

INFORMATION TO USERS

This manuscript has been reproduced from the microfilm master. UMI films the text directly from the original or copy submitted. Thus, some thesis and dissertation copies are in typewriter face, while others may be from any type of computer printer.

The quality of this reproduction is dependent upon the quality of the copy submitted. Broken or indistinct print, colored or poor quality illustrations and photographs, print bleedthrough, substandard margins, and improper alignment can adversely affect reproduction.

In the unlikely event that the author did not send UMI a complete manuscript and there are missing pages, these will be noted. Also, if unauthorized copyright material had to be removed, a note will indicate the deletion.

Oversize materials (e.g., maps, drawings, charts) are reproduced by sectioning the original, beginning at the upper left-hand corner and continuing from left to right in equal sections with small overlaps.

Photographs included in the original manuscript have been reproduced xerographically in this copy. Higher quality 6" x 9" black and white photographic prints are available for any photographs or illustrations appearing in this copy for an additional charge. Contact UMI directly to order.

**ProQuest Information and Learning
300 North Zeeb Road, Ann Arbor, MI 48106-1346 USA
800-521-0600**

UMI[®]

Naval Battle Simulation System Optimization

LinFang Wang

A Major Report

in

The Department

of

Computer Science

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

March, 2002

© Linfang Wang, 2002



**National Library
of Canada**

**Acquisitions and
Bibliographic Services**

**395 Wellington Street
Ottawa ON K1A 0N4
Canada**

**Bibliothèque nationale
du Canada**

**Acquisitions et
services bibliographiques**

**395, rue Wellington
Ottawa ON K1A 0N4
Canada**

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-68483-0

Canada

Abstract

This dissertation is a Computer Science Master degree major report by student Linfang Wang. The main objective of this project is to utilize the object oriented methodologies to design and implement a simple Naval Battle Simulation System. The project is based on a Software Engineering course project taught by Dr. Joey Paquet at Concordia University. The project consists in respecification of the system requirements, optimization of the system design, UML notation improvement, reorganization of the structure of documents and rewriting of the SRS, SDD and STD documents. This document can be taken as an integrated standard for requirement, design, and testing documents for Naval Battle Simulation System, or any other similar Software Engineering project. This will enable Dr. Paquet to re-use this document as a valuable information source for other Software Engineering projects in the future.

The project applies the object-oriented design and implementation for all the subsystems. The developing tool is MFC and OpenGL. For the requirements specification, a requirements identifier scheme is applied to improve the traceability for the whole system. For system implementation, function overloading, virtual function and pure virtual function, multithreading, inheritance and polymorphism are used to improve the system generality, reuseability and flexibility as well.

Acknowledgements

I would like to express that it was very beneficial to work on my major report under the direction of Dr. Joey Paquet. He gave me lots of important suggestions and advises. His guidance helped me to make significant progress and enhance my knowledge as well. Sincerely, I appreciated Dr. Joey Paquet for his great help during the process of this project for my master study.

Also, I would like to say thanks to all the COMP554 (Software Engineering, Summer 2001) students for their great contribution, which I took as blueprint to start my project. Without their contribution, the project would have had to be started from scratch and probably, it would not have been possible for me to finish it alone.

Finally, my best wish to my lovely 2 years old son—Ian. I hope he will grow up to know more and more from the world and keep growing healthy, happily.

Table of Contents

1. INTRODUCTION	1
1.1 PURPOSE.....	1
1.2 SCOPE.....	2
1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS	3
1.4 OVERVIEW.....	4
1.5 REFERENCES.....	5
2. GENERAL DESCRIPTION.....	6
2.1 PRODUCT PERSPECTIVE	6
2.2 PRODUCT FUNCTIONS	6
2.3 USER CHARACTERISTICS	7
2.4 GENERAL CONSTRAINTS.....	7
2.5 ASSUMPTIONS AND DEPENDENCIES	8
3. SPECIFIC REQUIREMENTS.....	9
3.1 REQUIREMENT IDENTIFICATION.....	9
3.2 HIGH LEVEL USE CASE DESCRIPTION.....	10
3.3 FUNCTIONAL REQUIREMENTS DESCRIPTION.....	16
3.3.1 <i>Simulation Controller Requirements</i>	17
3.3.1.1 Use Case Diagram	17
3.3.1.2 Requirement Breakdown	18
3.3.1.3 Use Case Description	22
3.3.2 <i>Communication/Detection Requirements</i>	33
3.3.2.1 Use Case Diagram	33
3.3.2.2 Requirement Breakdown	34
3.3.2.3 Use Case Description	37
3.3.3 <i>Aircraft Carrier Requirements</i>	51
3.3.3.1 Use Case Diagram	51
3.3.3.2 Requirement Breakdown	52
3.3.3.3 Use Case Description	56
3.3.4 <i>Aircraft Requirements</i>	64
3.3.4.1 Use Case Diagram	64
3.3.4.2 Requirement Breakdown	65
3.3.4.3 Use Case Description	70
3.3.5 <i>Destroyer Requirements</i>	79
3.3.5.1 Use Case Diagram	79
3.3.5.2 Requirement Breakdown	80
3.3.5.3 Use Case Description	84
3.3.6 <i>Cruiser Requirements</i>	91
3.3.6.1 Use Case Diagram	91
3.3.6.2 Requirement Breakdown	92
3.3.6.3 Use Case Description	96
3.3.7 <i>Battleship Requirements</i>	103
3.3.7.1 Use Case Diagram	103
3.3.7.2 Requirement Breakdown	104
3.3.7.3 Use Case Description	108
3.3.8 <i>Submarine Requirements</i>	115
3.3.8.1 Use Case Diagram	115
3.3.8.2 Requirement Breakdown	116
3.3.8.3 Use Case Description	120
3.3.9 <i>Weapons Requirements</i>	127
3.3.9.1 Use Case Diagram	127

