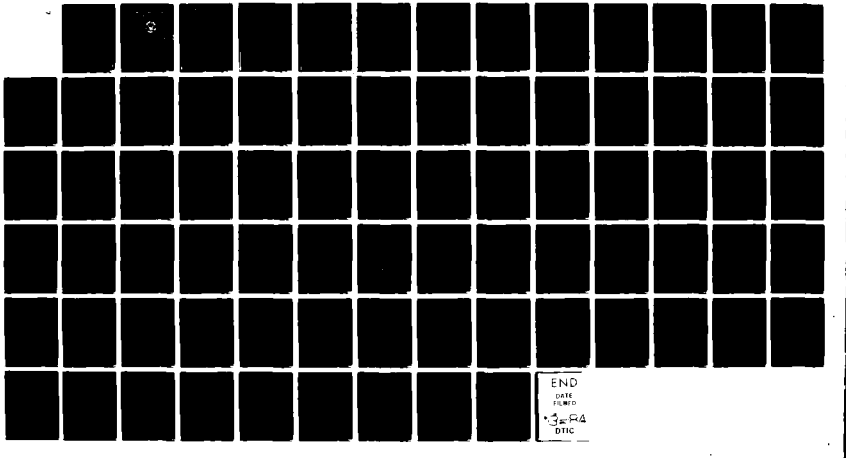
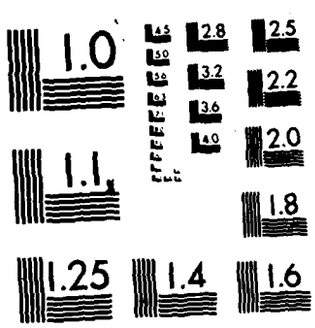
AD-A138 112 ANALYSIS OF THE NAVAL WARFARE GAMING SYSTEM'S SURFACE-TO-AIR MISSILE ROUTINE(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA D T STOKOWSKI SEP 83 1/1

UNCLASSIFIED F/G 15/7 NL

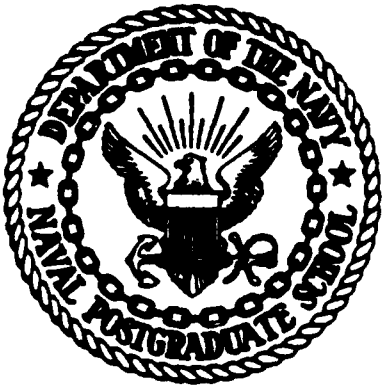


END
DATE FILMED
3-84
DTIC



(2)

NAVAL POSTGRADUATE SCHOOL
Monterey, California



THESIS

ANALYSIS OF THE NAVAL WARFARE GAMING SYSTEM'S SURFACE-TO-AIR MISSILE ROUTINE

by

Dennis Thomas Stokowski

September 1983

Thesis Advisor: A.F. Andrus

Approved for public release; distribution unlimited.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A134112	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Analysis of the Naval Warfare Gaming System's Surface-to-Air Missile Routine		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; September 1983
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Dennis Thomas Stokowski		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE September 1983
		13. NUMBER OF PAGES 76
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Naval Warfare Gaming System
NWGS
Surface-to-Air Missile
SAM
Computer simulation

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

This thesis is an examination of the surface-to-air missile engagement model in the Naval Warfare Gaming System installed at the Center for War Gaming, Naval War College, Newport, Rhode Island. Flow charts derived directly from the computer code are included. The intent is to verify the computer code with pertinent documentation as well as to determine its realism in modeling actual surface-to-air missile engagements. Modifications to

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-LF-014-6601

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

the Naval Warfare Gaming System surface-to-air model are proposed.

A-1

Approved for public release; distribution unlimited

Analysis of the Naval Warfare Gaming System's Surface-to-Air Missile Routine

by

Dennis Thomas Stokowski
Lieutenant, United States Navy
B.S., Southeastern Massachusetts University, 1976

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN OPERATIONS RESEARCH

from the

NAVAL POSTGRADUATE SCHOOL
September 1983

Author: Dennis T. Stokowski

Approved by: Alan I. Andrus
Thesis Advisor

Second Reader

Chairman, Department of Operations Research

K. T. Marshall
Dean of Information and Policy Sciences

ABSTRACT

This thesis is an examination of the surface-to-air missile engagement model in the Naval Warfare Gaming System installed at the Center for War Gaming, Naval War College, Newport, Rhode Island. Flow charts derived directly from the computer code are included. The intent is to verify the computer code with pertinent documentation as well as to determine its realism in modeling actual surface-to-air missile engagements. Modifications to the Naval Warfare Gaming System surface-to-air model are proposed.

TABLE OF CONTENTS

I. INTRODUCTION

A. PREFACE

An important responsibility of the U.S. Navy is to conduct exercises to train personnel and evaluate performance, tactics and weapons systems. As the Naval environment increases in complexity it becomes significantly more expensive to conduct exercises even with a limited number of platforms. Fortunately, the computer technology which spawned today's sophisticated combat systems also provides an alternative to the high costs associated with exercising fleet units. Interactive computer war gaming is not a satisfactory replacement for underway maneuvers but is a cost effective adjunct to them.

War gaming can be used for several basic tasks in support of defense readiness. These tasks include: training of personnel; providing quasi-combat experience to personnel; and formulating and analyzing scenario dependent problems. These problems may pertain to force procurement, strategic planning, or testing current operational doctrine.

Regardless of the purpose of a particular war game, the results are usually no better than the algorithms and assumptions that constitute the game. For this reason, verification, validation and modification of existing war games should be a continuous process.

B. NAVAL WARFARE GAMING SYSTEM OVERVIEW

The Naval Warfare Gaming System was developed by Computer Sciences Corporation of Moorestown, New Jersey under U.S. Navy contract. It was installed for acceptance testing in 1982 at the Center for War Gaming at the Naval War College in Newport, Rhode Island. It is an interactive, data base oriented computer simulation covering the entire spectrum of naval engagements. It can be played in real time or else in multiples faster or slower than real time.

The NWGS software consists of approximately 990 procedures (subroutines) and 156,000 lines of code. There are approximately 170 procedures and 50,000 lines of code in warfare area models alone. The code is written in Programming Language One (PL/I).

The central computer facility is a Honeywell Multics Level 68 Multiprocessing Computer system. The interactive display system consists of Sanders Associates Incorporated, high resolution, color graphics displays and Honeywell alphanumeric displays. There are 44 of these interactive console stations at the Center for War Games. Additionally, two console stations are operating at CINCPACFLT in Hawaii with plans for other remote installations such as at the Naval Postgraduate School. Different game scenarios may be played simultaneously at each of the terminal stations.

NWGS allows for several levels of game play: the Command Game, the Student Full Scale Game and the Student

One-on-One Game. The Command Game is suited towards major Naval staffs playing at the task force/theater level and lasting from one day to weeks. Only one Command Game may be played at a time as most of the 44 console stations would be needed to support the large number of players involved in just a single game. Ten Student Full Scale Games may be played simultaneuously and are generally at the battle group level. They are of four to eight hours duration. A typical Student One-to-One Game is aircraft versus submarine and lasts one to four hours. Ten student versus student or twenty computer opposed Student Games may be played simultaneously [Ref. 1].

Additionally NWGS allows the person preparing the game to select the level of detail of the warfare area models used in game play. Level One is the least detailed and Levels Two and Three are progressively more complex. Two levels are available in engagements, damage assessment, sonar and others, while there are three levels available in air operations.

The doctrinal control of forces is another feature of NWGS. The player accomplishes this by implementing strings of conditional commands. The conditional commands are built into NWGS and are readily available to the player throughout game play. The doctrinal control is especially useful for movement or engagement of a large number of platfroms. An example being upon detection of an air contact with speeds

in excess of 500 knots all ships in Task Force 70.1 have weapons free. The result of this example is an automatic engagement.

The NWGS data base consists of five files [Ref. 2].

Master File: contains all NWGS software and data on platforms, weapons, sensors, etc.

Game Design File: contains game objectives, pregame scenario, initial conditions, etc.

Game Play File: is created from the previous two files and remains fixed for the duration of the game. It contains all of the information necessary for the game to be played This information includes platform, environmental and geographical data.

Game Date File: initially contains information from the Play File; a current listing of platform position, detections, battle damage, fuel status, ammunition status, etc.

Game History File: contains event information for replay, rerun and postgame analysis.

C. PURPOSE

The purpose of this thesis is to assist the U.S. Navy in its procurement of NWGS by ensuring that the finished product is indeed a realistic simulation of Naval warfare. The purpose is therefore a thorough analysis of a logical section of the computer code, a warfare area model, namely, the Surface-to-Air Missile Routine. Additionally, the purpose is to recommend any changes to the model that are necessary to accurately reflect actual Naval operations.

This work is needed because the Navy Staff at the Center for War Gaming is involved in the daily operation of the

Center, providing war gaming services to Naval War College students, Fleet staffs, and others. The Staff conducts testing of NWGS by conducting games on it and watching for discrepancies. This, however, only reveals the most obvious discrepancies. The Staff just does not have the time nor, in many instances, the experience in computer modeling to thoroughly examine the system. Additionally, the personnel that designed NWGS, including those currently on assignment at the Center for War Gaming, may be proficient in computer programming but are limited in their knowledge of Naval warfare. For these reasons, a complete analysis of a part of NWGS by someone with both some modeling experience and a Naval background would be useful to the Navy.

D. *PROCEDURE*

The organization of this thesis follows the structure of the subject itself, the SAM Routine. Chapters 2, 3, and 4 correspond to the three phases in the SAM Routine: Shoot, Result, and Reload, respectively. These three chapters are divided into sections which each cover a single procedure (subroutine) of the NWGS computer code. Each subroutine, one at a time, is verified and validated. Proposed modifications as necessary are also included within each section.

In order to facilitate an analysis of the NWGS SAM Routine, a diagram showing the relationship of the SAM Routine to the rest of NWGS was created and is included as

Figure 1 of Appendix A. Figure 2 of the same appendix is an overview of the SAM Routine showing the phases and procedures that comprise it. A flow chart of each of the phases and procedures that are shown in Figure 2 was developed from the contractor supplied code. The flow charts reflect the NWGS PL/I code as of 5 April 1983. These are included as Figures 3 through 18 of Appendix A and will be referred to throughout this thesis. Variable names used are identical to those used in the PL/I code except where it would be disruptive to a smoothly-flowing chart. For a similar reason some of the intricate details of data storage and retrieval were omitted but are represented in general terms.

Actual description of the subroutines will be limited within the three chapters of analysis to areas of particular interest. This is because the flow charts already describe the subroutines sufficiently.

Verification of the SAM Routine entails corroboration of the PL/I code with model documentation. The three documents most relevant were among those provided by the system contractor, Computer Sciences Corporation. They are the Program Performance Specification (PPS), the Program Description Document (PDD) and the Student's Training Course. The PPS [Ref. 3], the most general of the three, is a broad description down to the NWGS Routine level. The PDD [Ref. 4], is more detailed in that it includes variables used and their meanings as well as an algorithm for the SAM

Routine. The Student's Training Course, consisting of the Guide [Ref. 2], and the video tape [Ref. 5], in which the NWGS senior designer uses the Course Guide to explain the system to the personnel at the Center for War Gaming, is the third piece of documentation. The Course is a description of how NWGS functions with emphasis on both the models and the reasons for the particular design chosen.

Validation implies answering the question of how accurately does NWGS model real world SAM engagements. This was accomplished by trying different feasible values for the variables in a subroutine and manually calculating them through the code to see if the results were reasonable. Reasonable here means mathematically sound as well as in accordance with personal Naval experience and common sense.

Proposed modifications naturally follow if the result of the computer code does not match a good model. Likewise if the code reflects the documentation accurately but really doesn't portray real world SAM engagements, changes will obviously be needed.

E. SAM ROUTINE STRUCTURE

The three phases in the SAM Routine are actually separate entry points into the routine. Each phase is called from outside the routine as necessary. The phases are related, however, in that they all model an essential aspect of an engagement using SAMs.

The Shoot Phase actually consists of two versions, Automatic and Player initiated. They are different in actual computer code and in how they are initiated. They are similar in that both determine the number of missiles fired at a hostile track by a group of SAM platforms and they also call exactly the same SAM Procedures. However, within most of those procedures there are points at which specific values are determined differently for automatically initiated SAM engagements than they are for player initiated ones.

The Player Shoot Phase is entered by a player manually specifying the engagement of a track by a particular SAM platform. The Automatic Shoot Phase is called when game conditions are as previously defined by the player using the doctrinal control of forces. An example being that upon detection of an air contact above a certain altitude and approaching from the East, all ships are to engage that contact.

The Hit Phase is subsequently called to determine the results of any SAM engagements. It calls procedures that account for various reliability factors and weapons system degradations due to the particular circumstances at the time. The procedures in this phase make use of several of the data base tables in determining the engagement results.

Lastly, the Reload Phase performs just what one would imagine by the name. It has an effect on the other phases

in that reload time or number of weapons readied may result in not being able to take another hostile track.

Figures 3 through 6 of Appendix A are flow charts of the four phases in the SAM Routine. The SAM Procedures (subroutines) that are called in these phases are flow charted in Figures 7 through 18 of Appendix A.

Figure 1 is a schematic, though not complete, representation of NWGS. The organization can be illustrated best by using a typical engagement of various hostile aircraft and missiles attacking a carrier battle group. Module 20, Air-to Air Engagements, accounts for the interactions of defending aircraft with the hostile platforms in the area outside the range that surface-to-air missile (SAM) ships can fire.

Next, the surviving hostiles are engaged by the ships in Module 21, Surface-to-Air Engagements. First, the SAM Routine in this module, the object of this thesis, accounts for engagements by SAMs that are of the area defense type.

Then one of the Surface-to-Air Weapon Routines considers point-defense-missile and close-in-weapon systems defending the ships. The Surface-to-Air Weapon 1 Routine considers firing by sectors at aggregates of hostile platforms whereas the Surface-to-Air Weapons 2 Routine models each individual weapon-hostile engagement. It is important to note that only weapons fired in the SAM Routine are allowed to engage attacking aircraft and missiles not targeted at the firing platform.

The previously described interactions are accomplished by the NWGS Strike Supervisor which aperiodically calls Modules 20 and 21. It also calls Module 22, Air-to-Surface Engagements, to execute the other half of the battle. The Strike Supervisor utilizes the Aircraft and Missile Monitors to update track geometry and delete platforms as necessary [Ref. 5].

Although the two other routines of the Surface-to-Air Module may appear in Figure 1 to be the same as the SAM Routine, they are not. The overall structure of the three is comparable, and indeed they share some PL/I code, but they are significantly different.

II. SHOOT PHASE (AUTOMATIC AND PLAYER INITIATED)

A. OVERVIEW

The purpose of this phase of the SAM Routine is to model the activities that would occur in an actual SAM engagement with the exception of the terminal phase of SAM flight, including hit probability, and the reloading after firing. The excluded activities are modeled in subsequent phases of the SAM Routine and will be discussed in Chapters III and IV. Following an examination of the computer code that comprises the central part of the shoot phase will be an analysis of each of the subroutines (procedures) in this phase.

Initially in the Automatic Shoot Phase the number of SAM platforms is determined for use in loops through all possible SAM shooters. Then a check is made to ensure that there are in fact additional strike platforms to be processed. If there are not any then processing is stopped, otherwise the type of strike that is inbound is determined. If the strike is one or more missiles then the SAM_Msl_Threat Procedure is called. If it is one or more aircraft then the SAM_Ac_Threat Procedure is called. If it is neither of the above, then an error message is returned.

In the Player Shoot Phase the system first ensures that the player did not initiate an inappropriate SAM engagement. It checks if the platform indicated to do the SAM firing has

a weapons-free rules-of-engagement status and that the specific weapon to be fired is on that particular platform. If it is not, then a return is caused with an error message. Next a check is made as to whether this weapon system requires a fire control illuminator, setting an indicator bit if it does. Unlike in the Automatic Shoot Phase, the number of firing SAM platforms here is always set to one because the player may only fire from a single platform at a time. Finally, SAM_Msl_Threat or SAM_Ac_Threat is called based on what the threat actually is.

Regardless of whether the system is processing an automatic or player initiated shooting, the appropriate Threat Procedure subsequently calls the Launcher_Loop, SAM_Allocation and Store_Data Procedures. The Launcher_Loop in turn calls SAM_Availability and through it the Range_Alt_Check and SAM_Max Procedures. These procedures will be discussed in detail in their respective sections.

B. SAM_AC_THREAT PROCEDURE

For each hostile aircraft track this subroutine determines the number of missiles shot at it by defending SAM ships. It is flow charted in Figure 7.

SAM_Ac_Threat loops through each threat track where a track consists of one or more similar platforms traveling together. The maximum total shots at each platform is determined and the track's speed, used in future calculations, is modified based on whether the track is inbound

or outbound. Next this subroutine calls the Launcher_loop Procedure which returns the number of SAMs that could be shot restricted by factors to be discussed later. This number is further restricted by whether a shoot-look-shoot or shoot-shoot-look policy is in effect. This final number fired is then used in allocating SAMs shot at each particular platform. The SAMs shot are also deducted from each platform's magazines either in this subroutine or by calling the SAM_Allocation Procedure. Next the Store_Data Procedure is called to retain the pertinent information in the appropriate files. Lastly, the impact time of the weapons fired is determined.

In Figure 7a one can see that if the SAM engagement was not player initiated then the number of SAMs fired at each platform in the threat track is limited to three minus any other pending SAM shots at that particular platform. The constraint seems to be a reasonable way to place an upper limit on the number of SAMs shot at a single platform. Should this prove to be unreasonable, merely changing the value of the variable MAX_SURF_SHOTS, declared at the beginning of the SAM Routine, would remedy this.

