: create a temporary entity. (a bullet or mnl)

Note - The amount of storage space is 1000 words. If you want to
change this you'll have to change all occurrences of 1000.

Also note that in .:: set these must be set - a=0, a(1)=1000.,

i=1.

logical trace, history, done
common /trace/ trace, history

1 format ('CREATE: Not enuf space to store',i5, ' attributes.')
2 format ('CREATE: i, j, a(i), a(j) =',2i,2f10.3)

if (trace) print '>', 'create'

c Initialize
done = .false.
istart = i
nreq = n+1

c Find empty space in the a-array
10 IF (.not.done) THEN

c Try next empty space
20 CONTINUE

c Catenate empty spaces if possible

c Find next space (and error off if we're back at start)
j = i+iabs(int(ai))
if (j.gt.1000) j=1
IF ((j.eq.1) .or. (a(i),lt.0) .or. (a(j),lt.0)) THEN

c Test this space for size.
IF (a(i),lt,float(nreq)) THEN

Move to next space.
i = j
if (i.eq.istart) print 1, n
IF (i.eq.istart) STOP
ELSE

c Reserve space.
done = .true.
item = i+nreq
if(a(i).ne.float(nreq))a(item) = a(i),float(nreq)
a(i) = -nreq
lent = i

ENDIF
ELSE

c Do catenation.
a(i) = a(i)+a(j)
if (a(i),gt.0.0, and a(j),gt.0.0) GOTO 20
print 2, i, j, a(i), a(j)
STOP
ENDIF
GOTO 10
ENDIF
if (trace) print '>', 'create'

END

SUBROUTINE CRESET

Creset - Reset variables used by create.

common /store/ a(1000), iholy

parameter (NN=20)

DO 20 i=2,1000
a(i)=0
20 CONTINUE

a(1)=NN
a(NN+1) = 1000-NN

iholy=NN+1

END
14.2. Anglef: Find the Angle Between Two Vectors. The dot product of two vectors is:
\[ \mathbf{a} \cdot \mathbf{b} = ab \cos \theta \]

where, \( \theta \) is the angle between them. So
\[ z = \cos \theta = \frac{\mathbf{a} \cdot \mathbf{b}}{|\mathbf{a}| |\mathbf{b}|} \]

To avoid round off errors, the result is trimmed so that \(-1 \leq \cos \theta \leq -1\). Next, the cosine is taken:
\[ y = \cos^{-1} z \]

Finally, the appropriate sign is attached:
\[ r_3 = a_1b_2 - a_2b_1 \]
\[ z = -\text{sign}(y, r_3) \]

Note that \( \mathbf{a}, \mathbf{b} \) are approximately in the ground plane and that \( r_3 \) is the 3rd component of the cross product. The result is in radians.

c V7.3
FUNCTION ANGLEF (a, b)
c 9 Anglef: find angle between two vectors.
dimension a(3), b(3)
vabsa = sqrt(a(1)**2 + a(2)**2 + a(3)**2)
vabsb = sqrt(b(1)**2 + b(2)**2 + b(3)**2)
dotab = a(1)*b(1) + a(2)*b(2) + a(3)*b(3)
dm = dotab/(vabsa*vabsb)
dm = amin1(1., amax1(-1., dm))
dm = acos(dm)
r3 = a(1)*b(2) - a(2)*b(1)
anglef = -sign(dm, r3)
END
14.3. Confb: Find the 90% Confidence Interval On a Binomial Outcome. Dixon and Massey give the method for finding a confidence interval on a binomial outcome.

<table>
<thead>
<tr>
<th>CODE</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>The number of desirable outcomes.</td>
</tr>
<tr>
<td>np</td>
<td>The number of trials.</td>
</tr>
<tr>
<td>hi, lo</td>
<td>The high and low ends of the confidence interval.</td>
</tr>
<tr>
<td>fail</td>
<td>True IFF a confidence interval cannot be found.</td>
</tr>
</tbody>
</table>

If most outcomes are desirable ones or most outcomes are undesirable or if the sample size is too small, no confidence interval can be found.

c V1.1
SUBROUTINE CONF (p, nr, hi, lo, fail)
Confb: Find the 90% binomial confidence interval.

p - the sample probability.

nr - the sample size.