3.3.9.2	Requirement Breakdown	128
3.3.9.3	Use Case Description	129
3.4	EXTERNAL INTERFACE REQUIREMENTS	132
3.4.1	<i>User Interface</i>	132
3.4.2	<i>Hardware Interface</i>	132
3.4.3	<i>Software Interface</i>	132
3.4.4	<i>Communication Interface</i>	133
3.5	PERFORMANCE REQUIREMENTS	133
3.6	DESIGN CONSTRAINTS	134
3.7	QUALITY ATTRIBUTES	134
3.8	OTHER REQUIREMENTS	134
4.	SOFTWARE DESIGN	135
4.1	DECOMPOSITION DESCRIPTION	135
4.1.1	<i>Module Decomposition</i>	135
4.1.1.1	Simulation Controller	138
4.1.1.2	Communication/Detection Description	138
4.1.1.3	Aircraft Carrier Description	138
4.1.1.4	Aircraft Description	139
4.1.1.5	Destroyer Description	139
4.1.1.6	Cruiser Description	139
4.1.1.7	Battleship Description	139
4.1.1.8	Submarine Description	140
4.1.1.9	Weapon Description	140
4.1.2	<i>Concurrent Process Decomposition</i>	140
4.1.3	<i>Data Decomposition</i>	140
4.1.3.1	Data entity description	140
4.2	DEPENDENCY DESCRIPTION	141
4.2.1	<i>Internal Module Dependency</i>	141
4.2.1.1	Simulation Controller dependency on BaseShip Subsystem	141
4.2.1.2	Simulation Controller dependency on BaseWeapon subsystem	141
4.2.1.3	Communication/Detection dependency on Simulation Controller	141
4.2.1.4	Communication/Detection dependency on BaseShip	141
4.2.1.5	BaseShip Subsystem dependency on Communication/Detection	141
4.2.1.6	BaseShip (except Aircraft Carrier) dependency on BaseWeapon	142
4.2.1.7	BaseWeapon dependency on BaseShip	142
4.2.1.8	BaseWeapon dependency on Communication/Detection	142
4.2.2	<i>Internal Process Dependency</i>	142
4.2.3	<i>Data Dependency</i>	142
4.3	INTERFACE DESCRIPTION	143
4.3.1	<i>Module Interface</i>	143
4.3.1.1	Simulation Controller	143
4.3.1.2	Communication/Detection	144
4.3.1.3	BaseShip Class	146
4.3.1.4	BaseWeapon	148
4.3.2	<i>Process Interface</i>	150
4.4	SYSTEM DETAILED DESIGN	151
4.4.1	<i>Simulation Controller Detailed Design</i>	151
4.4.1.1	Module Detailed Design	152
4.4.1.2	Class Definition	153
4.4.2	<i>Communication/Detection Detailed Design</i>	165
4.4.2.1	Module Detailed Design	165
4.4.2.2	Class Definition	166
4.4.3	<i>Ship and Aircraft Detailed Design</i>	184
4.4.3.1	Module Detailed Design	184
4.4.3.2	Class Definition	184

4.4.3.3	Description of Class Members and Member Functions.....	184
4.4.4	<i>Weapon Detailed Design</i>	212
4.4.4.1	Module Detailed Design.....	212
4.4.4.2	Class Definition.....	213
5.	SYSTEM TESTING	241
5.1	UNIT TESTING.....	241
5.1.1	<i>Unit Testing for Simulation Controller</i>	242
5.1.1.1	Unit Test Case for SetUpDlg Class Functions.....	242
5.1.1.2	Unit Test Case for Controller Class Functions.....	243
5.1.1.3	Other Unit Test Through User Interaction.....	245
5.1.2	<i>Unit Testing for Communication/Detection</i>	245
5.1.2.1	Unit Test Case for CDetected Class Functions.....	245
5.1.2.2	Unit Test Case for CDetectedDatabase Class Functions.....	246
5.1.2.3	Unit Test Case for CRadar Class Functions.....	247
5.1.2.4	Unit Test Case for CSonar Class Functions.....	248
5.1.2.5	Unit Test Case for CMessage Class Functions.....	248
5.1.2.6	Unit Test Case for CMessageDatabase Class Functions.....	249
5.1.2.7	Unit Test Case for CRadio Class Functions.....	250
5.1.3	<i>Unit Testing for All Vehicles</i>	252
5.1.3.1	Unit Test Case for Derived BaseShip Class Functions.....	252
5.1.3.2	Unit Test Case for Captain Class Functions.....	253
5.1.3.3	Unit Test Case for NavigationOfficer Class Functions.....	254
5.1.3.4	Unit Test Case for RadioOfficer Class Functions.....	256
5.1.3.5	Unit Test Case for Radar/SonarOfficer Class Functions.....	256
5.1.3.6	Unit Test Case for WeaponOfficer Class Functions.....	256
5.1.3.7	Unit Test Case for WeaponLauncher Class Functions.....	258
5.1.4	<i>Unit Testing for Weapons</i>	260
5.1.4.1	Unit Test Case for CWActiveStateController Class Functions.....	260
5.1.4.2	Unit Test Case for CWPositionController Class Functions.....	260
5.1.4.3	Unit Test Case for CWAutoAimController Class Functions.....	261
5.1.4.4	Unit Test Case for CWChargeController Class Functions.....	263
5.1.4.5	Unit Test Case for CWCharge Class Functions.....	264
5.1.4.6	Unit Test Case for CWRudder Class Functions.....	265
5.2	SUBSYSTEM TESTING.....	266
5.2.1	<i>Simulation Controller Subsystem Testing</i>	266
5.2.1.1	Test Cases and Results.....	266
5.2.1.2	Error Reports.....	268
5.2.1.3	Untested Components.....	268
5.2.2	<i>Communication/Detection Subsystem Testing</i>	269
5.2.2.1	Test Cases and Results.....	269
5.2.2.2	Error Reports.....	269
5.2.2.3	Untested Components.....	269
5.2.3	<i>Ship/Aircraft Subsystem Testing</i>	270
5.2.3.1	Test Cases and Results.....	270
5.2.3.2	Error Reports.....	270
5.2.3.3	Untested Components.....	270
5.2.4	<i>Weapon Subsystem Testing</i>	271
5.2.4.1	Test Cases and Results.....	271
5.2.4.2	Error Reports.....	271
5.2.4.3	Untested Components.....	271
5.3	SYSTEM INTEGRATION TESTING.....	272
5.3.1	<i>Integration scheme</i>	272
5.3.2	<i>Test Cases and Results</i>	272
5.3.3	<i>Error Reports</i>	274
6.	CONCLUSION	275

6.1	REQUIREMENTS GATHERING AND SPECIFICATION	275
6.2	DESIGN AND IMPLEMENTING	275
6.3	TESTING AND DELIVERY/MAINTENANCE.....	276
6.4	FURTHER IMPROVEMENTS	276