Prior to calling the Launcher_Loop Procedure in Figure 7b, the value of the variable, RDOT_FAC, is determined. This is set to -0.2 times threat track speed if the threat track is inbound or the SAM is player-fired. It is set to +0.2 times track speed if the track is outbound and it is

not a player initiated engagement. In the SAM_MAX Procedure, Figure 12a, RDOT_FAC is used in determining the time of flight of the SAM. The equation is:

TOF = 3600 * LAUNCH RANGE /
(Q_WEAPON.SPEED_AVE(WPN_ID) RDOT_FAC).

The RDOT_FAC is used to roughly take into account the fact that intercept of an outbound threat at a certain range takes longer than intercept of an inbound one at the same range.

There are two discrepancies with the above crude modeling. The first is that all targets at which a SAM engagement is manually initiated are modeled as inbound threats. This means that the resulting time of flight for a SAM fired in this manner is identical for inbound and outbound threats. It also means that the time of flight at an outbound threat is longer for automatically fired SAMs than it is for player fired ones.

PROPOSED MODIFICATION: As the first step in Figure 7a do not distinguish between player and automatically initiated SAMs; use TSK_IX for all SAM shots. Then in Figure 7b likewise don't distinguish between the two; set:

EG_F = Q.SUBTASK.EGRESS_F (TSK_IX)

for both types of threat.

The second discrepancy results from the crudeness of using the 0.2 factor. The time of flight against an outbound

slow threat, with a small closest point of approach (cpa) to the launching platform, would be less, not greater, than that against an inbound threat with a greater cpa. What is missing is the actual geometry of the engagement. This might slow processing time if greater detail in modeling were used with a large number of simultaneous engagements, but in smaller scale games this level of detail would be important.

PROPOSED MODIFICATION: Add a third level of detail to engagements which takes into account the actual geometry of the problem. The variables, Q_TRACK.LAT, Q_TRACK.LONG, Q_TRACK.SPEED, Q_TRACK.ALT_DEPTH and Q_TRACK.COURSE contain the appropriate data on the threat. For the launching platform, Q_TRACK.LONG, Q_TRACK.LAT and Q_WEAPON.SPD_AVE are available. The range to impact should then be calculated in a separate routine much as M30_Great_Circle_Range_ee calculates the range between two tracks.

In Figure 7b the Launcher_Loop Procedure is called. One of the variables it returns is TOT_SALVO, the total salvos shot by a group of SAM platforms against a track. In Figure 7c TOT_SALVO is further constrained. If the SAM platforms have a coordinated defense in effect then the number of SAMs shot is fewer than if an uncoordinated defense is in effect. This factor is determined at game

start by the person preparing the game though he may modify it during game play.

A second constraint is based on whether a shoot-look-shoot or shoot-shoot-look defense is in effect. This means whether a single salvo is fired or whether a second salvo is fired while the first is still in flight. Typically this would be determined by the number of weapons remaining on-board the firing platform. If a substantial number were available one might want to fire a double salvo to increase the probability of destroying the threat whereas if only a few weapons were remaining, a more conservative approach might seem prudent.

These two constraints are imposed by multiplying N_TARGS by one of the following four factors depending on the circumstances:

SLS_FAC_C = 1.0 (shoot-look-shoot & coordinated)

SLS_FAC = 1.3 (shoot-look-shoot & uncoordinated)

SSL_FAC_C = 2.0 (shoot-shoot-look & coordinated)

SSL_FAC = 2.3 (shoot-shoot-look & uncoordinated)

Then the total number of salvos is allowed to be no greater than the resulting number.

The intention of the NWGS designers was to automatically determine whether the shoot-shoot-look or shoot-look-shoot policy was in effect and therefore not burden the player with this decision. The intention was to make the decision based on the ratio of salvos remaining onboard the SAM

platforms to the total salvos they can carry [Ref. 6]. If the ratio is less than 0.5 then the shoot-look-shoot is in effect otherwise the shoot-shoot-look. This seems a reasonable way to simulate fleet doctrine.

Upon closer examination of the PL/I code or Figure 7c one can see that:

RATIO = SAM_SUM / SAM_CAPY.

Both are set to zero each time the SAM_Ac_Threat is called. In the SAM_Availability Procedure, Figure 10, SAM_CAPY is given the value of the full load of salvos that the firing SAM systems could carry. SAM_Sum is the total salvos that the SAM systems could shoot at the track during this particular call of the SAM Routine. It does not keep account of how many SAMs are left onboard. The value of RATIO is thus not calculated correctly.

PROPOSED MODIFICATION: Since SAM_SUM is used in calculations other than in determining RATIO, do not change it but rather add a new variable, SAMS_USED. In SAM_Availability, Figure 10, SAMS_USED should be calculated as:

```
SAMS_USED      =      SAM_USED      +
(Q_PROJ_ITEM.LEVEL    (PROJ_IX)     *
Q_WEAPON_SYSTEM.SALVO_SIZE
(SYSTEM_IX)).
```

In Figure 7c change the calculation of RATIO to read:

RATIO = SAMS_USED / SAM_CAPY.

The code in the SAM_Ac_Threat Procedure does not agree with the NWGS PDD, [Ref. 4: p. 3-661], which for example multiplies threat track speed by a factor of 0.9 before any further calculations. In addition the actual code uses an extremely different structure from that found in the PDD. The PDD lists only two procedures as being in the SAM Routine vice the actual twelve. Therefore the algorithm found in the PDD is practically useless. Unfortunately this means that any serious examination of the NWGS SAM Routine must be made by tediously examining the actual PL/I code or by following some description derived from the code such as the figures in this thesis.

C. SAM_MSL_THREAT PROCEDURE

In a manner similar to that in SAM_Ac_Threat, this procedure determines for each hostile missile track the number of missiles shot at it by the defending SAM ships. Figure 8 shows this subroutine.

The description of the SAM_Ac_Threat procedure generally applies here. One specific exception however is that in this routine no distinction is made between either inbound/outbound threats nor between player/automatically initiated engagements. In all cases RDOT_FAC, the relative speed enhancement factor, is assigned the value 0.4 times track speed. This is because the egress flag is never set to 1 for missiles. Unlike aircraft they are not assumed to be able to attack their target and depart afterwards. The flaw

with this is that time of flight calculations always consider the threat as inbound when actually it may have passed over a SAM platform on its way to the target and is actually outbound from the SAM shooter.

PROPOSED MODIFICATION: Add a third level of detail to engagements which takes into account the actual geometry of the problem. The variables, Q_TRACK.LAT, Q_TRACK.LONG, Q_TRACK.SPEED, Q_TRACK.ALT_DEPTH and Q_TRACK.COURSE contain the appropriate data on the threat. For the launching platform, Q_TRACK.LONG, Q_TRACK.LAT and Q_WEAPONS.SPD_AVE are available. The range to impact should then be calculated in a separate routine much as M30_Great_Circle_Range_ee calculates the range between two tracks.

As in SAM_Ac_Threat the criteria for choosing between a shoot-shoot-look and a shoot-look-shoot policy is calculated incorrectly.

PROPOSED MODIFICATION: Since SAM_SUM is used in calculations other than in determining RATIO, do not change it but rather add a new variable, SAMS_USED. In SAM_Availability, Figure 10, SAMS_USED should be calculated as:

```
SAMS_USED    =    SAMS_USED      +
(Q_PROJ_ITEM.LEVEL  (PROJ_IX)     *
Q_WEAPON_SYSTEM.SALVO_SIZE
(SYSTEM_IX)).
```

In Figure 7c change the calculation of RATIO to read:

RATIO = SAMS_USED / SAM_CAPY.

D. LAUNCHER_LOOP PROCEDURE

This procedure as outlined in Figure 9 basically keeps track of which SAM platform is shooting next. It calls SAM_Availability to loop through the SAM shooters starting with the designated platform.

Early in this procedure the egress flag, EG_F, is set for automatically initiated engagements to be 0 for inbound threat and 1 for outbound threat. For all player initiated engagements EG_F is set to 0. This becomes important in later processing because this procedure calls SAM_ Availability which calls Range_Alt_Check. In this last procedure the EG_F value is used in determining which route of subsequent calculations will be followed. Signifying all player shots as inbound results in the wrong calculations.

PROPOSED MODIFICATION: Do not distinguish between player and automatic fire. Allow EG_F to equal Q_SUBTASK.EGRESS_F(TSK_IX) in both cases. This would then correctly signify whether a track is inbound or outbound.

Verification of the Launcher_Loop against the documentation is not possible. None of these documents covers this procedure and the comment statements interspersed in the code is limited.

E. SAM_AVAILABILITY PROCEDURE

This procedure limits the number of SAMs that can be shot by a single launcher. Limitations imposed are due to the geometry of the engagement, the number of SAMs at the launcher, illuminators available, time of last shot, etc. These limitations can be seen in Figure 10.

In this subroutine a loop is made through the SAM platforms. A check is made to ensure that the platform has not been destroyed and has weapons free or else another platform is called. The range to the threat is calculated. Next a second loop is formed through the SAM launchers on that platform. For each launcher the program ensures that the weapon system has not been destroyed and that SAMs are onboard. Then the Range_Alt_Check Procedure is called to determine if the track is a legitimate target now or if it will be prior to the next call of the Shoot Phase.

Upon return from Range_Alt_Check, if the track has not been found to be engageable, processing drops down to the point where another launcher or SAM platform is called. Otherwise, a check is made to ensure that the weapons onboard are available to this system. The ready time, or earliest time that this weapon system can shoot is calculated. If the weapon system requires an illuminator, a check is made to see if one is available, if so, then the SAM_Max Procedure is called. It limits the number of SAMs that can be fired by this particular system based on several factors to be discussed later.

After return from SAM_Max, the procedure checks if any SAMs were actually shot, then as before, another launcher or SAM platform is processed. If SAMs were fired, the number shot by this launcher as well as the time of the last shot are recorded. SAM_CAPY and SAM_SUM are both incremented, SAM_CAPY being the full magazine salvo capacity for the SAM systems that have fired and SAM_SUM the number of salvos fired. Unfortunately, as mentioned previously, SAM_SUM as calculated is not correct for use in determining the shoot-look-shoot/shoot-shoot-look doctrine.

PROPOSED MODIFICATION: Change the calculation of RATIO in Figure 7c to read:

RATIO = SAMS_USED / SAM_CAPY.

Finally the subroutine checks if there are any additional launchers on this platform or else if any more SAM platforms are to be processed. If so, then the loop is started again. If not, then the procedure returns to the Launcher-Loop.

F. RANGE_ALT_CHECK PROCEDURE

As the name implies, this subroutine determines whether a target is within a SAM's altitude and range limits. To some extent it takes into account the relative movement of the target to the firing platform.

The flow of this procedure follows one of two parallel courses depending on whether the threat is inbound or outbound. A similar rough check is made in either case to ensure that the track is within both the SAM's maximum and

minimum for both range and altitude. If the result is negative for the inbound threat then a check is made to see if it will be within the SAM's maximum limits after the next increment of game time has elapsed. In the case of an outbound threat the check is to see if it will be within the SAM's minimum limits after the next increment of game time has elapsed.

One further factor is taken into account for inbound missiles. The angle of attack is considered in making the altitude check. By not doing this for aircraft the implicit assumption is that aircraft will not change altitude while attacking or that the change is not as significant as it is for missiles. This is not reasonable to assume, however, coding a better aircraft simulation into NWGS would not be easy. The angle of attack for most missiles can be placed in a table for look-up as it currently is in NWGS; but with aircraft the possible angles cannot be so easily predefined. A way to mitigate this problem, which would be especially important in modeling iron bombers, could be as follows:

> PROPOSED MODIFICATION: Provide the player controlling strike aircraft the option of selecting from several different attack profiles when initiating a strike plan. Examples would be high-low-high and high-low-low where the player would additionally select the altitude for these different stages as well as the distance out from the

target center at which the altitude change is to be initiated. The changes would be made using standard rate changes for the particular aircraft type as in Level 3 Kinematics and might be limited to this higher level option of game play.

PROPOSED MODIFICATION: In the case of iron bombers add a capability for the attacking aircraft to change altitude during the final approach to the target by means of randomly generated attack angle. This angle could be determined by limiting it to the maximum angle of attack range currently listed for the particular aircraft type in the NWGS data base.

Finally, if the checks of range and altitude are satisfied, then a more exact slant range is calculated to ensure that the track is within SAM range. If it is, then the variable CHECK is set to 1 and the variable LAUNCH_RANGE retains the slant range to the target. Both variables are used in the next procedure, SAM_MAX.

There is a discrepancy between the modeling in Range_Alt_Check and what one would expect fleet doctrine to be. This procedure does not simulate SAM engagements in such a manner as to allow for threat intercept at maximum SAM range. One would logically want to down an incoming threat as close as possible to maximum range to allow for the greatest number

of additional shots should the first fail. Also, in certain scenarios, intercept at maximum range could conceivably mean the difference between having to shoot one aircraft or multiple missiles launched from that aircraft.

PROPOSED MODIFICATION: The point of interest in the program, as in Figure 11a, is where a threat is determined to be: inbound, greater than MIN_RGE & MIN_ALT, not less than MAX_RGE, and such that (RANGE_DR) is less than or equal to SAM MAX_RGE. Change the calculation:

```
CHK_RGE = MAX_RGE
```

to read:

```
CHK_RGE = MAX_RGE + DR.
```

A similar change should be made for outbound threats at the corresponding point in that course of the procedure. Change:

```
CHK_RGE = MIN_RGE
```

to read:

```
CHK_RGE = MIN_RGE - DR.
```

As seen in Figure 11b, prior to computing the slant range to the threat, CHK_ALT must be converted from feet to nautical miles. The factor, 6072, is incorrect as there are actually 6076.1 feet in an international nautical mile [Ref. 7].

PROPOSED MODIFICATION: Change the line which reads:

```
CHK_ALT = CHK_ALT / 6072
```

to be:

```
CHK_ALT = CHK_ALT / 6076.
```

G. SAM_MAX PROCEDURE

The purpose of this procedure as shown in Figure 12 is to limit the number of SAMs shot by available time, salvos and illuminators as well as by the reliability of the weapon system. The time of the firing is computed taking into account the time of any previous shot.

In the right-hand side of Figure 12a the number of automatically fired salvos that can be fired is determined based on the amount of time that the threat is in the SAM envelope. For player fired salvos there is no limitation at this point in the procedure. There is no reason for this distinction.

PROPOSED MODIFICATION: Have both player and automatically fired salvos limited by TIME_AVAIL.

There is a further discrepancy with this section of the code. Only when time available is 1.5 times as great as cycle time is there a calculation to determine if greater than one salvo is shot. Realistically, the first salvo should intercept the threat right at maximum range and therefore if time available is equal to time per salvo, two salvos can be shot.

PROPOSED MODIFICATION: If TIME_AVAIL is greater than TIME_PER and TIME_PER is greater than zero

then calculate:

SALVOS = (TIME_AVAIL / TIME_PER) + 1.

At the beginning of the flow chart in Figure 12b SALVOS_SHOT is determined. In the case of an automatic engagement, this is SALVOS, as calculated earlier, decremented by the reliability of the weapon system. This decrement is implemented either deterministically or stochastically depending on the choice made by the game director at game start. In the case of a player initiated engagement however, SALVOS_SHOT simply equals SALVOS. There is no decrement. Logically there is no reason why missiles fired manually should have a higher success rate than those that are not. This is especially true when one considers that the automatic mode is really used to simulate multiple ships shooting at a rate which the player might not physically be able to "punch in" quickly enough.

PROPOSED MODIFICATION: Do not distinguish between automatic and player initiated SAM engagements in determining SALVOS_SHOT. Allow both types to take system reliability into account.

H. SAM_ALLOCATION PROCEDURE

This procedure is called after the number of SAMs shot at a track is determined. It is shown in Figure 13. It loops through the SAM systems allocating the missiles shot until all fired are accounted for. In addition, it decrements the number of weapons remaining onboard as well as the number of

available illuminators. Should there then be no more SAMs at the launcher a reload is executed.

The subroutine accurately follows the limited documentation in the Course Guide [Ref. 2, p. 297], though it is not covered at all in the PDD [Ref. 4]. The result appears to be a valid model of what occurs in the real AAW environment.

I. STORE_DATA PROCEDURE

As depicted in Figure 14, this procedure simply updates information in the Game Data and Game History Files. The data includes such things as the number of salvos fired, the number of weapons fired by each launcher, total salvos, etc. This is the last procedure called during the Shoot Phase of the SAM Routine.

III. HIT PHASE

A. OVERVIEW

The purpose of this phase is to model the terminal moments of a SAM engagement. It takes into account factors affecting the probability of a hit/kill. It also updates variables that affect future SAM firings such as illuminators available and time of last shot.

The PL/I code of this phase checks if any engagements have taken place since the last time it was called. If no engagements have taken place then a return is caused. Next the SAM_Msl_Result or the SAM_Ac_Result is called based on what target type the engaged platform actually is. SAM_Ac_Result in turn calls Weather_Factor in order to consider the effects of the environment on the engagements. It also calls Fill_Ert in which the kill probabilities are determined and the storage of results takes place. SAM_Msl_Result likewise calls Weather_Factor but it processes the functions contained in Fill_Ert internally and thus does not call this procedure. Figure 5 depicts the Hit Phase.

B. SAM_AC_RESULT PROCEDURE

This procedure determines the status of hostile aircraft after the engagement by the SAMs fired in previous procedures. Electronic countermeasures, environmental factors, as well as speed and size of the aircraft are taken into account.

