TECHNICAL REPORT BRL-TR-3292

BRL

THE SUSTAINED COMBAT MODEL: TANK WARS II PROGRAMMERS' MANUAL

FRED L. BUNN

NOVEMBER 1991

U.S. ARMY LABORATORY COMMAND

BALLISTIC RESEARCH LABORATORY
ABERDEEN PROVING GROUND, MARYLAND

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE November 1991	3. REPORT TYPE AND DATES COVERED Final, January 1965 - September 1991

4. TITLE AND SUBTITLE
The Sustained Combat Model: Tank Wars II Programmers' Manual

5. FUNDING NUMBERS
PR: 1L162618AH80

6. AUTHOR(S)
Fred L. Bunn

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Ballistic Research Laboratory
ATTN: SLCBR-DD-T
Aberdeen Proving Ground, MD 21005-5066

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
BRL-TR-3292

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

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.

14. SUBJECT TERMS
Tanks (Combat Vehicles), Warfare, Simulation, Armored Vehicles, Materiel

15. NUMBER OF PAGES
163

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT UNCLASSIFIED	18. SECURITY CLASSIFICATION OF THIS PAGE UNCLASSIFIED	19. SECURITY CLASSIFICATION OF ABSTRACT UNCLASSIFIED	20. LIMITATION OF ABSTRACT SAR

INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

TABLE OF CONTENTS (contd)

TABLE OF CONTENTS (contd)

[illegible]

A-1

INTENTIONALLY LEFT BLANK

LIST OF FIGURES

INTENTIONALLY LEFT BLANK

LIST OF TABLES

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

Armament/accuracy
- round reliability
- KE, HEAT, missiles (incl STAFF)
- time of flight
- probability of sensing miss
- S-S accuracy depends on prev round

Driver routines loop through
- scenarios
- opening ranges
- force ratios
- multiple replications
- waves of threat systems
- re-grouping

Fire cycle characteristics modeled
- ammo consumption
- burst fire
- first round time
- subsequent round times
- manual loading, load assist, automatic loading

Intermediate output
- event histories
- logic tracing
- event tracing
- as scheduled, canceled, and retrieved

Measures of effectiveness
- ammo consumption
- blue tanks killed
- red tanks killed
- blue win probabilities
- red win probabilities
- exchange ratios

Miscellaneous characteristics
- multiple kill levels
- pinpoint and non-firing detection
- various target disengagement policies
- model widely used
- 21 levels of tgt priority
- simple terrain

Program design
- event sequenced
- time stepping for detection
- stochastic
- Fortran 77
- modular
- structured
- indented code
- 3900 lines of code

Scenario characteristics modeled
- multiple waves of Red threat systems
- multiple combatants
- false targets
- passive and active decoys

Vehicle characteristics modeled
- disjointed turret & hull
- cardioid or other aspects
- moving targets
- fire on the move
- halt to fire
- gross motion
- full defilade, hull defilade, fully exposed
- red tanks killed

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.

Top Level	Events (cont)	Events (cont)	Model Utility	Time Adv.	Utility
Main	**Event**	**Impact**	**Abort**	**Reset**	**Anglef**
Input	**Finish**	**May hit**	**Diseng**	**Skedul**	**Angsum**
Rd misc	**Finsh2**	**Acc err**	**Newtgt**	**Event**	**Cango**
Pr misc	**Search**	**Acc ssb**	**Selecs**	**Cancel**	**Create**
Rd eror	**Searc2**	**Acc sss**			**Interp**
Rd pkh	**Detect**	**Acc sm**			**Intrp2**
Rd pkh2	**Select**	**Acc ms**			**Nrgf**
Rd pkh5	**Priorn**	**Is hit**			**Ranu**
Waves	**Priort**	**Bounds**			**Rann**
Reset	**Engage**	**Damage**			
Init	**Fire**	**Kill**			
Deploy	**Pinpnt**	**Kill5**			
Det rg	**Frd ssg**	**Damagm**			
Terain	**Frd bst**	**Damagf**			
Smoke	**Frd msl**	**Late kl**			
Searc0	**Reload**	**Hide**			
Events	**Slow up**	**Appear**			
Stats1	**Halt**	**Vanish**			
Stats2	**Accel**				
Stats3	**Max vel**				

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.
Devic2: 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 $a(j) <= x < a(j+1)$.
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 eror: 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.
Selecs: 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.
Terain: 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.

ACCELG	= 3 Identifies tank that is accelerating. integer
ALIVE	= 1 Implies fully functional. integer
ALL	= 0 Used to schedule or cancel an event for all firers or targets. integer
BATTAK	= 3 Blue attack scenario. integer
BLU	= 1 # of blue side. integer
DEG	= 57.29577951 Degrees/radian.
FD	= 1 Full defilade. integer
FE	= 3 Fully exposed. integer
FKILL	= 3 Firepower kill. integer
FLSTGT	=-1 ID # for false targets. integer
HD	= 2 Hull defilade. integer
HULL	= 2 Identifies hull box. integer
IKILL	= 5 M&F kill & recognized to be inactive.
KKILL	= 6 Catastrophic kill.
MAXVL	= 4 Identifies tank moving at maximum combat velocity.
MEETNG	= 1 Meeting scenario.
MFKILL	= 4 Mobility & firepower kill.
MKILL	= 2 Mobility only kill.
NN	= Maximum number of tanks playable.
NULL	= 0 No specific target associated with firer event.
PI	= 3.141592654 (π).
RATTAK	= 2 Red attack scenario. integer
RED	= 2 # of red side. integer
SLOWNG	= 1 Identifies tank that is slowing up. integer
STATNY	= 2 Identifies tank that is stationary (halted). integer
TURRET	= 1 Turret of tank. integer
TWOPI	= 6.283185308 (2π).
a(1000)	See routine **create**. Storage for temporary entities.
accel(2)	Acceleration of ith side (m/s**2).
accmax(2)	Maximum lateral acceleration (m/s**2).
ampl(2)	Amplitude of sinusoidal path (m).
angle	No longer used. Used to input crossing angle of attacker.
army(NN)	Army(i) is 1 if ith tank is Blue and 2 if it is Red. integer
busy(NN)	busy(i)=T iff ith tank is too busy to select a new tgt. logical
chanel(2,NN,5)	chanel(i,j,k) contains ID of missile for ith side, jth tank, kth guidance channel.
color(2)	color(n) is the color of the nth army. character*4
decel(2)	Deceleration of ith side (m/s**2).
empty(NN)	empty(i)=T iff out of ammo or empty missile pod. logical
fot(NN)	fot(i)=T iff ith tank is occupied firing on a target. logical
histry	=T iff event history will be printed. logical

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.
irginc	Range increment for detection, time of flight, tfirst, pkh tables. Usually 500 meters.
ishtfs(2)	Side is halt to fire. logical
keym(20)	=T iff print statement should be executed.
kindrd(2)	Side's kind of round.
knceal(NN)	knceal(i) is 1 if full defilade, 2 if hull defilade, 3 if fully exposed.
kshot(2,20)	Count of shot results.
kview(2)	kview(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. logical
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.
ndecoy(2)	# decoys on ith side.
ndet(NN)	# tgts ith tank has detected.
ndets(2)	# tgts tank on ith side can detect.
neval	# surviving blue tanks with ammo.
nflash(2)	# flashing decoys on ith side.
nhot(NN)	nhot(i) is # hits on ith tank after it's M&F killed.
nipod(NN)	# rounds in missile pod for ith side.
nipods(2)	# rounds in missile pod for ith side.
nprior(2)	nprior(i) = 1 if ith side using priority scheme 1, 2 if using scheme 2.
nrd(NN)	# 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.
nrtgt(NN)	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 HD tgt. pfalse(i,2) is for FE tgt.
pinfin(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.
recknz(2)	recknz(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.
tfirst(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.
touti(21,5)	Table of out-of-view segment lengths for infrared sensors.
touti1(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.
toutv1(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.
vx0(NN)	vx0(i) is the last computed speed of the ith tank. (m/s).
vy0(NN)	Velocity of tank.
wvlth(2)	Wave length of sinusoidal path (m).
x0(NN)	x0(i) is the last computed Easting coordinate of ith tank. (m).
y0(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.

```
c common.new
c V1.3
c      common.h file
       parameter (NN=20)
c Much used:
       character*4 color
       common /charc / color(2)
       common /consts/ PI, TWOPI, DEG
       integer ALL, NULL, FLS TGT,
     1    FD, HD, FE, TURRET, HULL, BLU, RED, MEETNG, RATTAK,
     2    BATTAK, ALIVE, MKILL, FKILL, MFKILL, IKILL, KKILL,
     3    SLOWNG, STATNY, ACCELG, MAXVL
```

```
      common /const2/ ALL, NULL, FLS TGT,
     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 /contrl/ nreps, keym(20), scene, tmax, meth sm
      common /cshot/ kshot(2,20)
      logical trace, histry
      common /ctrace/ trace, histry
      common /n sys/ nblu, nred
      common /states/ army(NN), life(NN), nrtgt(NN)
c Less used:
      common /aspekt/ iangd
      common /tstore/ a(1000), iholy
      common /vars6/ irginc, rgincr
c Vehicle:
      common /endgam/ sysdim(2,8)
      common /state2/ idecoy(NN), iflash(NN), ndecoy(2), nflash(2)
c Round:
      common /round/ kindrd(2), nrds(2), nrd(NN), reliab(2),
     1 nipods(2), nipod(NN), trelod(2), nrpb(2), nrib(NN), tof(2,8)
c Detection:
      logical los, see, repeat
      integer prevrd
      common /xx/ invisb, kview(2), ndets(2), ndet(NN),
     1 pinp(2), tlook(2), los(NN,NN),
     2 knceal(NN), prevrd(NN), rgvis(3,NN), see(NN,NN)
      common /sensor/ psense(2,8), pinfin(2,3,10), pdet(2,3,10),
     1    repeat
c Selection:
      logical busy, fot, mot, share
      common /choose/ busy(NN), fot(NN,NN), mot(NN,NN),
     1 nprior(2),pfalse(2,2),recknz(2),share(2),tfire(NN,NN),tfire2(NN)(2)
c Fire cycle:
      integer chanel
      logical empty
      common /fcycle/ ishtfs(2), loader(2), tfirst(2,8),
     1 rof(2), tvar(2), tcon(2), nbrst(NN), empty(NN),
     2 chanel(2,NN,5), nchans(2), nchan(NN)
c Target discard:
      integer tactic
      common /policy/ tactic(2), nrpt(2), nrot(NN), nhot(NN),
     1 tbump(2),nbump(2)
c Motion:
      common /cpath / accel(2),decel(2),
     1 speed(2), angle(2), accmax(2), wvlth(2), ampl(2),
     2 motion(NN), t0(NN), x0(NN), y0(NN), vx0(NN), vy0(NN)
      common /where2/ nrg, rg0, rg, s(3)
      common /yy/  thide(2)


c V7.4
      BLOCK DATA BLKDAT
      include 'common.h'
      data color,    pi,  twopi,      deg
     1    /'Blue', 'Red ', 3.141592654, 6.283185308, 57.29577951/
      data ALL, NULL, FLS TGT /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/
      dataSLOWNG, STATNY, ACCELG,MAXVL
     1 /1,2,3,4/
      data keym /20*0/
      END
```

INTENTIONALLY LEFT BLANK

2. TOP LEVEL ROUTINES

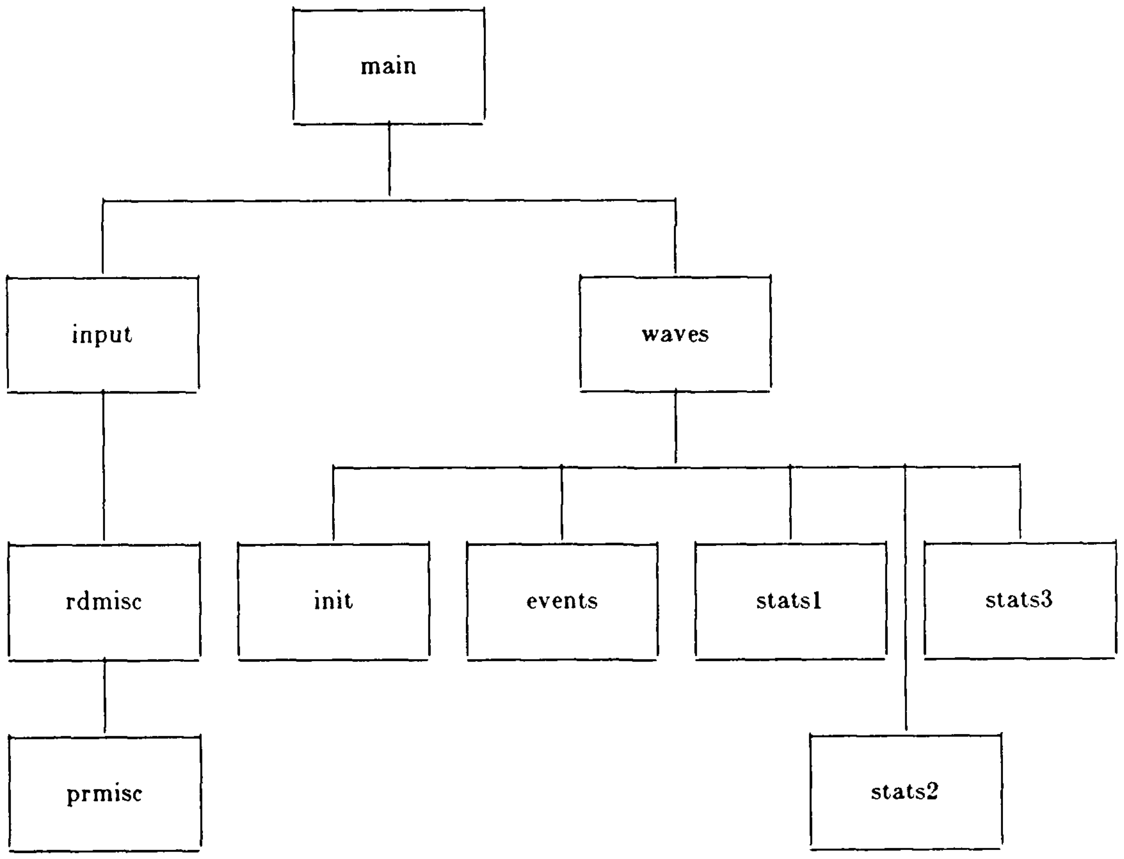
The Tank Wars routines are in a hierarchy. The routines discussed in this section are at the top of the hierarchy. They read input, loop through parameters, and produce summary statistics, rather than dealing with the specifics of a single engagement. **Main**, and **waves** take the burden of changing 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.



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:

Game file	Comment
.	
.	
.	
1000, 3000, 500	min range, max range, range increment
1, 5, 5	scene, #blue, #red
2, 4, 12	scene, #blue, #red
3, 12, 4	scene, #blue, #red

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
c     MAIN ROUTINE
c 9   Main: read input and simulate scenarios.
      character string(3)*11
      include 'common.h'
3     format(' #Blues=',i3,' #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  incrg
20    CONTINUE
        read(5,*,IOSTAT=io) scene, nblu, nred
        if (io.lt.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, incrg
          rg0 = irg
          call waves
30        CONTINUE
        GOTO 20
        END
```

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; **stats1, stats2,** and **stats3.** 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 1000 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 **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 nwaves > 1, multiple waves are being simulated. If the number of replications for the first wave is nreps=1000, and if the number of blue combatants in an engagement is 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:

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

Code.

```
c V7.14
       SUBROUTINE WAVES
c 7    Waves: loop thru waves of red tanks.
       include 'common.h'
       common /crandm/ irandm
       common /inpwav/ keyd(5), nwaves, neval, nused(3000)
       integer kount(2,20)
       logical done
c
       if (trace) print *,'>waves'
       IF (nreps*nblu.gt.3000 .and. nwaves.gt.1) THEN
         print *,'WAVES: Too many reps or blues.',nreps,nblu
       ELSE
c        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
         nreps3 = 0
         nreps2 = nreps
         nsurv = 0
         nwave = 0
40       CONTINUE
c        Loop thru up to n waves of red tanks
           nwave = nwave+1
           iseed2 = irandm
           nrg = rg0/irginc
           DO 50 i=15,20
             kount(2,i) = 0
50         CONTINUE
           DO 60 nrep = 1,nreps2
c          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)
60         CONTINUE
c          Update statistics and see if all waves are done.
             nreps3 = nreps3+nreps2
             call stats2(nwave,nwaves,nreps2,kount)
             done = nsurv.lt.nblu .or. nwave.eq.nwaves
         IF (done) GOTO 80
c        Find #reps, #Blue tanks, #unused Blues for next wave.
           nreps2 = nsurv/nblu
           iuse = nreps2*nblu
           kount(1,1) = kount(1,1)-iuse
           nsurv = nsurv-iuse
         GOTO 40
80       call stats1 (nwave,nreps3,kount)
       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:

CODE	COMMENT
what	The event name. character*6
who	The ID of the tank (or bullet). integer
whom	The ID of the target (if any). integer
t	Simulated time (sec).

Code.

```
c V7.5
      SUBROUTINE EVENTS
c 9   Events: call each event in sequence.
      include 'common.h'
      character*6 what
      integer who, whom
c
      if (trace) print *,'>events'
10    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 accelf (t,who)
        ELSEIF (what.eq.'maxvel') THEN
          call maxvel (t,who)
        ELSEIF (what.eq.'ikill ') THEN
          call latekl (t,who,wnom)
        ELSEIF (what.eq.'hide  ') THEN
          call hide  (t,who)
        ELSEIF (what.eq.'reload') THEN
          call reload (t,who)
        ELSEIF (what.eq.'popdn ') THEN
          call pop dn (t,who)
        ELSEIF (what.eq.'finish') THEN
          GOTO 99
        ELSE
          print*,'EVENTS: what=',what,' who='
     1    ,who,' whom=',whom,' time=',t
          print*,'Contact Fred Bunn'
          STOP
        ENDIF
      GOTO 10
99    if (trace) print *,'<events'
      END
```

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 Wars II Users' Manual.*

Code.

```
c V7.9
      SUBROUTINE INPUT
c 9   Input: read game control file.
      include 'common.h'
      character*32 fname
      integer indx(5)
      common /crandm/ irandm
      common /inpwav/ keyd(5), nwaves, neval, nused(3000)
1     format(i1,1x,a32)
2     format(2x,a,f6.2)
c
      read(5,*)(keyd(i),i=1,5)
      trace=keyd(4).gt.0
      histry = 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
20    CONTINUE
      read(5,*) irginc
      rgincr = irginc
      read(5,*) nreps, nwaves, iangd, METHSM, irandm
      read(5,*) tmax
        read *, sun
      read(5,1) k, fname
      invisb = 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.eq.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 rderor(fname,BLU,iecho)
c     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 rderor(fname,RED,iecho)
c     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**.

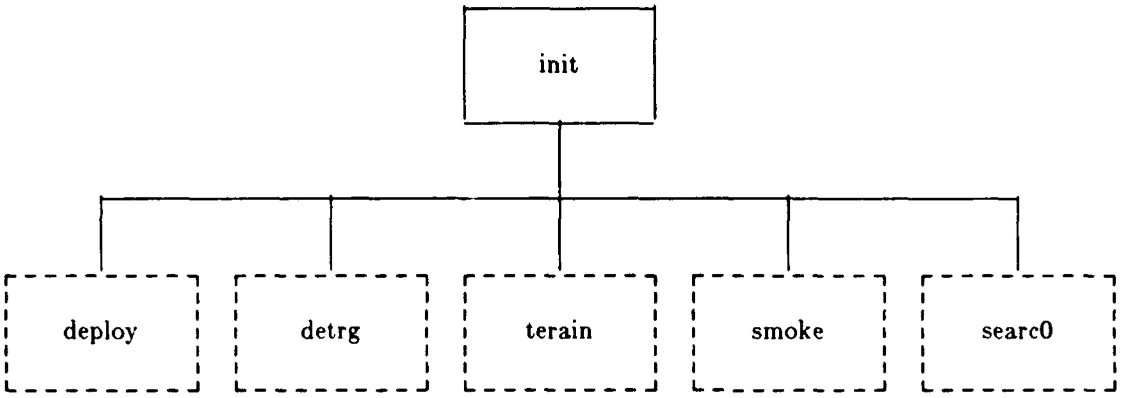
```
c V7.7
      SUBROUTINE RD MISC (fname,n,sun,iecho)
c 9   Rd misc: read miscellaneous tank characteristics.
c     fname - file name.
c     n - # of side. Blue=1, Red=2
      include 'common.h'
      character fname*32
      real high(2)
c
      if (trace) print *,'>Rdmisc'
      open(4, file=fname, 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), nrds(n), reliab(n)
        read(4,*) (tof(n,i),i=1,8)
c     Read acquisition characteristics.
        read(4,*) kview(n), ndets(n)
        read(4,*) pfalse(n,1), pfalse(n,2), pinp(n)
        read(4,*) share(n)
c     Read target selection criteria.
        read(4,*) nprior(n), recknz(n)
c     Read fire cycle characteristics.
        read(4,*) ishtfs(n), loader(n), tcon(n), tvar(n)
        read(4,*) nchans(n)
        read(4,*) (tfirst(n,i),i=1,8)
        read(4,*) nrpb(n), rof(n)
        read(4,*) nipods(n), trelod(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
```

2.6 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)
c 3   Pr misc: print misc tank characteristics.
      include 'common.h'
      integer irg(8)
      character*25 bar
      data bar/'-------------------------'/
1     format(i8,1x,a)
2     format(f8.2,1x,a)
3     format(f8.0,1x,a)
4     format(9x,a)
6     format(/,
   1  ' TARGET DIMENSIONS:',28x,'MOTION CHARACTERISTICS:',/,
   1  ' Distance (m) from center of',
   1  19x,'Acceleration',f6.2,' m/s**2',/,
   1  ' turret ring to:',31x,'Deceleration',f6.2,' m/s**2',/,
   1  ' Turret top  ',f6.2,'  Ground    ',f6.2,9x,
   1  ' Time to hide',f6.2,' sec',/,
   1  ' Turret Side ',f6.2,'  Hull side ',f6.2,9x,
   1  ' Combat speed',f6.2,' m/s',/,
   1  ' Turret front',f6.2,'  Hull front',f6.2,/,
   1  ' Turret back ',f6.2,'  Hull back ',f6.2,/)
7     format(/,
   1  9x,'---------DETECTION CAPABILITY---------',
   2  '  ------FIRING CYCLE-----'
   1  ,/,'   Rg     P-det (ever)      Pdet (1 sec) ',
   1  '    Tfirst  Tfixed   Tfly',/,
   1  '    (m)   HD    FE   FE-M     HD     FE   FE-M',
   1  '   (sec)   (sec)  (sec)',/,
   1  8(i7,f7.2,2f6.2,f8.2,2f7.2,f7.1,f8.1,f8.2,/))
8     format('  Temporarily discard after firing',i2,' shots.')
9     format('  2. M&F kill and',f4.0,' sec elapse or tank fires',
   1   i2,' rds at it.')

c     Write header
       print*
       if (n.eq.1) print *, bar//'BLUE SYSTEM CHARACTERISTICS'//bar
       if (n.eq.2) print *, bar//'RED SYSTEM CHARACTERISTICS'//bar
c     Write target dimensions and motion characteristics
       print 6, accel(n),decel(n),
   1    sysdim(3-n,1), sysdim(3-n,5), thide(n),
   1    sysdim(3-n,2), sysdim(3-n,6), speed(n),
   1    (sysdim(3-n,i), sysdim(3-n,i+4),i=3,4)
       k=ndecoy(n)
       if (k.gt.0) print 1,ndecoy(n),'decoys.'
       if (k.gt.0) print 1,nflash(n),'flashing decoys.'
c     Write range dependent values
       DO 50 i=1,8
        irg(i) = i*irginc
50     CONTINUE
       print 7,
   1    (irg(i), (pinfin(n,j,i),j=1,3), (pdet(n,j,i),j=1,3),
   1    tfirst(n,i),9.99, tof(n,i),i=1,8)
      print*,'ROUND CHARACTERISTICS:'
       k=kindrd(n)
       if (k.eq.1) print 1, nrds(n),'KE rounds/tank.'
       if (k.eq.2) print 1, nrds(n),'HEAT rounds/tank.'
       if (k.eq.4) print 1, nrds(n),'missiles/system.'
       if (k.eq.5) print 1, nrds(n),'top attack rounds/tank.'
       print 2, reliab(n),'Reliability of round.'
       print*
      print*,'ACQUISITION CHARACTERISTICS:'
       if (kview(n).eq.1) print 4,'Visual sensor'
       if (kview(n).eq.2) print4,'Thermal sensor'
       print 1, ndets(n),'detections at a time.'
       print 2, pfalse(n,1),'Probability of selecting false HD tgt.'
       print 2, pfalse(n,2),'Probability of selecting false FE tgt.'
       print 2, pinp(n),'Probability of pinpoint detection.'
       if (share(n)) print 4,'Systems communicate tgt locations.'
       if (.not.share(n))
   1    print 4,'Systems DO NOT communicate tgt locations.'
       print*
      print*,'TARGET SELECTION CRITERIA:'
       k=nprior(n)
       if (k.eq.1) print 4,'Selects old, hit tgts over new tgts.'
       if (k.eq.2) print 4,'Selects new tgts over old hit tgts.'
       print 3, recknz(n),'Recognition rg. (for tgt priorities.)(m)'
       print*
      print*,'FIRE CYCLE CHARACTERISTICS:'
       if(ishtfs(n).gt.0)print 4,'System halts to fire.'
       if(ishtfs(n).eq.0)print 4,'System fires on the move'
       k=loader(n)
       if (k.eq.1) print 2, tvar(n),
   1    'Median time between rounds for manual loader (sec).'
       if (k.eq.2) print 2, tcon(n),
   1    'Minimum time for load assist (sec).'
       if (k.eq.2) print 2, tvar(n),
   1    'Median additional time for load assist (sec).'
       if (k.eq.3) print 2, tcon(n),
   1    'Minimum time for autoloader (sec).'
       if (k.eq.3) print 2, tvar(n),
   1    'Median time for autoloader (sec).'
       print 1, nchans(n),'guidance channels.'
       k = nrpb(n)
       if (k.gt.1) print 1,nrpb(n),'Rounds/burst.'
       if (k.gt.1) print 2,rof(n),'between rds in a burst (sec).'
       k=kindrd(n)
       if (k.eq.4) print 1, nipods(n),'missiles/pod.'
       if (k.eq.4) print 2, trelod(n),'Time to change pods (sec).'
       print*
      print*,'TARGET SWITCHING CRITERIA:'
       print *, 'Permanently discard:'
       print *, '1. K-killed targets.'
       print 9, tbump(n), nbump(n)
       k = tactic(n)
       if (k.eq.2) print *, 'Temporarily discard after scoring a hit'
       if (k.eq.3) print 8, nrpt(n)
       if (k.gt.1) print 2, tlook(n),
   1    'Search time until re-engaging an old tgt (sec).'
       print *
       if (trace) print *,'<prmisc'
      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.



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.

CODE	INITIAL VALUE	COMMENT
		FOR EACH TANK I
busy	F	Tank is not busy. (busy inhibits selecting a new target)
empty	F	Tank has ammo.
idecoy	0	Tank is not a decoy.
iflash	0	Tank is not a flashing decoy.
life	ALIVE	Tank is mobile and lethal
ndet	0	Tank has detected zero targets.
nhot	0	Tank has achieved no hits on a target.
nrd	0	Tank has fired zero rounds.
nrtgt	0	ID of target is NULL.
nchan	0	Tank has no guidance channels busy.
nrot	0	Tank has achieved no rounds on target yet.
tfire2	0	Time tank last fired is NULL.
		CLEAR NN by NN MATRICES
los	F	Line-of-sight does not exist.
missed	F	Tank has not missed target.
mot	F	Firer has no missile assigned to target.
fot	F	Firer is not engaging target.
see	F	Firer doesn't see target.
tfire	F	Time tank last fired at tgt is NULL.

CODE	INITIAL VALUE	COMMENT
		FOR EACH TANK'S GUIDANCE CHANNEL
chanel	0	Channel assigned to guide NULL missile.
		FOR EACH BLUE-RED PAIR
los	T	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 **detrg** to find how far each tank can see. After that, it calls **terain** and **smoke** to set up terrain and smoke conditions. Next, it resets some values if some of the systems are decoys. Finally, it calls **searc0** so that each tank will begin searching for targets.