List of Tables

TABLE 3-1	REQUIREMENT IDENTIFIERS	9
TABLE 3-2	ATTACKER LIST	16
TABLE 3-3	WEAPON LIST.....	16
TABLE 3-4	COMMUNICATION/DETECTION LIST	16
TABLE 3-5	USE CASE SET UP OPERATIONAL PARAMETERS	22
TABLE 3-6	USE CASE DESCRIPTION FOR: START SIMULATION	24
TABLE 3-7	USE CASE DESCRIPTION FOR SIMULATE COMMUNICATION.....	25
TABLE 3-8	USE CASE DESCRIPTION FOR BASE SUPPLY	27
TABLE 3-9	USE CASE DESCRIPTION FOR PAUSE SIMULATION.....	29
TABLE 3-10	USE CASE DESCRIPTION FOR RESUME SIMULATION.....	30
TABLE 3-11	USE CASE DESCRIPTION FOR END SIMULATION	31
TABLE 3-12	USE CASE DESCRIPTION FOR REPORT STATISTICS	32
TABLE 3-13	USE CASE DESCRIPTION FOR TURN ON RADAR.....	37
TABLE 3-14	USE CASE DESCRIPTION FOR TURN OFF RADAR.....	38
TABLE 3-15	USE CASE DESCRIPTION FOR RADAR EMIT WAVE	39
TABLE 3-16	USE CASE DESCRIPTION FOR RADAR RECEIVE WAVE.....	40
TABLE 3-17	USE CASE DESCRIPTION FOR TURN ON SONAR	41
TABLE 3-18	USE CASE DESCRIPTION FOR TURN OFF SONAR	42
TABLE 3-19	USE CASE DESCRIPTION FOR SONAR EMIT WAVE.....	43
TABLE 3-20	USE CASE DESCRIPTION FOR SONAR RECEIVE WAVE	44
TABLE 3-21	USE CASE DESCRIPTION FOR TURN ON RADIO	45
TABLE 3-22	USE CASE DESCRIPTION FOR TURN OFF RADIO.....	46
TABLE 3-23	USE CASE DESCRIPTION FOR RADIO SEND MESSAGE.....	47
TABLE 3-24	USE CASE DESCRIPTION FOR RADIO RECEIVE MESSAGE	49
TABLE 3-25	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER NAVIGATION CONTROL.....	56
TABLE 3-26	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER COMMUNICATE WITH ALLIES	57
TABLE 3-27	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER MAKE DECISION.....	58
TABLE 3-28	USE CASE DESCRIPTION FOR AIRCRAFT CONTROL	60
TABLE 3-29	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER UPDATE STATUS.....	62
TABLE 3-30	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER REFUELING.....	63
TABLE 3-31	USE CASE DESCRIPTION FOR AIRCRAFT NAVIGATION CONTROL.....	70
TABLE 3-32	USE CASE DESCRIPTION FOR AIRCRAFT DETECT ENEMY.....	72
TABLE 3-33	USE CASE DESCRIPTION FOR AIRCRAFT COMMUNICATION WITH ALLIES	73
TABLE 3-34	USE CASE DESCRIPTION FOR AIRCRAFT MAKE DECISION.....	74
TABLE 3-35	USE CASE DESCRIPTION FOR AIRCRAFT WEAPON CONTROL	76
TABLE 3-36	USE CASE DESCRIPTION FOR AIRCRAFT UPDATE STATUS.....	77
TABLE 3-37	USE CASE DESCRIPTION FOR AIRCRAFT REFUELING.....	78
TABLE 3-38	USE CASE DESCRIPTION FOR DESTROYER NAVIGATION CONTROL	84
TABLE 3-39	USE CASE DESCRIPTION FOR DESTROYER NAVIGATION CONTROL	85
TABLE 3-40	USE CASE DESCRIPTION FOR DESTROYER COMMUNICATION WITH ALLIES	86
TABLE 3-41	USE CASE DESCRIPTION FOR DESTROYER MAKE DECISION.....	87
TABLE 3-42	USE CASE DESCRIPTION FOR DESTROYER WEAPON CONTROL	88
TABLE 3-43	USE CASE DESCRIPTION FOR DESTROYER UPDATE STATUS.....	89
TABLE 3-44	USE CASE DESCRIPTION FOR DESTROYER REARM AND REFUELING.....	90
TABLE 3-45	USE CASE DESCRIPTION FOR CRUISER NAVIGATION CONTROL.....	96
TABLE 3-46	USE CASE DESCRIPTION FOR CRUISER NAVIGATION CONTROL.....	97
TABLE 3-47	USE CASE DESCRIPTION FOR AIRCRAFT CARRIER COMMUNICATION WITH ALLIES	98
TABLE 3-48	USE CASE DESCRIPTION FOR CRUISER MAKE DECISION	99
TABLE 3-49	USE CASE DESCRIPTION FOR CRUISER WEAPON CONTROL.....	100
TABLE 3-50	USE CASE DESCRIPTION FOR CRUISER UPDATE STATUS.....	101
TABLE 3-51	USE CASE DESCRIPTION FOR CRUISER REARM AND REFUELING.....	102