As outlined in Figure 15, this procedure follows the PDD and the NWGS Course Guide to a greater extent than the previous procedures. It loops through the firing weapons freeing illuminators assigned during the Shoot Phase. Then it loops through the platforms in the threat track and checks if electronic counter measures for the specific platform or the entire track is operating. Weather_Factor is called which returns information to be used in the actual kill probability determination. Likewise the speed of the threat is categorized for later use as being 1, 2 or 3, that is, slow, medium, or fast.

This procedure also determines how many of the original platforms in the threat track are still valid. If fewer than the original number are found then an adjustment in the number of salvos shot is made prior to the kill probabilities being determined. This is realistic in that once the missiles are fired at a specific target, should the target be destroyed by some other means, the missiles in flight to it could not easily be retargeted to an extant threat.

A loop is then made through the threat platforms and within it through the SAM systems onboard. Fill_Ert is called inside these loops.

The final step in this procedure is a call to M25_BDA_Control_ee. The probability of kill for a single shot (pkss) for each of the SAMs fired as well as the number fired at each threat is used to determine the damage inflicted.

C. WEATHER_FACTOR PROCEDURE

This subroutine which is called by both of the Result Procedures is diagrammed in Figure 17. It simply sets two variables, EVIX1 and EVIX2, the first for weather and the second for sea-state, based on the conditions in the area of the SAM engagement. These variables are used in Fill-Ert for aircraft threats and in SAM-Msl-Threat in the case of missile threats. In either case the variables are used as indices in a table look-up to get a degredation for the SAM pk.

D. FILL_ERT PROCEDURE

This procedure as shown in Figure 18, is called for each SAM firing during SAM_Ac_Result. Pkss is initially determined by lock-up in a table specific to the SAM weapon. The table is entered using the target's speed (slow, medium, fast) and size (small, medium, large). The appropriate one of the nine listed Probabilities is then returned. There is a problem in that all nine probabilities listed though different from each other are identical for the same speed and size for the following missiles: SM-2-ER, SM-2-MR, SM-1-ER, SM-1-MR, Sea-Sparrow, SA-N-1, SA-N-3, and SA-N-4. NWGS has the potential of having separate tables for use in unclassified as well as classified games by using the appropriate set of tables. This does not mean, however, that the unclassified tables must have identical probabilities for different missiles but the same engagements.

PROPOSED MODIFICATION: Change the Probability of Kill of Surface Weapons Versus Aircraft tables to reflect the relative capabilities of the different SAMs.

Pkss is further modified by the guidance reliability of the SAM system and by environmental factors returned from Weather_Factor. Finally, a degredation due to electronic counter-measures and counter-counter-measures is imposed.

E. SAM_MSL_RESULT PROCEDURE

Basically this subroutine determines the status of hostile missiles engaged by SAMs. One major difference from the SAM_Ac_Result Procedure is that hostile missiles are either killed or not killed unlike aircraft in SAM_Ac_Result which may be merely degraded in capability after a hit.

Figure 16 is the flow chart of this procedure for which the first half is similar to SAM_Ac_Result. One difference as seen in Figure 16a is that altitude vice size is considered in determining pkss. The look-up tables, however, have the same flaw as those for aircraft.

PROPOSED MODIFICATION: Change the Probability of Kill of Surface Weapons Versus Missiles tables to reflect the relative capabilities of the different SAMs.

The major difference between the second half of this procedure and that of SAM_Ac_Result is that instead of calling Fill_Ert to account for the various factors used in

pk determination those calculations are done internally by this procedure. At the end of SAM_Ac_Result the Battle Damage Assessment Routine must be called to determine the effect of any successful SAM shots. This is because multi-engined threat aircraft are not necessarily destroyed by a single SAM. In this procedure however, hostile missiles if hit by SAMs are always considered destroyed and therefore it is unnecessary to call Battle Damage Assessment.

In SAM_Msl_Threat the percentage of missiles killed in the hostile track is determined. This can be seen near the end of Figure 16b with the calculations of interest being:

PK = (1-PK) ** (NO/N_TGTS)

PK_PROD = PK_PROD * PK

where NO is the number of salvos and N_TGTS in the number of missiles in the threat track. These are repeated until all SAM systems have fired at which time the percent killed is determined as:

PK_PROD = 1 - PK_PROD.

This formula is correct when the number of SAMs is greater than or equal to the number of targets, but incorrect otherwise.

As an example, if PKSS = 0.9, NO = 1, and N_TGTS = 8, and there is only one SAM system in the engagement, then 0.25 would be the final PK_PROD. In other words, a single SAM destroyed 25 percent of 8, or 2, hostile missiles. In reality this may be possible in certain circumstances but

since this capability is not allowed in SAM_Msl_Result it should not be allowed here.

Furthermore, the formula used in calculating percentage killed assumes that the SAM platforms are coordinated in their firing. As seen in the Threat Procedures this is not necessarily the case. Therefore, this too should be taken into consideration. These problems were not coding errors but rather modeling errors as the NWGS Course Guide, [Ref. 2, p. 320], the PPS, [Ref. 3, p. 249], and the PDD, [Ref. 4, p. 3-688], all contain the same discrepancy.

PROPOSED MODIFICATION: In Figure 16b, or in the actual PL/I code, changes should be made after the calculation:

```
PKSS = PKSS * (1-  (1-ECCM_EFF)  *
ECM_EFF)
```

Here each SAM fired must be recursively summed, say in PKSS_SUM, so that the average pkss of all shots fired can be calculated later. To do this PKSS must be multiplied by SHOTS, the number of SAMs fired in the particular salvo being considered this time through the loop and this number added to PKSS_SUM. Also, the number of shots fired must be recursively summed each pass through the loop, say in TOT_SAMS. The remaining two calculations of PK and PK_PROD in the loop as it is now must be removed.

Upon completion of the necessary number of passes through the previous loop, several calculations must be made.

PROPOSED MODIFICATION: PKSS_AVG can be calculated by dividing PKSS_SUM by the total number of shots fired. Then one needs to determine if the SAM platforms are coordinated in their fire by checking SE_STAT(10) which has the value "1"b if coordinated.

Additionally, if a coordinated defense is in effect, then one must determine if the number of SAMs fired is less than the number of targets.

PROPOSED MODIFICATION: Compare the total number of SAMs shot with N_TGTS and set a flag if the former is less than the latter.

PROPOSED MODIFICATION: Calculate the percentage of incoming missiles killed according to one of the following three formulas depending on the situation:

Percent killed (uncoordinated fire) =

1- ((1-(PKSS_AVG/N_TGTS))**TOT_SAMS)

[Ref. 8].

Percent killed (coor; SAMS >= targets) =

1-((1-PKSS_AVG)**(TOT_SAMS/N_TGTS))

[Ref. 8].

Percent killed (coor; SAMs < targets) =

PKSS_AVG*TOT_SAMS/N_TGTS.

IV. RELOAD PHASE

Prior to any reloading it is first determined whether the platform firing the SAM or the SAM system itself has been destroyed since firing. If both are still operable then the reload takes place. The number of rounds that should be reloaded is found using the salvo size times the number of rounds fired per salvo for this weapon as indicated in the appropriate weapon_system table. If this number is actually available to this system then a full reload occurs, otherwise the number reloaded equals the greatest integer number of salvos that is possible.

This final phase of the SAM Routine, shown in Figure 6, is not covered in the documentation. It does, however, appear to accurately model the reloading of SAM weapons.

V. CONCLUSION

Personal experience in playing the Naval Warfare Gaming System led to the conclusion that it is an excellent device for the education of Naval decision makers and for investigative purposes even though a few discrepancies became obvious during game play. Upon closer examination however, namely verification of the computer code with documentation as well as a check as to the validity of the model, additional discrepancies were found. None of these discrepancies were drastic when viewed singly but when one considers the total number found in the entire SAM Routine it leads to doubts as to how well this system really models SAM engagements. Even more importantly if the number of discrepancies found in this one small section of NWGS is representative of the number found throughout the system then one must consider the possible synergistic effect that such discrepancies have on the outcome of the games.

In every instance where a discrepancy was discovered a modification was proposed. These proposals are specific enough so contractor personnel can make these changes readily under the supervision of the Navy personnel at the Center for War Gaming.

Future analysis and improvements to NWGS could be made easier if the contractor was required to provide better

documentation. This could be accomplished by correcting the existing contractor supplied publications, the PPS [Ref. 3], and the PDD [Ref. 4], to accurately reflect what is realized in the computer code. It could also be accomplished by completing the comment blocks that exist inside the PL/I code but which have not been filled in. These blocks should be filled in well in advance of final acceptance of NWGS so that Center for War Gaming personnel have the opportunity to use this information to assist them in reviewing the product. The flow charts in Appendix A of this thesis were derived from the contractor supplied code since good documentation did not exist for the SAM Routine. The intention is that this work will assist others in further analysis of this routine.