```
c V7.5
      SUBROUTINE INIT
c 4   Init: Initialize scenario & schedule search at time zero.
      include 'common.h'
      logical missed, ok
      common /MayPri/ missed(NN,NN)
      save /MayPri/
c
      if (trace) print *,'>Init'
      call skedul(tmax,0,'finish',NULL)
c     Set state variables for both red and blue systems.
      DO 40 I=1,nblu+nred
        busy(I) = .false.
        empty(I) = .false.
        idecoy(I) = 0
        iflash(I) = 0
        life(I) = ALIVE
        ndet(I) = 0
        nhot(I) = 0
        nrd(I) = 0
        nrtgt(I) = 0
        nchan(I) = 0
        nrot(I) = 0
        tfire2(I) = 0.0
        DO 20 it=1,nblu+nred
          los(I,it) = .false.
          missed(I,it) = .false.
          mot(I,it) = .false.
          fot(I,it) = .false.
          see(I,it) = .false.
          tfire(I,it) = 0.0
20      CONTINUE
        DO 30 k=1,5
          chanel(1,I,k) = 0
          chanel(2,I,k) = 0
30      CONTINUE
40    CONTINUE
      DO 45 i=1,nblu
        DO 42 k=1,nred
          j = nblu+k
          los(i,j) = .true.
          los(j,i) = .true.
42      CONTINUE
45    CONTINUE
      call deploy
      call detrg(BLU,1,nblu)
      call detrg(RED,nblu+1, nblu+nred)
      ok = scene.eq.BATTAK .and. invisb.eq.1
      if (ok) call terain(1,nblu)
      ok = scene.eq.RATTAK .and. invisb.eq.1
      if (ok) call terain(nblu+1,nblu+nred)
      if (invisb.ne.1) call smoke
c     Initialize some as decoys.
        DO 50 I=1,ndecoy(BLU)
          idecoy(I) = 1
          if (I.le.nflash(BLU)) iflash(I) = 1
          if (I.gt.nflash(BLU)) nrd(I)=999
50      CONTINUE
        DO 60 j=1,ndecoy(RED)
          I = nblu+j
          idecoy(I) = 1
          if (j.le.nflash(RED)) iflash(I) = 1
          if (j.gt.nflash(RED)) nrd(I)=999
60      CONTINUE
      call searc0
      if (trace) print *,'<Init'
      END
```

2.8 Statsl: Print Summary Statistics.

Statsl 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.

Statsl generates and prints summary results as shown below:

```
RESULTS         #              90% CONFIDENCE
Blue  won      54 or  0.540  0.454  0.625
Red  won       46 or  0.460  0.376  0.547
Draw            0 or  0.000
All dead        0 or  0.000
TOTAL REPS    100 or  1.000
(Red k-killed)/(Blue k-killed) =   3.112

ROUNDS FIRED BY          Blue   Red        SYSTEM STATUS          Blue   Red
Fired                     809   709        Alive                   103   134
  Wasted                    0     0        M-killed only             0     0
    Aborted                 0     0        F-killed only             4     6
    False Tgts              0     0        M&F-killed only          23    31
    Hidden tgts             6     0        K-killed                170   529
  Impacting               803   709        TOTAL                   300   700
    Misses                  4   393        damaged                 197   566
    Hits                  799   316        Alive 1-5 rds             0     0
      Duds                  6     4        Alive no rds              0     0
      No damage            65    41        *future                   0     0
      M-kill only           2     0        *future                   0     0
      F-kill only          29     8        *future                   0     0
      M&F-kill only       163    90        *future                   0     0
      K-kill              534   173        *future                   0     0
```

First, **statsl** 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, **statsl** 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, **statsl** is 90% confident that the probability of blue winning is between the confidence interval .454 to .625. If the sample size is too small, **statsl** 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 engagments 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, **statsl** 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, **statsl** 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)	k_{ij}	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
	$k_{1,5}$, $k_{2,5}$	# Blue & Red K-killed
	$k_{1,5}>0$	Test to avoid divide by zero
exch	$e = k_{2,5}/k_{1,5}$	exchange ratio

```
c V1.2
      SUBROUTINE STATS1 (nwave,nreps3,kount)
c 0   Stats1: Update and print statistics for 1 wave.
      include 'common.h'
      integer kount(2,20)
      character str(3)*4, str2(4)*9
      character str3(14)*20, str4(14)*20
      logical fail
      real lo
      data str /'Mtg ','Ratk','Batk'/
      data str2/'Blue  won ','Red  won ','Draw     ','All dead '/
      data str3/'Fired',' Wasted','  Aborted','  False Tgts',
     1 '  Hidden tgts',' Impacting','  Misses','  Hits',
     2 '    Duds','    No damage','    M-kill only',
     3 '    F-kill only','    M&F-kill only','    K-kill'/
      data str4/'Alive','M-killed only','F-killed only',
     1 'M&F-killed only','K-killed','TOTAL','damaged',
     2 'Alive 1-5 rds','Alive no rds','*future','*future',
     3 '*future','*future','*future'/
1     format(2x,a,t14,i5,' or',3f7.3)
2     format(2x,a,t37,f6.3)
3     format(2x,a,t23,2i6,t42,a,t62,2i6)
4     format('  ROUNDS FIRED BY',8x,'Blue  Red',
     1 7x,'SYSTEM STATUS',9x,'Blue   Red')
c
      if (trace) print *,'>stats1'
c     Print summary of outcomes.
        print*
        print*,'SUMMARY'
        print *,'RESULTS      #         90% CONFIDENCE'
        DO 50 i=1,4
          n = kount(1,i+14)
          frac = float(n) /nreps3 + 0.0004999
          call confb(frac,nreps3,hi,lo,fail)
          if (fail) print 1, str2(i),n,frac
          if (.not.fail) print 1, str2(i),n,frac,lo,hi
50      CONTINUE
        print 1, 'TOTAL REPS',nreps3, 1.00
        exch = 0.0
        if (kount(1,5).gt.0) exch=float(kount(2,5))/kount(1,5)
        print 2,'(Red k-killed)/(Blue k-killed) =',exch
c     Print round results and weapon system status
c       Find system status totals (6th line of numbers)
          DO 30 i=1,5
            kount(1,6)=kount(1,6)+kount(1,i)
            kount(2,6)=kount(2,6)+kount(2,i)
30        CONTINUE
c       Fill lines 7..9
          kount(1,7)=kount(1,6)-kount(1,1)
          kount(2,7)=kount(2,6)-kount(2,1)
          kount(1,8)=kount(1,19)
          kount(1,9)=kount(1,20)
c       Print results
          print *
          print 4
          print 3, (str3(j),(kshot(i,j),i=1,2),
     1       str4(j),(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**.

CODE	COMMENT
nwave	number of current wave
nw	maximum number of waves
nreps2	number of replications in current wave
kount(i,j)	table of systems status

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.

```
c V1.2
      SUBROUTINE STATS2 (nwave,nw,nreps2,kount)
c 0   Wave2: Update and print statistics for 1 wave.
      include 'common.h'
      integer kount(2,20), n(4)
1     format(' 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_{ik} 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_{ik} 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 i 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

life(j)	Status	Result
1	Alive	n(1,k) = n(1,k)+1
2	M-only killed	n(2,k) = n(2,k)+1
3	F-only killed	n(3,k) = n(3,k)+1
4	M&F-only killed	n(4,k) = n(4,k)+1
5	M&F-only killed	n(4,k) = n(4,k)+1
6	K-killed	n(5,k) = n(5,k)+1

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.

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

```
c V1.3
      SUBROUTINE STATS3 (key,nrep,kount,iseed,nsurv,nused)
c 0   Stats3: update statistics at end of a single engagement.
      character str(4)*8
      integer ammo, n(5,2), kount(2,20), balive, ralive, result
      integer nused(3000)
      include 'common.h'
      data str /'Blue won','Red won ','Draw    ','All dead'/
1     format('   Rep  Result  ',2(' AL MO FO MF  K'),2x,'seed')
2     format(i6,1x,a10,2(5i3),i9)
c
      if (trace) print *,'>stats3'
c     Count tanks in each damage status at the end of an engagement
c       Zero the counter array.
         DO 5 j=1,5
           n(j,1) = 0
           n(j,2) = 0
5        CONTINUE
c       Count blue tanks by status.
         DO 10 j=1,nblu
c          Tally by damage status
           i = life(j)
           if (i.ge.5) i=i-1
           n(i,1) = n(i,1)+1
10       CONTINUE
c       Count red tanks by status.
         DO 20 j=1,nred
           i = life(j+nblu)
           if (i.ge.5) i=i-1
           n(i,2) = n(i,2)+1
20       CONTINUE
c     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.eq.0) result = 1
        if (balive.eq.0 .and. ralive.gt.0) result = 2
        if (balive.gt.0 .and. ralive.gt.0) result = 3
        if (balive.eq.0 .and. ralive.eq.0) result = 4
      IF (key.ge.1) THEN
c       Print results of 1 engagement.
          if (nrep.eq.1 .or. trace) print 1
          print 2, nrep, str(result), n, iseed
      ENDIF
c     Update statistics for all replications at 1 opening range.
        DO 30 j=1,5
          kount(1,j) = kount(1,j) + n(j,1)
          kount(2,j) = kount(2,j) + n(j,2)
30      CONTINUE
        kount(2,14+result) = kount(2,14+result)+1
c     Find which Blues can fight a subsequent wave of Red tanks.
        DO 40 j=1,nblu
            IF (life(j).eq.ALIVE) THEN
c           Count fully functional Blue tanks by ammo remaining.
              ammo = nrds(BLU)-nrd(j)
              if (ammo.ge.5) nsurv = nsurv+1
              if (ammo.ge.5) nused(nsurv) = nrd(j)
              if (ammo.lt.5.and.nrd(j).gt.0)kount(2,19)=kount(2,19)+1
              if (ammo.eq.0) kount(2,20) = kount(2,20)+1
            ENDIF
40      CONTINUE
      if (trace) print *,'<stats3'
      END
```

INTENTIONALLY LEFT BLANK

3. SEARCH AND DETECTION ROUTINES

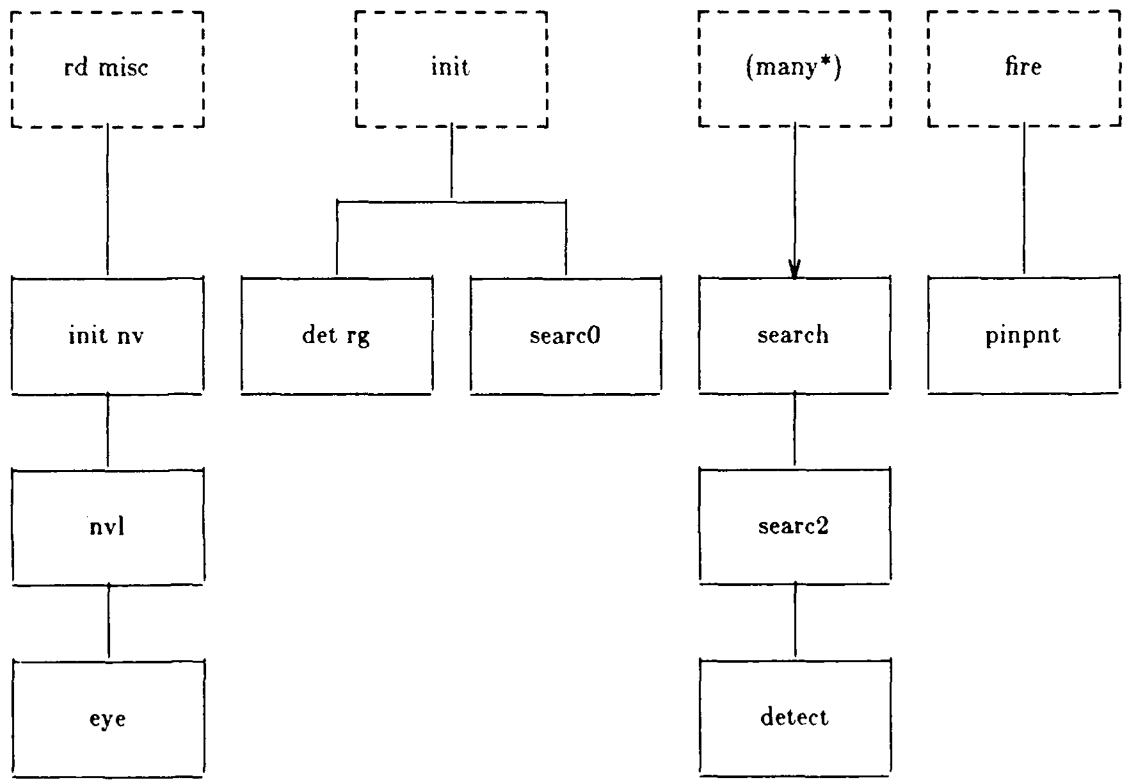
The search and detection routines simulate acquisition of non-firing targets. When undetected, unmasked targets are within detection range, the **search** routine is called once for each second of simulated time to find if a searcher detects a target in the next second.

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 it's 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 [illegible] great deal of run time by avoiding repeated calculation of exponentials later.

Further, the **det rg** and **searc0** 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.

Searc0 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, searc0, 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:

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

```
c 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 probabilities 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:

Ft-candles	Typical Day
1000	Clear day
100	Overcast day
10	Heavy overcast day
1	Sunset overcast day

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.

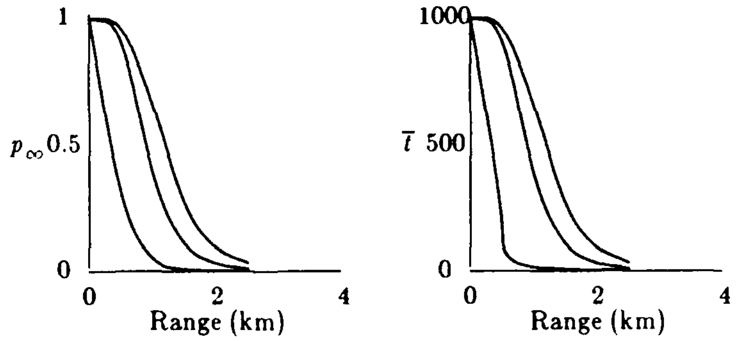


Figure 1. The Probability of Detecting a Target.

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 ($\bar{t}$) 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 moving, 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:

$$p_d = p_\infty e^{-t/\bar{t}}$$

```
c V7.2
      SUBROUTINE NVL (n,j,alumnc, dim, ajob, kind)
c     Nvl: find p-infinity, pdetect in 1 sec.
c     n - # of army
c     j - range band.
c     alumnc - illumination in ft-candles. (sun?)
c     dim - target height (m)
c     ajob - acquisition level
c     kind - kind of sensor. (1=eye)
      common /sensor/ psense(2,8), pinfin(2,3,10), pdet(2,3,10),
     1  ndets(2), tlook(2), pinp(2), repeat, recknz(2), pfalse(2,2)
      save rinfd, zone, visrg
      data zone, rinfd /225., .1/
      data visrg /7./
c
      DO 20 i=1,10
c     Find probabilities and median times for 8 ranges.
        rc   = 0.
        pinf = 0.
        tbarr = 999.
        range = 0.5*i
        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+0.7*x
          z = x**y
          pinf = z/(z+1.0)
          pinf = amin1(pinf,.99)
          tau = zone / (fov*amin1(5.,fov))
          tbarr = 3.4*tau/pinf
          if (pinf.gt.0.9) tbarr = tau*ajob*6.8/rc
        ENDIF
        if (tbarr.gt.999.0) tbarr=999.0
        pinfin(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, **nvl**, finds the actual probabilities after **eye** finds the resolvable cycles. Resolvable cycles are akin to scan lines used to paint an object on a TV screen.

CODE	MATH	COMMENT
alumnc	f	Illumination (ft-candles).
attn	a	Attenuation (?).
range	r	Range to target (km).
visrg	v	Visibility range (km).
sog	$s = (v+1)/3$	Sky over ground ratio. ($1 \leq s \leq 3$)
cntrst	$c = \dfrac{0.4}{1+s(e^{ar}-1)}$	Target to background contrast ratio.
i	i	Lower column for interpolation. $1 \leq i \leq 4$
j	j	Upper column for interpolation. $2 \leq j \leq 4$
k	k	Row index.
clog	$C = \ln c$	Natural logarithm of contrast ratio.
rlo	$r_l = \sum_{k=1}^{7} a_{i,k} C^{k-1}$	Resolvable cycles for lower power of 10.
rhi	$r_h = \sum_{k=1}^{7} a_{j,k} C^{k-1}$	Resolvable cycles for higher power of 10.
eye	$r_c = r_l+(r_h-r_l)(f-10^i/10)/(.9\times 10^i)$	Resolvable cycles interpolated for intermediate illumination.

It's not obvious that the equations for r_l and r_h above correspond to the calculation of **rlo and rhi**. 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:

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

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:

```
      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_c above yields an interpolated value between r_l, and r_h. To understand how r_c is interpolated, we'll have to do some backtracking. The equation used is:

$$r_c = r_l+(r_h-r_l)(f-10^i/10)/(.9\times 10^i)$$

$$r_c-r_l = (r_h-r_l)(f-10^i/10)/(.9\times 10^i)$$

$$\frac{r_c - r_l}{r_h - r_l} = \frac{f - 10^i/10}{.9\times 10^i}$$

$$\frac{r_c - r_l}{r_h - r_l} = \frac{f - 10^{i-1}}{10^i - 10^{i-1}}$$

This is simply the relationship for similar triangles used to perform linear interpolation in figure 2 below.

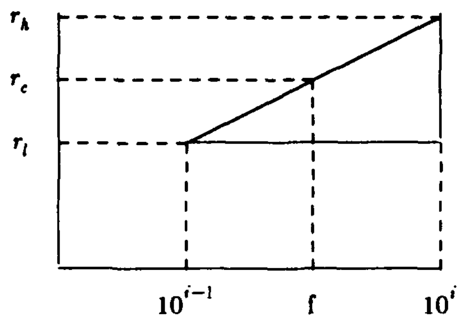
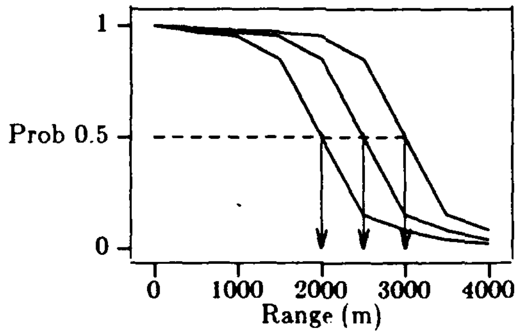


Figure 2. Linear Interpolation

```
c V7.1
      FUNCTION EYE (alumnc, attn, range, visrg)
c     Eye: find resolvable cycles for the human eye. (Device 1)
      real a(4,7)
      save a, acon
c     sunset o'cast  heavy o'cast   overcast day   clear day
      data a/
    1 1.2378091942, 1.7176916034, 1.9909928015, 2.0892716525,
    2 0.4694720809,  .4739084812,  .4484981232,  .2813866389,
    3  .0493317078, -.2102695514, -.4084256747,-1.0084578626,
    4 -.0601756751, -.4161055149, -.6856409935,-1.4323484287,
    5 -.0558327470, -.2696921300, -.4318233767, -.8450225947,
    6 -.0174190671, -.0756229822, -.1197712507, -.2235482536,
    7 -.0018530403, -.0077222394, -.0121729428, -.0218136690/
      data acon /.4/
c
c     Find sky-to-ground ratio
        sog = (visrg+1.0)/3.
        sog = amin1(3.,amax1(1.,sog))
      eye = 0.0
      cntrst = acon/(1.0+sog*(exp(attn*range)-1.0))
      IF (cntrst.ge.0.02) THEN
c     Target/Background contrast is sufficient to detect
        i = min0(4,1+int(alog10(alumnc)))
        ack = 10.**i
        j = min0(4,i+1)
        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
c       Interpolate & compute cycles across target
          eye = rlo+(rhi-rlo)*(alumnc-ack/10.)/(ack*.9)
      ENDIF
      END
```

3.4 *Detrg: Find Maximum Range to Which Each Firer Can Detect.* **Det rg** 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.

CODE	MATH	COMMENT
narmy	n	1 if Blue, 2 if Red
first		ID of first tank on the side
last		ID of last tank on the side
p	$p = ranu$	Random draw from a uniform distribution.
cond	c	1 if stationary HD, 2 if stationary FE, 3 if moving FE.
krg	k	Index such that $p_{n,c,k} \leq p < p_{n,c,k+1}$
p1	p_1	Lower bound of interpolation interval
p2	$p_1 = P_{n,c,k}$	Higher bound of interpolation interval.
r1	r_1	Range at lower bound (m).
incrg	Δr	Range increment in table (m).
r	$r = \Delta r \frac{p_1 - p}{p_1 - p_2}$	Range chosen.

```
c V7.5
      SUBROUTINE DET RG (narmy,first,last)
c     Det rg: Find the max ranges at which each firer in 'narmy' detects
      include 'common.h'
      integer narmy, first, last, tank, cond, krg
      real p1, p2, r, r1, p
1     format ('   Range to which tank can see',/,
    1 ' Tank  HDFE-SFE-Mranu')
2     format (i5,3f8.1,f8.4)
c
      if (trace) print *,'>detrg'
      if (histry) print 1
c     Loop thru all tanks on the side
        DO 80 tank = first,last
          p = ranu(0.0)
          DO 70 cond=1,3
            p1 = 1.0
c           Search for P-infinity values bounding x
              DO 60 krg=1,8
                p2 = pinfin(narmy,cond,krg)
                IF (p2 .lt. p) GOTO 65
                p1 = p2
60            CONTINUE
              p2 = 0.0
65            CONTINUE
c             Interpolate on p-infinity to find range.
                r1 = irginc*(krg-1)
                r  = r1 + irginc*(p1-p)/(p1-p2)
                rgvis(cond,tank) = r
70          CONTINUE
            if (histry) print 2,tank,(rgvis(cond,tank),cond=1,3),p
80        CONTINUE
        if (trace) print *,'<detrg'
        END
```

3.5 SearcO: Schedule Initial Search Event. The model executes the **searcO** routine once at the beginning of each engagement. **SearcO** 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.

SearcO initializes the table containing ranges between combatants as follows:

$$rgtbl(i,j) = r_{i,j} = \sqrt{(\,(x_i - x_j)^2 + (y_i - y_j)^2\,)}$$

Where,

x_i, y_i are the coordinates of the ith system.

Combatants are not ignored until they are killed to some level. So initially, 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.

SearcO 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 **searcO** schedules a finish event immediately. If they are beyond detection range and one side is moving, **searcO** finds when the sides are within detection range and schedules a search event at that time.

CODE	MATH	COMMENT
i	i	ID of Blue tank.
n	n	ID of Red tank.
x0(i), y0(i)	x_i, s_i	Coordinates of Blue tank (m).
x0(j), y0(j)	x_j, s_j	Coordinates of Red tank (m).
x, y	x, y	Coordinate of Blue tank W.R.T. to Red tank (m).
r	$r = \sqrt{[x^2+y^2]}$	Distance between tanks (m).
rgtbl(i,n)		Cell in table storing distance from tank i to tank n (m).
kred(scene)		Initial exposure of Red tanks in scenario 'scene'.
ni		Initial exposure of Blue tanks.
ignore(i)		True IFF tank i is no longer a threat.
rgvs(i)		Range to which tank i can detect (m).
dmax	d	Farthest distance any tank can detect a target (m)
nj		Initial exposure of Red tanks.
rg0	r_0	Opening range (m).
dist	$d = d - r_0$	Farthest distance a tank can detect beyond opening range (m).
	$r_0 < d$	Implies at least 1 tgt in detection range at time zero.
time		First possible detect time (sec).
scene		=1 for Meeting, 2 for Red attack, 3 for Blue attack

```
c V1.1
      SUBROUTINE SEARC0
c     Searc0: schedule initial search event.
c     ni, nj - 1 if HD, 2 if FE & stationary, 3 if FE & moving.
c     dist - distance to travel before entering detection range.
c     dmax = maximum distance any combatant can detect.
c     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'
c     Set up tables.
        DO 30 i=1,nblu
          DO 20 j=1,nred
            n = j+nblu
            x = x0(i)-x0(n)
            y = y0(i)-y0(n)
            r = sqrt(x**2+y**2)
            rgtbl(i,n) = r
            rgtbl(n,i) = r
20        CONTINUE
30      CONTINUE
c     Find maximum distance combatants can detect.
        ni = kred(scene)
        dmax = -1.e10
        DO 50 i=1,nblu
          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+nblu) = .false.
          rgvs(i+nblu) = rgvis(nj,i+nblu)
          dmax = amax1(dmax,rgvs(i+nblu))
60      CONTINUE
c     Find when search should begin.
        dist = rg0 - dmax
        IF (rg0.lt.dmax) THEN
c       In detection range at time 0.0
          time = 0.0
        ELSEIF (scene.eq.MEETNG) THEN
c       Outside & never enters detection range.
          time = 1.0e10
          call skedul(0.0,NULL,'finish',NULL)
        ELSEIF (scene.eq.RATTAK) THEN
c       Outside and red enters
          time = dist/speed(RED)
        ELSE
c       Outside and blue enters.
          time = dist/speed(BLU)
        ENDIF
        if (time.lt.tmax) call skedul(time,ALL,'search ',ALL)
      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 **search**ers to see which foe 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 **search**er and then Red as the **search**er.

If the Red tank is in detection range of the Blue **search**er and the Blue **search**er has not already detected the Red tank and the Blue **search**er 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.

CODE	MATH	COMMENT
t	t	Current time (sec).
repeat		True IFF **search** should reschedule itself.
time		First time any target can be detected (sec).
i	i	ID of tank.
ignore(i)		True IFF tank i is no longer a threat. (logical)
life(k)		Status of ith tank (>= IKILL implies it's known dead.)
x0(i), y0(i)		Coordinates of ith tank (m).
rgtbl(i,j)		Distance between ith & jth tanks (m).
ndeti		Maximum detections for a Blue tank.
ndetj		Maximum detections for a Red tank.
rgi		Distance Blue can detect Red target (m).
rgj		Distance Red can detect Blue target (m).
ok		Implies tgt is in detection range and not yet seen, and searcher is not loaded with detects.