Reference: Introduction to Statistical Analysis, 3rd edition,
Dixon and Massey, p246.

real lo, n
logical fail

data z/1.645/

c
n = float(nr)
fail = (n*p.lt.5) or. ((n-n*p).lt.5)
IF (.not.fail) THEN
Find confidence interval (sample size is big enough).
s1 = n/(n+z**2)
s2 = 0.5*z**2/n
s3 = (p+0.5/n) * (1.0-p-0.5/n)
s4 = (p-0.5/n) * (1.0-p+0.5/n)
s5 = z**2/(4.0*n)**2
lo = s1*(p-0.5/n+s2+z*sqrt(s3/n+s5))
hi = s1*(p+0.5/n+s2+z*sqrt(s4/n+s5))
ENDIF
END
14.4. Indexx: Find the Index \( j \), Where \( a(j) \leq x < a(j+1) \). To interpolate in tables, use the indexx function. Indexx assumes that the dependent variable is stored in a vector of reals, for example, \( x(1) \ldots x(n) \), in ascending or descending order. Given the arguments \( x \), \( n \), \( x_i \), where \( x \) is an ascending vector, it finds the value \( i \) such that \( x_i \leq z \leq x_{i+1} \) using binary search. If \( z < x_i \) or \( z > x_n \), it returns a zero value for \( i \).

Suppose we wish to linearly interpolate in table 21. We may use the following lines of code, where the second line is a statement function:

```
real f(10), x(10)
fj(xj) = f(i) + (f(i+1)-f(i)) * (xj-x(i)) / (x(i+1)-x(i))
i = indexx(x,10,xj)
y = fj(xj)
```

Table 21. Find an index

<table>
<thead>
<tr>
<th>x</th>
<th>0.0</th>
<th>0.5</th>
<th>1.0</th>
<th>1.5</th>
<th>2.0</th>
<th>2.5</th>
<th>3.0</th>
<th>3.5</th>
</tr>
</thead>
<tbody>
<tr>
<td>f(x)</td>
<td>0.0</td>
<td>0.4</td>
<td>0.81</td>
<td>1.23</td>
<td>1.68</td>
<td>2.10</td>
<td>2.52</td>
<td>2.95</td>
</tr>
</tbody>
</table>

Code.

```c
FUNCTION INDEXX(a, n, x)
integer n, lo, hi, mid
logical incres, above
real a(n), x
incres = a(n).gt.a(1)
lo=0
hi=n+1
10 IF (hi-lo.gt.1) THEN
   mid=(hi+lo)/2
   above=x.gt.a(mid)
   IF (incres.eqv.above) THEN
      lo=mid
   ELSE
      hi=mid
   ENDIF
GOTO 10
ENDIF
indexx=lo
END
```
14.5. Ranu: Draw a Random Number from the Standard Uniform Distribution. This subroutine uses a version of the uran3l uniform random number generator to pseudo-randomly draw a number from the uniform distribution extending from 0 to 1. The following explains how to "seed" the generator and shows some sample draws.

Why use a random number generator coded in Fortran? For the following reasons:

1. First, we believe this is one of the better random number generators. It is based on an algorithm by Pike\textsuperscript{5}. We also recommend the discussion of random number generators by Press\textsuperscript{6} and by Knuth\textsuperscript{7}.

2. If you are transporting a program from one computer to another, and it draws random numbers, you'll be more confident if test cases generate exactly the same results on each machine.

3. If a long run dies in mid-stream, and you've printed the random number seed periodically, you may be able to restart the run at the point it last printed the seed.

4. And finally, if you are debugging a run by turning on more and more print statements, you can suppress enormous volumes of printout by judiciously setting the random number seed and restarting the run in mid-stream.

Input/Output. Ranu requires the calling program to initialize the variable \( j \) in the common statement /crandm/ \( j \). We have used the value \( j=1111111 \), however other odd integers are legal. The common statement may be replaced with a data statement such as: data \( j /1111111/ \) if you do not wish to reset the seed. The calling statement: call ranuo, of course, requires no argument. Figure 17, below illustrates ten draws using ranu, and shows a plot of 20 draws using pairs of variates as the coordinates of the 10 points.

\[
\begin{array}{c}
\text{Seed} = 1111111 \\
\text{First ten draws from the function:} \\
0.7401378 & 0.9308131 \\
0.7908101 & 0.2813551 \\
0.2351466 & 0.8330226 \\
0.1955611 & 0.1284252 \\
0.3288300 & 0.5939014 \\
\end{array}
\]