APPENDIX A

NWGS FLOW CHARTS

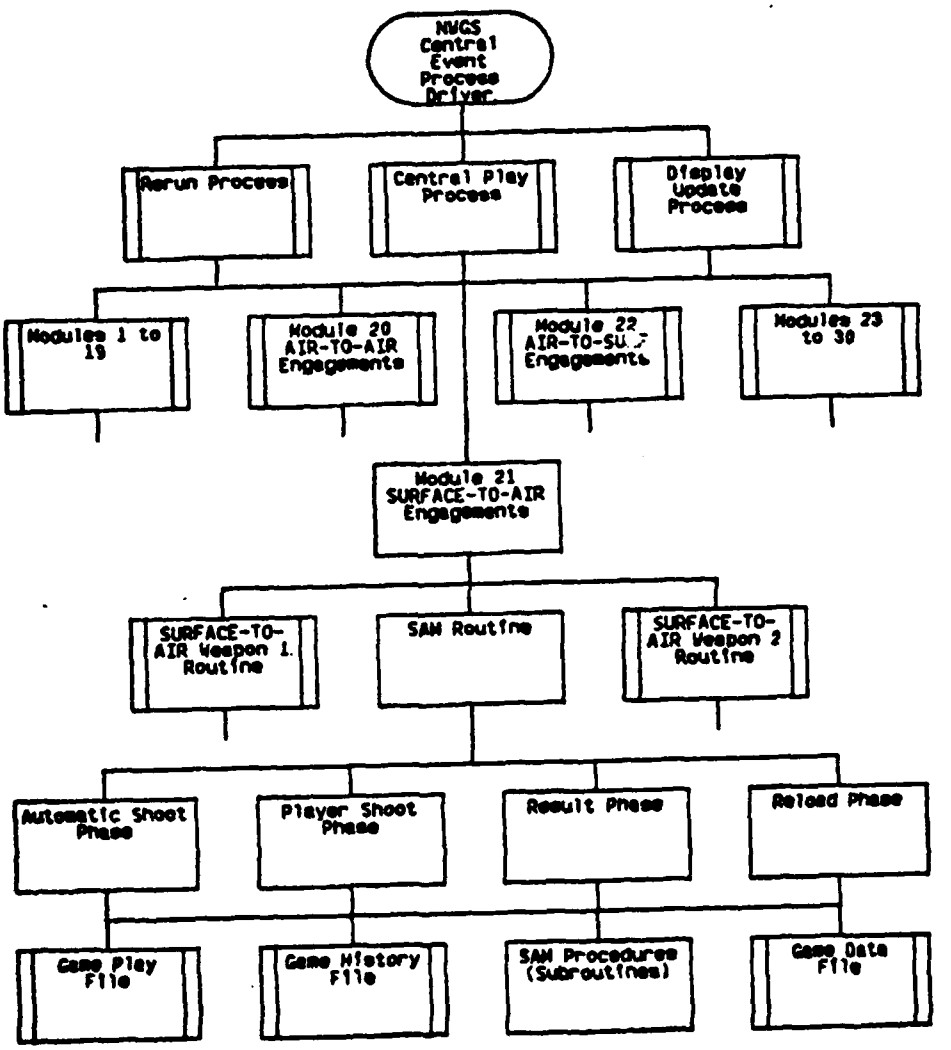


Figure 1: Naval Warfare Gaming System

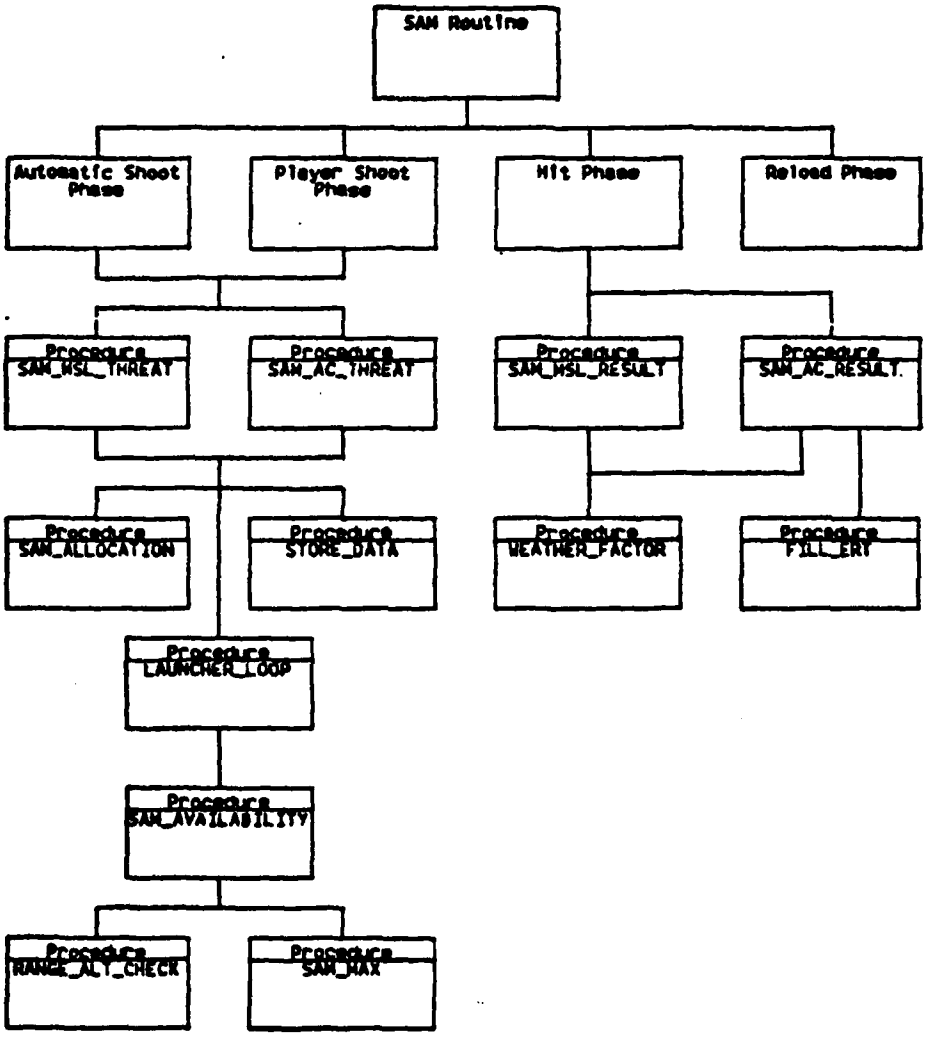


Figure 2: SAM Routine

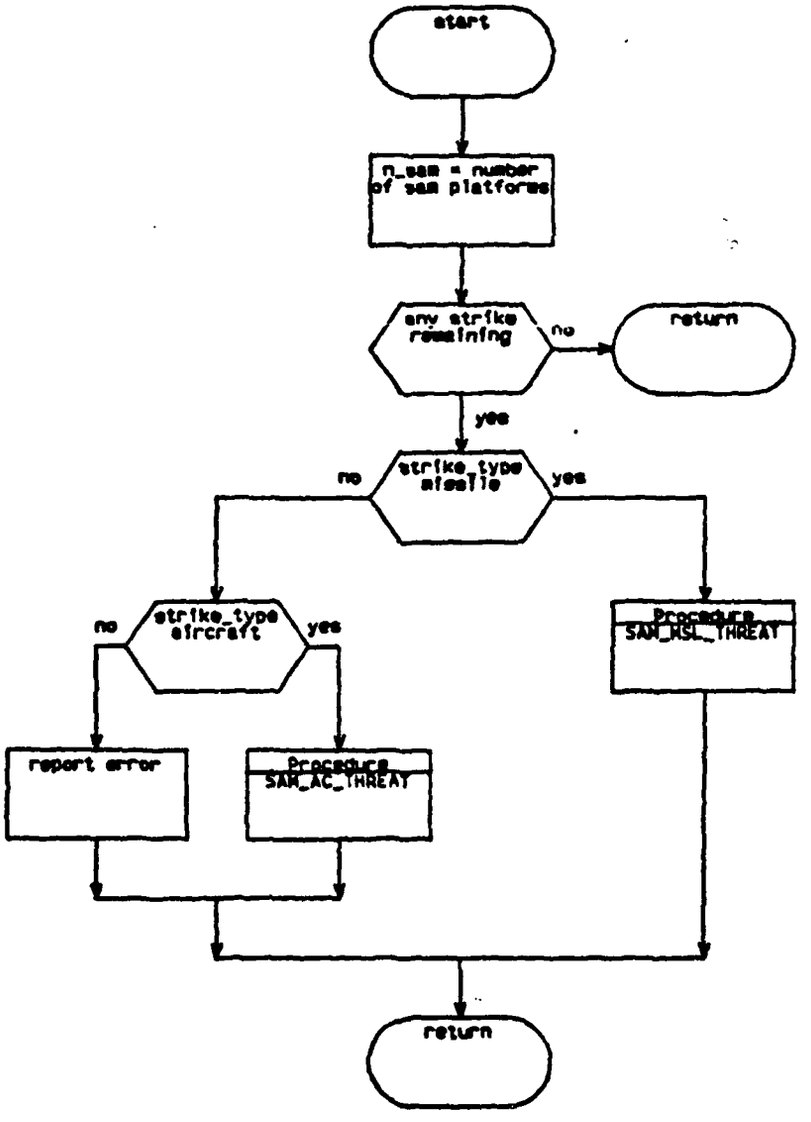


Figure 3: SAM Routine Automatic Shoot Phase

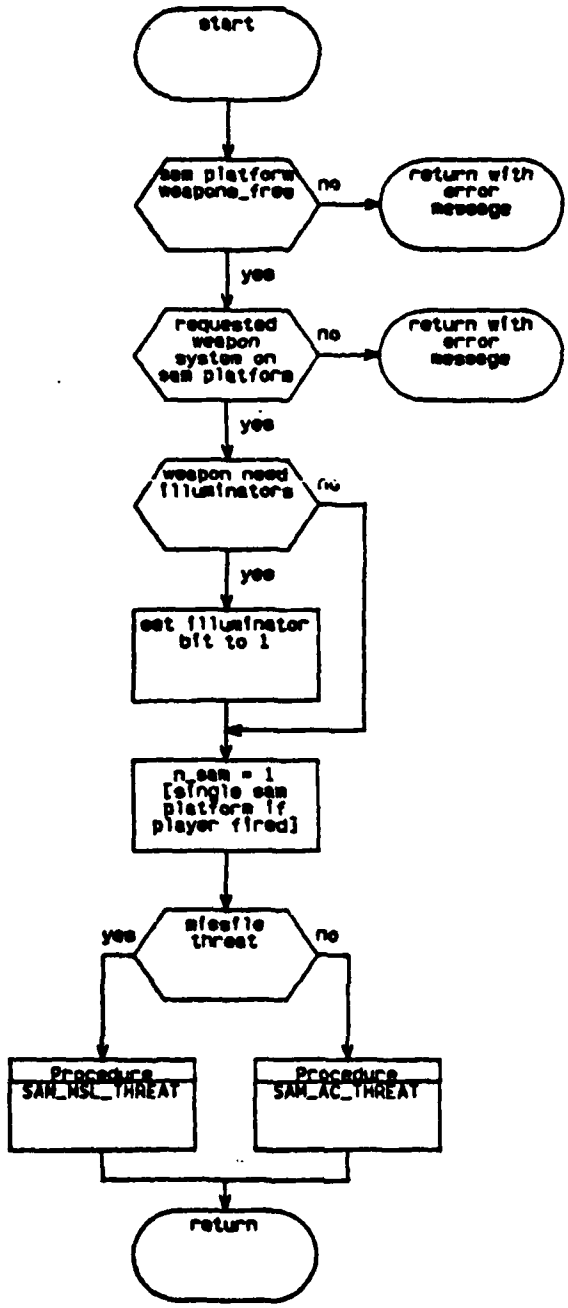


Figure 4: SAM Routine Player Shoot Phase

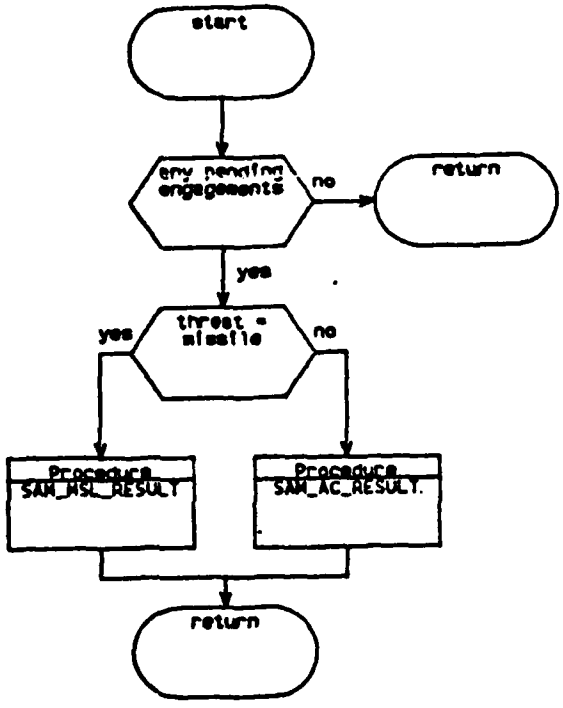


Figure 5: SAM Routine Hit Phase

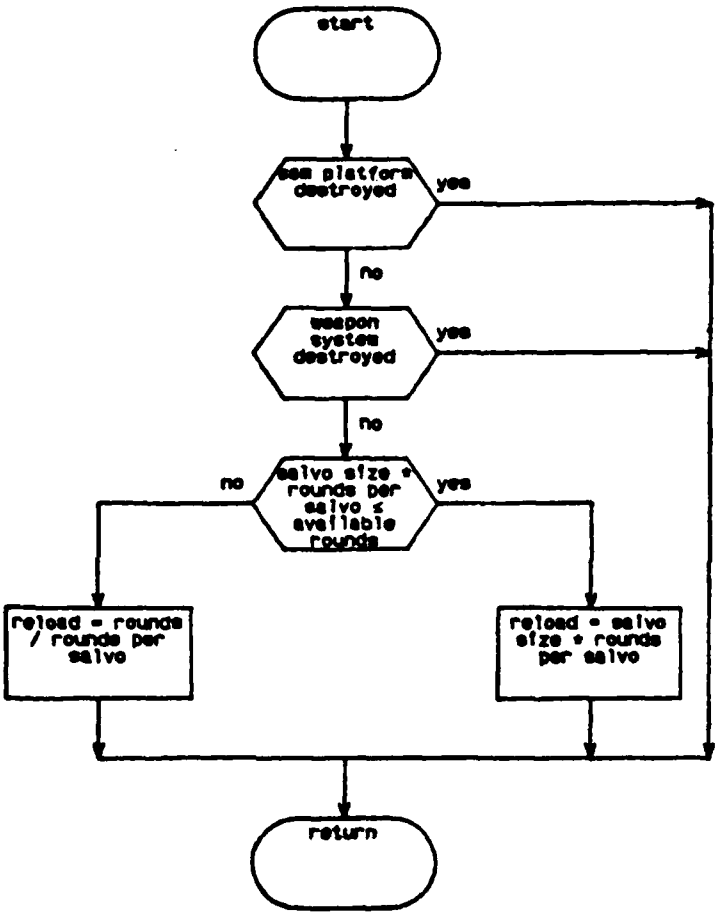


Figure 6: SAM Routine Reload Phase

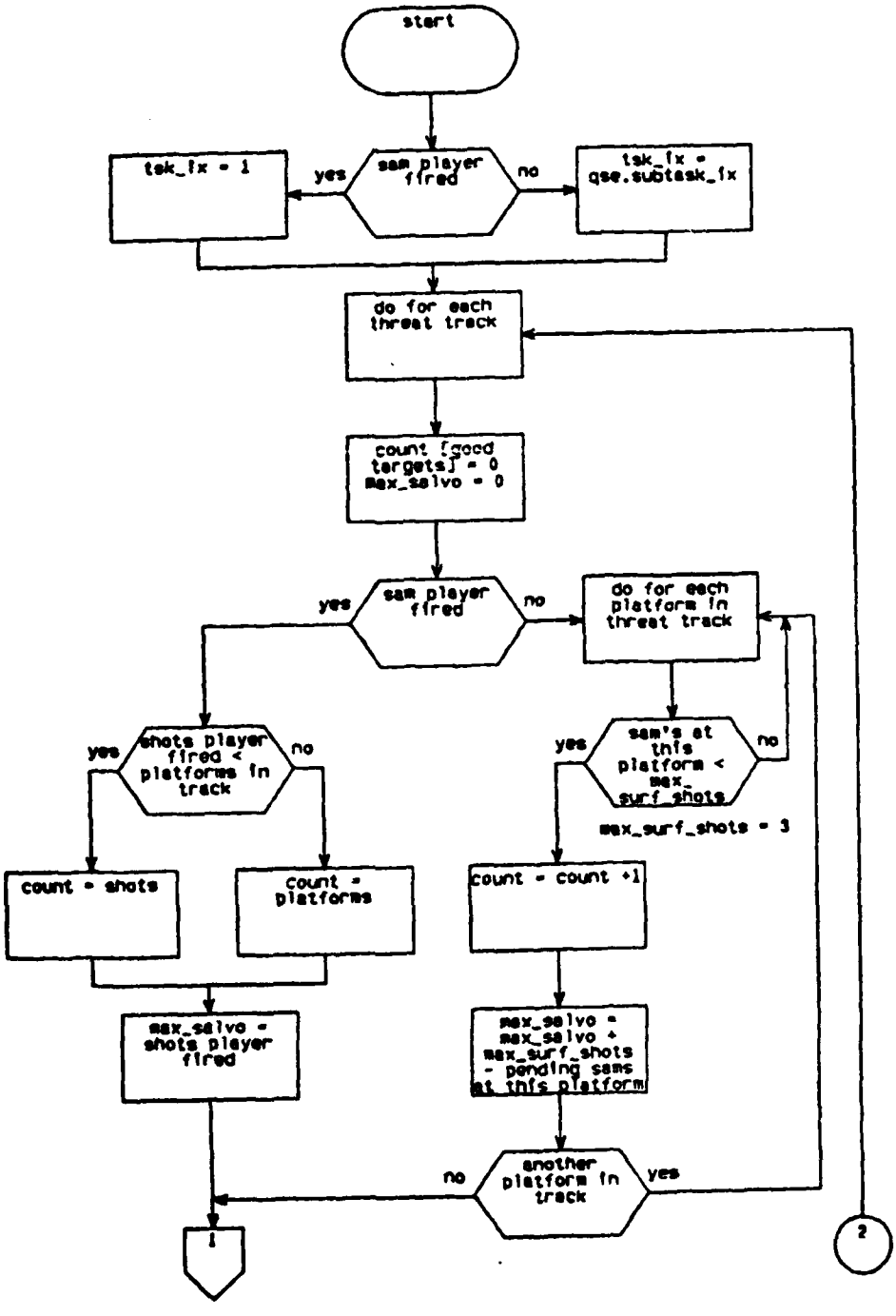


Figure 7a: Sam_Ac_Threat Procedure

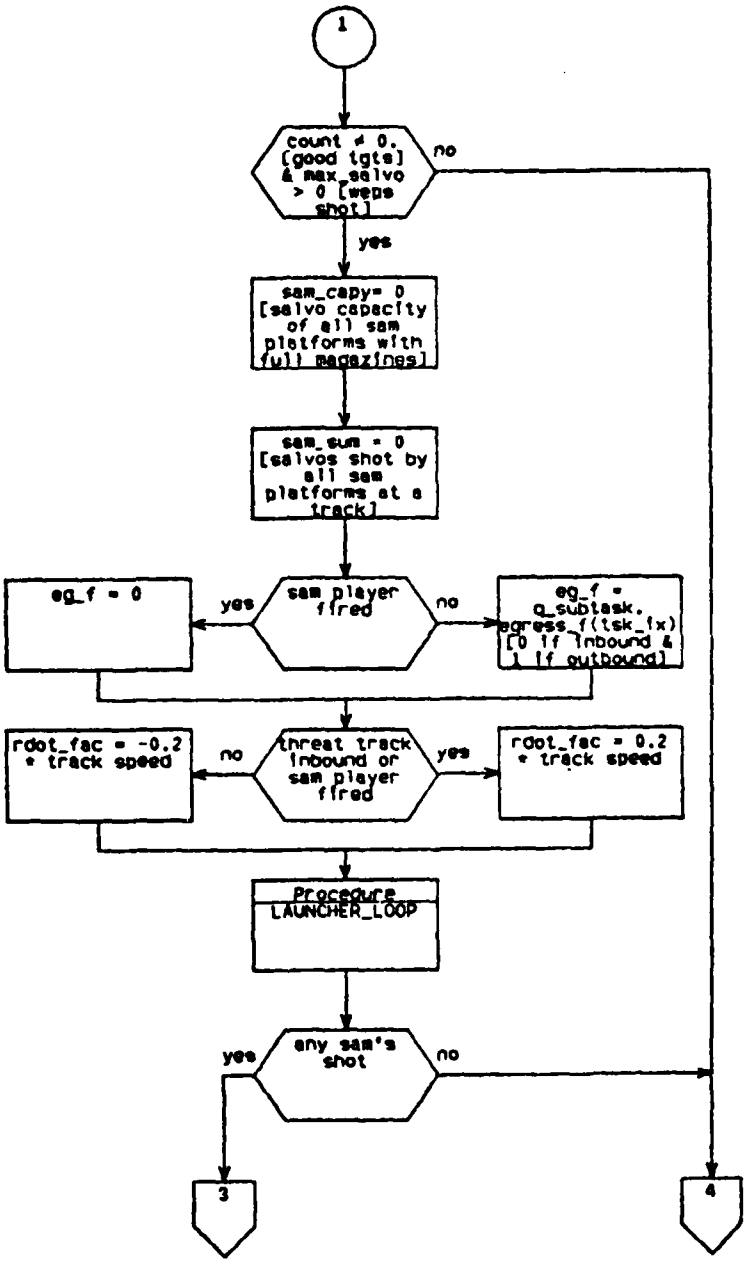


Figure 7b: Sam_Ac_Threat Procedure

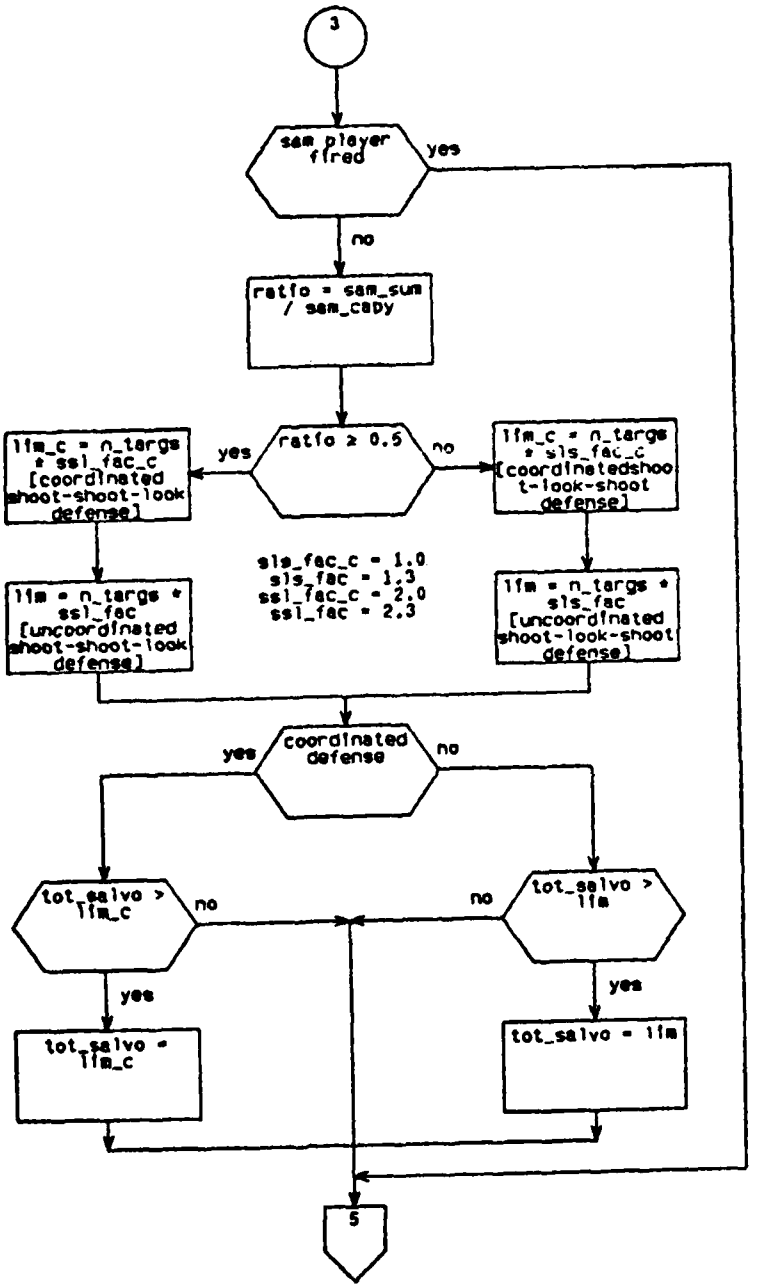


Figure 7c: Sam_Ac_Threat Procedure

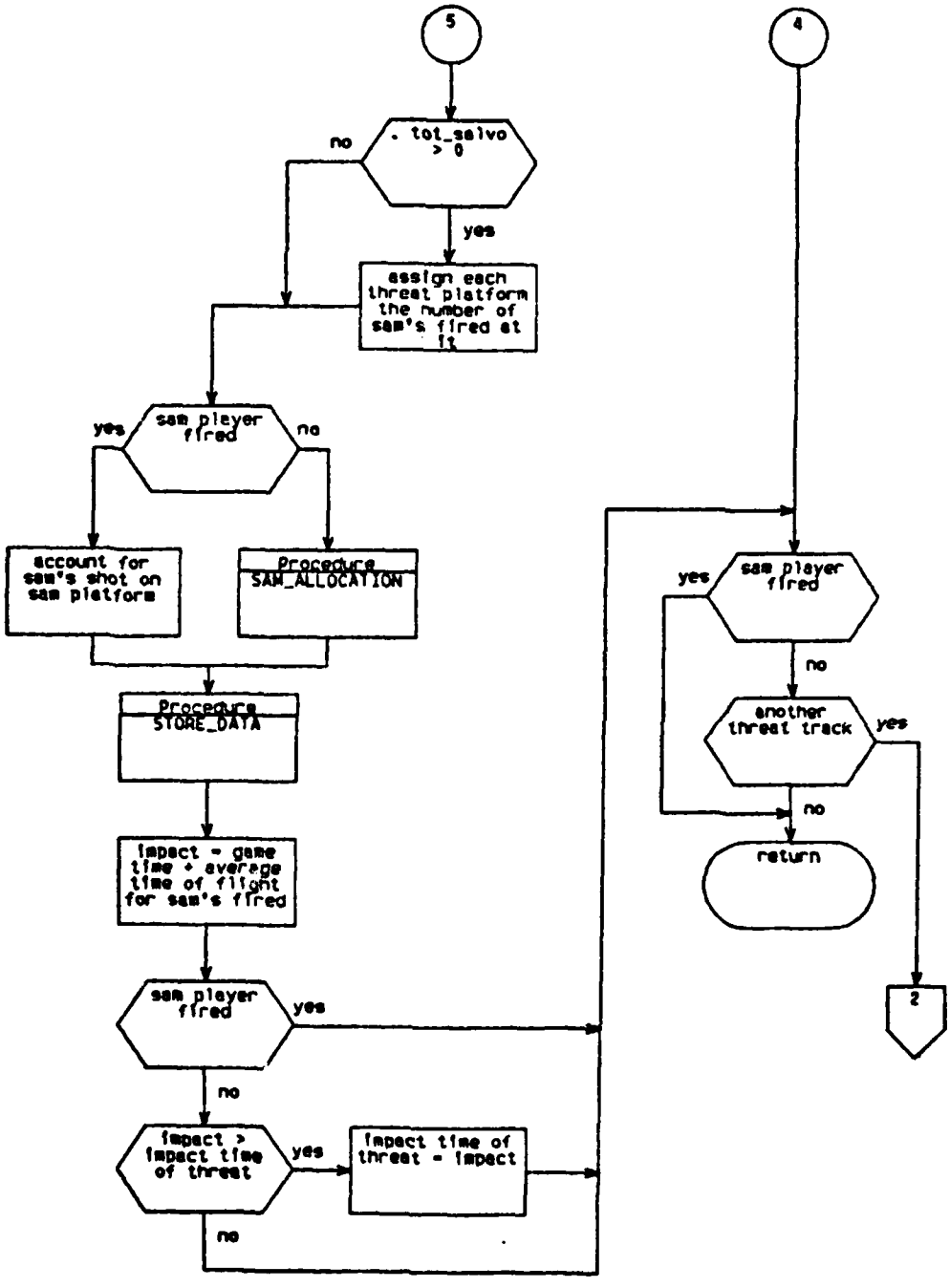


Figure 7d: Sam_Ac_Threat Procedure

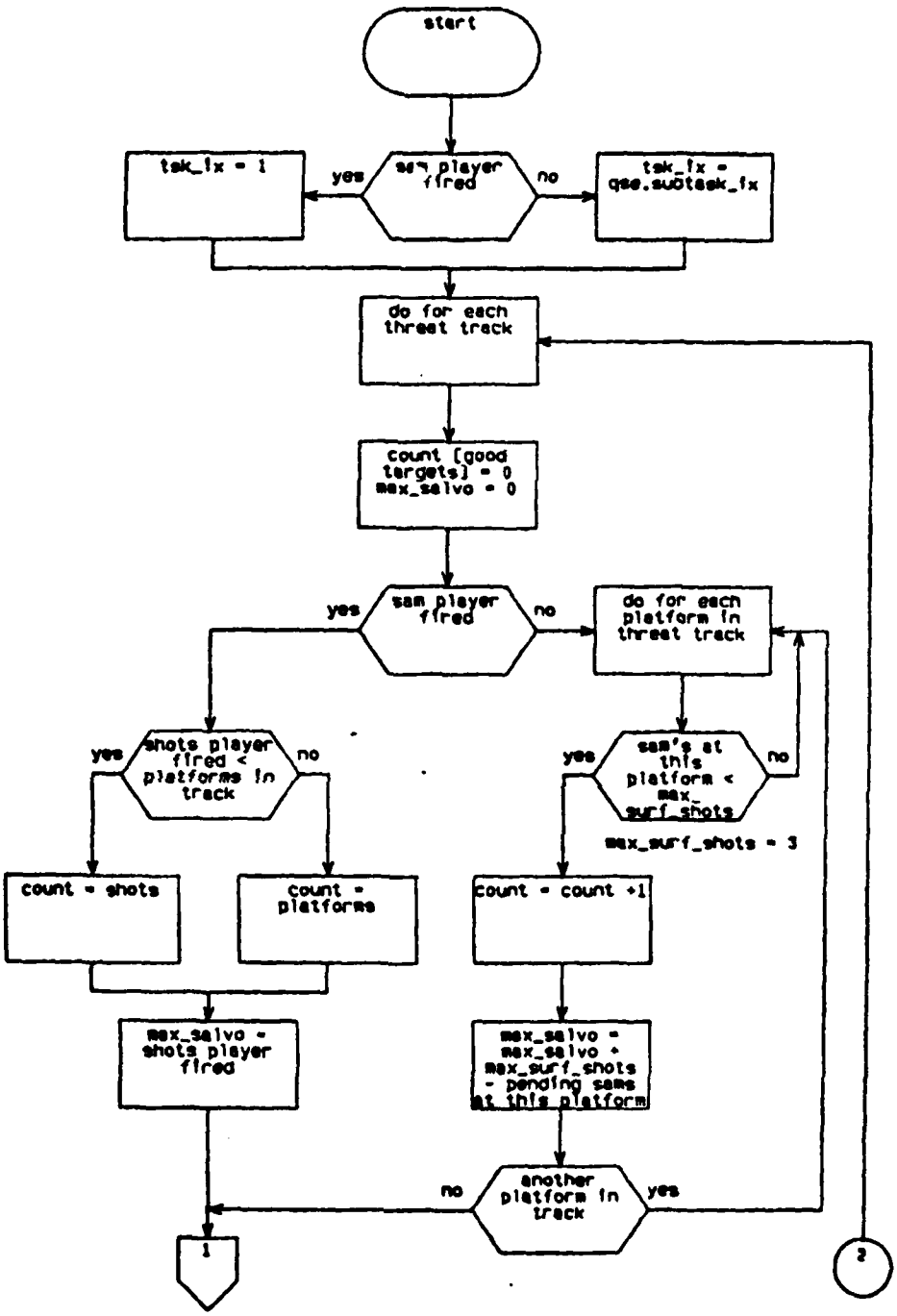


Figure 8a: Sam_Msl_Threat Procedure

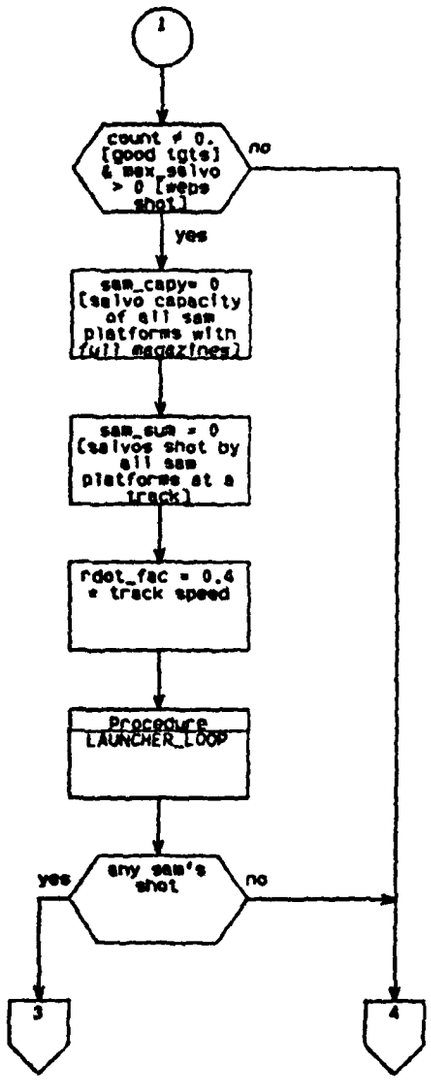


Figure 8b: Sam_Msl_Threat Procedure