TABLE 3-52	USE CASE DESCRIPTION FOR BATTLESHIP NAVIGATION CONTROL.....	108
TABLE 3-53	USE CASE DESCRIPTION FOR BATTLESHIP NAVIGATION CONTROL.....	109
TABLE 3-54	USE CASE DESCRIPTION FOR BATTLESHIP COMMUNICATION WITH ALLIES	110
TABLE 3-55	USE CASE DESCRIPTION FOR BATTLESHIP MAKE DECISION	111
TABLE 3-56	USE CASE FOR WEAPON CONTROL	112
TABLE 3-57	USE CASE DESCRIPTION FOR BATTLESHIP UPDATE STATUS	113
TABLE 3-58	USE CASE DESCRIPTION FOR BATTLESHIP REARM AND REFUELING	114
TABLE 3-59	USE CASE DESCRIPTION FOR SUBMARINE NAVIGATION CONTROL.....	120
TABLE 3-60	USE CASE DESCRIPTION FOR SUBMARINE DETECT ENEMY.....	121
TABLE 3-61	USE CASE DESCRIPTION FOR SUBMARINE COMMUNICATE WITH ALLIES	122
TABLE 3-62	USE CASE DESCRIPTION FOR SUBMARINE MAKE DECISION.....	123
TABLE 3-63	USE CASE DESCRIPTION FOR SUBMARINE WEAPON CONTROL.....	124
TABLE 3-64	USE CASE DESCRIPTION FOR SUBMARINE UPDATE STATUS.....	125
TABLE 3-65	USE CASE DESCRIPTION FOR SUBMARINE REARM AND REFUELING.....	126
TABLE 3-66	USE CASE DESCRIPTION FOR PROVIDE LOCATION	129
TABLE 3-67	USE CASE DESCRIPTION FOR AIM TARGET	130
TABLE 3-68	USE CASE DESCRIPTION FOR FIRE AND HIT TARGET.....	131
TABLE 5-1	UNIT STATIC TESTING	241
TABLE 5-2	UNIT DYNAMIC TESTING	242
TABLE 5-3	UNIT TEST CASE FOR SETUPDLG DRAW FUNCTION	243
TABLE 5-4	UNIT TEST CASE FOR SETUPDLG UNDO FUNCTION	243
TABLE 5-5	UNIT TEST CASE FOR CONTROLLER LOADTGA FUNCTION.....	244
TABLE 5-6	UNIT TEST CASE FOR CONTROLLER CALDIR FUNCTION	244
TABLE 5-7	UNIT TEST CASE FOR CONTROLLER ONKEYDOWN FUNCTION.....	244
TABLE 5-8	OTHER UNIT TEST THROUGH USER INTERACTION	245
TABLE 5-9	UNIT TEST CASE FOR CDETCCTED SETDETDATA FUNCTION.....	246
TABLE 5-10	UNIT TEST CASE FOR CDETCCTED OPERATOR <<OVERLOADING FUNCTION.....	246
TABLE 5-11	UNIT TEST CASE FOR CDETCCTEDDATABASE DELETEALL FUNCTION	246
TABLE 5-12	UNIT TEST CASE FOR CDETCCTEDDATABASE ADDDELETED FUNCTION	247
TABLE 5-13	UNIT TEST CASE FOR CRADAR EMITRECEIVE FUNCTION	247
TABLE 5-14	UNIT TEST CASE FOR CSONAR EMITRECEIVE FUNCTION.....	248
TABLE 5-15	UNIT TEST CASE FOR CMESSAGE VALIDTOSEND FUNCTION	248
TABLE 5-16	UNIT TEST CASE FOR CMESSAGE VALIDTOSEND FUNCTION	249
TABLE 5-17	UNIT TEST CASE FOR CMESSAGE VALIDTOSEND FUNCTION	249
TABLE 5-18	UNIT TEST CASE FOR CMESSAGE DELETEALLMSG FUNCTION.....	249
TABLE 5-19	UNIT TEST CASE FOR CMESSAGE ADDONEMSGINTHELIST FUNCTION.....	250
TABLE 5-20	UNIT TEST CASE FOR CMESSAGE GETMYMSG FUNCTION.....	250
TABLE 5-21	UNIT TEST CASE FOR CRADIO DELETEMESSAGES FUNCTION	250
TABLE 5-22	UNIT TEST CASE FOR CRADIO SENDMESSAGE FUNCTION	250
TABLE 5-23	UNIT TEST CASE FOR CRADIO RECEIVEMESSAGES FUNCTION	251
TABLE 5-24	UNIT TEST CASE FOR DERIVED BASESHIP CONSTRUCTOR FUNCTION.....	252
TABLE 5-25	UNIT TEST CASE FOR DERIVED BASESHIP UPDATESTATUS AND RESISTANCERECOVERY FUNCTION	253
TABLE 5-26	UNIT TEST CASE FOR DERIVED CAPTAIN IFATTACK FUNCTION	253
TABLE 5-27	UNIT TEST CASE FOR DERIVED CAPTAIN : ISONTHEWAY, ADJUSTNAVIGATION FUNCTION	253
TABLE 5-28	UNIT TEST CASE FOR NAVIGATIONOFFICER ADJUSTSPEED FUNCTION.....	254
TABLE 5-29	UNIT TEST CASE FOR NAVIGATIONOFFICER OTHER FUNCTION	255
TABLE 5-30	UNIT TEST CASE FOR WEAPONOFFICER PREPAREATTACK FUNCTION.....	257
TABLE 5-31	UNIT TEST CASE FOR WEAPONOFFICER SELECTWEAPON FUNCTION	257
TABLE 5-32	UNIT TEST CASE FOR WEAPONLAUNCHER AIMBYBALLISTIC FUNCTION.....	258
TABLE 5-33	UNIT TEST CASE FOR WEAPONLAUNCHER FIRECANNONHELL FUNCTION.....	258
TABLE 5-34	UNIT TEST CASE FOR WEAPONLAUNCHER FIREMISSILE FUNCTION.....	258
TABLE 5-35	UNIT TEST CASE FOR WEAPONLAUNCHER DELETEWEAPON FUNCTION.....	259

TABLE 5-36	UNIT TEST CASE FOR CWACTIVESTATECONTROLLER GET/SETSTATE FUNCTION	260
TABLE 5-37	UNIT TEST CASE FOR CWACTIVESTATECONTROLLER INITIALPOSITION FUNCTION	260
TABLE 5-38	UNIT TEST CASE FOR CWACTIVESTATECONTROLLER INITIALPOSITION FUNCTION	261
TABLE 5-39	UNIT TEST CASE FOR CWAUTOAIMCONTROLLER TRACETARGET FUNCTION	261
TABLE 5-40	UNIT TEST CASE FOR CWAUTOAIMCONTROLLER TRACETARGET (AIRCRAFT) FUNCTION	262
TABLE 5-41	UNIT TEST CASE FOR CWAUTOAIMCONTROLLER TRACETARGET (SUBMARINE) FUNCTION	262
TABLE 5-42	UNIT TEST CASE FOR CWCHARGECONTROLLER HITDETECT FUNCTION	263
TABLE 5-43	UNIT TEST CASE FOR CWCHARGECONTROLLER HITDETECT(AIRCRAFT) FUNCTION	263
TABLE 5-44	UNIT TEST CASE FOR CWCHARGECONTROLLER HITDETECT(SUBMARINE) FUNCTION	264
TABLE 5-45	UNIT TEST CASE CWCHARGE DETONATETARGET FUNCTION	264
TABLE 5-46	UNIT TEST CASE CWRUDDER CHANGEVELOCITY FUNCTION	265
TABLE 5-47	TEST CASE FOR SIMULAITON CONTROLLER(SETUPDLG) SUBSYSTEM	266
TABLE 5-48	TEST CASE FOR SIMULAITON CONTROLLER(VECTOR) SUBSYSTEM	266
TABLE 5-49	TEST CASE FOR SIMULAITON CONTROLLER (SC) SUBSYSTEM.....	267
TABLE 5-50	TEST CASE FOR SIMULAITON CONTROLLER (VEHICLEFACTORY) SUBSYSTEM	267
TABLE 5-51	TEST CASE FOR SIMULAITON CONTROLLER (CONTROLLER) SUBSYSTEM	268
TABLE 5-52	TEST CASE FOR COMMUNICATION/DETECTION SUBSYSTEM.....	269
TABLE 5-53	TEST CASE FOR SHIP/AIRCRAFT SUBSYSTEM.....	270
TABLE 5-54	TEST CASE FOR WEAPON(WTORPEDO) SUBSYSTEM	271
TABLE 5-55	TEST CASE FOR WEAPON (WCANNONSHELL) SUBSYSTEM.....	271
TABLE 5-56	TEST CASES AND RESULTS	273