Figure 17. Ten Pairs of Numbers Drawn from a Uniform Distribution

Mathematics. Ranu is a variant of the uran31 subroutine long used at BRL. It is discussed in the Collected Algorithms of the ACM\textsuperscript{6}. We have tested a number of random number generators and found Ranu to be the only one to pass all 5 tests. Ranu & uran31 will work on any computer with 31 or more bits per integer. Any odd seed between 1 and 67108863 was acceptable for the earliest version. Revisions to accommodate 31 bit machines will have reduced this upper limit and the cycle length. Cycle length of the current version is 18,777,215.
FUNCTION RANU (dm)

Ranu: A version of uran31 uniform random nr generator.

common /crandm/ j
real a1
j=3*25
j=j/(67108864)*67108864
j=3*25
j=j/(67108864)*67108864
j=3*25
j=j/(67108864)*67108864
a1=j
ranu=a1/67108864
END
14.8. **Rann: Draw a Random Number from a Normal Distribution.** Rann draws two random variates from the standard normal distribution using the Box-Muller method.

**Output.** This subroutine generates two real numbers randomly chosen from the normal distribution. Interpreting the output as $x$ and $y$ coordinates, we produced the twenty points drawn in Figure 18, which represent random shots.

![Figure 18. Draw of 20 Random Shots](image)

**Mathematics.** Rann is based on an algorithm by Bell. It produces two independent random variables, each from the normal distribution with mean 0 and standard deviation 1. The subroutine calls the real function twice. RANU is a pseudo-random number generator that produces a number lying strictly between 0 and 1. See section 14.5 for details. Algorithm 334 is a slightly faster, but more complex version of algorithm 357. See also algorithm 412 for a slower, but higher precision algorithm. Finally, see algorithm 148, which may be faster if one of the 2 random deviates must be discarded.

**Code.**

```fortran
SUBROUTINE RANN(p,q)
    REAL p,q
    REAL x,y
    REAL r1,r2

    DO 100 i=1,2
        r1=randi() ! generate a random number
        r2=randi() ! generate a random number
        x=sqrt(-2.*log(r2))*cos(2.*3.1415926535*r1)
        y=sqrt(-2.*log(r2))*sin(2.*3.1415926535*r1)
        p=x*cos(q)
        q=x*sin(q)
    100 CONTINUE
END
```

157
14.7. RndAng: Draw a Random Angle from a Cardioid or Other Distribution. The aspect angle is the angle of an incoming round measured from the nose of the target. Rnd ang chooses an aspect angle for the incoming round by randomly drawing from the cardioid distribution or a more frontally oriented distribution.

Peterson\textsuperscript{1,2} has analyzed the angular distributions of shots on hulls and on turrets during World War II. He found that the distributions were approximately cardioid with the distribution of shots on turrets slightly more tightly grouped for the turret than for the hull. Again, this distribution is not very helpful to the armor analyst or simulation builder because the initial orientations and subsequent motions are not defined, only the final orientations. If we assume a cardioid distribution initially, the straight line motion of the attackers tends to spread out the distribution of impacting rounds on the tanks. Further, the act of pointing the turret at a target tends to pinch the distribution of impacting rounds on the turrets. The net result in simulations is a distortion of the distribution of impact angles on the tanks, but it is not necessarily severe. This is how the initial orientations are chosen for each engagement and if they move this is how they will move.

The cardioid density function is:

\[ p = \frac{(1 + \cos \theta)}{2\pi} \]

The cardioid distribution function is the integral of the density function:

\[ P = \int_{-\pi}^{\theta} p \, d\theta = \frac{(\theta + \sin \theta + \pi)}{2\pi} \).

Both of these are illustrated in figure 19 below.