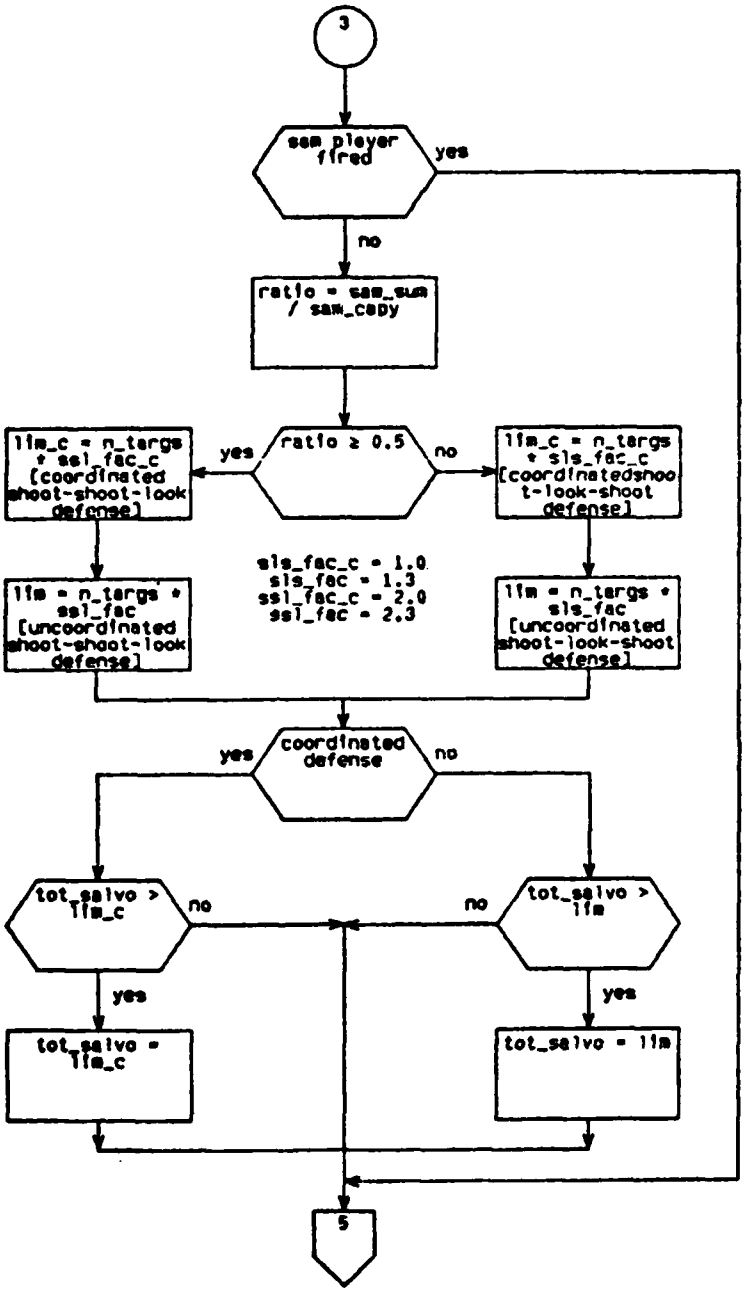


Figure 8c: Sam_Msl_Threat Procedure

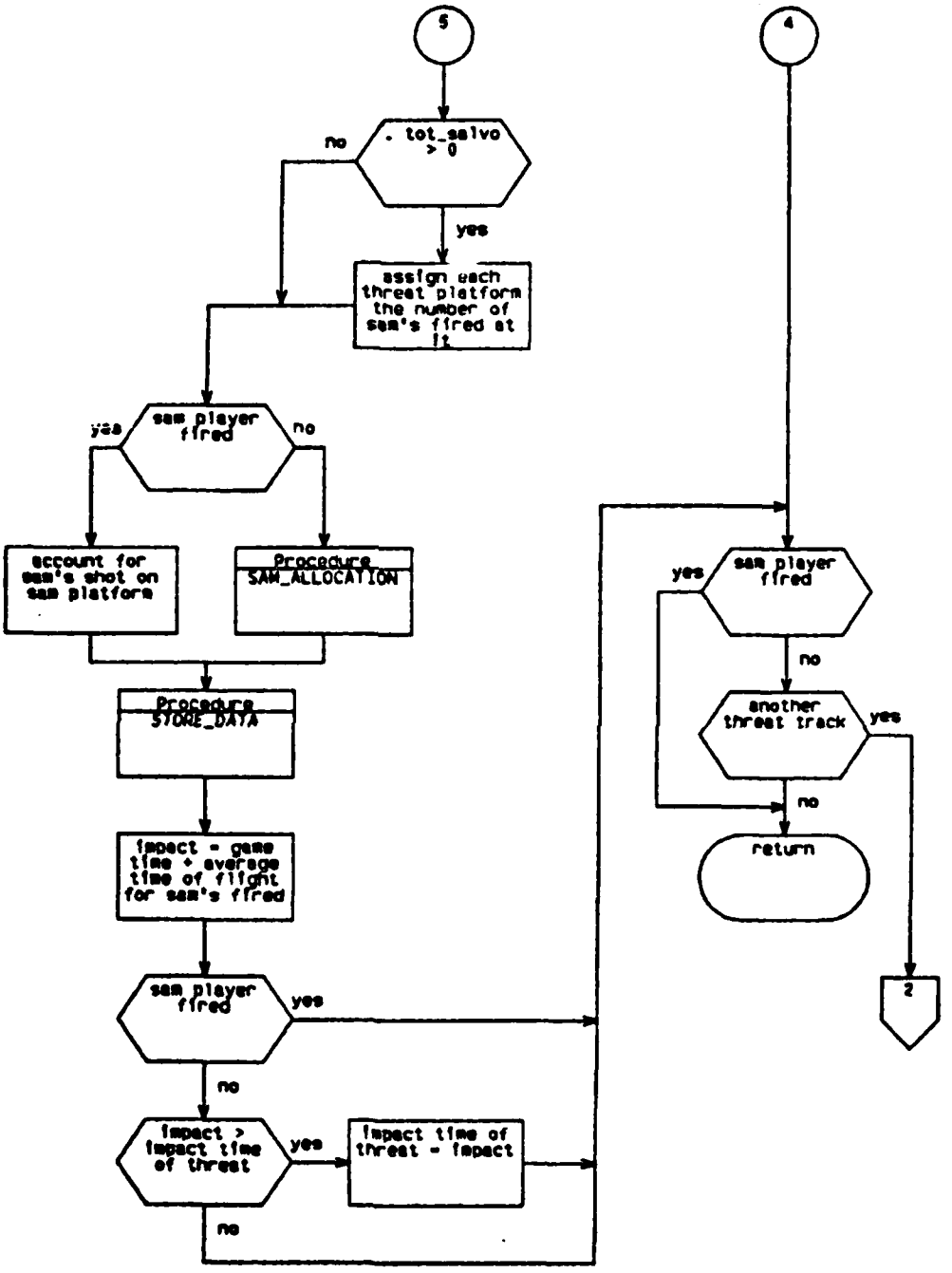


Figure 8d: Sam_Msl_Threat Procedure

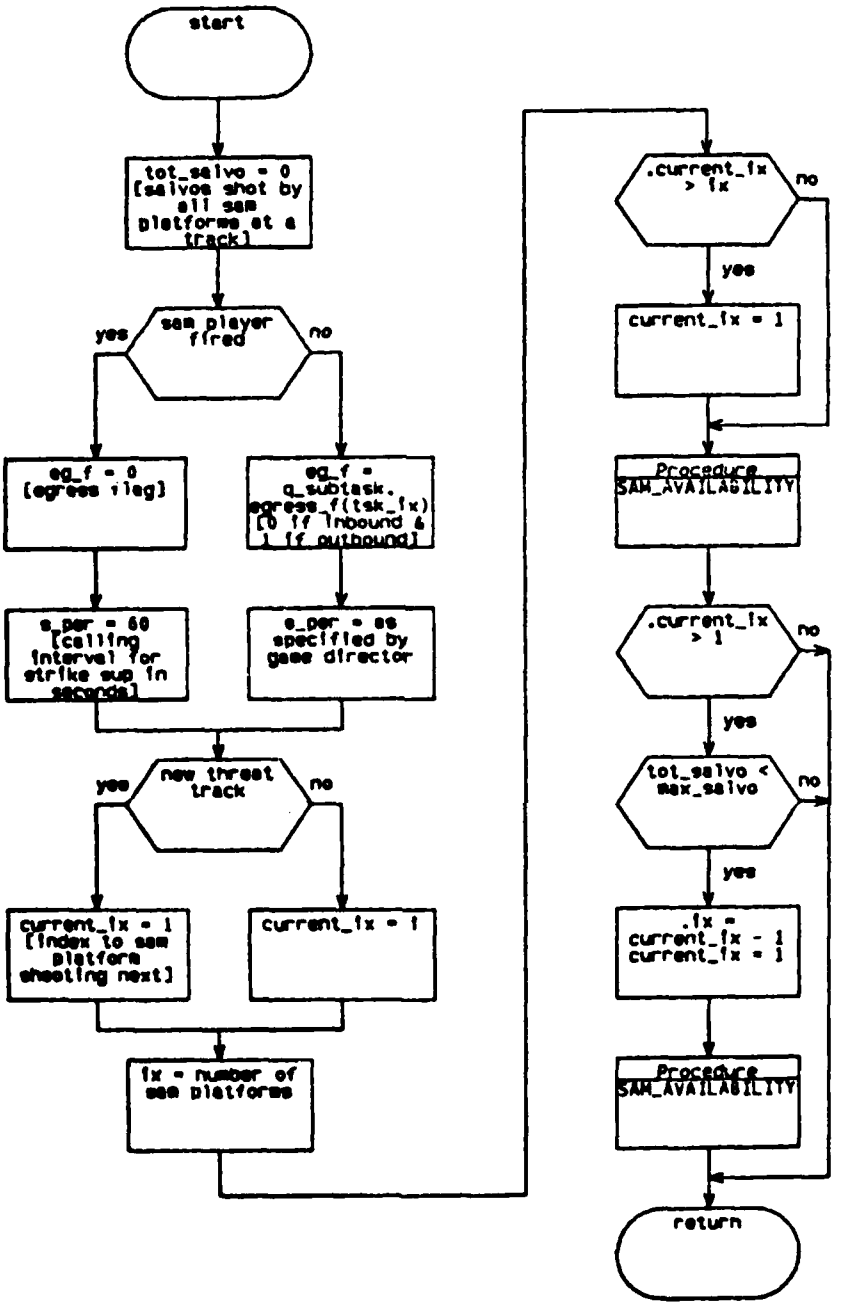


Figure 9: Launcher_Loop Procedure

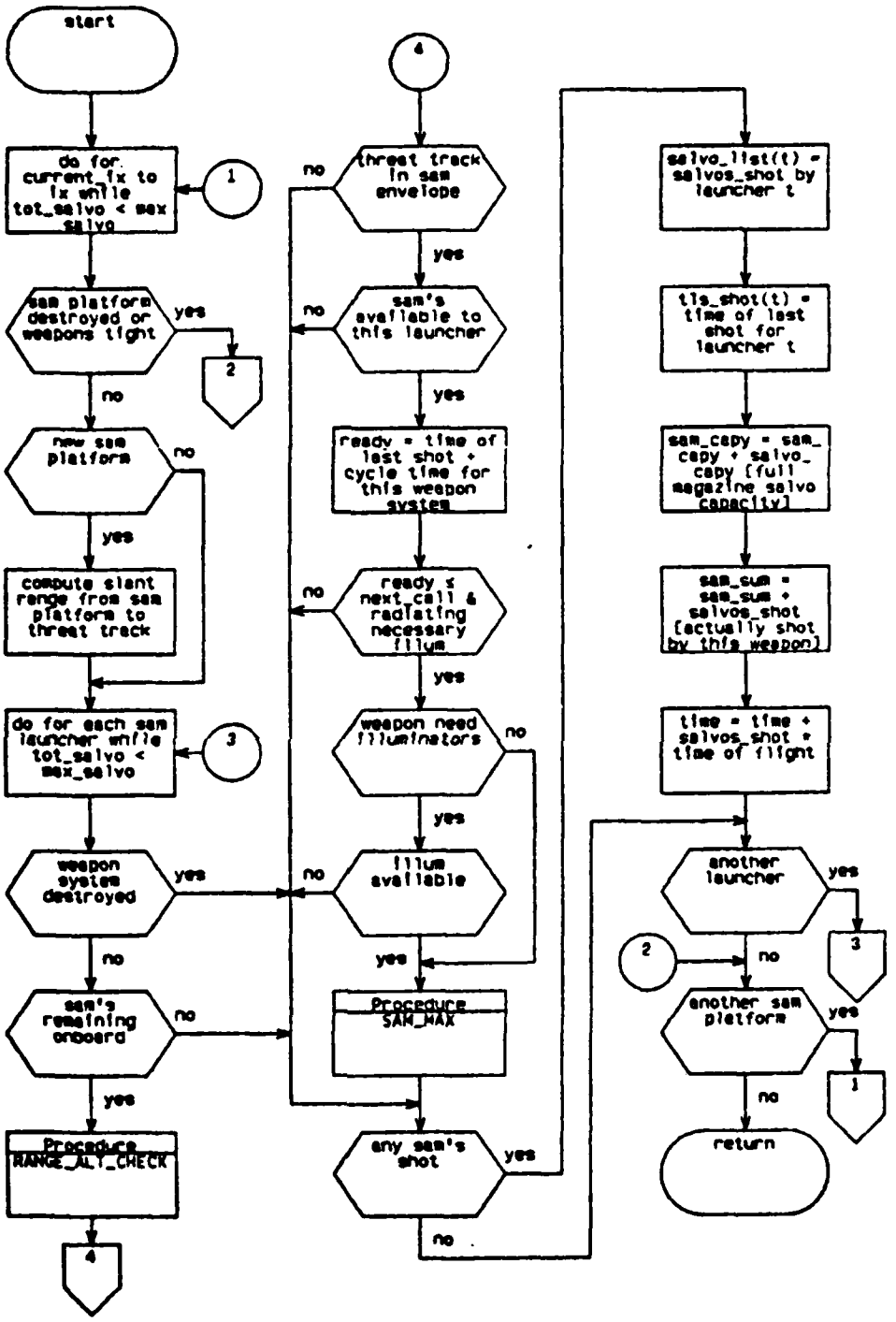


Figure 10: Sam_Availability Procedure

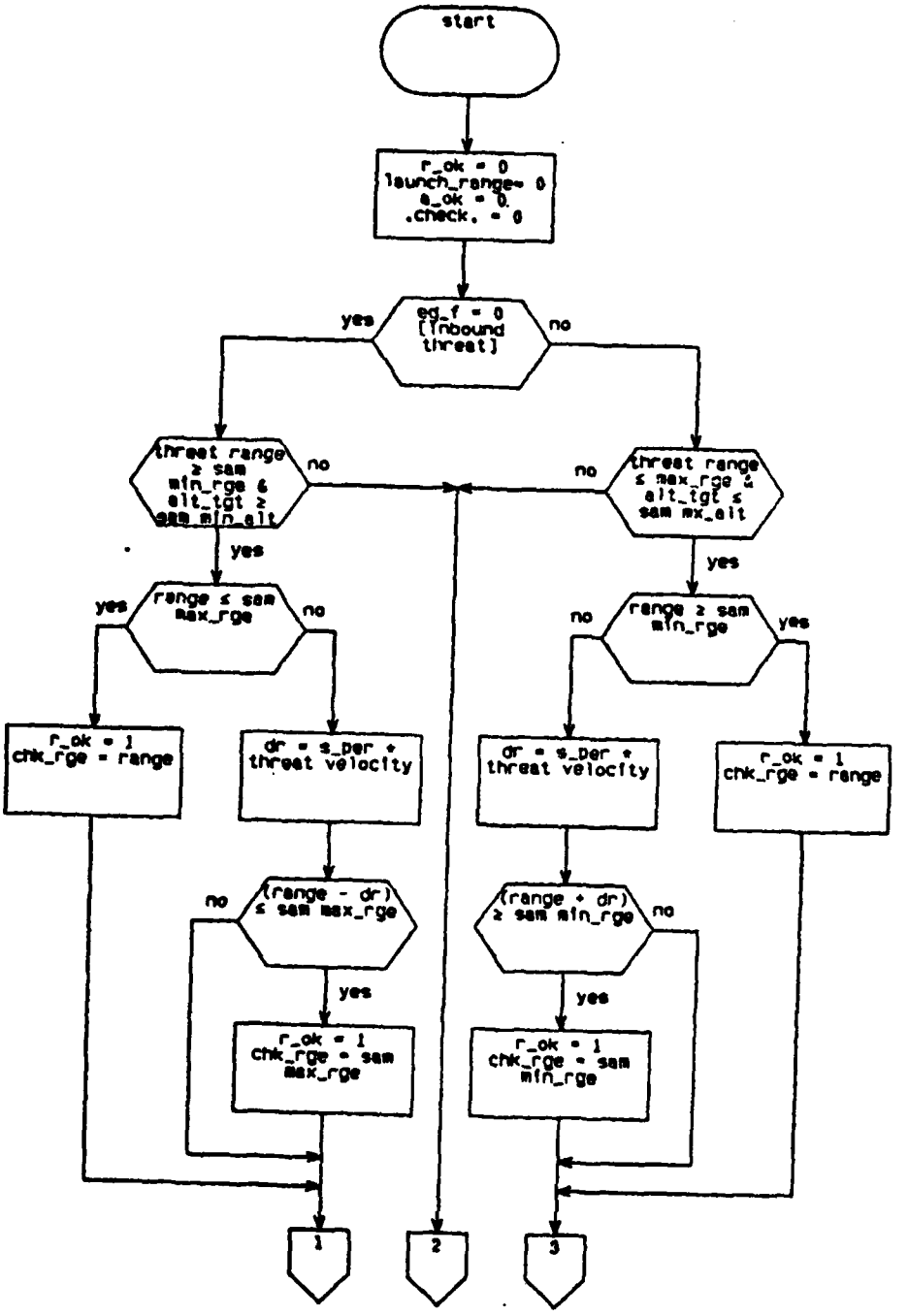


Figure 11a: Range_Alt_Check Procedure

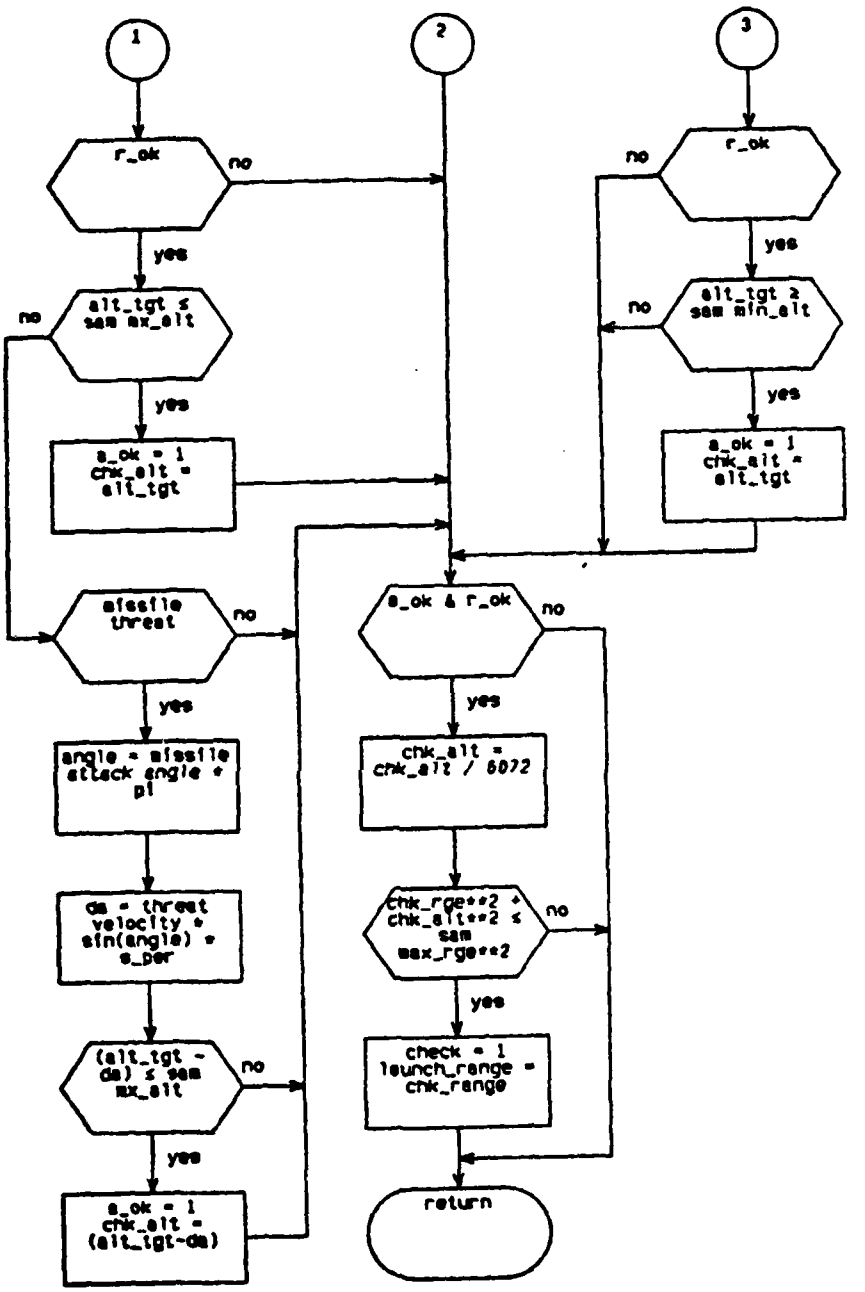


Figure 11b: Range_Alt_Check Procedure

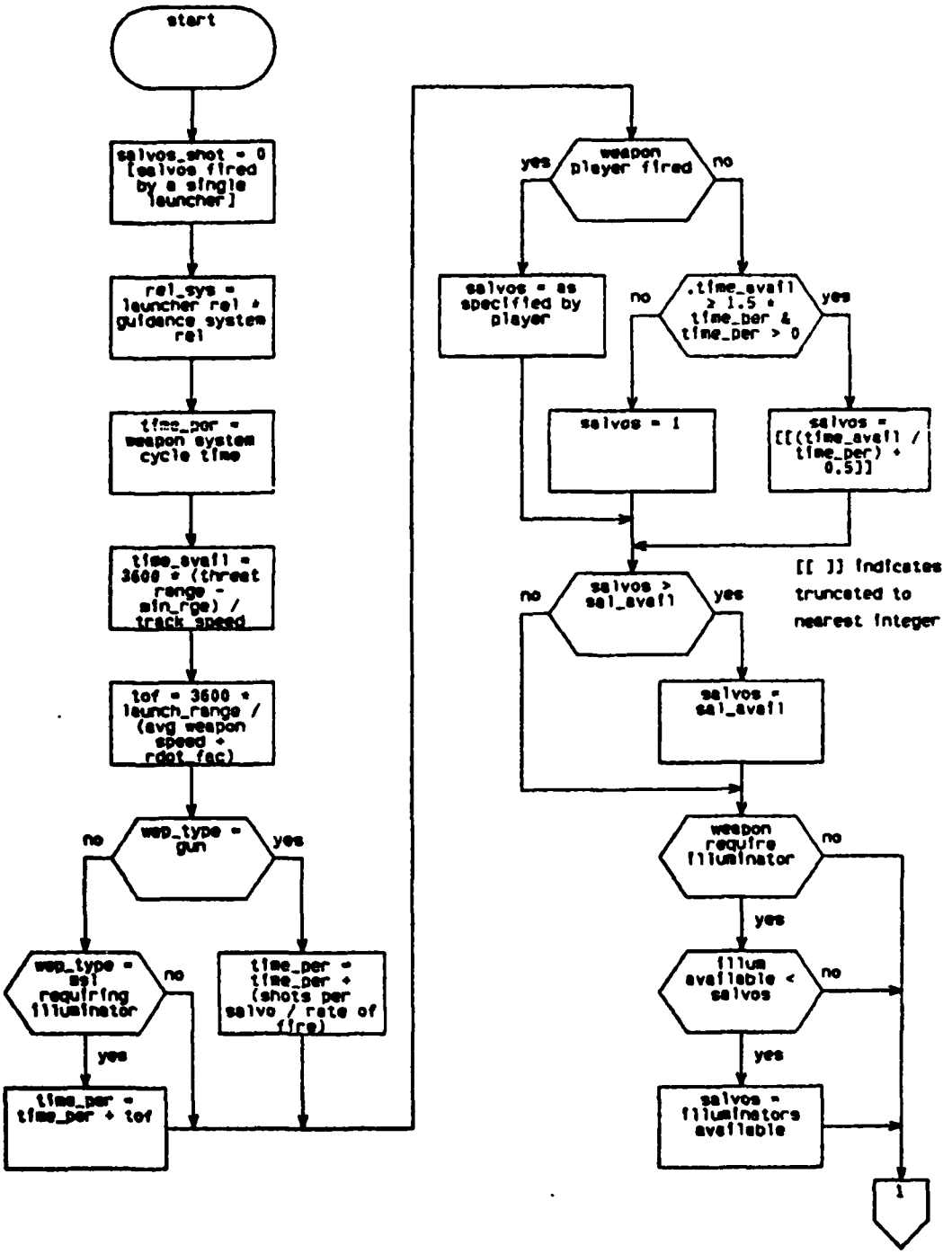


Figure 12a: Sam_Max Procedure

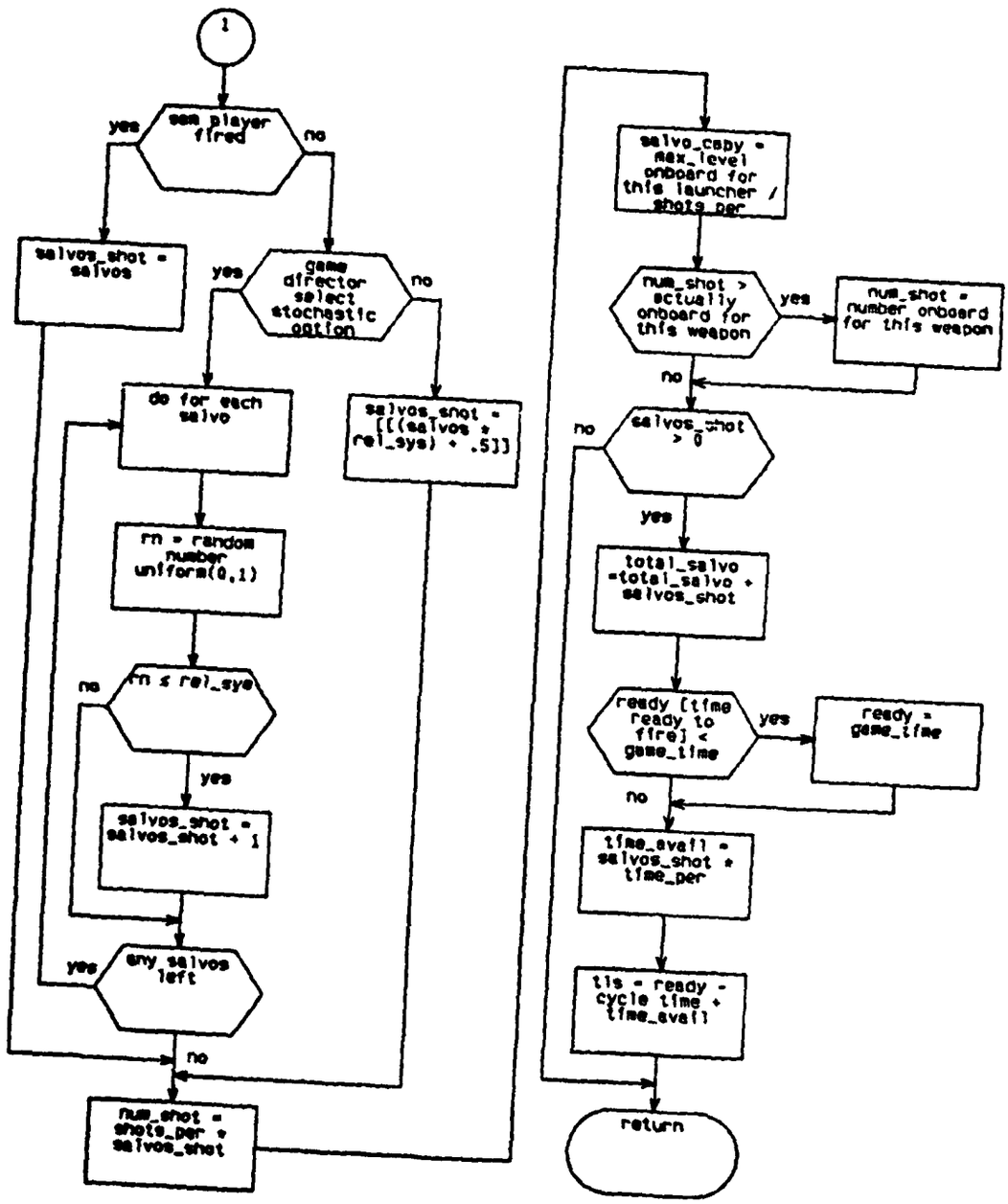


Figure 12b: Sam_Max Procedure

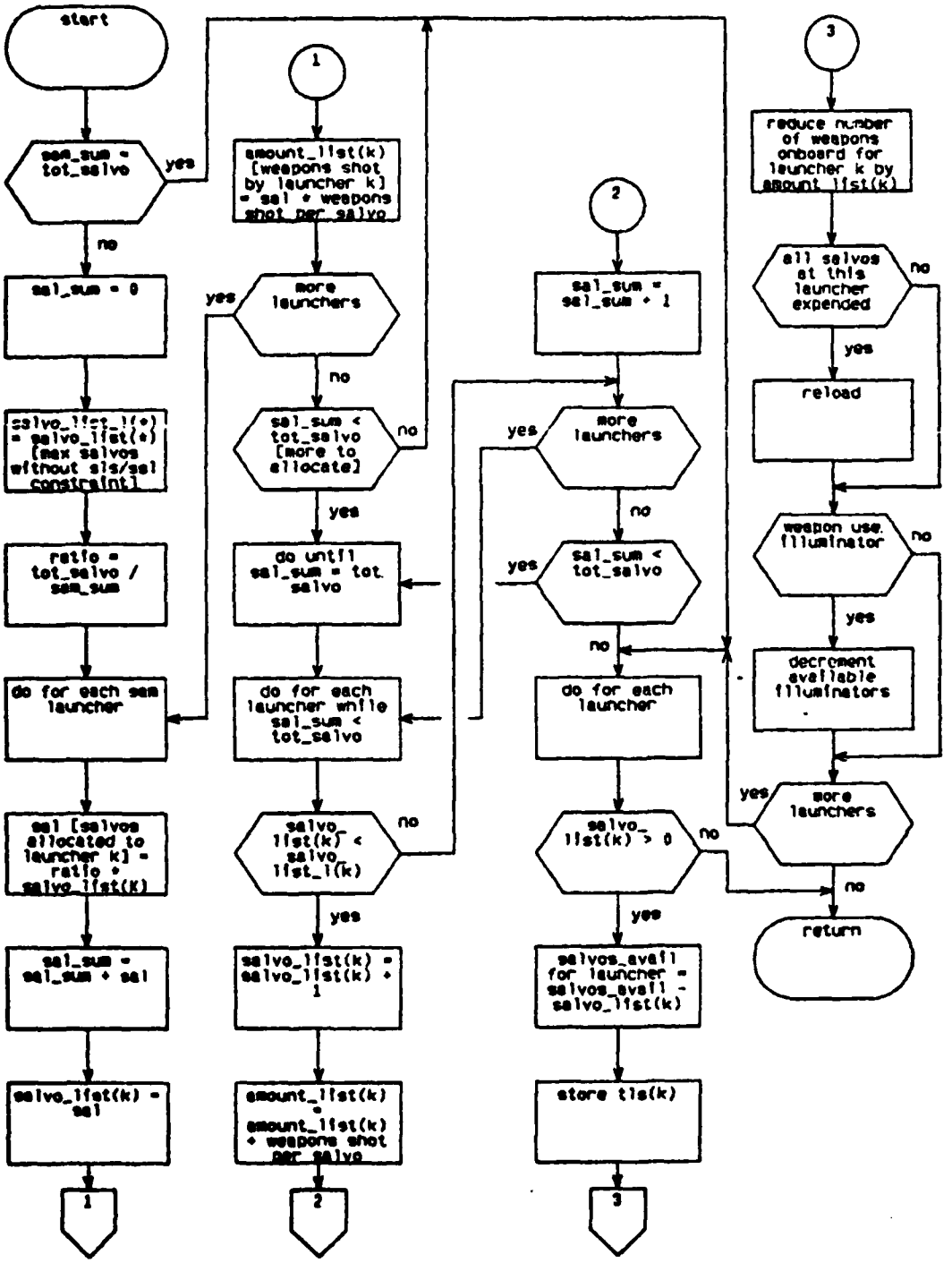


Figure 13: Sam_Allocation Procedure

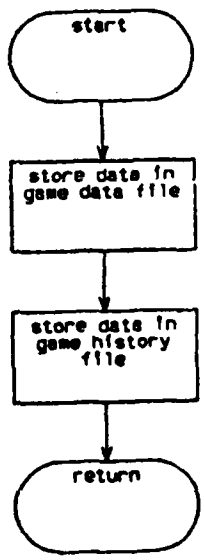


Figure 14: Store_Data Procedure

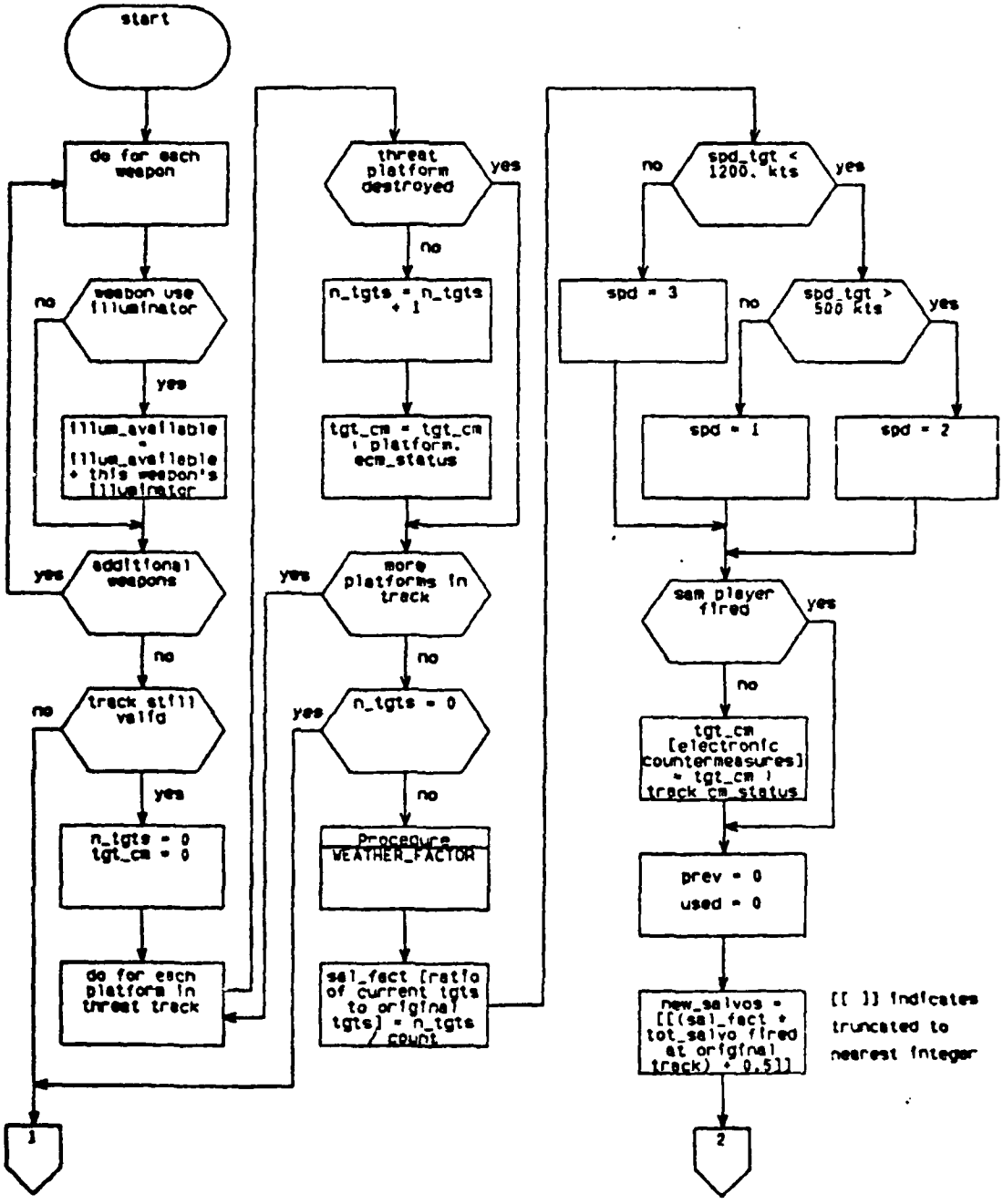


Figure 15a: Sam_Ac_Result Procedure

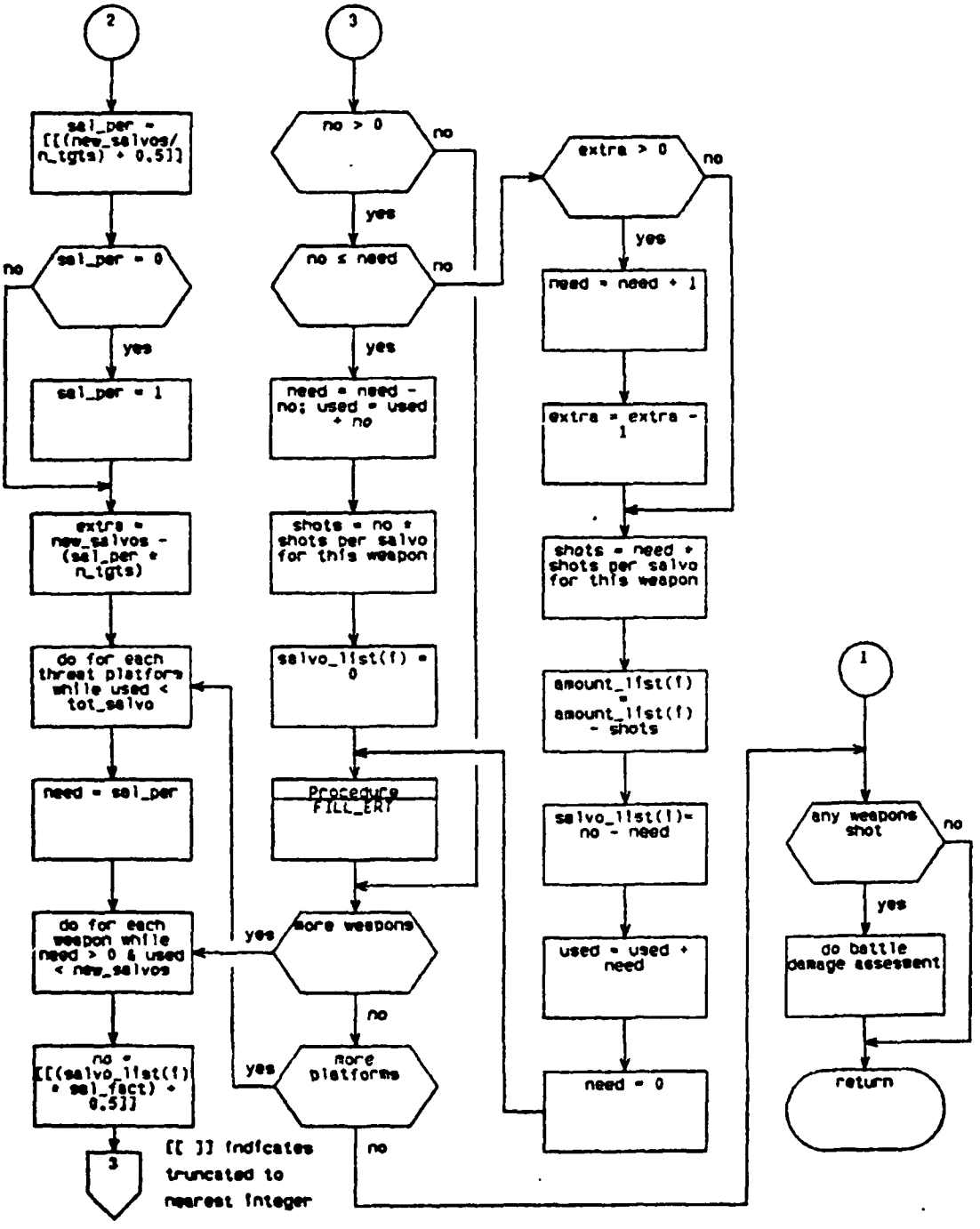


Figure 15b: Sam_Ac_Result Procedure