List of Figures

FIGURE 3-1	SEQUENCE DIAGRAM FOR USE CASE NAVIGATION CONTROL.....	10
FIGURE 3-2	SEQUENCE DIAGRAM FOR USE CASE DETECT ENEMY	11
FIGURE 3-3	SEQUENCE DIAGRAM FOR USE CASE COMMUNICATE WITH ALLIES.....	11
FIGURE 3-4	SEQUENCE DIAGRAM FOR USE CASE MAKE DECISION	12
FIGURE 3-5	SEQUENCE DIAGRAM FOR USE CASE WEAPON CONTROL	12
FIGURE 3-6	SEQUENCE DIAGRAM FOR USE CASE UPDATE STATUS	13
FIGURE 3-7	SEQUENCE DIAGRAM FOR USE CASE REARM AND REFUELING	13
FIGURE 3-8	SEQUENCE DIAGRAM FOR USE CASE TURN ON COMMUNICATION/DETECTION.....	14
FIGURE 3-9	SEQUENCE DIAGRAM FOR USE CASE TURN OFF COMMUNICATION/DETECTION	14
FIGURE 3-10	SEQUENCE DIAGRAM FOR USE CASE DETECTION EMIT WAVE	15
FIGURE 3-11	SEQUENCE DIAGRAM FOR USE CASE DETECTION RECEIVE WAVE.....	15
FIGURE 3-12	USE CASE DIAGRAM FOR SIMULATION CONTROLLER	17
FIGURE 3-13	SEQUENCE DIAGRAM FOR USE CASE: SET UP OPERATIONAL PARAMETERS.....	23
FIGURE 3-14	SEQUENCE DIAGRAM FOR USE CASE: START SIMULATION	24
FIGURE 3-15	SEQUENCE DIAGRAM FOR USE CASE: SIMULATE COMMUNICATION	26
FIGURE 3-16	SEQUENCE DIAGRAM FOR USE CASE: BASE SUPPLIER	28
FIGURE 3-17	SEQUENCE DIAGRAM FOR USE CASE: PAUSE SIMULATION	29
FIGURE 3-18	SEQUENCE DIAGRAM FOR USE CASE: RESUME SIMULATION	30
FIGURE 3-19	SEQUENCE DIAGRAM FOR USE CASE: END SIMULATION	31
FIGURE 3-20	SEQUENCE DIAGRAM FOR USE CASE: REPORT STATISTICS	32
FIGURE 3-21	USE CASE DIAGRAM FOR COMMUNICATION/DETECTION.....	33
FIGURE 3-22	SEQUENCE DIAGRAM FOR USE CASE RADIO SEND MESSAGE	48
FIGURE 3-23	SEQUENCE DIAGRAM FOR USE CASE RADIO RECEIVE MESSAGE	50
FIGURE 3-24	USE CASE DIAGRAM FOR AIRCRAFT CARRIER	51
FIGURE 3-25	SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT CARRIER MAKE DECISION	59
FIGURE 3-26	SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT CARRIER AIRCRAFT CONTROL.....	61
FIGURE 3-27	USE CASE DIAGRAM FOR AIRCRAFT	64
FIGURE 3-28	SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT NAVIGATION CONTROL.....	71
FIGURE 3-29	SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT MAKE DECISION	75
FIGURE 3-30	USE CASE DIAGRAM FOR DESTROYER	79
FIGURE 3-31	USE CASE DIAGRAM FOR CRUISER.....	91
FIGURE 3-32	USE CASE DIAGRAM FOR BATTLESHIP	103
FIGURE 3-33	USE CASE DIAGRAM FOR SUBMARINE	115
FIGURE 3-34	USE CASE DIAGRAM FOR WEAPON	127
FIGURE 3-35	SEQUENCE DIAGRAM FOR USE CASE WEAPON PROVIDE LOCATION	129
FIGURE 3-36	SEQUENCE DIAGRAM FOR USE CASE WEAPON AIM TARGET	130
FIGURE 3-37	SEQUENCE DIAGRAM FOR USE CASE WEAPON FIRE AND HIT TARGET.....	131
FIGURE 4-1	INTERACTION DIAGRAM BETWEEN SUBSYSTEMS	135
FIGURE 4-2	ARCHITECTURE OF THE NAVAL BATTLE SIMULATION SYSTEM	136
FIGURE 4-3	CLASS LEVEL INTERFACE DIAGRAM.....	137
FIGURE 4-4	SIMULATION CONTROLLER FOR WEAPON	143
FIGURE 4-5	SIMULATION CONTROLLER FOR COMMUNICATION/DETECTION	144
FIGURE 4-6	RADAR/SONAR_FOR_WEAPON	145
FIGURE 4-7	BASESHIP_FOR_SC.....	147
FIGURE 4-8	BASEWEAPON_FOR_SIMULATION CONTROLLER	148
FIGURE 4-9	BASEWEAPON_FOR_SHIP AND AIRCRAFT.....	150
FIGURE 4-10	CLASS DIAGRAM FOR SIMULATION CONTROLLER MODULE.....	152
FIGURE 4-11	CLASS DIAGRAM FOR COMMUNICATION/DETECTION MODULE	165
FIGURE 4-12	CLASS DIAGRAM FOR BASESHIP (SHIP AND AIRCRAFT) MODULE.....	185
FIGURE 4-13	CLASS DIAGRAM FOR WEAPON MODULE.....	212

1. Introduction

The Naval Battle Simulation System is a software system to simulate real life but yet simplified modern naval battle scenarios. This document follows the IEEE standards [2], [3] and Dr. Paquet SRD slides [4] to specify the system requirements and describe the system design. The whole document is based on the Software Engineering (COMP554, Summer 2001) project of the Computer Science Department in Concordia University. Care was taken to write this document in an organized and comprehensive structure, and also to fully list the system requirement and optimize the original system design. This document's objective is to practice the object oriented design methodology and to comply with the IEEE documentation standards for software.

1.1 Purpose

The purpose of this document is as following:

- Present in a precise and understandable manner the requirements, design, and testing procedure of the Naval Battle Simulation System.
- Demonstrate software documentation traceability among SRS, SDD and Software Testing Document.
- Show how the design is a translation of requirements into software structure, software components, interface, and data necessary for the implementation phase; show how testing is linked to requirements.
- The document is intended to be a baseline to supply sufficient design and implementation information for the future students in other Software Engineering courses offered in the Department.
- The system and documentation are to be designed in terms of extensibility and reusability as much as possible.

1.2 Scope

The software system that will be developed is called NBSS—Naval Battle Simulation System. This system simulates the activities and functions of many real life parties involved in (hypothetic) naval battles. The subsystem includes Simulation Controller, Aircraft, Aircraft Carrier, Battleship, Cruiser, Destroyer, Submarine, Weapon and Communication/Detection. The simulated behavior includes navigating, detecting enemies with Radars and Sonars, communicating and cooperating with allies, attacking enemies, and base supplier. The system allows the user to set the simulation parameters and interact with the system too. The deliverable products are the following:

Software System

A software package that fulfills the system requirements listed in section 3. It is implemented to comply with software design in section 4. It also meets the test goals listed in the testing document presented in section 5.