```
c V7.8
      SUBROUTINE SEARCH (t)
c 3   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/
      rss(x,y) = sqrt(x*x+y*y)
c
      if (trace) print *,'>search'
      repeat = .false.
      IF (t.lt.time) RETURN
c     Update status of tanks.
       DO 5 i=1,nblu+nred
c      (Next line shud eventually be updated in damage.f, ltkill.)
        ignore(i) = ignore(i).or.life(i).ge.IKILL
        if (.not.ignore(i) .and. motion(i).ne.STATNY)
     1   call path(i,t,motion(i),0.0,dm1,dm2,dm3,dm4)
5      CONTINUE
       DO 20 i=1,nblu
        IF (.not.ignore(i)) THEN
         DO 10 j=nblu+1,nblu+nred
          rgtbl(i,j) = rss(x0(i)-x0(j),y0(i)-y0(j))
          rgtbl(j,i) = rgtbl(i,j)
10       CONTINUE
        ENDIF
20     CONTINUE
      ndeti = ndets(BLU)
      ndetj = ndets(RED)
      DO 40 i=1,nblu
c     Loop thru Blue tanks.
       IF (.not.ignore(i)) THEN
c      Consider tank i (It is alive and can detect or be detected.)
        rgi = rgvs(i)
        DO 30 j=nblu+1,nblu+nred
         IF (.not.ignore(j)) THEN
c        Consider tank j (Also alive and can detect or be detected.)
          rgj = rgvs(j)
          rgmax = amax1(rgi,rgj)
          rg = rgtbl(i,j)
          IF (rg.lt.rgmax) THEN
c         At least one is in detection rg of the other.
           IF (los(i,j)) THEN
c          Treat Blue tank as searcher
            ok=rg.lt.rgi.and..not.see(i,j).and.ndet(i).lt.ndeti
            if (ok) call searc2(t,i,j,BLU,ni,dt)
            if (.not.ok) repeat = .true.
c          Treat Red as searcher
            ok=rg.lt.rgj.and..not.see(j,i).and.ndet(j).lt.ndetj
            if (ok) call searc2(t,j,i,RED,nj,dt)
            if (.not.ok) repeat = .true.
           ELSE
            repeat = .true.
           ENDIF
          ENDIF
         ENDIF
30      CONTINUE
       ENDIF
40    CONTINUE
      if (repeat) call skedul(t+1.0,0,'search', NULL)
      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:

	index	1	2	3	4	5	6	7	8
	Rg (m)	100	200	300	400	500	600	700	800
1	Stationary, Hull defilade								
2	Stationary, Fully exposed	.775	.514	.372	.286	.230	.190	.160	.138
3	Moving, Fully exposed								

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.

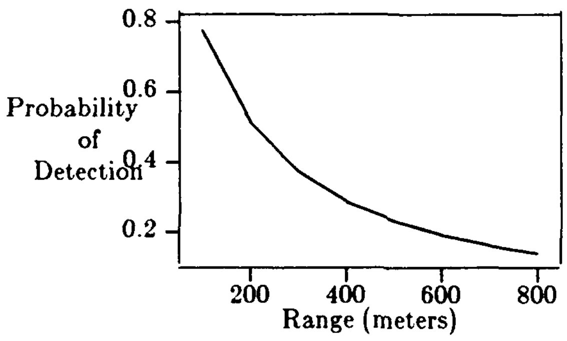


Figure 3. Probability of Detection in the Next Second

CODE	MATH	COMMENT
t		Simulation time (sec)
firer		ID of firer (integer)
tgt		ID of target (integer)
narmy		1 if firer is Blue, 2 if Red.
cond	c	1 if tgt is stationary HD, 2 if stationary FE, 3 if moving FE. (integer)
dt		Delay for rescheduling search (sec).
rg	r	Range from searcher to target (m).
rgincr	ΔR	Range increment in table (m).
temp	$z = r/\Delta R$	
indx	$i = int(z)$	Index of lower bound of range interval.
tlo	t_l	Probability of detecting at lower bound of range interval.
thi	t_h	Probability of detecting at upper bound of range interval.
frac	$f = z - int(z)$	Fraction of distance into range interval.
pdetct	$p = t_l + f(t_h - t_l)$	Probability of detection in current 1 second interval.
repeat		Flag to reschedule search. Set to .true. if detection does not occur. (logical)

Code.

```
c V7.4
      SUBROUTINE SEARC2 (t,firer,tgt,narmy,cond,dt)
c ?   Searc2: see if a tank detects a target during this second.
      include 'common.h'
      integer firer, tgt, cond

c     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
c     Interpolate in interval.
        frac = temp-aint(temp)
        pdetct = tlo + frac*(thi-tlo)
      IF (ranu(0.0).gt.pdetct) THEN
c     Set flag to repeat search. (At least one searcher didn't detect tgt.)
        repeat = .true.
        dt = 1.0
      ELSE
c     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.)

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

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.

```
c V7.3
      SUBROUTINE DETECT (t, I, it)
c 3   Detect: find if tgt detected and schedule subsequent events.
      include 'common.h'
1     format (f8.2,1x,a4,i3,' detects ',1x,a4,i3)
c
      if (trace) print *,'>detect'
      m = army(I)
      n = 3-m
      IF (los(I,it) .and. .not.see(I,it) .and.
   1    ndet(I).lt.ndets(m)) THEN
        if(histry)print 1,t,color(m),I,color(n),it
        ndet(I) = ndet(I)+1
        see(I,it) = .true.
        t human = 2.0*exp(rann(0.5))
        call selecs(t,I,thuman)
      ENDIF
      if (trace) print *,'<detect'
      END
```

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.

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

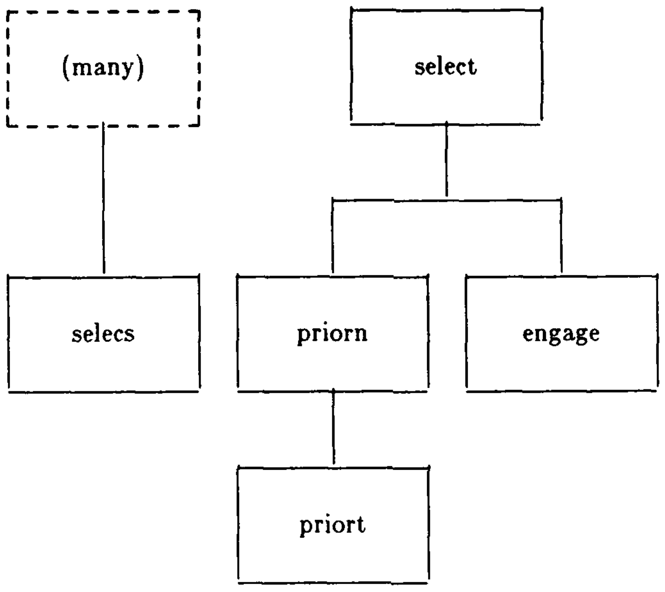
```
c V7.4
      SUBROUTINE PINPNT (t,I)
c 8   Pinpnt: Simulate firing signature (pinpoint) detection by some foes.
      include 'common.h'
      integer first
      logical wilsee
1     format (f8.2,1x,a4,i3,' sees    ',a4,i3,' muzzle flash')
c
      if (trace) print *,'>pinpnt'
      first = 1
      if (I.le.nblu) first = nblu+1
      last = nblu
      if (I.le.nblu) last = nblu+nred
      pinpxx = pinp(army(first))
      DO 20 k=first, last
        wilsee = pinpxx.gt.ranu(0.0)
        IF (life(k).lt.FKILL .and. wilsee .and.
     2    ndet(k).lt.ndets(army(k)) .and.
     1    los(k,I) .and. .not.see(k,I)) THEN
          if (histry) print 1,
     1      t, color(army(k)), k, color(army(I)), I
          see(k,I) = .true.
          ndet(k) = ndet(k) + 1
          thuman = 2.0*exp(rann(0.5))
          call selecs(t,k,thuman)
        ENDIF
20    CONTINUE
      if (trace) print *,'<pinpnt'
      END
```

INTENTIONALLY LEFT BLANK

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**.

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

```
c V7.5
      SUBROUTINE SELECT (t, I)
c 6   Select: gunner chooses most dangerous target it sees.
      include 'common.h'
      character*4 colort
      logical tgt fls, f alive, can go
      integer I, it, priorn, m
1     format(f8.2,1x,a4,i3,' selects  ',a4,i3,' with priority',i4,
   1    ' #tgts=',i2)
2     format(f8.2,1x,a4,i3,' selects  ',a4,' -1',
   1    ' & discards ',a4,i3, ' #tgts=',i2)
3     format(f8.2,1x,a4,i3,' selects',8x,'- (empty target set)')
4     format(' SELECT: ',a4,i3,' selects  ',a4,i3,' with priority',i4)
c
      if (trace) print *,'>select'
      m = army(I)
      kind = kindrd(m)
      f alive = life(I).lt.FKILL
      IF (f alive) THEN
c     Firer can shoot, so have him select.
        it = priorn(t,I,level)
        IF (it.eq.NULL) THEN
c       Firer has no targets to select so he moves if possible
          if (histry) print 3, t,color(m), I
          busy(I) = .false.
          if (kind.eq.4) nchan(I) = nchan(I)-1
          IF (can go(I,t) .and. (kind.le.2 .or.
   1        kind.eq.5 .or. nchan(I).eq.0)) THEN
            call cancel(I,'halt  ', NULL)
            call cancel(I,'accel ', NULL)
            call skedul(t,I,'accel ',NULL)
          ENDIF
        ELSE
c       Tgt has been selected
          colort = color(army(it))
          IF (tfire(I,it).le.0.) THEN
c         Tgt is new; replace with false tgt randomly.
            k = knceal(it)-1
            pf = ranu(0)
            tgt fls = pf .lt. pfalse(m,k)
            IF (tgt fls) THEN
              see(I,it) = .false.
              if (histry) print 2, t, color(m),
   1            I, colort, colort, it, nchan(I)
              it = fls tgt
c             Restart search if it is turned off
                IF (.not.repeat) THEN
                  repeat = .true.
                  call skedul(t,0,'search',NULL)
                ENDIF
            ELSE
              fot(I,it) = .true.
            if (histry) print 1, t, color(m),
     1        I,colort,it,level,nchan(I)
          ENDIF
        ELSE
c       Firer has previously serviced this target.
          fot(I,it) = .true.
          if (histry) print 1, t, color(m),
     1    I,colort,it,level,nchan(I)
        ENDIF
        call engage (t, t, I, it)
      ENDIF
      nrtgt(I) = it
    ENDIF
    if (trace) print *,'<select'
    END
```

4.2 Selecs: Start Target Selection if Appropriate. The subroutine **selecs** 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 **selecs** is the subroutine which calculates whether or not the target selection routines will be called. When **selecs** 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 **selecs** 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 **selecs** 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,I,dt)
      include 'common.h'
      logical loaded
1     format (f8.2,1x,a4,i3,' does not select; selecting already.')
2     format (f8.2,1x,a4,i3,' does not select; channels full.')
3     format (f8.2,1x,a4,i3,' does not select; pod empty.')
4     format (f8.2,1x,a4,i3,' begins selection.')
c
      if (trace) print *,'>selecs'
      m = army(I)
      kind = kindrd(m)
      if (kind.eq.4) loaded = nchan(I).ge.nchans(m)
      if (kind.ne.4) loaded = nrtgt(I).ne.0
      IF (busy(I) .or. empty(I) .or. loaded) THEN
c     Wait cause busy selecting, pod empty, or channels full.
        IF (histry) THEN
          IF (busy(I)) THEN
            print 1, t, color(m), I
          ELSEIF (loaded) THEN
            print 2, t, color(m), I
          ELSEIF (empty(I)) THEN
            print 3, t, color(m), I
          ENDIF
        ENDIF
      ELSE
c     Start selection: none in progress and a channel is free.
        busy(I) = .true.
        if (kind.eq.4) nchan(I) = nchan(I)+1
        call skedul(t+dt,I,'select', NULL)
        if (histry) print 4, t, color(m), I
      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:

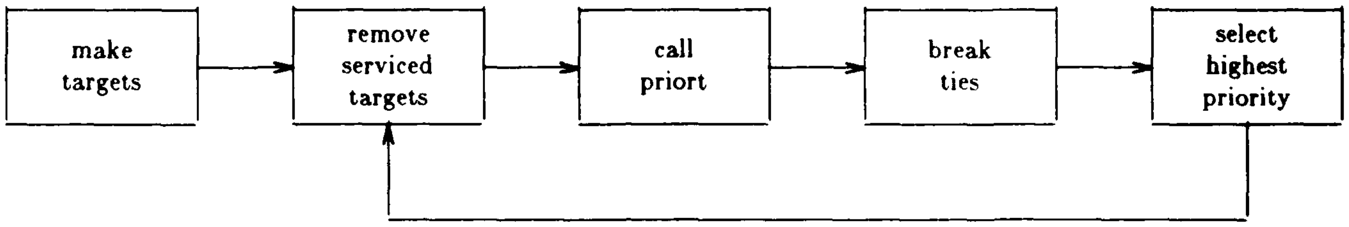


Figure 4. Selection of Highest Priority Target

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

```
c V7.3
      INTEGER FUNCTION PRIORN (t, I, lev old)
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)
c     '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
c     Compare all possible targets
        pick=.true.
        IF (share(armyf)) THEN
c       Don't select this tgt if anyone is already servicing it.
          DO 20 jfirer=1,last
            if(mot(jfirer,mtgt).or.fot(jfirer,mtgt))pick=.false.
20        CONTINUE
        ELSE
c       Don't select this target if I'm already servicing it.
          if(mot(I,mtgt).or.fot(I,mtgt))pick=.false.
        ENDIF
        rg tgt = rgf (t,I,mtgt)
        ck tgt = see(I,mtgt) .and. life(mtgt).lt.IKILL
     1    .and. rgtgt.le.4000.0 .and. pick
        IF (ck tgt) THEN
c       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*rann(1.0))
          t tgt = tfire(I,mtgt)
          better = level .lt. lev old
          IF (lev old.eq.level) THEN
c         Same priority class; now break ties
c           if new tgts pick closer
            if (t tgt.le. 0) better = rg tgt .lt. rg old
c           if old tgts, pick older (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
          ENDIF
        ENDIF
30    CONTINUE
      if (trace) print *,'<priorn'
      END
```

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

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

*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, MF, 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.

Table 3. Selection Priorities

1	1	Close	old tgt	missed	that fired in last 30 sec
2	2	Close	old tgt	missed	that has tgt
3	3	Close	old tgt	missed	that is stopped or slowing
4	4	Far	old tgt	missed	that fired in last 30 sec
5	5	Close	old tgt	missed	all others
6	6	Far	old tgt	missed	that is stationary
7	7	Far	old tgt	missed	all others
8	8	Close	new tgt		that fired in last 30 sec
9	9	Far	new tgt		that fired in last 30 sec
10	15	Close	old tgt	hit	that fired in last 30 sec
11	16	Close	old tgt	hit	that has tgt
12	17	Close	old tgt	hit	that is stopped or slowing
13	18	Far	old tgt	hit	that fired in last 30 sec
14	19	Close	old tgt	hit	all others
15	20	Far	old tgt	hit	that is stationary
16	21	Far	old tgt	hit	all others
17	10	Close	new tgt		that has tgt
18	11	Close	new tgt		that is stopped or slowing
19	12	Close	new tgt		all others
20	13	Far	new tgt		that is stationary
21	14	Far	new tgt		all others

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.

```
c V7.3
      SUBROUTINE PRIORT(I, it, rgtgt, t, L)
c 0   PRIORT:
      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,
     *         1,2,3,4,5,6,7,8,9,15,16,17,18,19,20,21,10,11,12,13,14/
1     format(' PRIORT: ',a4,i3,' considrs ',a4,i3,' with priority',
    1  i4,' (',i2,')')
c
      if (trace) print *,'>priort'
      j = nprior(army(I))
      m = motion(it)
      tactiv = 1.e35
      if (tfire2(it).gt.0.) tactiv = t-tfire2(it)
      IF (tfire(I,it).gt.0) 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.lt.recknz(army(I))) 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.lt.recknz(army(I))) 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
      ELSE
c     Target is a new target
        IF (rgtgt.lt.recknz(army(I))) 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
```

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.

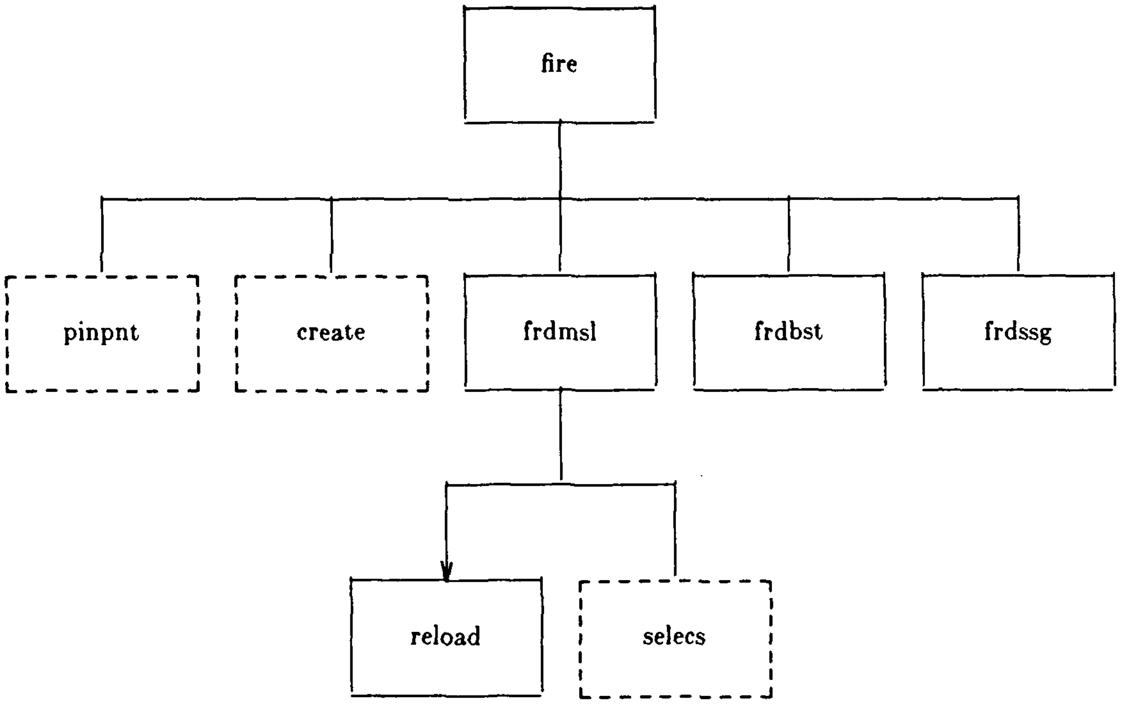
CODE	MATH	COMMENT
t1, t2		Current time (sec).
I		ID of firer.
it		ID of target.
m		Side of firer. 1 if Blue, 2 if Red.
n		Side of target. 1 if Blue, 2 if Red.
life(I)		Status of firer. Fully alive, mobility killed, etc.
nrd(I)		# rounds fired by firer.
nrds(m)		# rounds on board tanks on side m.
nbrst(I)		# Rounds fired in burst.
ishtfs(m)		1 if tanks on side m halt to fire. Zero otherwise.
motion(I)		1..4 if tank I is braking, stationary, accelerating, cruising
speed(m)		Combat cruise speed for tanks on side m.
rg		Range to target (m). Use opening range if false target.
nrg		Range band.
dt	$\Delta t = t_{first} e^{N(0.5)}$	Time to fire first round (sec).
prevrd(I)		=1 implies this is the first round fired at the target.
nrib(I)		=0 implies firer I is just beginning a burst (if it fires bursts).
nrot(I)		Count of rounds on target.

```
c V7.4
      SUBROUTINE ENGAGE (t1, t2, I, it)
c ?   Engage:
c     I - the firer.
c     it - the target.
c     t1, t2 - ?
      include 'common.h'
c
      if (trace) print *,'>engage'
      m = army(I)
      n = 3-m
      IF (life(I).lt.FKILL.and.nrd(I).lt.nrds(m)) THEN
        nbrst(I) = 1
        IF (ishtfs(m).gt.0 .AND. motion(I).ne.STATNY
     1    .and. speed(m).gt.0.0)THEN
c         halt to fire
          call cancel (I,'maxvel',NULL)
          call cancel (I,'accel ',NULL)
          call skedul(t1,I,'slowup',NULL)
        ELSE
c       Schedule a fire event otherwise
c         find range to target
            IF (it.eq.-1) THEN
              rg = rg0
              nrg = int(0.5+rg/irginc)
            ELSE
              dm = rgf(t1,it,I)
            ENDIF
          nrg = min0(8,nrg)
          dt = tfirst(army(I),nrg) * exp(rann(0.5))
          prev rd(I) = 1
          nrib(I) = 0
          nrot(I) = 0
c         if(kindrd(m).eq.4) dt=0.1
          call skedul (t2+dt,I,'fire  ',it)
        ENDIF
      ENDIF
      IF (trace) print *,'<engage'
      END
```

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

Range (m)	500	1000	1500	2000	2500	3000
Time (sec)	10.0	11.0	12.0	13.0	14.0	15.0

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 cannon 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:

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

The time between bursts is:

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

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+1st 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_c is the recharge time, then perhaps:

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

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.

CODE	MATH	COMMENT
t		Time (sec)
I		firer
it		Target
busy(I)		T/F. Tentatively set to false. This permits the tank to select a new target.
m,n		Side of firer, target
nrd(I)	$r_i = r_i+1$	Number of rounds fired by tank
nrib		Number of rounds fired in burst
nrpb		Number of rounds per burst
nrot(I)		Number of rounds fired at current target
	$it > 0$	=true if a real target. ID of false tgt is -1
tfire(I,it)	$t_f = t$	Save time firer fired at target. When switching targets, firer will give priority to least recently serviced tgt.
tfire2(I)	$t_2 = t$	Save time firer last fired. When selecting tgt, foes will give priority to tgts that fired recently.
	$it = FLSTGT$	Implies target is a false target. Code must generate position and set velocity to zero.
rg		Range of target (m).
nrg		Number of range band
irginc		Size of range bins (m).
s(i)	i=1,2,or3	Position of tank in question.
iflash(I)		T for flashing decoy, F for passive decoy.
bullet		ID of bullet.
tfly		Time of flight (sec).
tof		Time of flight table (sec).
psense		Probability of sensing the impact location of a miss
vx0,vy0	v	Current velocity of firer or tgt (m/s).
kindrd		1 for KE,2 for HEAT,4 for missile,5 for STAFF

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.

```
c V7.8
      SUBROUTINE FIRE (t,I,it)
c 7   Fire: Simulate firing of a round & schedule effects.
      include 'common.h'
      integer bullet
1     format(f8.2, 1x, a4, i3, ' fires at ', a4, i3)
2     format(f8.2, 1x, a4, i3, ' ran out of ammo.')

      if (trace) print *,'>fire'
      busy(I)=.false.
      m = army(I)
      n = 3-m
      if (histry) print 1,t,color(m),I, color(n),it
c     Update rd counts, time of last fire.
        nrd(I) = nrd(I)+1
        nrib(I) = nrib(I)+1
        if(nrib(I).gt.nrpb(m)) nrib(I)=1
        nrot(I) = nrot(I)+1
        if (it.gt.0) tfire(I,it) = t
        tfire2(I) = t
c     Update positions & velocities.
        IF (it.eq.FLSTGT) THEN
          rg = rg0
          nrg = max0(1,int(0.5+rg/irginc))
          s(1) = 0.0
          s(2) = rg0
          s(3) = 0.0
          if ((m.eq.BLU .and. scene.eq.BATTAK) .or.
     1      (m.eq.RED .and. scene.ne.BATTAK)) s(2) = -rg0
        ELSE
          dm = rgf(t,it,I)
        ENDIF
        call pinpnt (t,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(1)+tfly*x0(I)
          a(bullet+4) = s(2)+tfly*y0(I)
          a(bullet+7) = psense(m,nrg)
          a(bullet+9) = sqrt(vx0(it)**2+vy0(it)**2)
          if (it.eq.-1) a(bullet+10) = 0.0
          if (it.gt.0) a(bullet+10) = sqrt(vx0(I)**2+vy0(I)**2)
          kshot(m,1) = kshot(m,1) + 1
c       Schedule impact for rd & allot guidance channel.
          call skedul (t+tfly,bullet,'impact',it)
          IF (kindrd(m).eq.4) THEN
            if (it.gt.0) mot(I,it) = .true.
            DO 20 k=1,5
              IF (chanel(m,I,k) .eq. 0) GOTO 25
20          CONTINUE
25          chanel(m,I,k) = bullet
          ENDIF
       ENDIF
c      Move, fire, or switch targets as required
        IF (kindrd(m).le.2 .or. kindrd(m).eq.5) THEN
          if (nrpb(m) .le.1) call frdssg(t,I,it,m)
          if (nrpb(m) .ge.2) call frdbst(t,I,it,m)
        ELSEIF (kindrd(m) .eq. 4) THEN
c       Simultaneous missiles branch
          if (nchan(I).lt.nchans(m)) call frdmsl(t,I,it,m)
        ENDIF
      if (histry .and. nrd(I).ge.nrds(m)) print 2,t,color(m),I
      if (trace) print *,'<fire'
      END
```

5.3 Frdssg: Results of Firing a Single Shot Gun.

Frd ssg 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.

CODE	MATH	COMMENT
nrd(I)	r	# rounds fired by tank.
nrds(n)	$r_{\max}$	magazine capacity (Ammo load).
	$r < r_{\max}$	firer has more rounds to shoot.
tactic(n)	3	Side n fires a fixed number of rounds at a target.
nrot(I)		Number of rounds fired at current target.
nrpt(n)		Number of rounds per target before switching targets.
busy(I)		False for tank not busy; True for tank is busy.
loader(n)	k	1 for manual loader, 2 for automatic loader then manual, 3 for automatic loader parallel with manual gunning.
tvar	$N[0.5]$	Median between rounds of tank cannon.
dm	$d_m = t_v e$	Random human reaction time (sec).
tcon(n)	t_c	Minimum time to fire tank cannon (sec).
dt	$\Delta t = d_m$	For $k = 1$ (manual loading).
	$\Delta t = d_m + t_c$	For $k = 2$ (series auto-loading).
	$\Delta t = max(d_m, t_c)$	for $k = 3$ (parallel auto-loading).
empty(I)		No ammo.

Does the firer have more rounds to fire? The number of rounds it fired is **nrd(I)** and the number of rounds it started with is **nrds(n)**. It has more rounds iff **nrd(I) < ndrs(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 **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 **nrpt(n)**, so if **nrot(I).eq.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 **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 thide(n) seconds. Since the tank cannot shoot any more it seeks cover.

```
c V7.3
      SUBROUTINE FRD SSG (t, I, it, n)
c 6   Frd ssg: Schedule effects after firing single shot gun.
c     t - time (sec).
c     I - firer
c     it - target
c     n - side firer is on
      include 'common.h'
      logical can go, done, tactc3
1     format('FRD SSG:  t,I,it,n=',f7.2,3i3)
2     format(f8.2, 1x, a4, i3, 'is out of ammo. Will attempt',
   1   ' to hide if mobile.')
c
      if (trace) print *,'>frd ssg'
      IF (nrd(I).lt.nrds(n)) THEN
c     Have ammo branch
        tactc3 = tactic(n).eq.3
        done = nrot(I).eq.nrpt(n)
        IF ((tactc3 .and. done)) THEN
c       Switch targets after firing a fixed nr of rds at it
          busy(I) = .false.
          call dis eng (t, I, it,.true.,.true.)
c         If no other it and can move, skedul acceleration
          if (can go(I,t) .and. ishtfs(n).eq.1)
     1      call skedul(t,I,'accel ',NULL)
          nrot(I) = 0
        ELSEIF (it.gt.0) THEN
c       Schedule next round fired
          k=loader(n)
          dm = tvar(n)*exp(rann(0.5))
          if (k.eq.1) dt=dm
          if (k.eq.2) dt=tcon(n) + dm
          if (k.eq.3) dt=amax1(tcon(n),dm)
          call skedul (t+dt,I,'fire  ',it)
        ENDIF
      ELSE
c     Out-of-ammo branch
        empty(I) = .true.
        IF (cango(I,t)) THEN
          call skedul (t,I,'accel ',NULL)
          call skedul (t+thide(n),I,'hide  ',NULL)
        ENDIF
```

```
ENDIF
if (trace) print *,'<frd ssg'
END
```

5.4 Frdbst: Results of Firing a Round of a Burst. Frd bst 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.