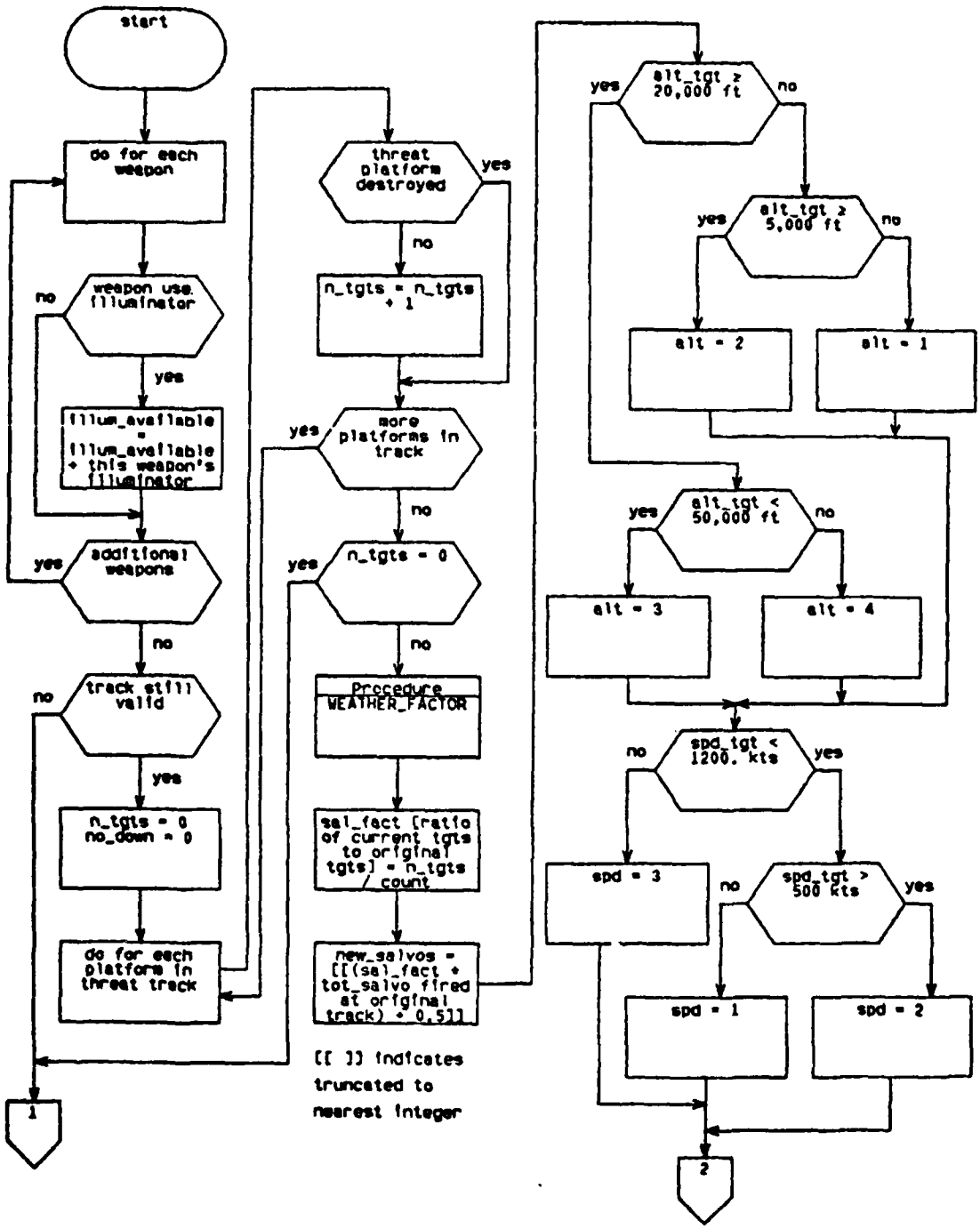


Figure 16a: Sam_Msl_Result Procedure

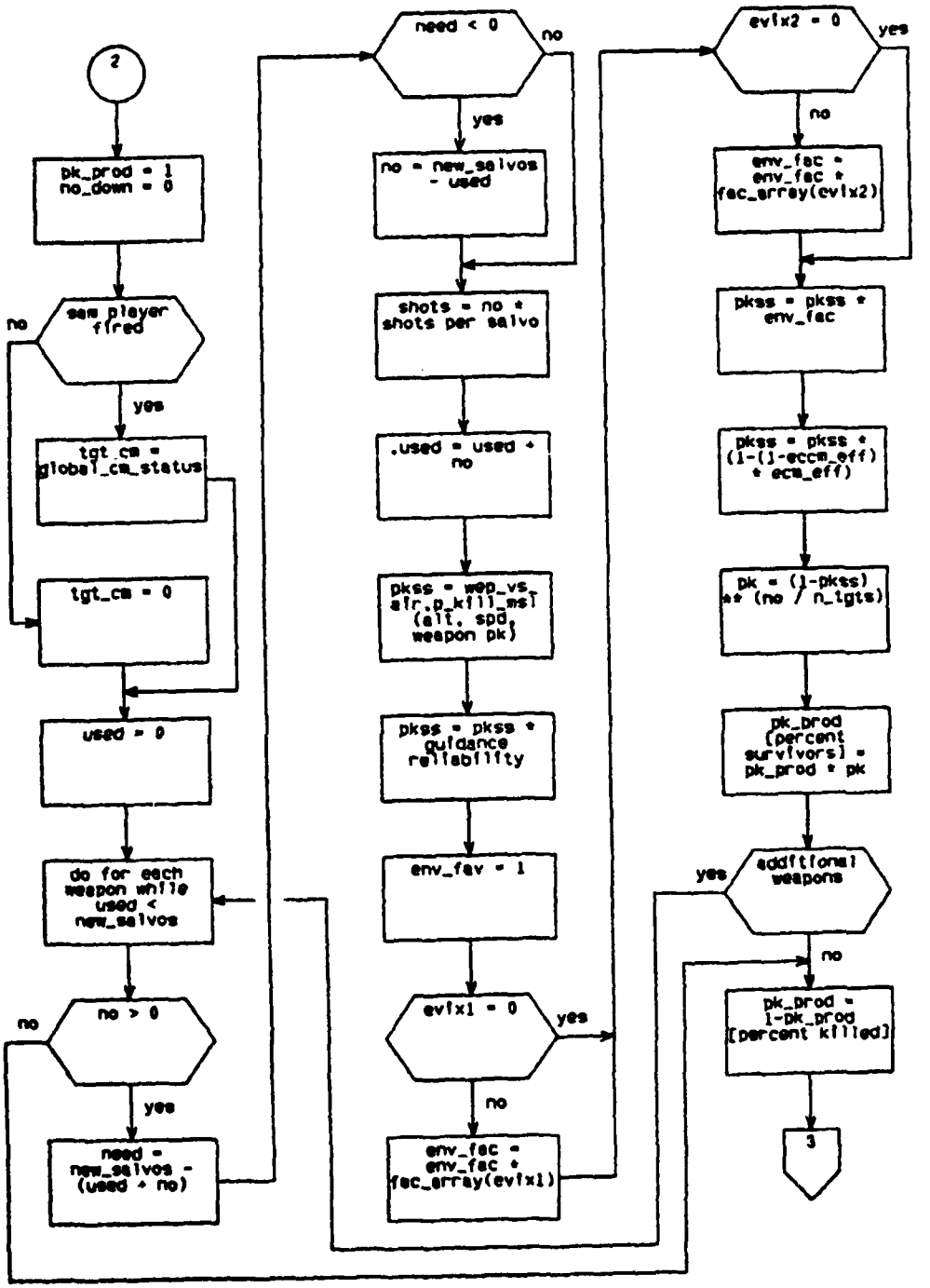


Figure 16b: Sam_Msl_Result Procedure

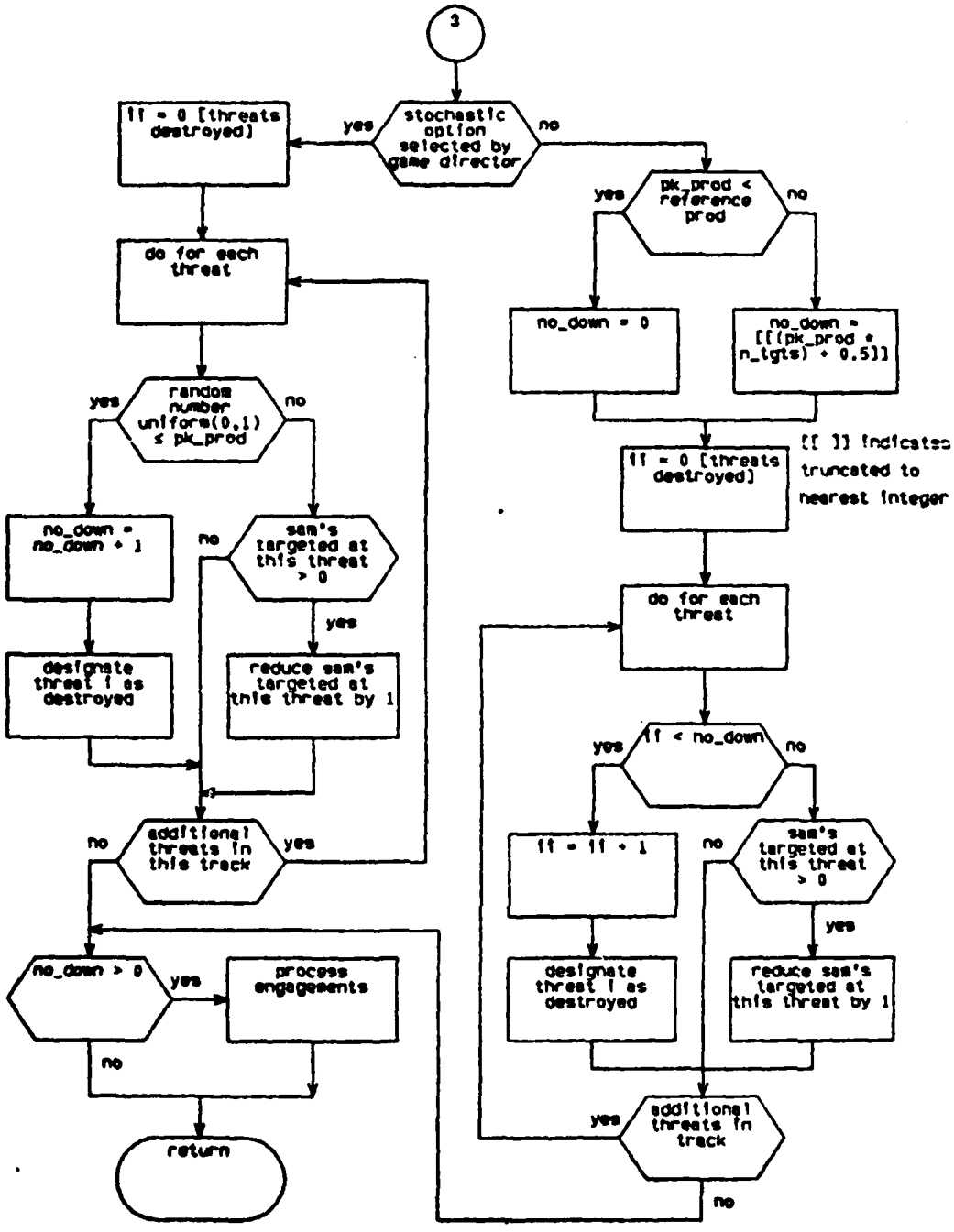


Figure 16c: Sam_Msl_Result Procedure

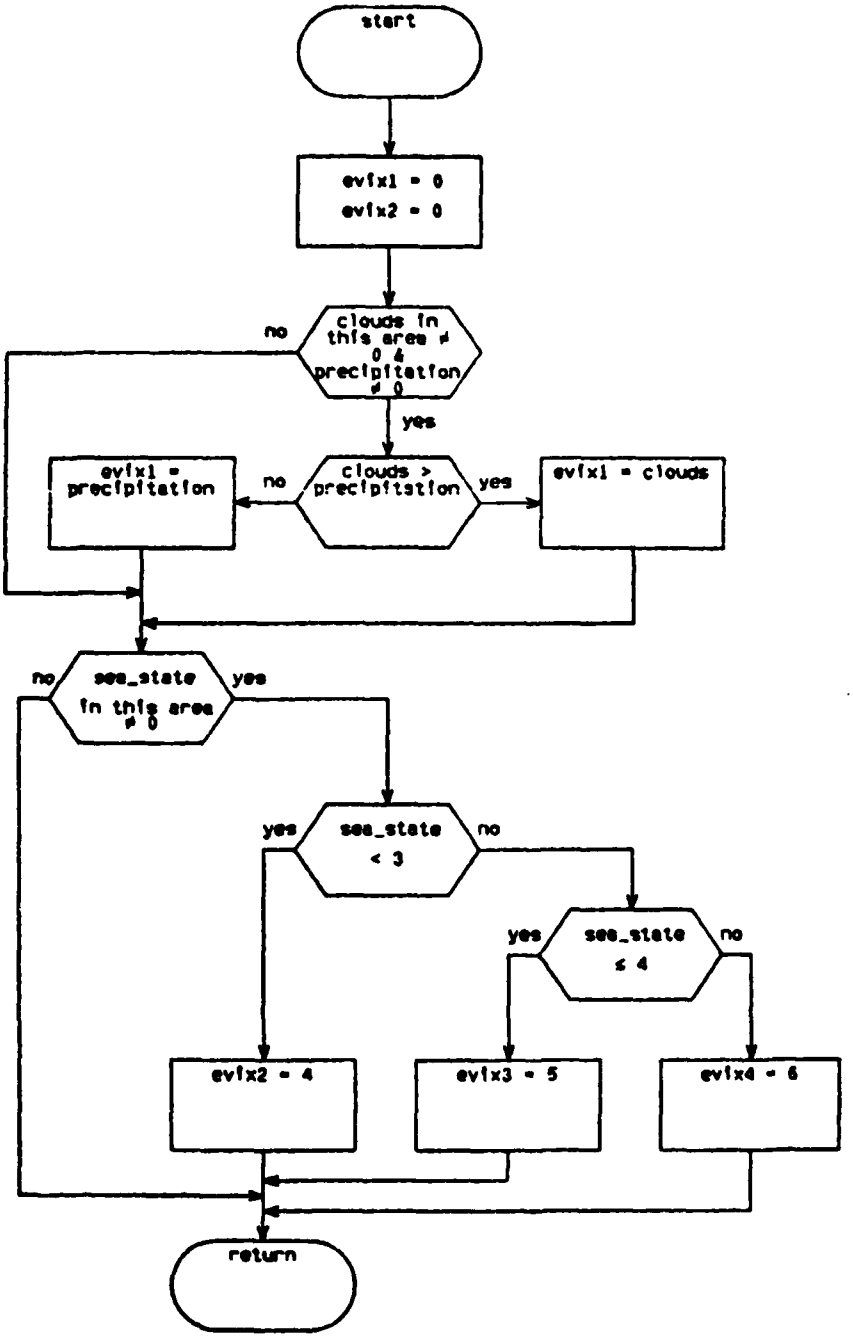


Figure 17: Weather_Factor Procedure

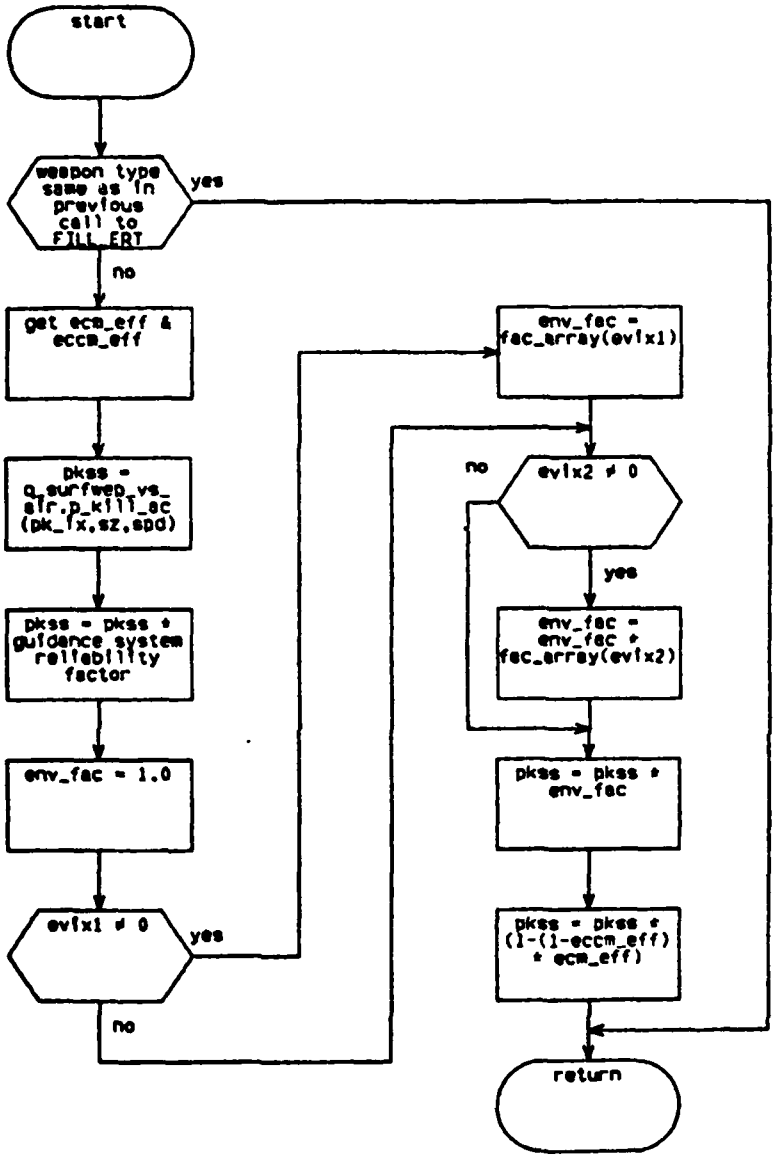


Figure 18: Fill_Ert Procedure

LIST OF REFERENCES

1. Computer Sciences Corporation, Command and Staff Users Manual for Naval Warfare Gaming System (NWGS), Center for War Gaming, U.S. Naval War College, Newport, RI, June 1981.

2. Computer Sciences Corporation, Naval Warfare Gaming System (NWGS) Student's Training Course Guide for Application Software, Center for War Gaming, U.S. Naval War College, Newport, RI, October 1982.

3. Computer Sciences Corporation, Naval Warfare Gaming System (NWGS) Program Performance Specification (PPS), Center for War Gaming, U.S. Naval War College, Newport, RI, January 1981.