Software Document

A complete and understandable document that describes the whole system in terms of requirement specification, software design, implementation, and testing. It will also be an aid reference for future maintenance and updating.

1.3 Definitions, Acronyms, Abbreviations

Acronym	Definition
ANSI	American National Standards Institute
Class Diagram	Used to display some of the classes and packages of classes in the system
Design Entity	An element (component) of a design that is structurally and functionally distinct from other elements
IEEE	The institute of Electrical and Electronics Engineers
IMD	Internal Module Design
MFC	Microsoft Foundation Class Library
MID	Module Interface Design
NA	Not Applicable
NBSS	Naval Battle Simulation System
Open GL	Open Graphics Library
SC	Simulation Controller
Sequence Diagram	Used to graphically show the flow of event in a use case (Functional requirements specifications)
SRS	Software Requirement Specification Document
SRD	Software Requirements Document
Use Case Diagram	Used to describe the functionality of a system, or one of its components
UML	Unified Modeling Language
Vehicle	Aircraft Carrier, Aircraft, Destroyer, Cruiser, Battleship, Submarine, Weapons
Weapon	Sea-Sub Missile/Torpedo, Sea-Air Missile, Heavy Cannon Shell, Sea-Sea Missile, Torpedo, Sub-Sea Torpedo/Missile, Air-Sea Missile, Air-Air Missile

1.4 Overview

This document is organized in six major sections. *Section 1 Introduction* introduces the main purpose, scope, overview, and references of the whole document. References are presented there to comply with the IEEE standards for software documentation. *Section 2 General Description* describes the system from different aspects: product perspective, product functions, user characteristics, general constraints and assumptions and dependencies. *Section 3 Specific Requirements* defines the specific requirements and all detailed need to build the system design for all the subsystems. *Section 4 Software Design* describes the system in terms of decomposition description, dependency description, interface description, scenario for major functionality and detailed design. *Section 5 Testing* describes the unit test cases and integrated testing plan.

1.5 References

- [1] Peter Freeman, Anthony I. Wasserman, ***Tutorial on Software Design Techniques. 4th Edition***, IEEE Computer Society Press, 1983.
- [2] ***Institute of Eletrical and Electonics Engineers Inc., An American National Standard IEEE Guide to Software Requirements Specification***, Software Engineering Standars Committee of the IEEE Computer Society, 1984.
- [3] ***Institute of Eletrical and Electonics Engineers, IEEE Recommended Practice for Software Design Descriptions***, Software Engineering Standars Committee of the IEEE Computer Society, September 1998.
- [4] Joey Paquet, ***SRD Document Standard & Guidelines Slides***, course material, Concordia University, Department of Computer Science, 2000.
- [5] Martin Fowler with Kendail Scott, ***UML Distilled Second Edition(A Brief Guide to the Standard Object Modeling Language)***, ADDISON-WESLEY, 1999.
- [6] **www.naval-technology.com**, the Website for defence industries – Navy, 2001.
- [7] Terry Quatrani , ***Visual Modeling With Rational Rose and UML***, ADDISON-WESLEY, 1999.
- [8] **www.rational.com/uml/index.jsp**, Rational Software Corporation, 2001.
- [9] **www.fas.org/man/dod-101/sys/**, the Federation of American Scientists
- [10] James Rumbaugh, Michel Balha, Premerlani, Eddy, Lorensen, ***Object Oriented Modeling and Design***, Prentice Hall, 1991.

2. General Description

2.1 Product Perspective

The Naval Battle Simulation System is divided into several subsystems. Each of these subsystems can further be divided into functional tasks.

The identified subsystems are:

- Simulation Controller: provides a user interface and controls the performance of the whole system. It also acts as the communication medium.
- Communication/Detection: is responsible for detecting enemies and communicating with allies, also simulates aiming system for Weapons.
- Aircraft Carrier: cooperates with Aircraft to locate and destroy enemy ships and Aircraft.
- Aircraft: cooperates with Aircraft Carrier to locate and destroy enemy ships and Aircraft.
- Destroyer: detects and destroys the underwater threats.
- Cruiser: detects and destroys the airborne threats.
- Battleship: detects and destroys the sea borne threats.
- Submarine: detects and destroys sea borne and underwater threats.
- Weapons: provides different kinds of Weapons that can be used by all ships (except Aircraft Carrier) and Aircraft to attack enemies.

2.2 Product Functions

Simulation Controller:

1. Provides an interactive user interface
2. Simulates the communication media
3. Generates the vehicles for both sides
4. Animates the movements of ships
5. Generates the fuel and Weapon upon request

Ships and Aircraft (Battleship, Cruiser, Destroyer, Submarine, Aircraft)

1. Navigates on the map
2. Detects the enemy
3. Communicates with allies
4. Launches Weapon to attack targets
5. Makes strategic decisions

Aircraft Carrier

1. Navigates on the map
2. Manages Aircraft take-offs
3. Manages Aircraft landings
4. Assigns missions to Aircrafts
5. Communicates with allies
6. Makes strategic decisions

Communication/Detection:

1. Passes information to the Simulation Controller
2. Detects vehicles
3. Enables communication between vehicles
4. Simulates the detecting system for Weapons

Weapon:

1. Aims at a target
2. Fires at a target
3. Hit sa target
4. Inflicts damage to a vehicle

For the product function definitions, refer to [6] and [9].

2.3 User Characteristics

Users of NBSS can be various: some users are Software Engineering students who need to access the system for maintenance and updating; some users are the end users who will play with the system as a game, and they may not have any background knowledge with computers. For the former, this document will act as a reference manual. For the latter, the system will provide the necessary help to them.

2.4 General Constraints

- The user interface of the vehicle subsystems is provided by the Simulation Controller subsystem. The user has limited access rights for vehicle subsystems.
- The vehicle subsystems have to interact with the Simulation Controller, Weapons, and Communication/Detection subsystems to perform its functions.
- The language used for the implementation of the system is C++.
- The platform of the system is Microsoft Windows 95/98/NT/2000.

2.5 Assumptions and Dependencies

Since the NBSS is composed of nine subsystems, the cooperation and coordination of all the subsystems is a key factor to ensure the success of the whole system. We assume that all subsystems will meet its own requirements and comply with the interface of the other subsystems.

Other assumptions and dependencies:

- The development requires the Microsoft Windows NT 4.0 operating system.
- There will be only two sides, enemy and friend, participating in the battle.
- The simulation will proceed fully automatically, the user can interact the simulation in very limited ways.
- There is no consideration of natural interferences in the simulation, e.g. weather, wind, lighting.