```
c V7.4
      SUBROUTINE FRD BST (t, firer, tgt, armyf)
c 0   Frd bst: just fired a gun burst, now schedule effects.
      include 'common.h'
      logical can go, done, tactc3
      integer armyf, firer, tgt
1     format('FRD BST:  t,firer,tgt,armyf=',f7.2,3i3)
2     format(f8.2, 1x, a4, i3, 'is out of ammo. Will attempt',
    1  ' to hide if mobile.')
c
      if (trace) print *,'>frd bst'
      IF (nrd(firer).ge.nrds(armyf)) THEN
c     Out-of-ammo branch
        IF (cango(firer,t)) THEN
          call skedul (t,firer,'accel ',NULL)
          call skedul (t+thide(armyf),firer,'hide  ',NULL)
        ENDIF
      ELSE
c     Have ammo branch
        tactc3 = tactic(armyf).eq.3
        done = nrot(firer).eq.nrpt(armyf)
        IF ((tactc3 .and. done)) THEN
c         Switch targets after firing a fixed nr of rds at it
          call dis eng (t, firer, tgt,.true.,.true.)
c         If halt-to-fire & no tgts & cango, skedul acceleration
            if (ishtfs(armyf).eq.1 .and. can go(firer,t))
    1     call skedul(t,firer,'accel ',NULL)
          nrot(firer) = 0
        ELSE
c         Schedule next round fired
          timea = tcon(armyf)
          timeb = tcon(armyf)
          timec = tvar(armyf) * exp(rann(0.5))
          dt = amax1(timea,timeb+timec)
          call skedul (t+dt,firer,'fire  ',tgt)
        ENDIF
      ENDIF
      if (trace) print *,'<frd bst'
      END
```

5.5 Frdmsl: Results of Firing a Missile.

Frd msl 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.

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

```
c V7.6
      SUBROUTINE FRD MSL (t, I, it, m)
c 0   Frdmsl: Fired a missile. Now schedule effects.
      include 'common.h'
      logical done, tactc3
2     format(f8.2, 1x, a4, i3, ' begins to reload.')

      if (trace) print *,'>frdmsl'
      IF (nrd(I).lt.nrds(m)) THEN
c     System has more rounds on board.
        IF (mod(nrd(I),nipods(m)).gt.0 .or.nrd(I).eq.0) THEN
c       System has more rounds in pod.
          tactc3 = tactic(m).eq.3
          done = nrot(I).eq.nrpt(m)
          IF (tactc3 .and. done) THEN
c         Switch targets after firing a fixed nr of rds at it
            if (it.ne.FLSTGT) fot(I,it) = .false.
            call selecs (t,I,0.0)
          ELSE
c         Schedule next round fired
            call skedul (t+0.1,I,'fire  ',it)
          ENDIF
        ELSE
c       Treat empty missile pod
          empty(I) = .true.
          call cancel(I,'fire  ',it)
          call cancel(I,'select',NULL)
          nrot(I) = 0
c         shud htf that is slowing to engage speed up now?
          call skedul (t+trelod(m),I,'reload',NULL)
          if (histry) print 2,t,color(m),I
        ENDIF
      ENDIF
      if (it.gt.0) fot(I,it) = .false
      if (trace) print *,'<frdmsl'
      END
```

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.

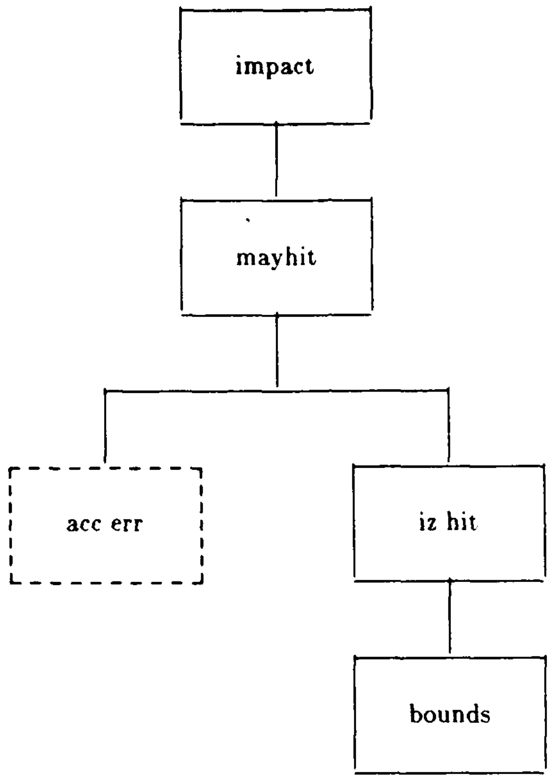
CODE	MATH	COMMENT
t		Time (sec)
I		ID of tank
firer		Number of the firer
nrtgt		Number of current target of tank
thuman	$\Delta t = 2e^{N[0.5]}$	Human reaction time

```
c V7.2
      SUBROUTINE RELOAD (t,I)
c 6   Reload: simulates completion of reloading
c     30 Oct 85  Fixed statement printing error message
      include 'common.h'
      logical defndr
1     format(f8.2,1x,a4,i3,' finishes reloading')
2     format(f8.2,1x,a4,i3,' pops-up')
c
      if (trace) print *,'>reload'
      m = army(I)
      if (histry) print 1,t,color(m),I
      nrtgt(I) = 0
      empty(I) = .false.
      defndr = (scene.eq.BATTAK .and. m.eq.RED) .or.
     1  (scene.eq.RATTAK .and. m.eq.BLU)
      IF (defndr) THEN
c     Defender pops back up and will start searching.
        if (histry) print 2,t,color(m),I
        call aprter(t,I,tgt,HD)
      ELSE
c     Attacker or 'meeter' never popped down.
        thuman = 2.0*exp(rann(0.5))
        call selecs(t,I,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.

CODE	COMMENT
t	Time (sec)
bullet	ID of bullet
it	ID of target
I	ID of firer
n	Side of firer (1=Blue, 2=Red)
k	Kind of round (1=KE, 2=HEAT, 4=msl, 5=TOP ATK)
expose	Exposure of tank (1=FD, 2=HD, 3=FE)
rgx	Range to target (m).
mot(I,it)	True iff missile on target
fot(I,it)	True iff firer on target
nchan(I)	Number of busy guidance channels for firer
nchans(m)	Number of guidance channels for side m tanks
empty(I)	True iff current missile pod or system is out of ammo
hit	True iff current missile hit target.
tactic(m)	Target switching policy for side m.
ndet(I)	Number of detections Ith tank has.
nrtgt(I)	ID of firer's latest target.

```
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)
      I = atr(2)
      n = army(I)
      k = kindrd(n)
      expose = knceal(it)
      rgx = 0.0
c     Find what bullet does.
      IF (it.eq.FLS TGT) THEN
```

```
c        Round does nothing.
          kshot(n,4) = kshot(n,4)+1
         ELSEIF (expose.eq.FD .and. k.le.4) THEN
c        Count round hitting berm.
          kshot(n,5) = kshot(n,5)+1
          if (histry) print *, 'Tgt in full deflade.'
         ELSE
c        See if round hits.
          call mayhit(t,I,it,n,k,atr(9),atr(10),expose,hit,rgx)
         ENDIF
         a(bullet) = -a(bullet)
c       Find what firer does.
         IF (k.eq.4) THEN
c        Missile
c         Clear guidance channel.
           DO 20 j=1,5
            IF (chanel(n,I,j).eq.bullet) GOTO 30
20         CONTINUE
           print *, 'IMPACT: Msl not assigned a channel.'
           print *, 'Channels assigned to',(chanel(n,I,j),j=1,5)
           print *, 'Msl #=',bullet,' Contact Fred Bunn'
           STOP
30         CONTINUE
           chanel(n,I,j) = NULL
          loaded = nchan(I).ge.nchans(n)
          call diseng (t,I,it,.true.,loaded)
          mot(I,it)=.false.
          fot(I,it)=.false.
          if (knceal(I).eq.HD .and. nchan(I).eq.0 .and. empty(I))
    1      call skedul(t,I,'popdn ',NULL)
         ELSE
c        KE, HEAT, or STAFF  {rethink this for STAFF}
          IF (it.eq.FLS TGT .or. hit.and.tactic(n).eq.2 .or.
    1      rgx.gt.4000.0) THEN
c          Switch targets if false target or rd hit & I switch on a hit.
c          Won't go here if I hit the berm; fls tgts don't go behind the
c          berm, and if true tgts do, the rd won't hit.
           ndet(I) = ndet(I)-1
           nrtgt(I)=0
           call diseng(t,I,it,.true.,.true.)
          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.

Treating 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(I,it) is set to false. This information will be used to select or reject this target later. Prevrd(I) 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.

Treating 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.)

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

CODE	COMMENT
life(it)	1=Alive, 2=M-kill, 3=F-kill, 4=M&F-kill, 5=I-kill, 6=K-kill
nhot(it)	Number of hits on target after M&F-killed.
nbump(n)	Number rounds to fire at M&F-killed tgt before discarding it.
missed(I,it)	True iff the round missed the target.
prevrd(I)	1=1st round on tgt, 2=previous was a hit, 3=previous was a sensed miss, 4=previous was lost miss.
reliab(n)	Probability round is reliable for nth side.
nrg	Number of range band.
psense(n,nrg)	Probability of sensing miss for tank.

```
c V7.9
      SUBROUTINE MAYHIT (t,I,it,n,k,v1,v2,expose,hit,rgx)
c 0   Mayhit: Find what the round does.
      include 'common.h'
      common /cimpct/ x,y,theta, disp
      logical missed
      common /MayPri/ missed(NN,NN)
      save /cimpct/, /MayPri/
      integer expose
      logical hit, izhit
1     format(f8.2,1x,a4,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,I,it)
        crs ang = 0.0
        if (vx0(it).ne.0.0) crs ang = anglef(s,vt)
        call accerr(n,rgx,I,crsang,v1,v2,x,y,disp)
c      Find position of round w.r.t. center of turret ring.
        if (expose.eq.FE) y=y-0.3
        if (expose.eq.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(HULL,5,n,x,y,theta)
          ELSE
            if (histry) print 1, t, color(n), I
          ENDIF
        ENDIF
      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,it,'ikill ',NULL)
       missed(I,it) = .false.
       prevrd(I) = 2
       IF (reliab(n) .ge. ranu(0)) THEN
         call skedul (t,I,'damage',it)
       ELSE
c      Round is a dud.
         kshot(n,9) = kshot(n,9)+1
       ENDIF
      ELSE
c     Treat miss.
       kshot(n,7) = kshot(n,7)+1
       missed(I,it) = .true.
       nrg = max0(1,int(0.5+rgx/rgincr))
       IF (psense(n,nrg) .gt.ranu(0.0)) THEN
         prevrd(I) = 4
         if (histry) print *,' Miss is sensed.'
       ELSE
         prevrd(I) = 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(I).ne.0.0 .or. vx0(it).ne.0.0) prevrd(I)=1
      if (v1.gt.0 .or. v2.gt.0) prevrd(I)=1
      if (trace) print *, '<mayhit'
      END
```

6.3 Izhit: Find if the Target Is Hit. **Izhit** discards rounds that are too high or too low. For other rounds, it finds the orientation of the hull or turret and its horizontal bou daries. If the round is within the horizontal boundaries, **izhit** 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.

Distance from Center of Turret Ring to

turret		hull	
i	sysdim(i)	i	sysdim(i)
1	top	5	bottom
2	side	6	side
3	front	7	front
4	rear	8	rear

The code tentatively sets izhit=.false. It then checks to see if the vertical error of the round is greater that 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	COMMENT
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	X coordinate of bullet on target (m).
y	Y coordinate of bullet on target (m).
theta	Angle at which round struck target (rad).
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).

```
c V7.2
      LOGICAL FUNCTION IZHIT (nbox, ndim, n, x, y, theta)
c 6   Iz hit: find if the target is hit.
      include 'common.h'
1     format (' IZHIT: the round is high. y, ylimit, x =',3f7.3)
2     format (' IZHIT: the round is low.  y, ylimit, x =',3f7.3)
3     format (' IZHIT: the round is wide. y, ylimit =', 2f7.3,/
   1  '        x, xleft, xright = ', 3f7.3)
4     format (' IZHIT:  the round hits.   y, ylimit =', 2f7.3,/
   1  '        x, xleft, xright = ', 3f7.3)
c
      if (trace) print *,'>izhit'
      izhit = .false.
      ylimit = sysdim(n,ndim)
      IF (ylimit.le.abs(y)) THEN
c       Too high or too low
        IF ( keym(6).gt.0) THEN
          if (y.gt.0.0) print 1, y, ylimit, x
          if (y.le.0.0) print 2, y, ylimit, x
        ENDIF
      ELSE
c     Height ok
        theta = rndang(iangd)
        call bounds (n, nbox, theta, xleft, xright)
        izhit = xleft.lt.x .and. x.lt.xright
        IF (keym(6).gt.0) THEN
          if(izhit) print 3, y,ylimit,x,xleft,xright
          if(izhit) print 4, y,ylimit,x,xleft,xright
        ENDIF
      ENDIF
      if (trace) print *,'<izhit'
      END
```

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.

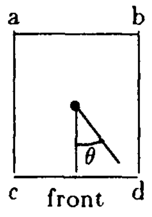
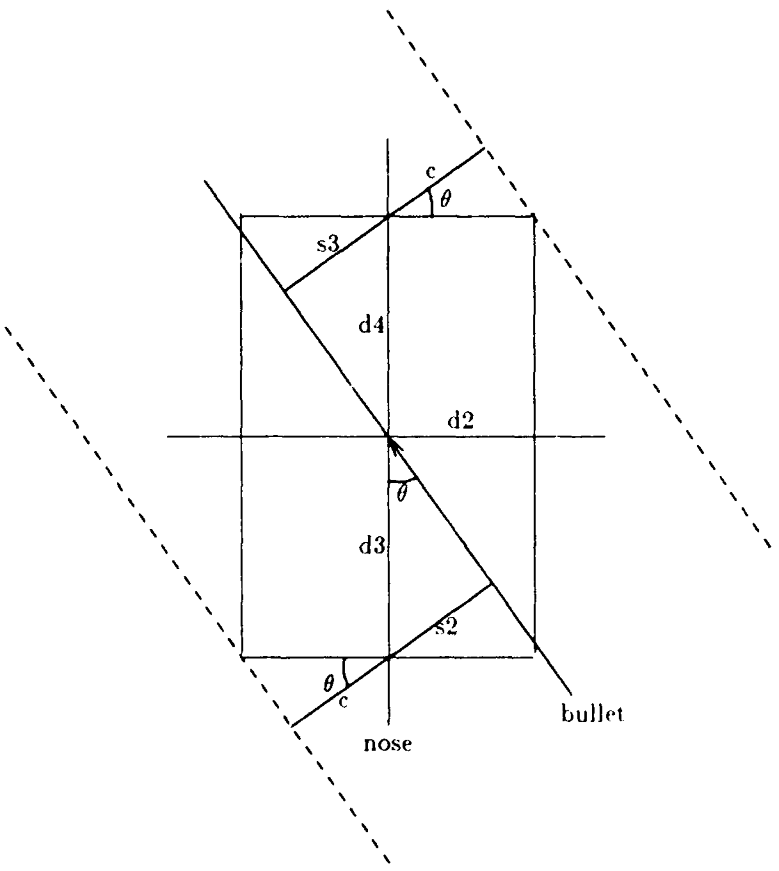


Table 5. Horizontal Boundaries

	Quadrants			
boundary	$0<\theta<90$	$90<\theta<180$	$180<\theta<270$	$270<\theta<360$
left	c	d	b	a
right	b	a	c	d

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

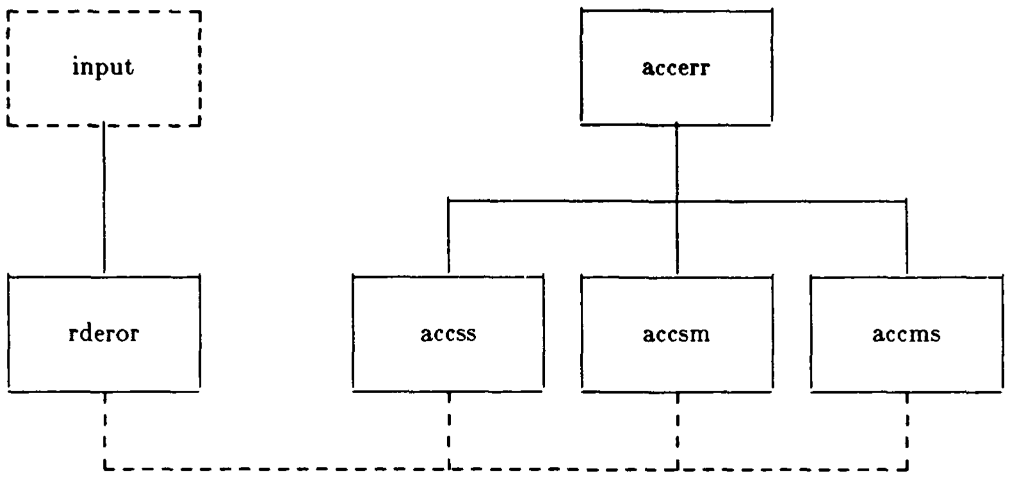
d3= distance from center of turret ring to turret rear
d4= distance from center of turret ring to turret front
c= d2cosθ
s2= d3sinθ
s3= d4sinθ

```
c V7.1
      SUBROUTINE BOUNDS (narmy, box, angll, r1, r2)
c 6   Bounds: find the horizontal bounds of hull or turret.
c     Definitions:
c       angll - angle off the nose of the box (rad).
c       box - 1 means turret box, 2 means hull box.
c       narmy - 1 means blue firers, 2 means red firers.
c       c, s2, s3 - temporary variables.
c       r1, r2 - left and right boundaries of boxes (m).
      include 'common.h'
      integer box
c
      if (trace) print *,'>bounds'
c     initialize
        temp = (angll+twopi)/twopi
        theta= (temp-aint(temp))*twopi
c       theta = amod (angll+twopi,twopi)
        c = sysdim(narmy,4*(box-1)+2) * cos(theta)
        s2= sysdim(narmy,4*(box-1)+3) * sin(theta)
        s3= sysdim(narmy,4*(box-1)+4) * sin(theta)
c
      IF (theta.le.0.25*twopi) THEN
c     case 0 < theta <= 90
        r1 = -s2 - c
        r2 = s3 + c
      ELSEIF (theta.le.0.5*twopi) THEN
c     case 90 < theta <= 180
        r1 = -s2 + c
        r2 = s3 - c
      ELSEIF (theta.le.0.75*twopi) THEN
c     case 180 < theta <= 270
        r1 = s3 + c
        r2 = -s2 - c
      ELSE
c     case 270 < theta <= 360
        r1 = s3 - c
        r2 = -s2 + c
      ENDIF
      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 **mayhit** 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.

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

```
c V7.4
      SUBROUTINE RDEROR (dbname, m, iecho)
c 3   Rd eror: read accuracy data for a side.
c     dbname - name of file containing error (accuracy) data.
c     m - 1 iff for Blue, 2 iff for Red.
c     iecho - print input echo iff true.
      include 'common.h'
      character*72 descr
      character*32 dbname
      character*8 q
      real mstbl
      common /comss/ nss(2), sstbl(10,7,2)
      common /comsm/ nsm(2), smtbl(10,17,2)
      common /comms/ kindms(2), nms(2), mstbl(10,5,2)
      save /comss/, /comsm/, /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 nrow=1,nrows
          read (4,2)q, (sstbl(ncol, nrow, m), ncol=1,ncols)
          if(iecho.ge.2) print 3, q,(sstbl(ncol,nrow,m),
     1      ncol=1,ncols)
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)
          if(iecho.ge.2) print 3, q,(smtbl(ncol,nrow,m),
     1      ncol=1,ncols)
20      CONTINUE
c     Read moving-stationary errors
        read (4,1) nrows, ncols, descr
        if(iecho.ge.1) print 1, ncols, nrows, descr
        nms(m) = ncols
        kindms(m) = nrows
        DO 30 nrow=1,nrows
          read (4,2) q,(mstbl(ncol,nrow,m),ncol=1,ncols)
          if(iecho.ge.2) print 3, q,(mstbl(ncol,nrow,m),
     1      ncol=1,ncols)
30      CONTINUE
      print *, ' Acc file is:', dbname
      close(4)
      if (trace) print *,'<rderor'
      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.

CODE	MATH	COMMENT
m		Firer's side
r		Range to target (m)
I		Firer ID
theta		Crossing angle (rad)
vtgt		Speed of target (m/s)
vfirer		Speed of firer (m/s)
x	$x = .001xr$	Linear horizontal error (m)
y	$y = .001yr$	Linear vertical error (m)
rex	$r_x = .001r_xr$	Linear horizontal dispersion (m)
rey	$r_y = .001r_yr$	Linear vertical dispersion (m)
disp	$d = 3.28\sqrt{.5(r_x^2+r_y^2)}$	RMS dispersion (ft)

```
c V7.4
      SUBROUTINE ACC ERR(m,r,I,theta,vtgt,vfirer,x,y,disp)
c 1   Acc err: find the linear error for a single round.
      include 'common.h'
      logical fmove, fstat, 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
c
      IF (burst) THEN
c     Burst fire branch
        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
c     Either KE or HEAT gun system. (kindrd=1,2)
        if (fstat.and.tstat) call accss (m,r,I,x,y,rex,rey)
        if (fstat.and.tmove) call accsm (m,r,theta,x,y,rex,rey)
        if (fmove.and.tstat) call accms (m,r,vfirer,x,y,rex,rey)
      ELSE
c     Direct fire or top attack missile. (kindrd=4,5)
        call acc ss (m,r,I,x,y)
      ENDIF
c     Convert from angular to linear errors.
        x = x*r*.001
        y = y*r*.001
        rex = rex*r*.001
        rey = rey*r*.001
        disp = 3.28*sqrt(0.5*(rex**2+rey**2))
      if (trace) print *,'<acc err'
      END
```

7.3 Accss: Find Angular Accuracy for Stationary Firer vs Stationary Target. Accss 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

First Round Biases, Dispersions, and First Round Probability of Hit									
			Horizontal (mils)			Vertical (mils)			
Range (Meters)	Fixed Biases (mils) Horizontal	Vertical	Random Error	Variable Biases	Total Dispersion	Random Error	Variable Biases	Total Dispersion	P_{H1}
250	1.072	0	1.3702	.5728	1.4272	1.3702	.6284	1.4504	.9927
500	.357	0	.7260	.6940	1.0043	.7260	.8572	1.1233	.9343
1000	.000	0	.4652	1.1345	1.2262	.4652	1.8468	1.9015	.3019
1500	-.119	0	.3929	1.7860	1.8287	.3929	3.4496	3.4719	.0580
2000	-.178	0	.3621	2.6669	2.6914	.3621	6.2610	6.2715	.0126
2500	-.214	0	.3459	3.8062	3.8757	.3459	11.1232	11.1286	.0032
3000	-.238	0	.3362	5.4729	5.4832	.3362	19.1972	19.2001	.0009
	μ_x	μ_y	σ_x	ν_x		σ_y	ν_y		

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

```
7   7 S-S Errors for Blue xxx
rg(m) ->  250      500      1000     1500     2000     2500     3000
1st mux 1.072    .357     .000     -.119    -.178    -.214    -.238
    muy 0        0        0        0        0        0        0
    nux .5728    .6940    1.1345   1.7860   2.6669   3.8062   5.4729
    nuy .6284    .8572    1.8468   3.4496   6.2610   11.1232  19.1972
h|1 sgx 1.3702   .7260    .4652    .3929    .3621    .3459    .3362
    sgy 1.3702   .7260    .4652    .3929    .3621    .3459    .3362
```

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.

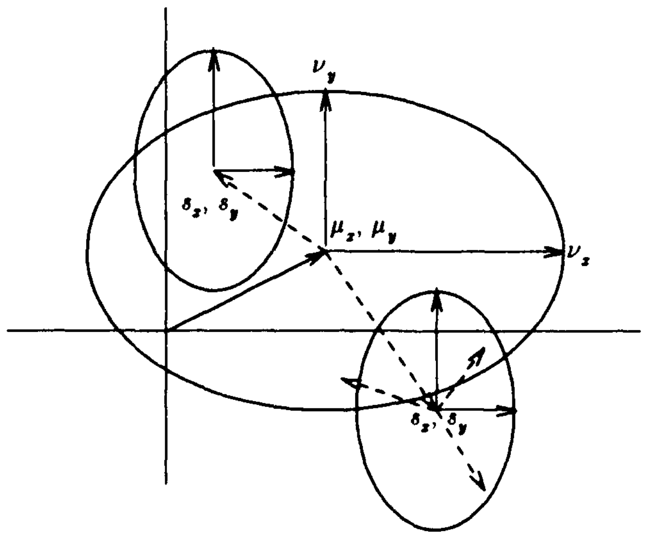


Figure 5. Bias and Dispersion Errors

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.

CODE	MATH	COMMENT
m		Side of firer
r		Range to target (m)
I		ID of firer.
sstbl(n,k,m)	$S_{n,k,m}$	Errors for nth range, kth type, and mth side (mils).
k	k	Interpolate for range in columns k, k+1
frac	$f = (r-r_k)/(r_{k+1}-r_k)$	Fraction of distance in interval
mux, muy	μ_x, μ_y	Horizontal, vertical fixed bias (mils)
sgx, sgy	σ_x, σ_y	Horizontal, vertical dispersion (mils)
sx, sy	s_x, s_y	Ràndom draw for horizontal, vertical random error (mils)
nux, nuy	ν_x, ν_y	Horizontal, vertical variable bias (mils)
vx, vy	$v_x = \nu_x[N]$	Random draw for variable bias (mils)
tx, ty	$t_x = \sqrt{\sigma_x^2+\nu_x^2}$	Total dispersion (mils)
ax, ay	$a_x = \mu_x+n_x+s_x$	Angular error (mils)

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.

n	k=	1	2	3	4	5	6	7	8	9	10
1	r	$s_{1,1,1}$	$s_{2,1,1}$								
2	μ_x	$s_{1,2,1}$									
3	μ_y										
4	ν_x										
5	ν_y										
6	σ_x										
7	σ_y	$s_{1,7,1}$									

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.)

```
$f(k,n) = sstbl_{k,n,m} + frac(sstbl_{k+1,n,m} - sstbl_{k,n,m})$

IF (missile) THEN Find missile errors
  $k=$
  $frac=$
  $a_z = f(k,2) + f(k,4)N$
  $t_z = \sqrt{(f(k,2)^2 + f(k,4)^2)}$
ELSE Find gun errors
  IF (first round) THEN Save values
    $k=$
    $frac=$
    $\mu_z' = f(k,2)$
    $v_z' = f(k,4)N$
    $\sigma_z' = f(k,6)$
    $t_z' = \sqrt{(f(k,4)^2 + f(k,6)^2)}$
  ENDIF
  $a_z = \mu_z' + v_z' + \sigma_z' N$
  $t_z = t_z'$
ENDIF
```

Before the first executable statement is a statement function which does a linear interpolation between the kth and k+1st 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.