Figure 19. Cardioid Density and Distribution Functions

A more frontally oriented distribution, often used is given by:

\[ p = \frac{(1 + \cos 3\theta)}{6\pi}, \quad -\frac{\pi}{3} \leq \theta \leq \frac{\pi}{3} \]

\[ P = \frac{(3\theta + 3\cos 3\theta)}{2\pi} \)

There may be no way to analytically solve for \( \theta \) given \( P \) and Newton's method for solving misbehaves, so the code uses binary search. The last 2 equations in the table below are repeated 10 times, each time saving the value of \( \theta \) in \( l_t \) or \( l_s \) as appropriate.
<table>
<thead>
<tr>
<th>CODE</th>
<th>MATH</th>
<th>COMMENT</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pi</td>
<td>$\pi$</td>
<td></td>
</tr>
<tr>
<td>denom</td>
<td>$d = 0.5/\pi$</td>
<td>Random draw from a uniform distribution.</td>
</tr>
<tr>
<td>p</td>
<td>$p = \text{ranu()}$</td>
<td>Lower limit of angular value (rad).</td>
</tr>
<tr>
<td>tlo</td>
<td>$\theta_l = -\pi$</td>
<td>Upper limit of angular value (rad).</td>
</tr>
<tr>
<td>thi</td>
<td>$\theta_h = \pi$</td>
<td></td>
</tr>
<tr>
<td>theta</td>
<td>$\theta = (\theta_l + \theta_h)/2$</td>
<td>REPEAT FOLLOWING LINES 10 TIMES Limit (rad).</td>
</tr>
<tr>
<td>px</td>
<td>$p_x = (\theta + \sin \theta \cdot d)$</td>
<td>True iff $\theta$ is too low.</td>
</tr>
</tbody>
</table>

**FUNCTION RNDANG(iangd)**

Do binary search to find theta associated with random draw

```plaintext
tlo = Pi
if (iangd.gt.1) tlo = -Pi/3.
else
  tlo = Pi/3.
endif

DO 20 i=1,10
theta = 0.5*(tlo+thi)
if (iangd.eq.1) px = (theta + sin(theta) + Pi)*denom
else px = (3.0*theta + sin(3.0*theta) + Pi)*denom
if (px.lt.p) THEN
  tlo = theta
else
  thi = theta
ENDIF

20 CONTINUE
rndang = theta
END```

```plaintext
c V7.2
FUNCTION RNDANG(iangd)
c Rnd ang: Draw a random angle from a cardioid/other distribution.
P1=3.1415926536
denom = 0.5/Pi
p=ranu(dummy)
c Do binary search to find theta associated with random draw
tlo = Pi
if (iangd.gt.1) tlo = -Pi/3.
else
  tlo = Pi/3.
endif
DO 20 i=1,10
theta = 0.5*(tlo+thi)
if (iangd.eq.1) px = (theta + sin(theta) + Pi)*denom
else px = (3.0*theta + sin(3.0*theta) + Pi)*denom
if (px.lt.p) THEN
  tlo = theta
else
  thi = theta
ENDIF

20 CONTINUE
rndang = theta
END```
15. REFERENCES


<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
</table>
| 2             | Administrator  
Defense Technical Info Center  
ATTN: DTIC-DDA  
Cameron Station  
Alexandria, VA 22304-6145 |
| 1             | Commander  
U.S. Army Materiel Command  
ATTN: AMCAM  
5001 Eisenhower Avenue  
Alexandria, VA 22333-0001 |
| 1             | Commander  
U.S. Army Laboratory Command  
ATTN: AMSLC-DL  
2800 Powder Mill Road  
Adelphi, MD 20783-1145 |
| 2             | Commander  
U.S. Army Armament Research,  
Development, and Engineering Center  
ATTN: SMCAR-IMI-I  
Picatinny Arsenal, NJ 07806-5000 |
| 2             | Commander  
U.S. Army Armament Research,  
Development, and Engineering Center  
ATTN: SMCAR-TDC  
Picatinny Arsenal, NJ 07806-5000 |
| 1             | Director  
Benet Weapons Laboratory  
U.S. Army Armament Research,  
Development, and Engineering Center  
ATTN: SMCAR-CCB-TL  
Watervliet, NY 12189-4050 |
| 1             | Commander  
U.S. Army Armament, Munitions and Chemical Command  
ATTN: AMSMC-IMF-L  
Rock Island, IL 61299-5000 |
| 1             | Director  
U.S. Army Aviation Research and Technology Activity  
ATTN: SAVRT-R (Library)  
M/S 219-3  
Ames Research Center  
Moffett Field, CA 94035-1000 |