4. Computer Sciences Corporation, Naval Warfare Gaming System (NWGS) Program Description Document Data Base (Models) Computer Program Configuration Item, Center for War Gaming, U.S. Naval War College, Newport, RI, November 1980.

5. Naval Warfare Gaming System (NWGS) Student's Training Course, video tape of, Center for War Gaming, U.S. Naval War College, Newport, RI, October 1982.

6. Paul E. Rubin, telephone conversation with, Computer Sciences Corporation, Moorestown, NJ, May 10, 1983.

7. Nathaniel Bowditch, American Practical Navigator, An Epitome of Navigation, Vol. 1, Defense Mapping Agency Hydrographic Center, Pub. no. 9, 1977.

8. Alan R. Washburn, Professor of Operations Research, Combat Models and Games, class notes on, Naval Postgraduate School, Monterey, CA, October 20, 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Professor A.F. Andrus, Code 55As Naval Postgraduate School Monterey, California 93940	1
4.	Cdr. G.R. Porter, USN, Code 55Pt Naval Postgraduate School Monterey, California 93940	1
5.	Cdr. R. Adams, USN Center for War Gaming Naval War College Newport, Rhode Island 02840	2
6.	Lcdr P. Craig, USN Center for War Gaming Naval War College Newport, Rhode Island 02840	1
7.	Lcdr. M. Thomas, USN Center for War Gaming Naval War College Newport, Rhode Island 02840	1
8.	Professor J.N. Eagle, Code 55Er Naval Postgraduate School Monterey, California 93940	1
9.	Capt. W.P. Hughes, USN, Code 55Hi Naval Postgraduate School Monterey, California 93940	1
10.	Professor M.G. Sovereign, Code 55Zo Naval Postgraduate School Monterey, California 93940	1
11.	Professor J.K. Hartman, Code 55Hh Naval Postgraduate School Monterey, California 93940	1

12. Professor R.H. Moose, Code 39A — 1
Naval Postgraduate School
Monterey, California 93940

13. Professor J.M. Wozencraft, Code 62Wz — 1
Naval Postgraduate School
Monterey, California 93940

14. Lt. Dennis Stokowski, USN — 2
326 Swanee Dr.
North Dighton, Massachusetts 02764


DA
FILM