```
c V7.4
      SUBROUTINE ACC SS(m,r,I,ax,ay,tx,ty)
c 0   Acc ss: find Angular accuracy for stationary-stationary single shots.
      include 'common.h'
      real rrgs(10), brgs(10)
      common /comss/ nss(2), sstbl(10,7,2)
      equivalence (sstbl(1,1,1),brgs(1))
      equivalence (sstbl(1,1,2),rrgs(1))
      save /comss/, smux, smuy, svx, svy, ssgx, ssgy, stx, sty
      real smux(NN),smuy(NN),ssgx(NN),ssgy(NN),svx(NN),svy(NN),
    1   stx(NN),sty(NN), mux, muy, nux, nuy
      f(k,n) = sstbl(k,n,m) +
    1   frac*(sstbl(k+1,n,m)-sstbl(k,n,m))
      rss(x,y) = sqrt(x*x+y*y)

      if (trace) print *,'>accss'
      IF (kindrd(m).eq.4) THEN
c     Find errors for a missile system.
c       Find index for interpolation.
          if(m.eq.1)k=indexx(brgs,nss(m),r)
          if(m.eq.2)k=indexx(rrgs,nss(m),r)
          if (k.lt.1) k=1
c       Find interpolation constant.
          if(m.eq.1)frac = (r-brgs(k)) / (brgs(k+1)-brgs(k))
          if(m.eq.2)frac = (r-rrgs(k)) / (rrgs(k+1)-rrgs(k))
c       Interpolate to find angular errors.
          tx = f(k,4)
          ty = f(k,5)
          ax = f(k,2) + rann(tx)
          ay = f(k,3) + rann(ty)
      ELSE
c     Find errors for a gun system.
        IF (prevrd(I).eq.1) THEN
c       Save first round values.
c         Find index for interpolation.
            if(m.eq.1)k=indexx(brgs,nss(m),r)
            if(m.eq.2)k=indexx(rrgs,nss(m),r)
            if (k.lt.1) k=1
c         Find interpolation constant.
            if(m.eq.1)frac = (r-brgs(k)) / (brgs(k+1)-brgs(k))
            if(m.eq.2)frac = (r-rrgs(k)) / (rrgs(k+1)-rrgs(k))
c         Interpolate to find angular errors.
            smux(I) = f(k,2)
            smuy(I) = f(k,3)
            svx(I)  = rann(f(k,4))
            svy(I)  = rann(f(k,5))
            ssgx(I) = f(k,6)
            ssgy(I) = f(k,7)
            stx(I)  = rss(f(k,4),f(k,6))
            sty(I)  = rss(f(k,5),f(k,7))
        ENDIF
c       Now find angular errors.
          tx = stx(I)
          ty = sty(I)
          ax = smux(I) + svx(I) + rann(ssgx(I))
          ay = smuy(I) + svy(I) + rann(ssgy(I))
      ENDIF
      if (trace) print *,'<accss'
      END
```

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 7. Stationary-Firer vs Moving-Target Accuracy

Stationary Firer vs. Moving Target
Evasive Factor = .25
Target Crossing Direction = counterclockwise
Target Crossing Angle = 0 degrees
Bias and Dispersion in mils

Target Speed (KPH)		ACCURACY DATA AS A FUNCTION OF RANGE (METERS)						
		250	500	1000	1500	2000	2500	3000
2	H BIAS	1.0547	.3194	-.0903	-.2843	-.4510	-.6415	-.8751
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.6466	.7561	1.1735	1.8110	2.6837	3.8718	5.4811
	V DISP	.6994	.9107	1.8725	3.4636	6.2689	11.1277	19.1999
	P(H)	1.000	.9853	.2156	.0581	.0125	.0032	.0009
10	H BIAS	.7800	-.2740	-1.4549	-2.5467	-3.5063	-3.8250	-3.2910
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.6472	.7566	1.1739	1.8114	2.6840	3.8721	5.4983
	V DISP	.7693	.9702	1.9082	3.4875	6.2855	11.1399	19.2091
	P(H)	1.000	.9801	.1784	.0231	.0055	.0020	.0008
20	H BIAS	.7794	-.2785	-1.5177	-2.8526	-4.5495	-6.6631	-8.8527
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.6468	.7583	.1.1752	1.8124	2.6849	3.8729	5.4821
	V DISP	.9552	1.1361	2.0154	3.5609	6.3371	11.1775	19.2378
	P(H)	1.000	.9560	.1617	.0178	.0031	.0007	.0002
30	H BIAS	.7793	-.2806	-1.5296	-2.9125	-4.7684	-7.3413	-10.6929
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.6521	.7611	1.1774	1.8142	2.6864	3.8742	5.4833
	V DISP	1.2028	1.3685	2.1824	3.6799	6.4129	11.2398	19.2852
	P(H)	.9999	.9076	.1491	.0164	.0026	.0005	.0001
40	H BIAS	.7792	-.2809	-1.5338	-2.9337	-4.8470	-7.5917	-11.4060
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.7573	.7650	1.1805	1.8163	2.6885	3.8761	5.4801
	V DISP	1.4814	1.6393	2.3965	3.8400	6.5387	11.3262	19.3512
	P(H)	.9983	.1411	.1365	.0155	.0025	.0005	.0001

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 θ into a θ' 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 θ'. After interpolating for range, **accsm** interpolates for crossing angle.

Table 8. Stationary Firer vs Moving Target

(KPH)		250	500	1000	1500	2000	2500	3000
0 deg	H BIAS	.7800	-.2740	-1.4549	-2.5467	-3.5063	-3.8250	-3.2910
	V BIAS	.0000	.0000	.0000	.0000	.0000	.0000	.0000
	H DISP	.6472	.7566	1.1739	1.8114	2.6840	3.8721	5.4983
	V DISP	.7693	.9702	1.9082	3.4875	6.2855	11.1399	19.2091
30 deg	H BIAS							
	V BIAS							
	H DISP							
	V DISP							
60 deg	H BIAS							
	V BIAS							
	H DISP							
	V DISP							
90 deg	H BIAS							
	V BIAS							
	H DISP							
	V DISP							

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	$f = (r - s_{k,1,m})/(s_{k+1,1,m} - s_{k,1,m})$	Fraction of distance into range interval.
i		Index selecting rows in table.
mux, muy	μ_x, μ_y	Horizontal, vertical biases (mils).
ax, ay		Horizontal and vertical angular errors (mils).
tx, ty		Horizontal and vertical total errors (mils).

```
c V7.3
      SUBROUTINE ACC SM(m,r,theta,ax,ay,tx,ty)
c 1   Acc sm: find angular error for stationary-firer moving-tgt.
c     m - 1 iff Blue firer, 2 iff Red firer.
c     r - range from firer to target (m).
c     theta - crossing angle (deg). ??
c     ax, ay - angular error of round (mils).
      include 'common.h'
      real brgs(10), rrgs(10)
      real interp, mux, mux1, mux2, muy, muy1, muy2
      common /comsm/ nsm(2), smtbl(10,17,2)
      save /comsm/
      equivalence (smtbl(1,1,1),brgs(1))
      equivalence (smtbl(1,1,2),rrgs(1))
      interp(f1,f2) = f1 + frac*(f2-f1)
c
      if (trace) print *,'>acc sm'
c     Find right range column
        if (m.eq.1) k=indexx(brgs,nsm(m),r)
        if (m.eq.2) k=indexx(rrgs,nsm(m),r)
        if (k.lt.1) k=1
        if (m.eq.1) frac = (r-brgs(k)) / (brgs(k+1)-brgs(k))
        if (m.eq.2) frac = (r-rrgs(k)) / (rrgs(k+1)-rrgs(k))
c     Find errors
        i = 0
        IF (meth sm.eq.1) THEN
          temp = abs(theta/PI)
          temp = (temp-aint(temp))*PI
          temp = 0.5*PI - abs(0.5*PI-temp)
          i = 4*int(temp*deg/30.)
        ENDIF
          mux1 = interp(smtbl(k,i+2,m), smtbl((k+1),i+2,m))
          muy1 = interp(sm:bl(k,i+3,m), smtbl((k+1),i+3,m))
          tx1 = interp(smtbl(k,i+4,m), smtbl((k+1),i+4,m))
          ty1 = interp(smtb!(k,i+5,m), smtbl((k+1),i+5,m))
        IF (methsm.eq.1) THEN
          mux2 = interp(smtbl(k,i+6,m), smtbl((k+1),i+6,m))
          muy2 = interp(smtbl(k,i+7,m), smtbl((k+1),i+7,m))
          tx2 = interp(smtbl(k,i+8,m), smtbl((k+1),i+8,m))
          ty2 = interp(smtbl(k,i+9,m), smtbl((k+1),i+9,m))
          frac = temp*6./PI
          mux = interp(mux1,mux2)
          muy = interp(muy1,muy2)
          tx = interp(tx1,tx2)
          ty = interp(ty1,ty2)
        ELSE
          mux = mux1
          muy = muy1
          tx = tx1
          ty = ty1
        ENDIF
        ax = mux + rann(tx)
        ay = muy + rann(ty)
      if (trace) print *,'<acc sm'
      END
```

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

MOVING FIRER ADD-ON DISPERSION(mils) ESTIMATE												
Velocity	TT I		TT II		TT III		TT IV		TT V		TT VI	
(KPH)	H	V	H	V	H	V	H	V	H	V	H	V
4	.40	.40	.49	.61	4.8	6.0	.40	.40	8.5	10.6	.40	.49
8	.40	.40	.49	.61	-	-	.40	.40	-	-	.84	1.04
12	.40	.49	.78	.97	-	-	.40	.40	-	-	.83	1.04
16	.45	.56	1.15	1.44	-	-	.43	.66	-	-	1.75	2.20
20	.54	.67	4.30	5.50	-	-	.91	1.14	-	-	-	-
24	.76	.95	-	-	-	-	1.45	1.80	-	-	-	-
32	1.70	2.10	-	-	-	-	10.9	13.7	-	-	-	-
40	-	-	-	-	-	-	-	-	-	-	-	-

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

```
  3   7 M-S Errors for Blue 9999
vel(m/s)      1.11     2.22     3.33     4.44     5.55     6.66     7.77
Adx            .40      .40      .40      .43      .91     1.45     10.9
Ady            .40      .40      .40      .66     1.14     1.80     13.7
```

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

```
  5   8 M-S Errors for Blue 9999
rg(m) ->     500.     1000.    1500.    2000.    2500.    3000.    3500.    4000.
FxBh        .1000     .1000    .1000    .1000    .1000    .1000    .1000    .1000
   v        .0000     .0000    .0000    .0000    .0000    .0000    .0000    .0000
Te h        .8000     .8000    .8000    .8000    .8000    .8000    .8000    .8000
   v        .8000     .8000    .8000    .8000    .8000    .8000    .8000    .8000
```

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:

$$frac = \frac{5-4.44}{5.55-4.44} = 0.504$$

$$Adx = interp(.43,.91) = .43+.504(.91-.43) = .672$$

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

CODE	MATH	COMMENT
m		Side of firer (1=Blue, 2=Red).
r		Range from firer to target (m).
v		Speed of firer at fire time (m/s).
ax,ay		Angular error of round (mils).
rex,rey	r_x, r_y	Dispersion of round (mils).
mstbl	$t_{k,l,m}$	MS data table (mils).
kindms		Number of lines of MS data.
		3 if data is a function of velocity
		5 if data is a function of range
brgs	r_k	Blue ranges in data table (m).
k		Index such that $r_k < r < r_{k+1}$.
frac	$f = (r-r_k)/(r_{k+1}-r_k)$	Fraction of distance into interval.
mux	$\mu_x = t_{k,2,m}+f(t_{k+1,2,m}-t_{k,2,m})$	Horizontal bias (mils).
rex	$r_x = \sqrt{(\nu_x^2+\sigma_x^2+Adx^2)}$	Random error horizontally (mils).
	N	Random draw from the standard normal distribution.
ax	$a_x = \mu_x+r_xN$	Horizontal error of round (mils).

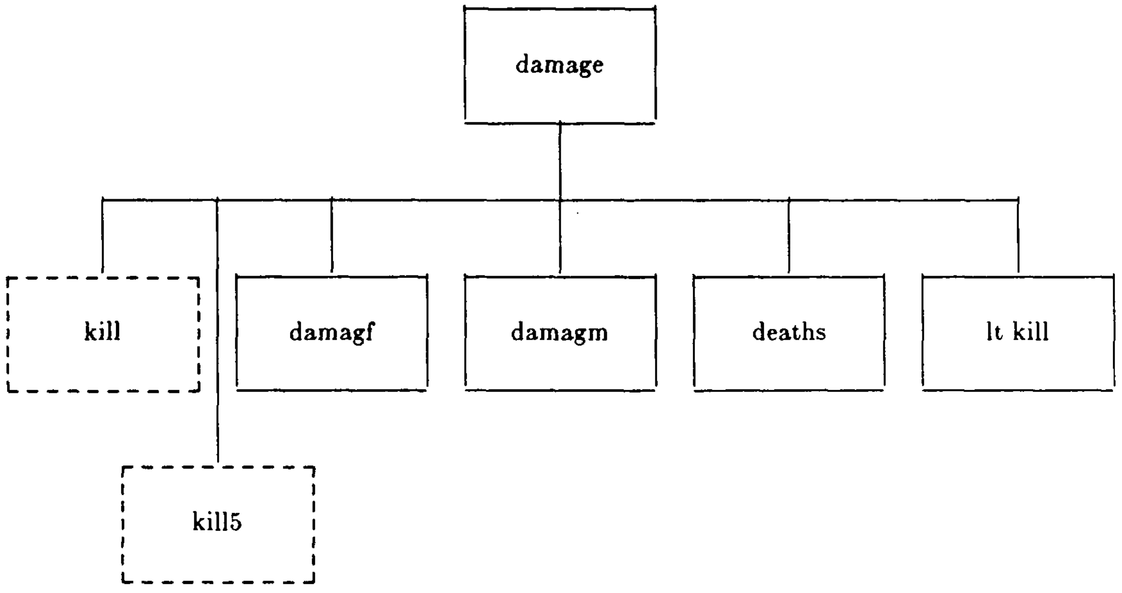
MS error as a function of range. When the input data is in this format, there is no need to access the SS data. Again the routine performs linear interpolation. In this case, it directly finds μ_x, *rex* and a_x, as well as the corresponding y-values.

```
c V7.4
      SUBROUTINE ACC MS(m,r,v,ax,ay,rex,rey)
c 3   Acc ms: find angular error for moving-firer stationary-tgt.
      include 'common.h'
      common /comss/ nss(2), sstbl(10,7,2)
      common /comms/ kindms(2), nms(2), mstbl(10,5,2)
      save /comss/, /comms/
      real bprm(10), rprm(10), brgs(10), rrgs(10), mstbl
      real interp, mux, muy, nux, nuy
      equivalence (sstbl(1,1,1),brgs(1))
      equivalence (sstbl(1,1,2),rrgs(1))
      equivalence (mstbl(1,1,1),bprm(1))
      equivalence (mstbl(1,1,2),rprm(1))
      interp(f1,f2) = f1 + frac*(f2-f1)

      if (trace) print *,'>acc ms'
      IF (kindms(m).eq.3) THEN
c     MS data is a function of velocity.
c       Find SS errors.
          if(m.eq.1)k=indexx(brgs,nss(m),r)
          if(m.eq.2)k=indexx(rrgs,nss(m),r)
          if (k.lt.1) k=1
          if(m.eq.1)frac = (r-brgs(k)) / (brgs(k+1)-brgs(k))
          if(m.eq.2)frac = (r-rrgs(k)) / (rrgs(k+1)-rrgs(k))
          mux = interp(sstbl(k,2,m), sstbl(k+1,2,m))
          muy = interp(sstbl(k,3,m), sstbl(k+1,3,m))
          nux = interp(sstbl(k,4,m), sstbl(k+1,4,m))
          nuy = interp(sstbl(k,5,m), sstbl(k+1,5,m))
          sgx = interp(sstbl(k,6,m), sstbl(k+1,6,m))
          sgy = interp(sstbl(k,7,m), sstbl(k+1,7,m))
c       Find MS add-ons.
          if (m.eq.1) k=indexx(bprm,nms(m),v)
          if (m.eq.2) k=indexx(rprm,nms(m),v)
          if (k.lt.1) k=1
          if (m.eq.1) frac = (v-bprm(k)) / (bprm(k+1)-bprm(k))
          if (m.eq.2) frac = (v-rprm(k)) / (rprm(k+1)-rprm(k))
          Adx= interp(mstbl(k,2,m),mstbl(k+1,2,m))
          Ady= interp(mstbl(k,3,m),mstbl(k+1,3,m))
c       Combine SS data and MS add-ons.
          rex  = sqrt(nux**2 + sgx**2 + Adx**2)
          rey  = sqrt(nuy**2 + sgy**2 + Ady**2)
          ax = mux + rann(rex)
          ay = muy + rann(rey)
      ELSEIF (kindms(m).eq.5) THEN
c     MS data is a function of range.
        if (m.eq.1) k=indexx(bprm,nms(m),r)
        if (m.eq.2) k=indexx(rprm,nms(m),r)
        if (k.lt.1) k=1
        if (m.eq.1) frac = (r-bprm(k)) / (bprm(k+1)-bprm(k))
        if (m.eq.2) frac = (r-rprm(k)) / (rprm(k+1)-rprm(k))
        mux = interp(mstbl(k,2,m),mstbl(k+1,2,m))
        muy = interp(mstbl(k,3,m),mstbl(k+1,3,m))
        rex= interp(mstbl(k,4,m),mstbl(k+1,4,m))
        rey= interp(mstbl(k,5,m),mstbl(k+1,5,m))
        ax  = mux + rann(rex)
        ay  = muy + rann(rey)
      ELSE
        print*,'ACC MS: There is no ms error type', kindms(m)
        STOP
      ENDIF
      if (trace) print *,'<acc ms'
      END
```

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 iin 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 ıound, 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.

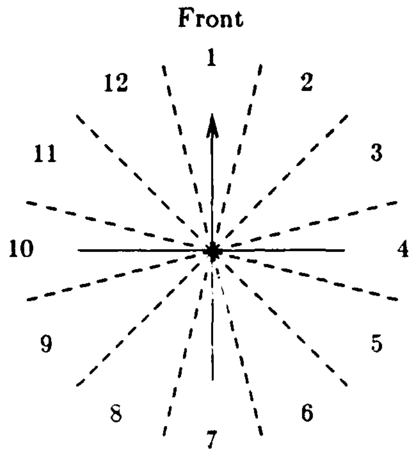


Figure 6 Attack Angle Bands Around the Target

CODE	MATH	COMMENT
t		Time (sec).
I		ID of firer.
it		ID of target.
n		1 if firer is Blue, 2 if Red
m		1 if target is Blue, 2 if Red
k		1 for KE, 2 for HEAT, 4 for missile, 5 for STAFF
theta	θ	Aspect angle of incoming round (rad) $(-\pi \leq \theta \leq \pi)$
theta2	$\theta_2 = 180\theta/\pi$	Aspect angle of incoming round (deg) $(-180 \leq \theta_2 \leq 180)$
nangls(n)	N_n	Number of angles tabled for side n.
	$N_n = 7$	True FF symmetrical target.
nang	$N = \lfloor \theta_2/30+1.5$	1 - 12 for 30 degree bands from head-on to tail-on
rex, rey	x, y	Horizontal, vertical dispersion of rounds (m)
disp	$3.28\sqrt{[x^2+y^2)/2]}$	Root mean square of dispersions (ft)
ndisp		Dispersion bands (1 - 10 ft)
k		Kind of round (1=KE, 2=HEAT, 4=missile, 5=top attack).
life(it)		Status of target (1=ALIVE, 2=MKILL, 3=FKILL, 4=M&FKILL, 5=IKILL, 6=KKILL).
injury		Damage caused by latest round.
injold		Damage incurred before latest round hit.

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 inac.ivity kill a short time in the future, which simulates foes recognizing that an inactive target is no longer a threat.

Table 12. Cumulative Damage

Previous	Damage Caused by Current Round				
Damage	none	M	F	M&F	K
none	-	M	F	M&F	K
M	-	-	M&F	M&F	K
F	-	M&F	-	M&F	K
M&F	-	-	-	-	K
I	-	-	-	-	K
K	-	-	-	-	-

- indicates no change

```
c V7.5
      SUBROUTINE DAMAGE (t, I, it)
C 0   Damage: find if it is hit & schedule effects.
      include 'common.h'
      character*2 kt(6)
      common /cpkh2/ nangls(2), pkh(2,7,12,2,4,11)
      common /cimpct/ x,y,theta, disp
      save /cimpct/, /cpkh2/
      data kt /'no','M-','F-','MF','I-','K-'/
1     format(f8.2,1x,a4,i3,1x,'Hits     ' ,a4,i3,' (no damage).')
2     format(f8.2,1x,a4,i3,1x,a2,'-kills ',a4,i3)
c
      if (trace) print *,'>damage'
      n=army(I)
      m = 3-n
      k = kindrd(n)
      theta2 = theta*DEG
      if (nangls(n).eq.7) theta2=abs(theta2)
      if (theta2.lt.-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 = 3.28*sqrt(0.5*(rex**2+rey**2))
      ndisp = max0(1,min0(10,int(0.5+disp)))
c     Find kill type (if any) and branch accordingly
        if (k.ne.5) injury = kill(I, it, y, ndisp, nang)
        if (k.eq.5) injury = kill5(I, x, y, nang)
        IF(histry) THEN
          if (injury.eq.1) print 1,t,color(n),I,color(m),it
          if (injury.gt.1) print 2,t,color(n),I,kt(injury),color(m),it
        ENDIF

      injold = life(it)
      IF (injury.eq.KKILL .and. injury.ne.injold) THEN
c     Treat first catastrophic kill.
        life(it) = KKILL
        call damagf(t,it,m)
        call damagm(t,it)
        call cancel(it,'ikill ',NULL)
        call newtgt (t,I,it)
        call deaths(t)
      ELSEIF (injury.ne.injold .and. injold.lt.MFKILL) THEN
c     Treat new damage (less than catastrophic).
        IF (injury.eq.MKILL) THEN
          if (injold.eq.FKILL) life(it) = MFKILL
          if (injold.eq.ALIVE) life(it) = MKILL
          if (injold.eq.ALIVE .or. injold.eq.FKILL) call damagm (t,it)
        ELSE IF (injury.eq.FKILL) THEN
          if (injold.eq.ALIVE) life(it)=FKILL
          if (injold.eq.MKILL) life(it)=MFKILL
          call damagf(t,it,m)
        ELSE IF (injury.eq.MFKILL) THEN
          if (injold.lt.MFKILL) life(it) = MFKILL
          if (injold.ne.MKILL) call damagm (t, it)
          if (injold.ne.FKILL) call damagf (t,it,m)
        ENDIF
        if (life(it).eq.MFKILL.and.injold.lt.MFKILL)
     1    call skedul(t+tbump(n),it,'ikill ',NULL)
      ENDIF
      if (trace) print *,'<damage'
      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.

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

```
c V7.1
      SUBROUTINE DAMAGF (t,it,n)
c     Damagf - Discard activities due to firepower kill.
      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,nblu+nred
          fot(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.BLU .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**.

CODE	MATH	COMMENT
t		Time (sec).
it		ID of target.
vx0	v	Current speed of target
motion	m	1 = slowing, 2 = stopped, 3 = accelerating, 4 = cruising
SLOWNG	s	1
	$v_{it} = 0$	Implies it is stopped
	$m_{it} = s$	Implies it is slowing down

```
c V7.4
      SUBROUTINE DAMAGM (t, it)
c 9   Damagm - Simulate mobility kill on the tgt.
      include 'common.h'
      logical sos
c     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 skedul (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.