3. Specific Requirements

3.1 Requirement Identification

Each requirement is represented by a requirement identifier, and a requirement name. It is described by a requirement statement and a requirement support comment. They are defined as:

Requirement Identifier

Requirements are distinguished from explanatory text via the requirement identifier. Requirement identifiers are made up of two alphabetic characters, which identify the subsystem the requirement belongs to, followed by a hyphen, and followed by a three digit number, which distinguishes it among requirements within that subsystem.

Subsystem	Prefix
Simulation Controller	SC
Communication/Detection	CD
Aircraft Carrier	AC
Aircraft	AT
Destroyer	DT
Cruiser	CS
Battleship	BS
Submarine	SM
Weapons	WP

Table 3-1 Requirement Identifiers

Requirement numbers are assigned sequentially. Sub requirements will be identified by requirement number and a hyphen that is followed by another two digit number (e.g. SC-001-01).

Requirement Name

The requirement name provides a short title description. Note that many requirements are similar across subsystems (e.g. all vehicles have to implement navigation). In these cases, the requirement names are worded so as to refer to the specific subsystem it describes.

Requirement Statement

The requirement statement is identified by being below the requirement name, in normal font. The requirement statement provides a full but high-level description of the requirement.

Requirement Support Comments

The requirement supporting comments are identified by being below the requirement statements, in an *italic* and somewhat smaller font. The requirement supporting comment provide further explanation and/or supporting discussion of the requirement.

3.2 High Level Use Case Description

For use case diagram and sequence diagram notation refer to reference [5] and [7].