<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
</tr>
</thead>
</table>
| 1             | Commander  
U.S. Army Missile Command  
ATTN: AMSMI-RD-CS-R (DOC)  
Redstone Arsenal, AL 35898-5010 |
| 1             | Commander  
U.S. Army Tank-Automotive Command  
ATTN: ASQNC-TAC-DIT (Technical Information Center)  
Warren, MI 48397-5000 |
| 1             | Director  
U.S. Army TRADOC Analysis Command  
ATTN: ATRC-WSR  
White Sands Missile Range, NM 88002-5502 |
| 1             | Commandant  
U.S. Army Field Artillery School  
ATTN: ATSF-CSI  
 Ft. Sill, OK 73503-5000 |
| 1             | Commandant  
U.S. Army Infantry School  
ATTN: ATSH-CD (Security Mgr.)  
Fort Benning, GA 31905-5660 |
| 1             | Commandant  
U.S. Army Infantry School  
ATTN: ATSH-CD-CSO-OR  
Fort Benning, GA 31905-5660 |
| 1             | Air Force Armament Laboratory  
ATTN: WL/MNOI  
Eglin AFB, FL 32542-5000  
Aberdeen Proving Ground |
| 2             | Dir, USAMSAA  
ATTN: AMXSY-D  
AMXSY-MP, H. Cohen |
| 1             | Cdr, USAFTECOM  
ATTN: AMSTE-TC |
| 3             | Cdr, CRDEC, AMCCOM  
ATTN: SMCCR-RSP-A  
SMCCR-MU  
SMCCR-MSI |
| 1             | Dir, VLAMO  
ATTN: AMSLC-VL-D |
| 10            | Dir, BRL  
ATTN: SLCBR-DD-T |
<table>
<thead>
<tr>
<th>No. of Copies</th>
<th>Organization</th>
<th>Address/Contact</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>General Dynamics</td>
<td>Land Systems Division</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: David Stremling</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.O. Box 2045</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Warren, MI 48090</td>
</tr>
<tr>
<td>1</td>
<td>General Defense Corporation</td>
<td>Tactical Systems Division</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: Ray Edmondson</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.O. Box 21606</td>
</tr>
<tr>
<td></td>
<td></td>
<td>St. Petersburg, FL 34664</td>
</tr>
<tr>
<td>1</td>
<td>LTV Aerospace and Defense Company</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: C. H. McKinley</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.O. Box 655907</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Dallas, TX 75265-5907</td>
</tr>
<tr>
<td>1</td>
<td>Booz Allen and Hamilton, Inc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: Mike McGinnes</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Suite 1610</td>
</tr>
<tr>
<td></td>
<td></td>
<td>1300 17th St.</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Rosslyn, VA 22209</td>
</tr>
<tr>
<td>1</td>
<td>Military Vehicles Operation</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: Dan Bitz</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.O. Box 420</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mail Code 01</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Indianapolis, IN 46206</td>
</tr>
<tr>
<td>1</td>
<td>Technical Solutions, Inc.</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ATTN: George Ober</td>
</tr>
<tr>
<td></td>
<td></td>
<td>P.O. Box 1148</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Mesilla Park, NM 88047</td>
</tr>
<tr>
<td>1</td>
<td>Mr. Harry Reed</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>338 Carter Street</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Aberdeen, MD 21001</td>
</tr>
</tbody>
</table>

6 Dir, USAMSAA
ATTN: AMXSY-GC,
G. Comstock
L. Harrington
ATTN: AMXSY-GA,
W. Brooks
K. Tarquini
ATTN: AMXSY-GI,
C. Ehrig
E. Walker
This laboratory undertakes a continuing effort to improve the quality of the reports it publishes. Your comments/answers below will aid us in our efforts.

1. Does this report satisfy a need? (Comment on purpose, related project, or other area of interest for which the report will be used.)

2. How, specifically, is the report being used? (Information source, design data, procedure, source of ideas, etc.)

3. Has the information in this report led to any quantitative savings as far as man-hours or dollars saved, operating costs avoided, or efficiencies achieved, etc? If so, please elaborate.

4. General Comments. What do you think should be changed to improve future reports? (Indicate changes to organization, technical content, format, etc.)

BRL Report Number BRL-TR-3292

Division Symbol

Check here if desire to be removed from distribution list. ___

Check here for address change. ___

Current address: Organization
                  Address

DEPARTMENT OF THE ARMY
Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

OFFICIAL BUSINESS

BUSINESS REPLY MAIL
FIRST CLASS PERMIT No 0001, APG, MD
Postage will be paid by addressee

Director
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066