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

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)
c 0   Deaths: Find death toll on each side. A tank is considered
c     dead if it is I-killed, K-killed, or F-killed & hidden.
      include 'common.h'
      logical dead1, dead2
      integer dead(2)
1     format (i3,' Blu dead,',i3,' Red dead.')
c
      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))
     1 call skedul(t+5.,NULL,'finish',NULL)
      if (trace) print *,'<deaths'
      END
```

8.5 LateKl: 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.

CODE	COMMENT
t	Time (sec).
tgt	ID of target (integer)
jj	Unused dummy variable
firer	ID of target's first foe. (Any will do.)
life(tgt)	Status of target is bumped to I-killed.

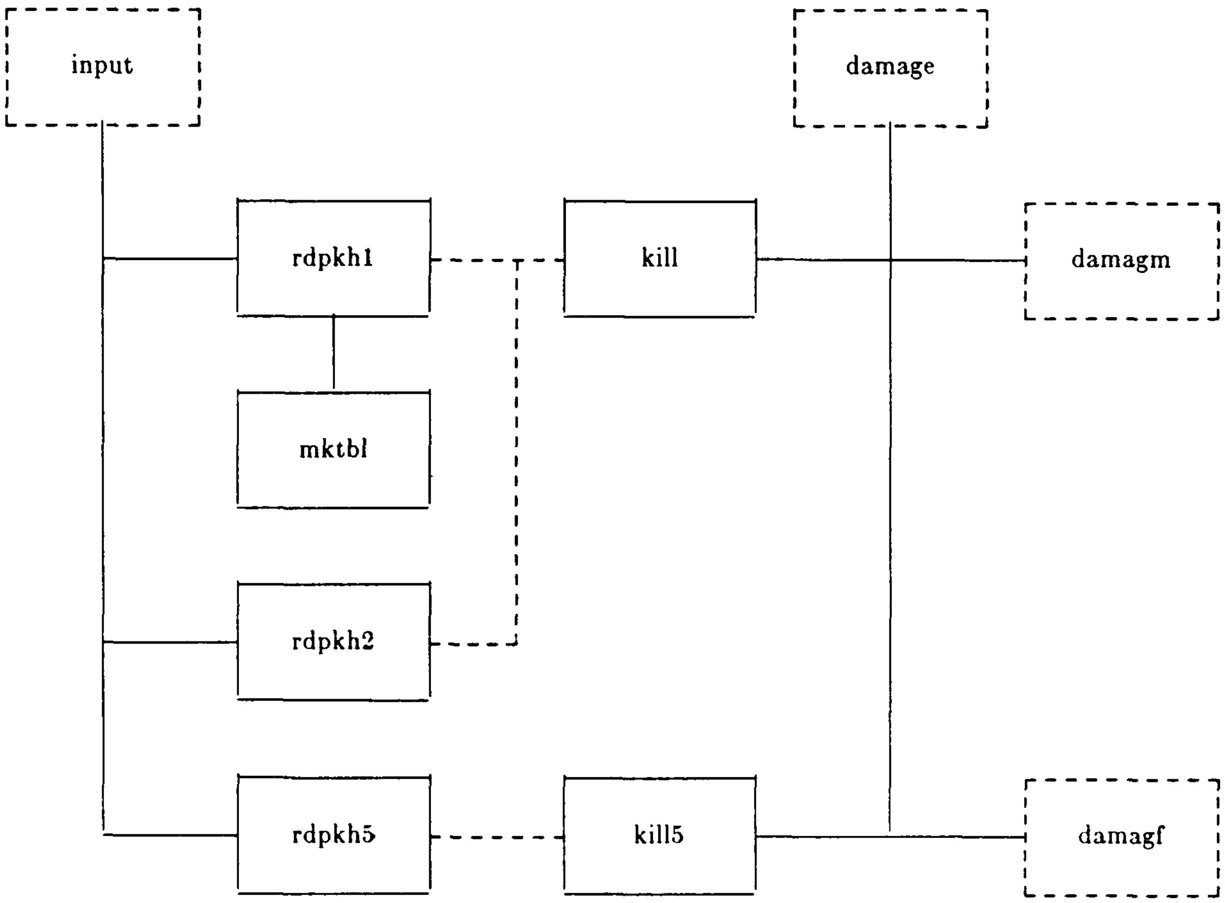
```
c V7.2
      SUBROUTINE LATE KL (t, tgt,jj)
c 3   Late kl: Simulate recognition of m&f kill after period of inactivity.
      include 'common.h'
      integer firer, tgt
1     format(f8.2,1x,a4,i3,' I-killed.')
c
      if (trace) print *,'>latekl'
      if (histry) print 1, t, color(army(tgt)), tgt
      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
```

INTENTIONALLY LEFT BLANK.

9. LETHALITY ROUTINES

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



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 **rd pkh1** and **mk tbl** and the variables are:

table(4,12) - table(i,j) is the lethality of ith type at jth aspect angle.
echo(2,7,7) - echo(i,j,k) lethality data for ith side, jth range, and kth type.
jrg(2,7) - jrg(i,j) ith side, jth range (meters).
jdisp(2,7) - jdisp(i,j) ith side's jth dispersion (ft).

The block /cpkh2/ is known only to **rd pkh1, rd pkh2,** and **kill**. The read routines store the processed lethality data in its variables. The arrays are:

nangls(2) - nangles(i) contains the number of angles of data stored for the ith 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 RdPkhl: 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)

Kill	Aspect Angle					
Level	0	30	...	180	...	330
M F M|F K						

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 = probability of a mobility kill given a hit
F = probability of a firepower kill given a hit
E = probability of either mobility or firepower kill as above
K = probability of a catastrophic kill given a hit
M'= probability of a mobility kill only given a hit
F'= probability of a firepower kill only given a hit
B = 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:

$M' = E - F$
$F' = E - M$
$B = F - F' - K \quad (= M - M' - K = E - M' - F' - K)$
$K = K$

The Venn diagrams in Figure 7 illustrate how the raw kills are separated into specific kills.

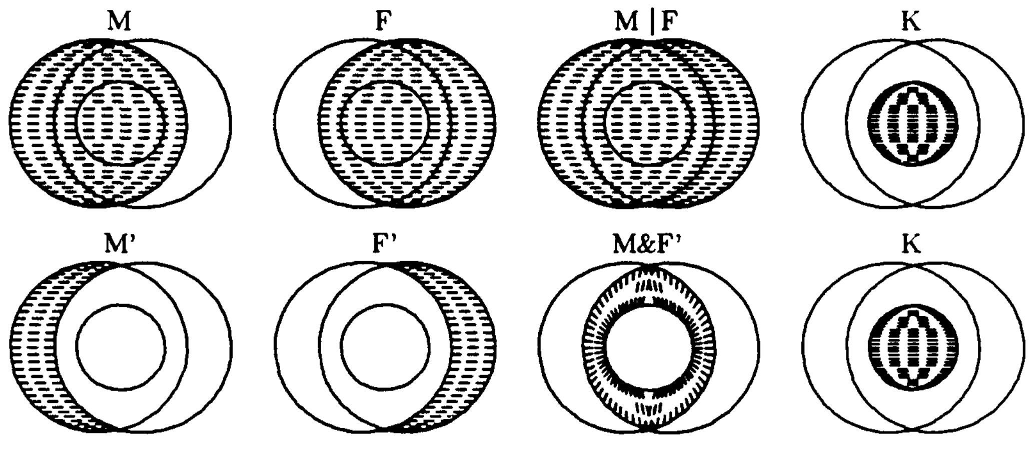


Figure 7. Raw and Specific Probabilities of Kill.

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.

CODE	COMMENT
dbname	Character name of lethality data file
narmy	1 for Blue, 2 for Red
iecho	Echos sample lethality data if > 2
fandm	Probability of M and F kill but not K-kill.
header	Character header line for sample data
irg	1 to 7 for the appropriate range bands. Normally, they are 0, 500, 1000, ..., 3000 meters.
	ktgt, kproj, krg, kexp, kdisp, ktype are read in but never set. They may be used in the future for error checking.
line	Line number (for reporting bad data lines)
mang	Number of column containing aspect angle data.
maxexp	2 (Hull defilade and fully exposed).
mdisp	1 to 10 for 1 to 10 foot dispersions, 11 for uniform dispersion.
nerr	Number of errors found
nhdfe	1 for hull defilade, 2 for fully exposed data
nrrgs	KE lethality is a function of range, so read 7 sets of data for KE. Only one is required for HEAT rounds.
nrow	Row of sample data to fill in table.
onlym	Probability of M-kill only.
onlyf	Probability of F-kill only.

```
c V7.4
      SUBROUTINE RDPKH1 (dbname, narmy, iecho)
c 3   Rd pkh: read & write probability-of-kill-given-a-hit data.
      include 'common.h'
      character*32 dbname
      character*78 header
      common /cpkh/ table (4,12), echo(2,7,7), jrg(2,7), jdisp(2,7)
      common /cpkh2/ nangls(2), pkh(2,7,12,2,4,11)
1     format (2i4,i5,i2,i3,i2,1x,12f6.3)
2     format (i7,7f7.3,i4)
3     format (i2,a78)
4     format ('    Rg   ',7('-'),'Sample Head-on ',
   1  'Kill Probabilities',7('-'), ' disp',/,'   ',
   2  ' (m)   M     F    M|F    K   Monly  Fonly   M&F  (ft)')
5     format(/,20x,a20)
8     format(' Rdpkh: line',i4,i2,'nd value is bad (=',f6.3,')')
9     format(' Rdpkh: lines',i4,'-',i4,' col',i3,'give m,f,m&f=',
   1    3f6.3)
c
      if (trace) write(*,*)'>rdpkh'
      open (4, file=dbname, status='old')
      rewind 4
      read(4,3) nangl, header
      if (iecho.gt.1) write(*,3) header
      line = 1
      nerrs = 0
      nr rgs=7
      nangls(narmy) = nangl
      if(kind rd(narmy).gt.1)nr rgs=1
      DO 80 irg=1,nr rgs
        maxexp = 2
        DO 70 nhdfe=1,maxexp
```

```
          DO 60 mdisp=1,11
           DO 30 i=1,4
            read (4,1) ktgt,kproj,krg,kexp,kdisp,ktype,
     1        (table(i,j),j=1,nangl)
30         CONTINUE
c          Check table for pkh's <0 or >1.
           DO 40 i=1,4
            line = line+1
            DO 35 j=1,nangl
             IF (table(i,j).lt.0. .or. table(i,j).gt.1.) THEN
              write(*,8)line,j,table(i,j)
              nerrs = nerrs+1
              if (nerrs.gt.20) write(*,*)'RDPKH: too many errors'
              if (nerrs.gt.20) STOP
             ENDIF
35          CONTINUE
40         CONTINUE
c          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,mang)-table(4,mang)-onlyf
c
             pkh(narmy,irg,mang,nndfe,1,mdisp)=onlym
             pkh(narmy,irg,mang,nhdfe,2,mdisp)=onlyf
             pkh(narmy,irg,mang,nhdfe,3,mdisp) = fandm
             pkh(narmy,irg,mang,nhdfe,4,mdisp) =
     1         table(4,mang)
             IF (onlym.lt.0 .or. onlyf.lt.0 .or. fandm.lt.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.7) nrow = irg
             if (mdisp.eq.1 .and. nrrgs.eq.1) call mk tbl(
     1         onlym,onlyf,fandm,mdisp,nhdfe,1)
             if (mdisp.le.6 .and. nrrgs.eq.1) nrow = mdisp+1
             if (nrow.gt.0) call mk tbl(
     1         onlym,onlyf,fandm,mdisp,nhdfe,nrow)
            ENDIF
60        CONTINUE
70       CONTINUE
80      CONTINUE
        close(4)
        IF (iecho.ge.2) THEN
         write(*,5)'Hull Defilade'
         write(*,4)
         write(*,2)(jrg(1,i),(echo(1,i,j),j=1,7),jdisp(1,i),i=1,7)
         write(*,5)'Fully Exposed'
         write(*,4)
         write(*,2)(jrg(2,i),(echo(2,i,j),j=1,7),jdisp(2,i),i=1,7)
        ENDIF
        IF (nerrs.gt.0) THEN
        write(*,*) 'RDPKH: errors found in data so program stops.'
         STOP
        ENDIF
        print*,' Pkh file is:',dbname
90      if (trace) write(*,*)'<rdpkh'
        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

Range	Kill Probability by Category							Dispersion
(m)	M-kill	F-kill	MF-kill	K-kill	M'-kill	F'-kill	M&F'-kill	(ft)
0								
500								
1000								
1500								
2000								
2500								
3000								

```
c V7.1
      SUBROUTINE MK TBL (onlym,onlyf,fandm,mdisp,k,nrow)
c 3   Mk tbl: make head-on pkh table for echo.
      include 'common.h'
      common /cpkh/ table (4,12), echo(2,7,7), jrg(2,7), jdisp(2,7)
c
      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
90    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     ldh... 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/ nangls(2), pkh(2,7,12,2,4,11)
1     format(6f5.2)

      if (trace) print *,'>rdpkh2'
      read(dbname,1)pk1,pk2,pk3,pk4,pk5,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,1,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:

CODE	COMMENT
y1	The distance from the target centroid to the lower bound of the fans.
y2	The distance from the target centroid to the 'base' of the fans.
y3	The distance from the target centroid to the aim point.
bands	The number of bands in a fan (1-7).
sectrs	The number of sectors in a band (1-4).
fans	The number of fans (7 or 12).

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

band/sector/kill	FAN DATA						
	0 deg	30 deg	60 deg	90 deg	120 deg	150 deg	180 deg
2.,8.,20.,	distances to fan, base, aim point						
7,4,7,	#bands, sectors, fans						
45.0,52.5,67.5,82.5,	sector lower bounds						
60 45.0 m	0.35	0.35	0.35	0.35	0.35	0.35	0.35
60 45.0 f	0.92	0.92	0.92	0.92	0.92	0.92	0.92
60 45.0 e	0.94	0.94	0.94	0.94	0.94	0.94	0.94
60 45.0 k	0.30	0.30	0.30	0.30	0.30	0.30	0.30
60 52.5 m	0.45	0.45	0.45	0.45	0.45	0.45	0.45
60 52.5 f	0.88	0.88	0.88	0.88	0.88	0.88	0.88
60 52.5 e	0.90	0.90	0.90	0.90	0.90	0.90	0.90
60 52.5 k	0.25	0.25	0.25	0.25	0.25	0.25	0.25
60 67.5 m	0.52	0.52	0.52	0.52	0.52	0.52	0.52
60 67.5 f	0.74	0.74	0.74	0.74	0.74	0.74	0.74
60 67.5 e	0.78	0.78	0.78	0.78	0.78	0.78	0.78
60 67.5 k	0.20	0.20	0.20	0.20	0.20	0.20	0.20
60 82.5 m	0.62	0.62	0.62	0.62	0.62	0.62	0.62
60 82.5 f	0.62	0.62	0.62	0.62	0.62	0.62	0.62
60 82.5 e	0.70	0.70	0.70	0.70	0.70	0.70	0.70
60 82.5 k	0.15	0.15	0.15	0.15	0.15	0.15	0.15
50 45.0 m	0.35	0.35	0.35	0.35	0.35	0.35	0.35
.							
.							
0 45.0 m	0.35	0.35	0.35	0.35	0.35	0.35	0.35
.							
.							
0 82.5 k	0.15	0.15	0.15	0.15	0.15	0.15	0.15

```
c V7.5
      SUBROUTINE RDPKH5 (dbname,narmy,iecho)
c     Rdpkh5: read STAFF lethality data.
      integer fan, fans, band, bands, sector, sectrs
      logical trace, histry
      real tbl(4,12)
      character*32 dbname
      common /cpkh5/ anglim(4), pkh5(2,7,4,4,12), y1, y2, y3, fans
      common /ctrace/ trace, histry
      save /cpkh5/
1     format(12x,12f6.2)
2     format(5f6.2)
3     format('LETHALITY: head-on fan, middle band.',/,
   1   ' angle  onlym onlyf onlymf k-kill')
c
      if (trace) print *,'>rdpkh5'
c     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
c           Read 1 row for each of the 4 kill categories.
              read(4,1) (tbl(i,fan),fan=1,fans)
5           CONTINUE
            DO 10 fan=1,fans
c           Separate each kill criteria in each fan.
              onlym = tbl(3,fan)-tbl(2,fan)
              onlyf = tbl(3,fan)-tbl(1,fan)
              pkh5(narmy,band,sector,1,fan) = onlym
              pkh5(narmy,band,sector,2,fan) = onlyf
              pkh5(narmy,band,sector,3,fan) =
   1            tbl(3,fan)-onlym-onlyf-tbl(4,fan)
              pkh5(narmy,band,sector,4,fan) = tbl(4,fan)
10          CONTINUE
20        CONTINUE
30      CONTINUE
        close(4)
c     Print sample data in 4th band if desired.
        IF (iecho.gt.1) THEN
          print 3
          DO 60 sector=1,4
            print 2, anglim(sector),(pkh5(narmy,4,sector,i,1),i=1,4)
60        CONTINUE
        ENDIF
      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.**

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

```
c V7.5
      FUNCTION KILL (I, it, y, ndisp, nang)
c 9   Kill: find kill type for a hit on a tgt.
      include 'common.h'
      common /cpkh2/ nangls(2), pkh(2,7,12,2,4,11)
      dimension pksave(4)
c
      if (trace) print *,'>kill'
      pk = ranu(0.0)
      nhdfe = knceal(it)-1
c     Find probabilities for a production run
        m = army(I)
        jrg=min0(6,nrg)+1
        if(kind rd(m).gt.1) jrg=1
        pksave(1) =          pkh(m,jrg,nang,nhdfe,1,ndisp)
        pksave(2) = pksave(1)+pkh(m,jrg,nang,nhdfe,2,ndisp)
        pksave(3) = pksave(2)+pkh(m,jrg,nang,nhdfe,3,ndisp)
        pksave(4) = pksave(3)+pkh(m,jrg,nang,nhdfe,4,ndisp)
c     Find which kill type occurs.
        if (pk.lt.pksave(4)) kill = KKILL
        if (pk.lt.pksave(3)) kill = MFKILL
        if (pk.lt.pksave(2)) kill = FKILL
        if (pk.lt.pksave(1)) kill = MKILL
        if (pk.ge.pksave(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.MFKILL) kshot(m,13) = kshot(m,13)+1
        if (kill.eq.KKILL) 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.

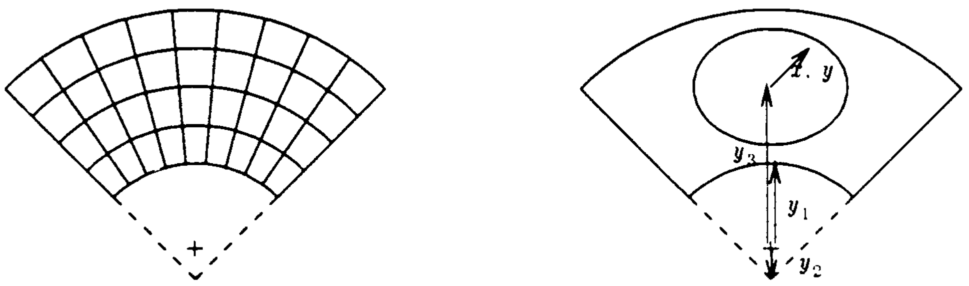
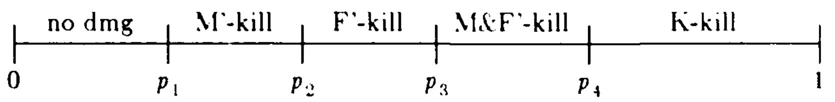


Figure 8. Lethality Fan Above Target

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.



The program uses the following symbols and relations:

CODE	MATH	COMMENT
I		ID of firer (1..NN)
x, y	x, y	Coordinates of round from center of impact (m).
nang		# of angular sector in which round aproaches target.
	y_1	Distance from target to lower edge of fan.
	y_2	Distance from target to apex of fan.
	y_3	Distance round is aimed above target.
h	$h = y+y_2+y_3$	Height of round above fan vertex (m).
d	$d = \sqrt{x^2+h^2}$	Distance of the round from the base of the fan (m).
	$d<y_1+y_2$	Implies round below fan.
	$d\geq y_1+y_2+70$	Implies round is above fan.
e	$\epsilon = atan(h/x)$	Elevation angle of the round.
	$\epsilon>45$	Implies round is left or right of fan.
fan	l	ID of fan perpendicular to approach angle.
fans		# of fans defined by data.
sector	j	ID of sector.
anglim(n)	ϵ_n	Lower angular limit of the nth cell in the fan.
band	i	ID of band.
pk		Random draw from uniform distribution
m		1 if Blue firer, 2 if Red firer
pkh5()	$P_{m,i,j,k,l}$	Probability of kill when the round passes through the i,j cell. (For side m, kill level k, and fan l.)
pksave(4)	p_i	Sums of kill probabilities.
kill5		Damage caused by round. 1 if none, 2 if mobility kill, 3 if firepower kill, 4 if mobility & firepower, 6 if catastrophic.
kshot()		Tallies statistical results.

```
c V7.5
      FUNCTION KILL5 (I, x, y, nang)
c 9   Kill5: find kill type for a STAFF-like round.
c     sector - angle band
c     band - range band (distance above tgt).
      include 'common.h'
c     Uses: trace, army, ALIVE, DEG , KKILL, MFKILL, FKILL, MKILL, kshot
      integer fan, fans, band, sector
      common /cpkh5/ anglim(4), pkh5(2,7,4,4,12), y1, y2, y3, fans
      save /cpkh5/
      dimension pksave(4)
c
      if (trace) print *,'>kill5'
      h = y+y2+y3
      d = sqrt(x**2 + h**2)
      kill5 = ALIVE
      IF (d.gt.y1+y2 .and. d.lt.y1+y2+70.0) THEN
c     Round is within distance limits, proceed.
        e = atan2(h,abs(x))*DEG
        IF (e.gt.45.) THEN
c       Round is within angular limits, proceed.
          fan = nang
          if (nang.gt.fans) fan=14-nang
          sector = indexx(e,4,anglim)
          band = 7-int((d-y1-y2)/10.0)
          pk = ranu(0.0)
          m = army(I)
          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(1)) kill5 = MKILL
          if (pk.gt.pksave(2)) kill5 = FKILL
          if (pk.gt.pksave(3)) kill5 = MFKILL
          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
      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 it's 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_{bsmp} 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_{bsmp} hits or t_{bsmp} 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 **ltkill** 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_{bsmp} seconds or pumps another n_{bsmp} 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_{bsmp}, and n_{bsmp} 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_{bump} additional rounds have been fired at the target, and again permanent disengagement occurs. Note that t_{bump} 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

Tactic	Condition	Event	Action
1	K-kill	Impact	Disengage target forever and select new target.
	M&F-kill	Impact	Schedule a late kill in t seconds. Also set a flag to switch after n_2 more shots.
	M t_{bump} sec	F-kill +	LateKl select new target.
	M&F-kill, and n_{bump} shots	Fire	Disengage target forever and select new target.
2*	Hit	Impact	Disengage target for now and select new target.
3*	Fired n_{rpt} rds	Fire	Disengage target for now and select a new target.

* 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 17. Switching Logic for Simple Missiles

Tactic	Condition	Event	Action
1	K-kill	Impact	Disengage target forever and select new target.
	M&F-kill	Impact	Schedule a late kill in t seconds. Also set a flag to switch after n_2 more shots.
	M t_{bump} sec	F-kill +	LateKl select new target.
	M&F-kill, and n_{bump} shots	Fire	Disengage target forever and select new target.
2*	(Don't use)		
3*	Fired n_1 rds	*Impact*	Disengage target for now and select a new target. (Since msl is guided to impact switching cannot occur until then)

* 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

Tactic	Condition	Event	Action
1	Don't use		
2*	Don't use		
3*	Fired n_{bsmp} rds	*Impact*	Disengage target for now and select a new target. (Since msl is guided to impact switching cannot occur until then)

* n_{bsmp} 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:

VARIABLE	QUESTION
busy(i)	Is i too busy to select a new target?
nrtgt(i)	What's firer i's latest target?
fot(i,j)	Is firer i engaging target j?
mot(i,j)	Does firer i have a missile enroute to target j?
loaded	Are firer i's guidance channels full? (missile firers)
	Is firer i engaging a target now? (gun systems)
empty(i)	Is firer i's upraised missile pod empty? (missiles)
	Is firer i out of ammo? (guns or missiles)

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, **selecs** 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.

	busy	*nrtgt*	*fot*	mot	loaded (local)	*empty*
Init	unset	unset	unset	unset		unset
Selecs	set	-	-	-	-	-
Select	unset	set	set	-	-	-
Fire	unset	-	-	set	-	-
-Frdmsl	-	-	unset	-	-	set
-Frdssg	*unset*	-	-	-	-	*set*
Impact	-	*unset*	*unset*	unset	-	-
-Diseng	-	unset	unset	-	-	-
--Frdmsl						
Reload	-	*unset*	-	-	-	unset
Newtgt	*unset*	*unset*	unset	-	-	*set*
Abort	-	-	-	unset	-	-
Damagf	-	unset	unset	-	-	-
Vanter	-	unset	unset	-	-	-
Vansmk	unset	-	-	-	-	-

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 engagment, 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 fixed 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 4km. 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 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.

CODE	COMMENT
t	Time (sec).
I	ID of firer.
it	ID of target.
drop	True IFF target should be dropped (logical).
take	True IFF new target should be selected (logical).
mytgt	ID of firer's actual 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.
nrds(m)	Number of rounds firers on side m start with.
inbrst	True IFF a burst firer is in the middle of a burst. (logical)
nrpb(m)	Number of rounds in a burst. If < 2 it is a single shot gun.
nrib(I)	Number of rounds fired in the burst so far.
on tgt	True IFF firer is engaging this target. (logical)
kind	Kind of round. (1=KE, 2=HEAT, 3 unused, 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(I)	Number of rounds firer has fired at target since engaging it.

```
c V7.3
      SUBROUTINE DIS ENG (t, I, it,drop,take)
c 7   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.
      include 'common.h'
      logical in brst, hav amo, on tgt, drop, take, cango
3     format (f8.2,1x,a4,i3,' dis-engs ',a4,i3,20x,'#tgts=',i2)
c
      if (trace) print *,' -diseng'
```

```
c     Set useful local variables
       my tgt = nrtgt(I)
       m = army(I)
       n = 3-m
       hav amo = nrd(I).lt.nrds(m)
       inbrst = nrpb(m).gt.1 .and. (0.ne.mod(nrib(I),
     1   nrpb(m)))
       if (it.eq.FLS TGT) on tgt = .true.
       if (it.ne.FLS TGT) on tgt = fot(I,it) .or.
     1    (kindrd(m).eq.4 .and. mot(I,it))
```

```
        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
c           Single shot gun system or STAFF fire & forget system.
              IF (it.ne.FLS TGT) THEN
                if (fot(I,it)) call cancel (I,'fire  ',it)
                fot(I,it) = .false.
              ENDIF
              hav amo = nrd(I).lt.nrds(m)
              IF (hav amo) THEN
                thuman = 2.*exp(rann(0.5))
                call selecs(t,I,thuman)
              ELSEIF (can go(I,t).and.ishtfs(m).gt.0) THEN
c             Firer moves on.
                if(histry)print 3, t,color(m),I,
     1            color(n),it,nchan(I)
                call skedul(t,I,'accel ',NULL)
              ENDIF
              nrot(I) = 0
              nrtgt(I) = 0
            ELSE
c           Burst fire gun system.
              print *,'DISENG: Not implemented for burst fire guns.'
              STOP
            ENDIF
          ELSEIF(kind.eq.4) THEN
c         Guided missile system.
            if (drop) nchan(I) = nchan(I)-1
            IF (it.ne.FLS TGT) THEN
              IF (fot(I,it)) THEN
                call cancel(I,'fire  ',it)
                fot(I,it) = .false.
              ENDIF
            ENDIF
            if(histry)print 3, t,color(m),I,
     1        color(n),it,nchan(I)
c           Firer attempts to select a new target
              IF (take) THEN
                call frdmsl(t,I,it,m)
c               The firer begins to select a new target right now and
c               finishes the selection in a few seconds.
              ENDIF
          ENDIF
        ENDIF
        IF (.not.repeat) THEN
          repeat = .true.
          call skedul (t+.01,0,'search',NULL)
        ENDIF
        if (trace) print *,'<diseng'
        END
```

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.

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.
nrds(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 unused, 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(I)	Number of rounds firer has fired at target since engaging it.

```
c V7.5
      SUBROUTINE NEWTGT (t, I, it)
c 3   Newtgt: redirect all 'attackers' of it to a new target.
c     Newtgt called for non-false tgts only and only if it condition
c     warrants it. It should only be called if tgt is V-killed,
c     vanishes, or hides.
c     Maybe it should be called if the tgt is I-killed by a gun system.
      include 'common.h'
      integer first
      logical loaded, hav amo, cango
1     format(f8.2,1x,a4,i3,' dis-engs ',a4,i3,20x,'#tgts=',i2)
2     format(f8.2, 1x, a4, i3, ' begins to reload.')
c
      if (trace) print *,'> newtgt'
c     Find first and last 'attacker'
        first = 1
        if (I.gt.nblu) first = nblu+1
        last = nblu
        if (I.gt.nblu) last = nblu+nred
      m = army(first)
      n = 3-m
      kind = kindrd(m)
```

```
        nrpb2 = nrpb(m)
        DO 20 j=first, last
          IF ((mot(j,it) .or. fot(j,it)) .and. life(j).lt.FKILL) THEN
            IF (kind.le.2 .or. kind.eq.5) THEN
c             Single shot gun system or other fire & forget system.
              IF (nrpb(m).le.1) THEN
c               Single shot gun system.
                call cancel(j,'fire  ',it)
                if (nrtgt(j).eq.it) busy(j) = .false.
                if (nrtgt(j).eq.it) nrtgt(j) = 0
                hav amo = nrd(j).lt.nrds(m)
                IF (hav amo) THEN
                  thuman = 2.*exp(rann(0.5))
                  call selecs(t,j,thuman)
                ELSEIF (can go(j,t).and.ishtfs(m).gt.0) THEN
c                 Move out
                  call skedul(t,j,'accel ',NULL)
                ENDIF
                nrot(j) = 0
                fot(j,it) = .false.
                if (histry) print 1, t, color(m), j,
     1            color(n), it, nchan(j)
              ELSE
c               Burst fire gun.
                print *,'NEWTGT: Not implemented for burst fire.'
                STOP
              ENDIF
            ELSEIF (kind.eq.4) THEN
c             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. nchans(m)
              IF ((.not.empty(j) .and. mot(j,it) .and. loaded) .or.
     1          (.not.empty(j) .and. fot(j,it))) THEN
                IF ((mod(nrd(j),nipods(m)) .gt. 0)  .or.
     1            fot(j,it)) THEN
c                 More rds in pod
                  IF (fot(j,it)) THEN
                    call cancel(j,'select',NULL)
                    busy(j) = .false.
                    fot(j,it) = .false.
                  ENDIF
C                 if (it.ne.FLS TGT) fot(j,it) = .false.
                  thuman = 2.0*exp(rann(0.5))
                  call selecs(t,j,thuman)
                ELSE
c                 Treat empty missile pod
                  empty(j) = .true.
                  call cancel(j,'fire  ',it)
                  call cancel(j,'select',NULL)
                  busy(j) = .false.
                  nrot(j) = 0
c                 shud htf that is slowing to engage speed up now?
                  call skedul (t+trelod(m),j,'reload',NULL)
                if (histry) print 2,t,color(m),j
                ENDIF
              ENDIF
              nchan(j) = nchan(j) -1
              nrtgt(j) = 0
              fot(j,it) = .false.
            ENDIF
          ENDIF
          if (see(j,it)) ndet(j) = ndet(j) - 1
          see(j,it) = .false.
20      CONTINUE
        IF (.not.repeat) THEN
          repeat = .true.
          call skedul (t+.01,0,'search',NULL)
        ENDIF
        if (trace) print *,'< newtgt'
        END
```

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:

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

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.

```
c V7.4
      SUBROUTINE ABORT (t,I,it)
c 6   Abort: abort msl from I to it (to all its if it=0)
      include 'common.h'
      logical defndr
1     format(f8.2,1x,a4,i3,' msl for  ',a4,i3,' aborted.')
2     format ('ABORT: I,it,i,chanel=',4i3)
3     format ('ABORT: msl approaching it',i3)
c
      if (trace) print *,'>abort'
      m = army(I)
      it = 3-m
      DO 20 k=1,5
c     Check all 5 missile pointers for this firer
        msl = chanel(m,I,k)
        if (keym(19).gt.0) print 2, I, it, k, msl
        IF (msl.gt.0) THEN
c       Missile found (pointer is non-zero)
          msl tgt = a(msl+1)
          if (keym(19).gt.0) print 3, msl tgt
          IF (it.eq.0 .or. it.eq.msl tgt) THEN
c           Abort this missile
            kshot(m,3) = kshot(m,3)+1
            call cancel (msl,'impact', NULL)
            chanel(m,I,k) = 0
            if (msl tgt.ne.FLS TGT) mot(I,msl tgt) = .false.
c           Release area for storage of missile data
              a(msl) = -a(msl)
              if (histry) print 1,t,color(m),I,
     1          color(it),msl tgt
c           Pop-down to reload if defender, pod empty, & fully alive
            defndr = (scene.eq.BATTAK .and. m.eq.RED) .or.
     1        (scene.eq.RATTAK .and. m.eq.BLU)
            if (defndr.and.empty(I).and.(life(I).le.ALIVE))
     2        call skedul(t,I,'popdn ',NULL)
          ENDIF
        ENDIF
20    CONTINUE
      if (trace) print *,'< abort'
      END
```

INTENTIONALLY LEFT BLANK

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[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.

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

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.

```
c V7.5
      SUBROUTINE DEPLOY
c     Deploy: set initial exposure, position, and motion.
      include 'common.h'
      common /inpwav/ keyd(5), nwaves, neval, nused(3000)

      if (trace) print*, ' >deploy'
c     Initialize Blue tanks.
```

```
      sp = 100.0
      c = -sp*(1+nblu)/2.0
      DO 20 i=1,nblu
        army(I) = BLU
        knceal(I) = FE
        if (scene.eq.RATTAK) knceal(I) = HD
        t0(I) = 0.0
        x0(I) = 0.0
```

```
          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.1) nrd(i) = nused(neval+1)
          if (nwaves.gt.1) neval = neval+nblu
20      CONTINUE
c     Initialize Red tanks.
        sp = 100.0
        c = -sp*(1+nred)/2.0
        DO 30 j=1,nred
          I = nblu+j
          army(I) = RED
          knceal(I) = FE
          if (scene.eq.BATTAK) knceal(I) = HD
          t0(I) = 0.0
          x0(I) = rg0
          y0(I) = sp*j+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
        wvlth(BLU)  = 50.
        wvlth(RED)  = 50.
      ampl(BLU) = accmax(BLU)/(TWOPI*speed(BLU)/wvlth(BLU))**2
      ampl(RED) = accmax(RED)/(TWOPI*speed(RED)/wvlth(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:

$$dt = |v_y|/decel$$

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.

CODE	MATH	COMMENT
t	t	Time (sec)
I		ID of tank.
kindmv		Motion of tank (1=slowing, 2=stationary, 3=accelerating, 4=cruising).
n		Side of tank (1=Blue, 2=Red).
decel(n)	d	Deceleration (ms^{-2}).
vy	v_y	Current speed (m/s).
dt	$\Delta t = \|v_y\|/d$	Time to halt from acceleration (sec).
dt	$\Delta t = s/d$	Time to halt from cruise speed (sec).

```
c V7.4
      SUBROUTINE SLOW UP (t, I)
c 8   Slow up: simulate tank starting to slow down.
      include 'common.h'
1     format (f8.2,1x,a4,i3,' continues to slow up.')
2     format (f8.2,1x,a4,i3,' would slow up if it weren''t',
   1    ' already stopped.')
3     format (f8.2,1x,a4,i3,' brakes',11x,'(was accelerating)')
4     format (f8.2,1x,a4,i3,' brakes',11x,'(was cruising)')
c
      if (trace) print *,' >slowup'
      kind mv = motion(I)
      n = army(I)
      IF (kind mv.eq.SLOWNG) THEN
        if(histry)print 1, t, color(n), I
      ELSEIF (kind mv.eq.STATNY) THEN
        if(histry)print 2, t, color(n), I
      ELSEIF (kind mv.eq.ACCELG) THEN
        if(histry)print 3, t, color(n), I
        call path (I,t,motion(I),0.0,x,y,vx,vy)
        dt = abs(vy)/decel(n)
        motion(I) = SLOWNG
        call skedul(t+dt,I,'halt  ', NULL)
      ELSEIF (kind mv.eq.MAXVL) THEN
        if(histry)print 4, t, color(n), I
        call path (I,t,motion(I),0.0,x,y,vx,vy)
c       schedule halt time
        dt = speed(n)/decel(n)
        call skedul(t+dt,I,'halt  ', NULL)
        motion(I) = SLOWNG
      ENDIF
      if (trace) print *,'< slowup'
      END
```

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 (ishtf(m)=1), and it can shoot (life(I)<FKILL), and it has ammo (nrd(I).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 the first round (sec),
$N[0.5]$ is a draw from a normal distribution with $\sigma = 0.5$.

But wait, $dt = \max(.01, dt-3)$ What'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.

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

```
c V7.5
      SUBROUTINE HALT (t, I)
c 7   Halt: simulate tank halting.
      include 'common.h'
      logical cango, threat
1     format (f8.2,1x,a4,i3,' halts',12x,'(x=',f8.1,'  y=',f8.1',)')
c
      if (trace) print *,'>halt'
      if(invisb.eq.1)call cancel (I,'vanish',NULL)
      m = army(I)
      call path (I,t,motion(I),0.0,x,y,vx,vy)
      if (histry) print 1, t, color(m), I, x, y
      motion(I) = STATNY
c     See if fire is a halt-to-fire-system and can still shoot
      IF (ishtfs(m).eq.1 .and.
     1  life(I).lt.FKILL .and. nrd(I).lt.nrds(m)) THEN
c       This is a halt-to-fire system. schedule firing if target is
c       still available.
        if (nrtgt(I).eq.FLSTGT) threat = knceal(I).ne.FD
        if (nrtgt(I).gt.0) threat = fot(I,nrtgt(I))
        IF (threat) THEN
          dt = tfirst(m,nrg)*exp(rann(0.5))
          dt = amax1(.01,dt-3.0)
          prev rd(I) = 1
          nrib(I) = 0
          it=nrtgt(I)
          call skedul (t+dt, I, 'fire  ', it)
        ELSE
c       Move firer because it's target has vanished.
          if(cango(I,t))call skedul (t, I, 'accel ', NULL)
        ENDIF
      ENDIF
      if (trace) print *,'<halt'
c     Move firer if tgt vanished may be redundant.
      END
```

11.4 Accel: Begin Acceleration. **Accelf** 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):

$$dt = (speed - |v_y|)/accel$$

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.

CODE	MATH	COMMENT
t	t	Time (sec).
I		ID of tank.
life(I)		1 if ALIVE, 2 if M-killed, etc
invisb		1 if terrain, 2 if smoke causes NLOS.
knceal(I)		1 if FD, 2 if HD, 3 if FE.
n		Side of tank (1=Blue, 2=Red).
motion(I)		1=Slowing, 2=stationary,
		3=accelerating, 4=cruising.
x,y		Coordinates of tank (m).
vx,vy	v_x, v_y	Velocity of tank (m/s).
speed	s	Cruise speed (m/s).
accel(i)	a	Acceleration for side m (m/s^2).
dt	$\Delta t = (s-v)/a$	Time to reach cruise speed from current velocity (sec).
dt	$\Delta t = v/a$	Time to reach cruise speed from halt (sec).

```
c V7.4
      SUBROUTINE ACCELF (t, I)
c 9   Accelf: simulate tank starting to accelerate.
      include 'common.h'
1     format (f8.2,1x,a4,i3,' speed up',9x,a)
c
      if (trace) print *,'>accel'
      if(life(I).ne.FKILL.and.invisb.eq.1.and.knceal(I).ne.FD)
     1  call skedul (t,I,'vanish',NULL)
      n = army(I)
      IF (motion(I).eq.SLOWNG) THEN
        if (histry) print 1, t, color(n), I, '(was slowing)'
        call path(I,t,motion(I),0.0,x,y,vx,vy)
        dt = (speed(n)-abs(vy))/accel(n)
        call skedul (t+dt,I,'maxvel',NULL)
        motion(I) = ACCELG
      ELSEIF (motion(I).eq.STATNY) THEN
        if (histry) print 1, t, color(n), I, '(was halted)'
        call path(I,t,motion(I),0.0,x,y,vx,vy)
c       schedule time full velocity reached (max vel)
          dt = speed(n)/accel(army(I))
          call skedul(t+dt,I,'maxvel',NULL)
          motion(I) = ACCELG
      ELSEIF (motion(I).eq.ACCELG) THEN
        if (histry) print 1, t, color(n), I, '(was speeding up)'
      ELSEIF (motion(I).eq.MAXVL) THEN
        if (histry) print 1, t, color(n), I, '(is cruising)'
      ENDIF
      if (trace) print *,'<accel'
      END
```

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**.

CODE	COMMENT
t	Time (sec).
I	ID of tank.
motion(I)	1=slowing, 2=halted, 3=accelerating, 4=cruising
it	ID of tank's target.

```
c V7.2
      SUBROUTINE MAX VEL(t, I)
c 6   Max vel: simulate tank reaching cruise speed.
      include 'common.h'
1     format (f8.2,1x,a4,i3,' at full speed.')
c
      if (trace) print *,'>maxvel'
      if (histry) print 1, t, color(army(I)), I
      call path(I,t,motion(I),0.0,x,y,vx,vy)
      motion(I) = MAXVL
      it = nrtgt(I)
      IF (it.eq.FLSTGT) THEN
        call engage(t,t,I,it)
      ELSEIF (life(it).lt.IKILL) THEN
        call engage(t,t,I,it)
      ENDIF
      if (trace) print *,'<maxvel'
      END
```

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:

Tank	t_0	x_0	y_0	v_{x_0}	v_{y_0}
1					
2					
...					
NN					

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 updat[illegible]d [illegible]er than delt. For purposes of this routine, defenders and tanks in a meeting engagement don[illegible] [illegible]e, [illegible]o their position and velocity never needs to be updated. When attackers are mobility kill[illegible] [illegible] [illegible] is updated, so it doesn't have to be updated later for those tanks. This means **path** must be [illegible] just before the program changes life(I) to indicate a mobility kill or worse. The last update time for the ith tank is t0(I) and the current time is t, so the time since last update is: dt = t-t0(I). If this is greater than delt then the data is old. If the data needs to be updated, the code stores the current time t as the update time t0(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.

CODE	MATH	COMMENT
decel(n)	a	Deceleration of tank (m/s**2).
dt	Δt	Time since last update (s).
dv	$\Delta v = a\Delta t$	Velocity change (m/s).
dv	$\Delta v = -\Delta v$	Change sign because Red moves in negative direction.
x0(I)	x	Old x coordinate (m).
vx0(I)	v	Old velocity (m/s).
x0(I)	$x' = \Delta t(v - \Delta v/2)$	New x coordinate (m).
v	$v' = v - \Delta v$	New velocity (m/s)
	$\|v\| < 0$	Force velocity to zero if round off gives almost zero speed.
		This avoids choosing moving accuracy data for stationary tank.
vx0(I)	$v'' = v'$	Save new velocity in 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.

CODE	MATH	COMMENT
dv	$\Delta v = -\Delta v$	Change sign because Red moves in negative direction
vx0(I)	v	Old velocity (m/s).
x0(I)	$x' = \Delta t(v - \Delta v/2)$	New x coordinate (m).
v	$v' = v - \Delta v$	New velocity (m/s)

If the tank is at cruise speed the code finds the new position and velocity as shown in the chart below.

CODE	MATH	COMMENT
x0(I)	$x' = x+v\Delta t$	New x coordinate (m).
vx0(I)	$v' = s$	Current velocity is combat cruise speed (m/s).

```
c V7.3
      SUBROUTINE PATH (I,t, motio2, delt, x, y, vx, vy)
c 4   Path: search path table for position and vel at time t.
      include 'common.h'
      logical is atkr, kan go, old
c
      if (trace) print *,'>path'
      n = army(I)
      is atkr = (scene.eq.RATTAK .and. n.eq.RED) .or.
     1  (scene.eq.BATTAK .and. n.eq.BLU)
      kan go  = (motio2.ne.STATNY .or.
     1  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
c     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*(vx0(I)-0.5*dv)
          v = vx0(I)-dv
          if (abs(v).lt.0.001) v = 0.0
          vx0(I) = v
        ELSEIF (motio2.eq.STATNY) THEN
          vx0(I) = 0.0
        ELSEIF (motio2.eq.ACCELG) THEN
          dv = accel(n)*dt
          if (n.eq.RED) dv=-dv
          x0(I) = x0(I)+dt*(vx0(I)+0.5*dv)
          vx0(I) = vx0(I)+dv
        ELSEIF(motio2.eq.MAXVL) THEN
          x0(I) = x0(I)+vx0(I)*dt
          vx0(I) = speed(n)
          if (n.eq.RED) vx0(I)=-vx0(I)
        ELSE
          print *,'PATH: no such motion. motio2=',motio2
          STOP
        ENDIF
      ENDIF
      x=x0(I)
      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

Range band	0-750	750-1250	1250-1750	1750-2250	2250-2750	2750-3250	3250-3750	3750-4250
nrg	1	2	3	4	5	6	7	8
Range used	500	1000	1500	2000	2500	3000	3500	4000

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

```
c V7.4
      FUNCTION RGF (t, I, it)
c 3   Rgf: find the position of the firer w.r.t. the tgt.
      include 'common.h'
      common /pathc / xf, yf, xt, yt
      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 (I,t,motion(I),0.0,xf,yf,vfx,vfy)
      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, vfy, 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'.

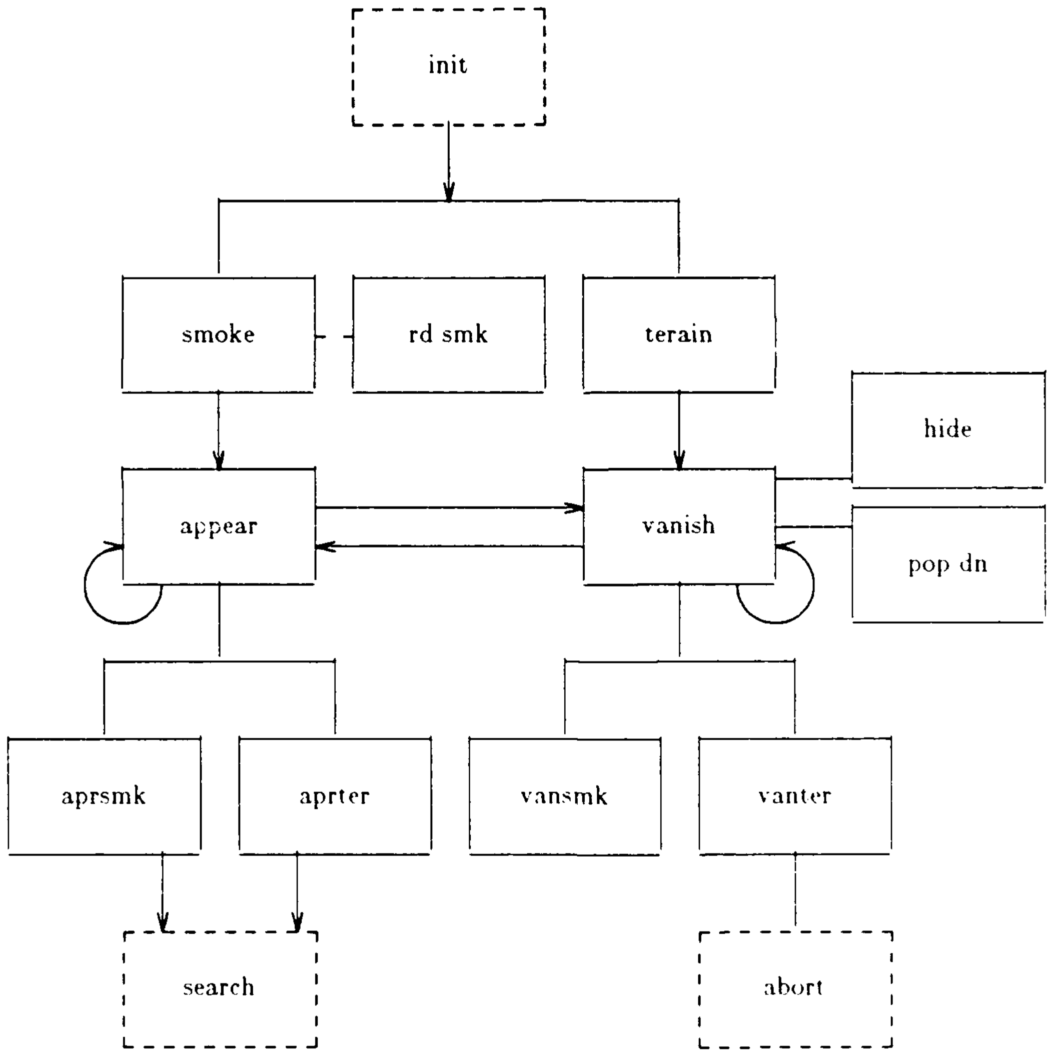
CODE	COMMENT
t	Time (sec).
I	ID of firer.
m	Side of firer (1=Blue, 2=Red).
scene	1=Meeting, 2=Blue attack, 3=Red attack.
isatkr	True iff firer is moving toward foes.
malive	True iff firer mobile.
faster	True iff firer is able to go faster.
cango	True iff firer can go faster than it is going now.

```
c V7.1
      LOGICAL FUNCTION CAN GO (I, t)
c 6   Can go: True iff is stationary and can move.
      include 'common.h'
      logical is atkr, m alive, faster

      m = army(I)
      is atkr = (m.eq.BLU .and. scene.eq.BATTAK) .or.
   1    (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.
   1    motion(I).eq.SLOWNG)
      can go = is atkr .and. m alive .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).
3. Subsequent out-of-view for IR band sensors.
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 20. Time In-View for a Visual Band Sensor

	Range (meters)				
Prob	0	1000	2000	3000	4000
0.00	500.	210.	150.	125.	100.
0.05	500.	190.	85.	80.	75.
0.10	500.	170.	63.	58.	42.
0.15	500.	150.	44.	40.	26.
0.20	500.	130.	33.	28.	10.
0.25	500.	110.	26.	16.	0.
0.30	500.	90.	18.	10.	0.
0.35	500.	80.	13.	4.	0.
0.40	500.	68.	9.	0.	0.
0.45	500.	55.	4.	0.	0.
0.50	500.	42.	0.	0.	0.
0.55	500.	34.	0.	0.	0.
0.60	500.	25.	0.	0.	0.
0.65	500.	15.	0.	0.	0.
0.70	500.	8.	0.	0.	0.
0.75	500.	4.	0.	0.	0.
0.80	500.	0.	0.	0.	0.
0.85	500.	0.	0.	0.	0.
0.90	500.	0.	0.	0.	0.
0.95	500.	0.	0.	0.	0.
1.00	500.	0.	0.	0.	0.

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.

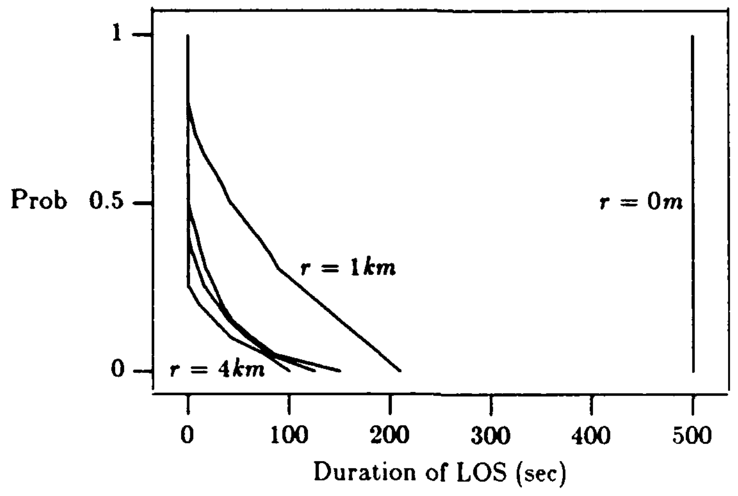


Figure 9. Probability LOS exists for t seconds or more.

CODE	COMMENT
toutil(21,5)	Table of out-of-view segment lengths for infrared
toutvl(21,5)	Table of out-of-view segment lengths for visible band
touti(21,5)	Table of out-of-view segment lengths for infrared band
toutv(21,5)	Table of out-of-view segment lengths for visible band
tini(21,5)	Table of in-view segment lengths for infrared band
tinv(21,5)	Table of in-view segment lengths for visible band
ptbl(21)	Vector of probabilities.
rtbl(5)	Vector of ranges (m)

```
c V1.3
      SUBROUTINE RDSMK (fname)
c     Rd smk: Read intervisibility data for smoke.
      character*32 fname
      include 'common.h'
      common /smoke1/ toutil(21,5),toutvl(21,5),touti(21,5),
    1  toutv(21,5),tini(21,5),tinv(21,5),ptbl(21),rtbl(5)

      open(4,file=fname, status='old')
      rewind 4
      print *,' Smoke causes intervisibility.'
      read *
      read *, ((toutil(i,j),j=1,5),i=1,21)
      read *
      read *, ((toutvl(i,j),j=1,5),i=1,21)
      read *
      read *, ((touti(i,j),j=1,5),i=1,21)
      read *
      read *, ((toutv(i,j),j=1,5),i=1,21)
      read *
      read *, ((tini(i,j),j=1,5),i=1,21)
      read *
      read *, ((tinv(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.

CODE	COMMENT
p	For all Blue/Red pairs a random number is drawn
r	Opening range.
dt	Time smoke blocks LOS between searchers and target (sec)

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**.

```
c V7.6
      SUBROUTINE SMOKE
c 0   Smoke: Find path lengths where attacker is hidden by smoke.
      include 'common.h'
      common /smoke1/ touti1(21,5),toutv1(21,5),touti(21,5),
    1   toutv(21,5),tini(21,5),tinv(21,5),ptbl(21),rtbl(5)
      data ptbl /0.,.05,.1,.15,.2,.25,.3,.35,.4,.45,.5,.55,.6,.65,.7,
    1   .75,.8,.85,.9,.95,1.0/
      data rtbl /0.,1000.,2000.,3000.,4000./

      if (trace)print *,'>smoke'
      DO 80 nb= 1,nblu
        DO 70 nr =nblu+1,nblu+nred
c       Find first time window for LOS between tanks nb, nr.
          p=ranu(din)
          r=rg0
          if (kview(RED).eq.1) dt=tdintp(ptbl,rtbl,toutv1,p,r,21,5)
          if (kview(RED).eq.2) dt=tdintp(ptbl,rtbl,touti1,p,r,21,5)
          call skedul(dt,nb,'appear',nr)
          if (kview(BLU).eq.1) dt=tdintp(ptbl,rtbl,toutv1,p,r,21,5)
          if (kview(BLU).eq.2) dt=tdintp(ptbl,rtbl,touti1,p,r,21,5)
          call skedul(dt,nr,'appear',nb)
70      CONTINUE
80    CONTINUE
      if(trace)print *,'<smoke'
      END
```

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.

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

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.

CODE	MATH	COMMENT
d1	Δ_1	Random in view time for visual (sec).
d2	Δ_2	Random out-of-view time for visual (sec).
d4	Δ_4	Random out-of-view time for IR (sec).
d3	$\Delta_3 = \Delta_1 + .5(\Delta_2 - \Delta_4)$	In view time for IR (sec).

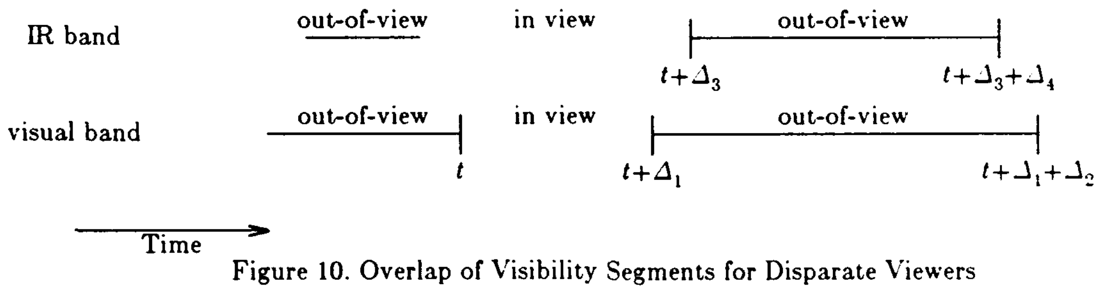


Figure 10. Overlap of Visibility Segments for Disparate Viewers

```
c V7.7
      SUBROUTINE APPEAR(t,it,I)
c 0   Appear: if tgt appears treat, otherwise reschedule appearance
      include 'common.h'
      common /smoke1/ touti1(21,5),toutv1(21,5),touti(21,5),
   1    toutv(21,5),tini(21,5),tinv(21,5),ptbl(21),rtbl(5)
      common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
      rss(x,y) = sqrt(x*x+y*y)
1     format(f8.2,1x,a4,i3,' appears ',9x,'(x=',f8.1,'   y=',f8.1,')')
2     format(f8.2,1x,a4,i3,' LOS to   ',a4,i3,' starts.')

      if (trace) print *,'>appear'
      n = army(it)
      m = 3-n
      IF (invisb.eq.1) THEN
c     Terrain causes intermittent LOS.
        if(speed(n).le.0.)print *,'APPEAR: n,speed=',n,speed(n)
        if(speed(n).le.0.) STOP
        call path(it,t,motion(it),0.2,x,y,vx,vy)
        travel = rss(x-xold(it), y-yold(it))
        IF (travel.gt.dist(it)) THEN
c       Tgt is no longer masked by terrain
          if (histry) print 1,t,color(n),it,x,y
          xold(it) = x
          yold(it) = y
          iseg(it) = iseg(it)+1
          if (iseg(it).gt.40) iseg(it)=iseg(it)-40
          dist(it) = d(iseg(it))
          call aprter(t,it,I,FE)
c         Schedule next disappearance
            dt = dist(it)/speed(n) + 0.01
            call skedul(t+dt,it,'vanish',NULL)
        ELSE
c       Still masked by terrain, so reschedule mask end
          IF (life(it).eq.ALIVE) THEN
            dt = (dist(it) - travel) / speed(n) + 0.01
            call skedul (t+dt,it,'appear',NULL)
          ENDIF
        ENDIF
      ELSE
c     Tgt is no longer masked by smoke
        if (histry) print 2,t,color(3-n),I, color(n),it
        call aprsmk(t,it,I)
c       Schedule next disappearance
        r = rgf(t,I,it)
        p = ranu(0)
        pout = ranu(0)
        IF (kview(RED).eq.kview(BLU)) THEN
          IF (m.eq.BLU) THEN
            IF (kview(m).eq.2) THEN
              dtin=tdintp(ptbl,rtbl,tini,p,r,21,5)
              dtout=tdintp(ptbl,rtbl,touti,pout,r,21,5)
            ELSE
              dtin=tdintp(ptbl,rtbl,tinv p,r,21,5)
              dtout=tdintp(ptbl,rtbl,toutv,pout,r,21,5)
            ENDIF
            call skedul(t+dtin,it,'vanish',I)
            call skedul(t+dtin,I,'vanish',it)
            call skedul(t+dtin+dtout,it,'appear',I)
            call skedul(t+dtin+dtout,I,'appear',it)
          ENDIF
        ELSE
          IF (kview(m).eq.1) THEN
            d1=tdintp(ptbl,rtbl,tinv,p,r,21,5)
            d2=tdintp(ptbl,rtbl,toutv,pout,r,21,5)
            d4=tdintp(ptbl,rtbl,touti,pout,r,21,5)
            d3=d1+(d2-d4)*0.5
            call skedul(t+d1,it,'vanish',I)
            call skedul(t+d1+d2,it,'appear',I)
            call skedul(t+d3,I,'vanish',it)
            call skedul(t+d3+d4,I,'appear',it)
          ENDIF
        ENDIF
      ENDIF
      if (trace) print *,'<appear'
      END
```

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.

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

```
c V7.1
      SUBROUTINE APRSMK(t,it,I)
c 0   Aprsmk: Tgt appears out of smoke, reset.
      include 'common.h'
      common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
c
      if (trace) print *,'>aprsmk'
      n = army(it)
c     Restore line-of-sight from firer to tgt.
        los(I,it) = army(I).ne.n
c     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 Aprter: 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. Aprter 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.

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

```
c V7.1
      SUBROUTINE APRTER(t,it,I,jexpos)
c 0   Aprter: Tgt has appeared from behind terrain, reset.
      include 'common.h'
      integer it,firer
      common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
1     format(f8.2,1x,a4,i3,' aprter ',9x,'(x=',f8.1,'  y=',f8.1,')')
c
      if (trace) print *,'>aprter'
      n = army(it)
      knceal(it) = jexpos
c     Restore all lines-of-sight involving it
        DO 20 i=1,nblu+nred
          IF (knceal(i).ne.FD) THEN
            los(it,i) = army(i).ne.n
            los(i,it) = army(i).ne.n
          ENDIF
20      CONTINUE
c     Turn search on if it is off
        IF (.not.repeat) THEN
          repeat = .true.
          call skedul(t+.01,0,'search',NULL)
        ENDIF
      if (trace) print *,'<aprter'
      END
```

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. **Terrain** 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.

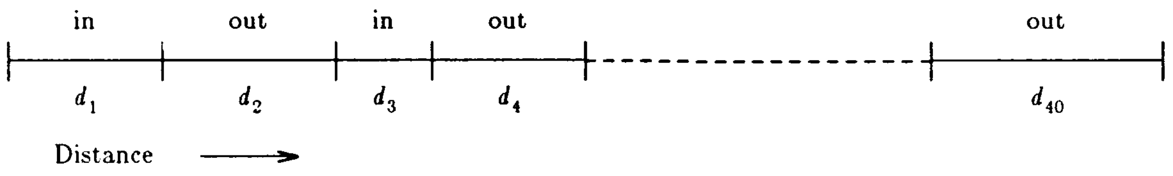


Figure 11. Alternating In View and Out-of-View Segments

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	COMMENT
ifirst	ID of first attacker.
last	ID of last attacker.
d(i)	Length of segment (m)
rg0	Opening range (m).
x0(i), y0(i)	Position of attacker (m).
xold(i), yold(i)	Beginning of segment (m).
dist(i)	Length of segment (m).
iseg(i)	Number of segment ith attacker is in.

```
c V7.6
      SUBROUTINE TERAIN (ifirst,last)
c 0   Terain: find path lengths where attacker is masked by terrain
      include 'common.h'
      common /terane/ d(40), xold(20), yold(20), dist(20), iseg(20)
      common /terrac/ a1, b1, a2, b2
1     format (' visible for',f5.0,'m, then hidden for',f5.0,'m.')
c
      if (trace) print *,'>terain'
c     Find segment length at start of each engagement.
      DO 20 i=1,39,2
c       Hunfeld terrain constants
        f = -alog(ranu(0.0))
        f = a1*f**b1
        d(i) = amin1(f,.3*rg0)
        f = -alog(ranu(0.0))
        f = a2*f**b2
        d(i+1) = amin0(f,.3*rg0)
        if (histry) print 1, d(i), d(i+1)
20    CONTINUE
c     Initialize data for each tank
      DO 30 i=ifirst,last
        xold(i) = x0(i)
        yold(i) = y0(i)
        dist(i) = d(1)
        iseg(i) = 1
        call skedul (0.,i,'vanish',NULL)
30    CONTINUE
      if (trace) print *,'<terain'
      END
```

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 simple 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.

CODE	MATH	COMMENT
t		Time (sec).
I		ID of searcher.
it		ID of target.
n		Side target is on. (1 if Blue, 2 if Red)
invisb		1 if terrain blocks LOS, 2 if smoke blocks LOS
speed(n)	v	Combat cruise speed of target (m/s).
x, y		Position of target (m).
xold, yold	x_o, y_o	Beginning of in view segment (m).
travel	$d = \sqrt{[(x-x_o)^2+(y-y_o)^2]}$	Distance traversed in segment (m).
dist(it)	d_{it}	Length of in view segment target is in (m).
dt	$\Delta t = d_{it}/v+.01$	Time to travel out-of-view segment if target vanished (sec).
dt	$\Delta t = (d_{it}-d)/v+.01$	Time to finish traveling in view segment if target paused enroute (sec).

```
c V7.4
      SUBROUTINE VANISH(t,it,I)
c 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 (invisb.eq.1) THEN
       if(speed(n).le.0.)print *,'VANISH: n,speed=',n,
     1   speed(n)
       IF (speed(n).le.0.) STOP
       call path(it,t,motion(it),0.0,x,y,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) = d(iseg(it))
        call vanter(t,it,I)
        dt = dist(it)/speed(n) + 0.01
        call skedul (t+dt,it,'appear',NULL)
       ELSE IF (life(it).eq.ALIVE) 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,I)
      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 taigets detected is decrementcd. 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.

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

```
c V7.3
      SUBROUTINE VANSMK(t,it,I)
c 0   Vansmk: Treat tgt that vanished behind smoke.
      include 'common.h'
1     format(f8.2,1x,a4,i3,' LOS to   ',a4,i3,' broken by smoke.')
c
      if (trace) print *,'>vansmk'
      m = army(it)
      n = 3-m
      if (histry) 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.

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

```
c V7.5
      SUBROUTINE VANTER(t,it,I)
c 0   Vanter: Treat tgt that vanished behind terrain.
      include 'common.h'
1     format(f8.2,1x,a4,i3,' vanishes',9x,'(x=',f8.1,'  y=',f8.1,')')
c
      if (trace) print *,'>vanter'
      n = army(it)
      if (histry) print 1, t, color(n), it,
     1  x0(it), y0(it)
      knceal(it) = FD
      nrtgt(it) = 0
      ndet(it) = 0
c     Cancel all lines-of-sight and sightings involving tgt
        DO 20 i=1,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(t,it,ALL)
        nchan(it) = 0
c     Abort incoming rounds & disengage tanks firing at tgt
        ifirst=1
        if (n.eq.1) ifirst = nblu+1
        call newtgt(t,ifirst,it)
        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.

CODE	COMMENT
t	Time (sec)
it	ID of target.
knceal(it)	Concealment is set to full defilade.
I	ID of first foe.
last	ID of last foe.
los(i,j)	Line-of-sight is broken (set to .false.)

```
c V7.2
      SUBROUTINE HIDE (t, it)
c 5   Hide: Simulate tank hiding.
      include 'common.h'
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 (it,'all   ',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**.

CODE	COMMENT
t	Time (sec)
I	ID of defender.

```
c V7.1
      SUBROUTINE POP DN (t,I)
c 0   Pop dn: Have defender pop down to reload
      include 'common.h'
c
      if (trace) print *,'>pop dn'
      call vanter(t,I,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:

```
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[2] 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 Ω. The 'next event' pointer points to Ω 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:

real when(100)	Time of the event
integer who(100)	The entity performing the event
character*6 what(100)	The type of event performed
integer whom(100)	The entity receiving the event
integer next(100)	The pointer to the next link

when(9)	who(9)	what(9)	whom(9)	next(9)
18.32	4	fire..	6	31

Figure 12. Contents of a Link

13.2 Reset: Re-initialise 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.

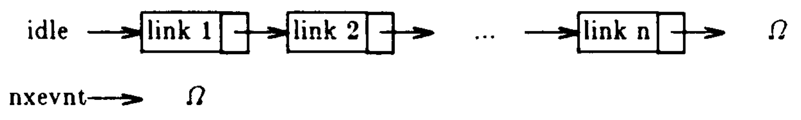


Figure 13. The Initial Linked Lists

Code.

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

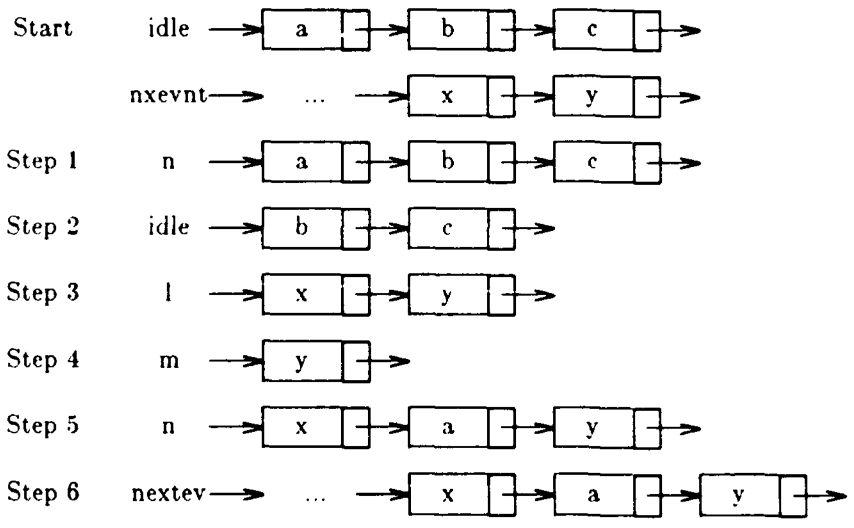


Figure 14. Scheduling an Event

Code.

```
c V7.3
      SUBROUTINE SKEDUL (t,I,act,it)
c 9   Schedule: Schedule an event for later execution.
      include 'clock.h'
      character*6 act
1     format(9x,'skedul ',i3,' ',a6,i3,' at time',f8.2)
c
      if (prflag) print 1, I, act, it, t
      IF (nxidle.eq.0) THEN
c     If storage all used stop
        print *,' Storage overloaded with too many events.'
        STOP
      ELSE
c     Store the event
c       Cut storage unit from empties
          n = nxidle
          nxidle = next(nxidle)
c       Then find where to insert this event in the event list.
        IF (nxevnt.le.0) THEN
c       New event is only event
          next(n) = 0
          nxevnt = n
        ELSE
c       Then find where to insert it.
c         Point to first 2 events
            l = nxevnt
            m = next(l)
c         Find where to insert them
            IF (t.ge.when(l)) THEN
c           See if between 2 scheduled events.
c             Loop till found.
20            IF (m.ne.0 .and. t.ge.when(m)) THEN
                l = m
                m = next(m)
                GOTO 20
              ELSE
c             Splice new event into list
                next(n) = m
                next(l) = n
              ENDIF
            ELSE
c           Place new event as most imminent
              next(n) = nxevnt
              nxevnt = n
            ENDIF
        ENDIF
c       Finally store event info
          when(n) = t
          what(n) = act
          who(n) = I
          whom(n) = it
      ENDIF
      END
```

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.

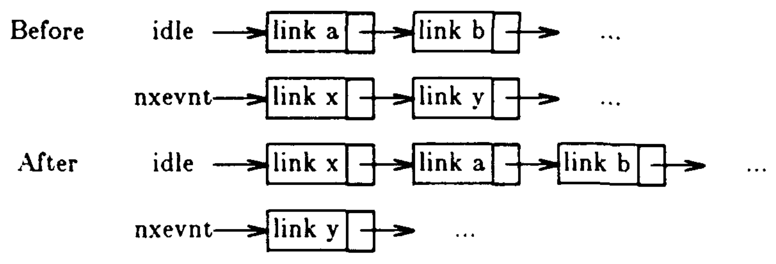


Figure 15. Selecting the Next Event

Code.

```
c V7.2
      SUBROUTINE EVENT (I,act,it,t)
c 0   Event: Find the next scheduled event.
      include 'clock.h'
      character*6 act
c
c     Fill arguments
        I = who(nxevnt)
        act = what(nxevnt)
        it = whom(nxevnt)
        t = when(nxevnt)
c     Drop storage unit from active storage chain
        n = nxevnt
        nxevnt = next(nxevnt)
c     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.

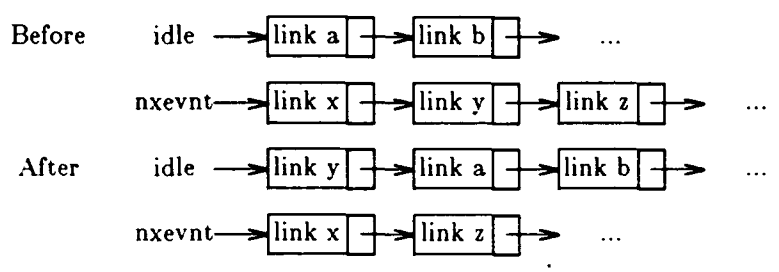


Figure 16. Cancelling an Event

Code.

```
c V7.1
      SUBROUTINE CANCEL (I, act, it)
c 0   Cancel: cancel 'act' events for 'I' entity.
c       (all events if act='')
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
1     format(9x,'cancel ',i3,' ',a6,i3,' at time',f8.2)
c
      m = 0
      n = nxevnt
10    IF (n.ne.0) THEN
c     Continue until n=0
        is who  = I .eq.who(n)
        is what = act.eq.what(n) .or. act.eq.'all   '
        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 (prflag )print 1, 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
          n = next(n)
        ENDIF
        GOTO 10
      ENDIF
      END
```

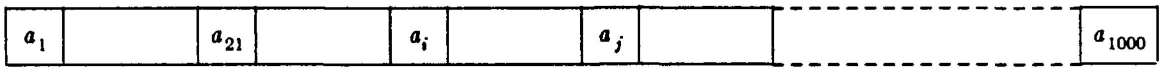
INTENTIONALLY LEFT BLANK

14. OTHER UTILITY ROUTINES

The routines in this section are generᵃˡ ⁿurpose 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 excerption 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.



The ID of the temporary entity is the index of the first word storing information for that entity. For example, 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 $a_i = \pm j$. 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 $a_i = a_i + a_j$.

CODE	COMMENT
n	The number of attributes to be stored.
a	The vector used to store attributes in.
done	True iff the routine is done; if it finds space to store the attributes in.
i	The index of the storage space we are currently looking at.
ient	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.
istart	The starting point for the search. If we get back to istart without a find, we have a storage overload and we error off.
j	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.
nreq	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.

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.

```
c V7.4
      SUBROUTINE CREATE (n, ient)
c 8   Create: create a temporary entity. (a bullet or msl)
c     note - The amount of storage space is 1000 words. If you want to
c      increase this you'll have to change all occurrences of 1000.
c      Also note that in creset these must be set - a=0, a(1)=1000.,
c      i=1.
c
      logical trace, histry, done
      common /ctrace/ trace, histry
      common /tstore/ a(1000), i
1     format (' CREATE:  Not enuf space to store',i5, ' attributes.')
2     format (' CREATE: i, j, a(i), a(j) =',2i5,2f10.3)
c
      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(a(i)))
          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
c           Move to next space.
              i = j
              if (i.eq.istart) print 1, n
              IF (i.eq.istart) STOP
            ELSE
c           Reserve space.
              done = .true.
              itemp = i+nreq
              if(a(i).ne.float(nreq))a(itemp) = a(i)-float(nreq)
              a(i) = -nreq
              ient = i
              i = j
            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
c V7.1
      SUBROUTINE CRESET
c 0   Creset - Reset variables used by create.
      common /tstore/ a(1000), iholy
      parameter (NN=20)
      DO 20 i=2,1000
        a(i)=0.0
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, θ is the angle between them. So

$$x = \cos\theta = \frac{\mathbf{a}\cdot\mathbf{b}}{ab}$$

To avoid round off errors, the result is trimmed so that $-1 \leq \cos\theta \leq -1$.
Next, the cosine is taken:

$$y = \cos^{-1} x$$

Finally, the appropriate sign is attached:

$$r_3 = a_1 b_2 - a_2 b_1$$
$$z = -sign(y, r_3)$$

Note that **a**, **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[4] give the method for finding a confidence interval on a binomial outcome.

CODE	COMMENT
p	The number of desirable outcomes.
np	The number of trials.
hi, lo	The high and low ends of the confidence interval.
fail	True IFF a confidence interval cannot be found.

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 CONFB (p, nr, hi, lo, fail)
c     Confb: Find the 90% binomial confidence interval.
c     p - the sample probability.
c     nr - the sample size.
c     Reference: Introduction to Statistical Analysis, 3rd edition,
c       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
c     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*n)
        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) <= 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) .. x(n), in ascending or descending order. Given the arguments x, n, x_s, where x is an ascending vector, it finds the value i such that $x_i \leq x_s \leq x_{i+1}$ using binary search. If $x_s < x_i$ or $x_s > x_n$, 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

x	0.0	0.5	1.0	1.5	2.0	2.5	3.0	3.5
f(x)	0.0	0.4	0.81	1.23	1.68	2.10	2.52	2.95

Code.

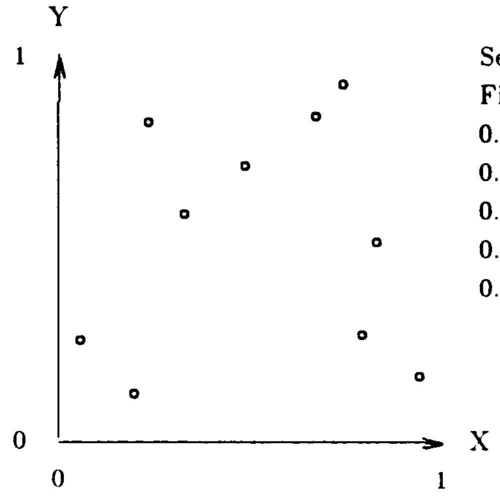
```
c V7.2
      FUNCTION INDEXX(a, n, x)
c     Find the index j, where a(j) <= x < a(j+1)
      integer n ,lo, hi, mid
      logical incres, above
     real a(n), x

      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 uran31 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[5]. We also recommend the discussion of random number generators by Press[6] and by Knuth[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 ranu(), 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.



```
Seed= 1111111
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
```

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[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 16,777,215.

Code.

```
      FUNCTION RANU (dm)
c     Ranu: A version of uran31 uniform random nr generator.
      common /crandm/ j
      real a1

      j=j*25
      j=j-(j/67108864)*67108864
      j=j*25
      j=j-(j/67108864)*67108864
      j=j*5
      j=j-(j/67108864)*67108864
      a1=j
      ranu= a1/67108864
      END
```

14.6. 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.

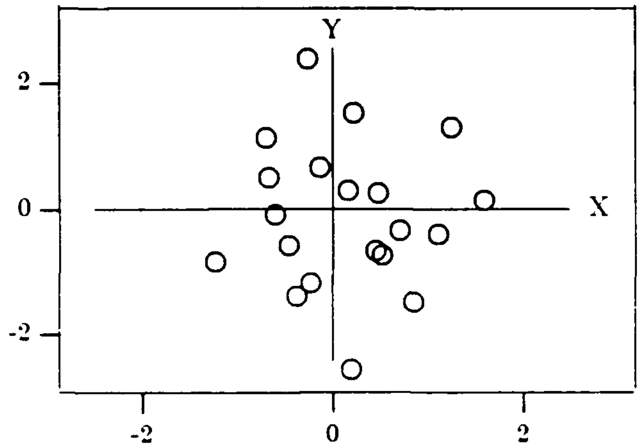


Figure 18. Draw of 20 Random Shots

Mathematics. Rann is based on an algorithm by Bell[8]. It produces two independent random variables, each from the normal distribution with mean 0 and standard deviation 1. The subroutine calls the real function ranu 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 267. See also algorithm 442 for a slower, but higher precision algorithm. Finally, see algorithm 448, which may be faster if one of the 2 random deviates must be discarded.

Code.

```
      SUBROUTINE RANN(p,q)
c 7   Rann: draw two random numbers from the std normal distribution.
c     Box-Muller method

      x = sqrt(-2.*alog(ranu(dm)))
      y = 2.*3.1415926535*ranu(dm)
      p = x*cos(y)
      q = x*sin(y)
      END
```

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[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:

$$p = (1+cos\theta)/2\pi$$

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

$$P = \int_{-\pi}^{\theta} p\ d\theta = (\theta+\sin\theta+\pi)/\ 2\pi.$$

Both of these are illustrated in figure 19 below.

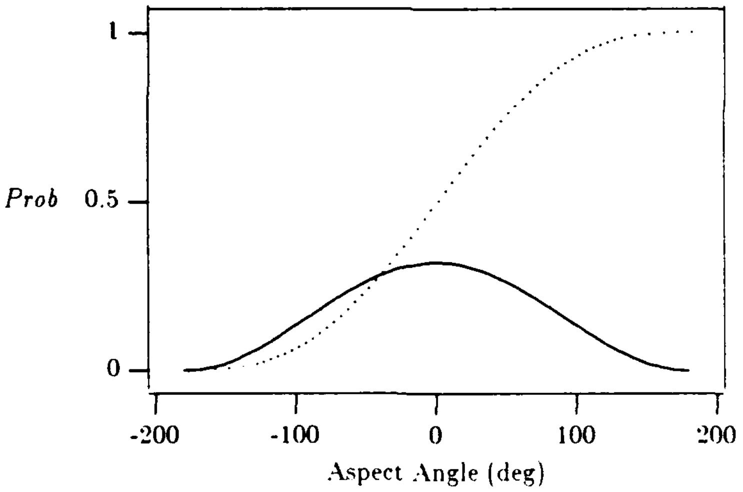


Figure 19. Cardioid Density and Distribution Functions

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

$$p = (1 + \cos 3\theta)/6\pi, \quad -\pi/3 \leq \theta \leq \pi/3$$
$$P = (3\theta + 3\cos 3\theta)/2\pi$$

There may be no way to analytically solve for θ 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 θ in t_l or t_h as appropriate.

CODE	MATH	COMMENT
PI	π	
denom	$d = 0.5/\pi$	
p	$p = ranu()$	Random draw from a uniform distribution.
tlo	$\theta_l = -\pi$	Lower limit of angular value (rad).
thi	$\theta_h = \pi$	Upper limit of angular value (rad).
		REPEAT FOLLOWING LINES 10 TIMES
theta	$\theta = (\theta_l+\theta_h)/2$	Limit (rad).
px	$p_x = (\theta+\sin\theta+\pi)d$	
	IF $p_x < p$ THEN	True iff θ is too low.
	$\theta_l = \theta$	
	ELSE	
	$\theta_h = \theta$	
	ENDIF	

```
c V7.2
      FUNCTION RNDANG(iangd)
c     Rnd ang: Draw a random angle from a cardioid/other distribution.

      PI=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.
        thi = PI
        if (iangd.gt.1) thi = PI/3.
        DO 20 i=1,10
          theta = 0.5*(tlo+thi)
          if (iangd.eq.1) px = (theta+sin(theta)+PI)*denom
          if (iangd.gt.1) px = (3.*theta+sin(3.*theta)+PI)*denom
          IF (px.lt.p) THEN
            tlo = theta
          ELSE
            thi = theta
          ENDIF
20      CONTINUE
      rndang=theta
      END
```

INTENTIONALLY LEFT BLANK

15. REFERENCES

1. Peterson, R. H., *The Range and Angular Distribution of A.P. Hit Tanks,* BRL-RPT-590A, U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, December 1951, (UNCLASSIFIED)

2. Peterson, R. H., Hardison, D. C., Benvienuto, A. A., *Terrain and Ranges of Tank Engagements,* U.S. Army Ballistic Research Laboratory, Aberdeen Proving Ground, MD, June 1953, (UNCLASSIFIED)

3. McCormack, W. M., and Sargent, R. G., "Comparison of Future Event Set Algorithms for Simulations of Closed Queuing Systems", *Current Issues in Computer Simulation,* Academic Press, NY, 1979, pp 71-82.

4. Dixon, W. J., and Massey, F. J., *Introduction to Statistical Analysis,* 3rd edition, McGraw Hill, New York, 1969, p. 246.

5. Pike, M. C., and Hill, I. D., "Algorithm 266, Psuedo-Random Numbers," page 266-P 1- 0, "Remark on Algorithm 266, Psuedo-Random Numbers," page 266-P 2- R1, *Collected Algorithms from CACM,* volume II, Association for Computing Machinery, NY, 1980.

6. Press, W. H., Flannery, B. P., Teukolsky, S. A., & Vetterling, W. T., *Numerical Recipes,* Cambridge University Press, NY, 1986, pp 191-199.

7. Knuth, D. E., *Seminumerical Algorithms, The Art of Computer Programming,* Addison-Wesley Publishing Company, Reading MA, 1969, pp 1-157.

8. Bell, J. R., "Algorithm 267 Random Normal Deviate", page 267-P 2- 0. *Collected Algorithms from CACM,* volume II, Association for Computing Machinery, NY, 1980.

INTENTIONALLY LEFT BLANK

No. of Copies | Organization

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

(Unclass. only) 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

No. of Copies | Organization

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

(Class. only) 1 Commandant
U.S. Army Infantry School
ATTN: ATSH-CD (Security Mgr.)
Fort Benning, GA 31905-5660

(Unclass. only) 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, USATECOM
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

No. of Copies	Organization
1	General Dynamics Land Systems Divsion ATTN: David Stremling P.O. Box 2045 Warren, MI 48090
1	General Defense Corporation Tactical Systems Division ATTN: Ray Edmondson P.O. Box 21606 St. Petersburg, FL 34664
1	LTV Aerospace and Defense Company ATTN: C. H. McKinley P.O. Box 655907 Dallas, TX 75265-5907
1	Booz Allen and Hamilton, Inc. ATTN: Mike McGinnes Suite 1610 1300 17th St. Rosslyn, VA 22209
1	Military Vehicles Operation ATTN: Dan Bitz P.O. Box 420 Mail Code 01 Indianapolis, IN 46206
1	Technical Solutions, Inc. ATTN: George Ober P.O. Box 1148 Mesilla Park, NM 88047
1	Mr. Harry Reed 338 Carter Street Aberdeen, MD 21001

<u>Aberdeen Proving Ground</u>

6	Dir, USAMSAA ATTN: AMXSY-GC, G. Comstock L. Harrington AMXSY-GA, W. Brooks K. Tarquini AMXSY-GI, C. Ehrig E. Walker

USER EVALUATION SHEET/CHANGE OF ADDRESS

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

NO POSTAGE NECESSARY IF MAILED IN THE UNITED STATES

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