Navigation Control

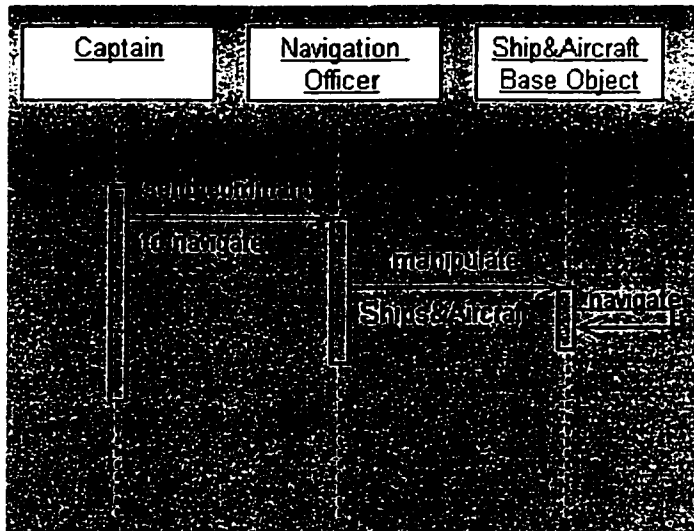


Figure 3-1 Sequence Diagram for Use Case Navigation Control

Detect Enemy

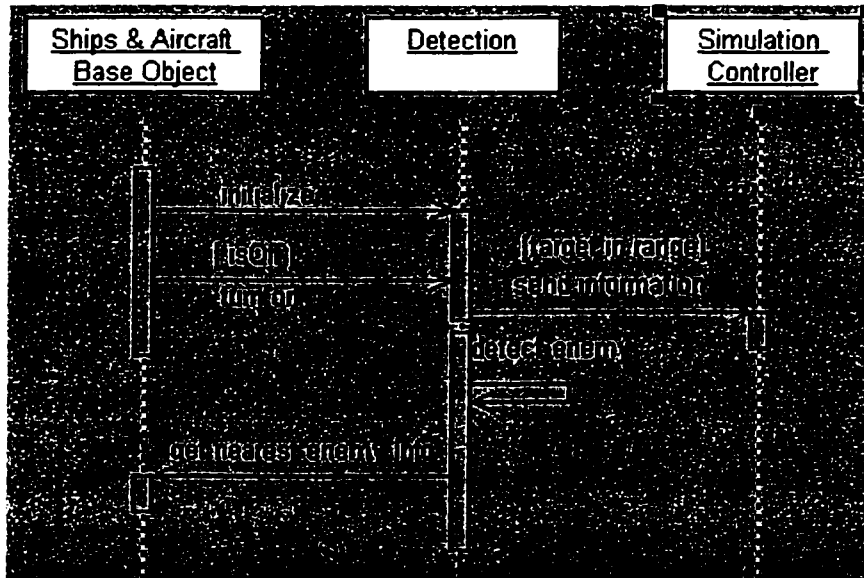


Figure 3-2 Sequence Diagram for Use Case Detect Enemy

Communicate with Allies

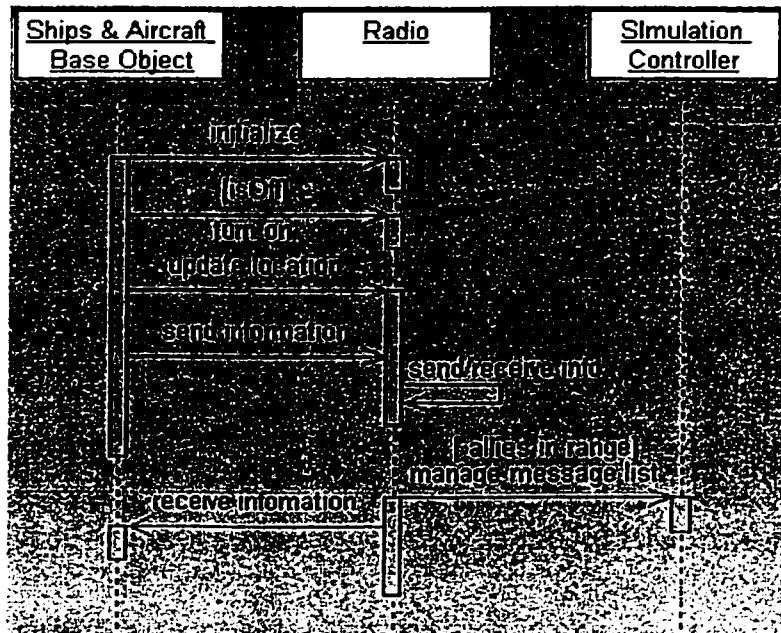


Figure 3-3 Sequence Diagram for Use Case Communicate with Allies

Make Decision

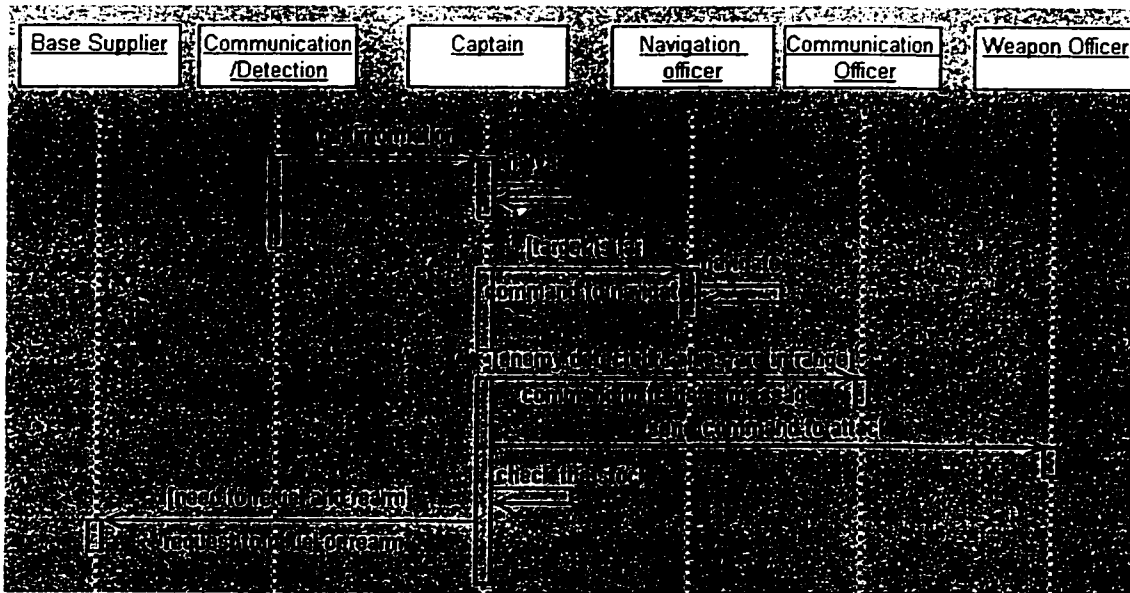


Figure 3-4 Sequence Diagram for Use Case Make Decision

Weapon Control

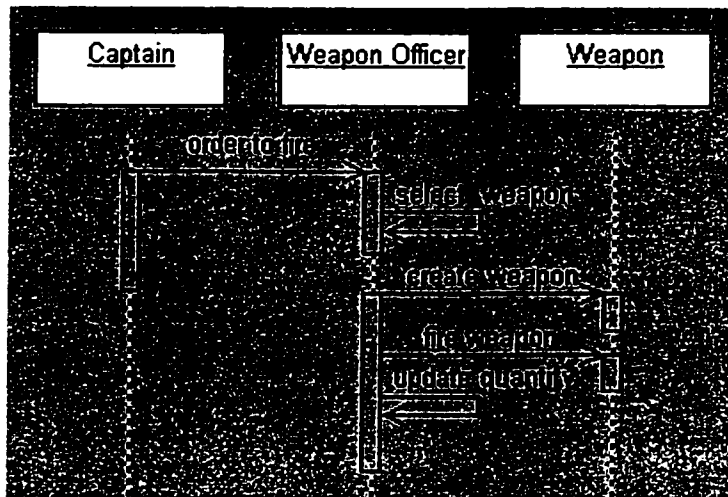


Figure 3-5 Sequence Diagram for Use Case Weapon Control

Update Status

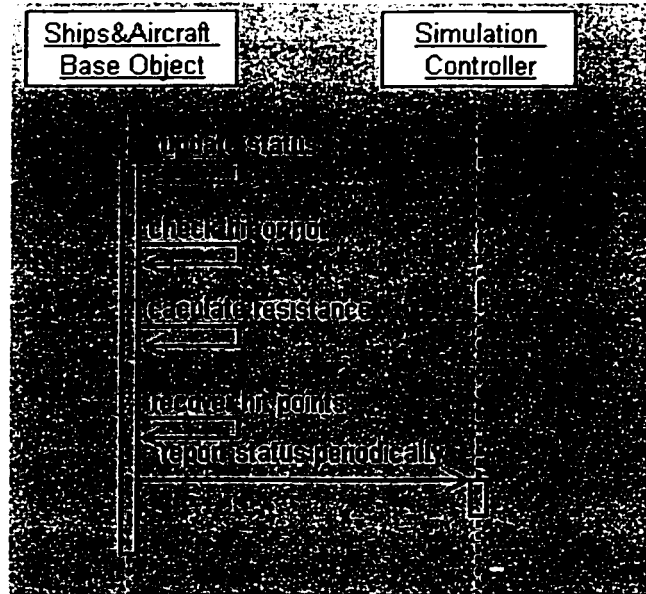


Figure 3-6 Sequence Diagram for Use Case Update Status

Rearming and Refueling

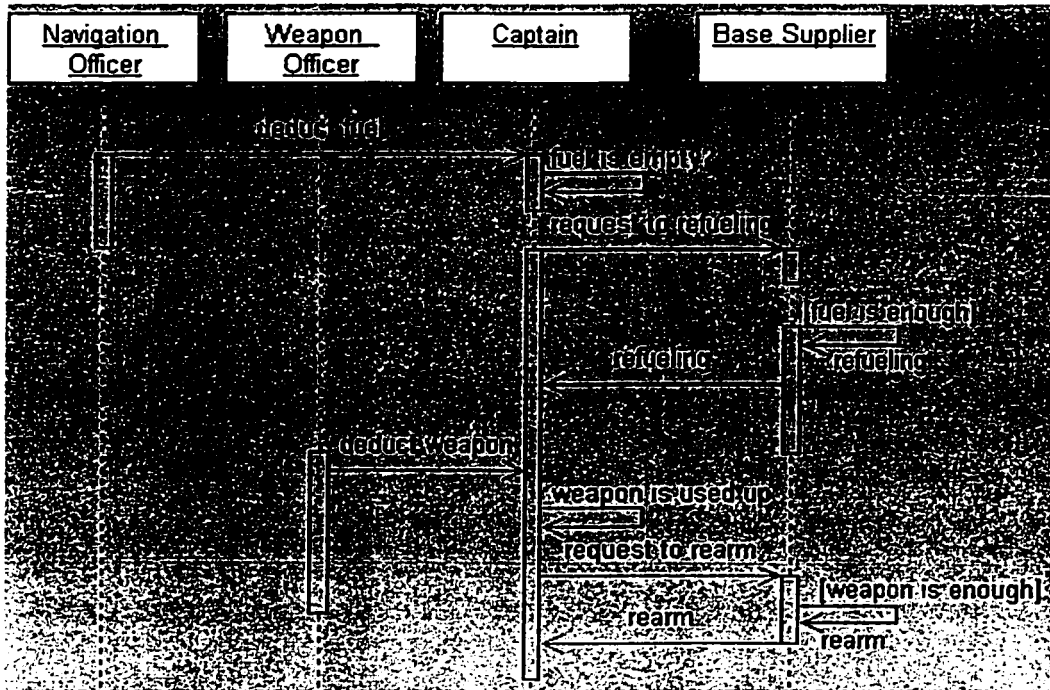


Figure 3-7 Sequence Diagram for Use Case Rearm and Refueling

Turn on Communication/Detection

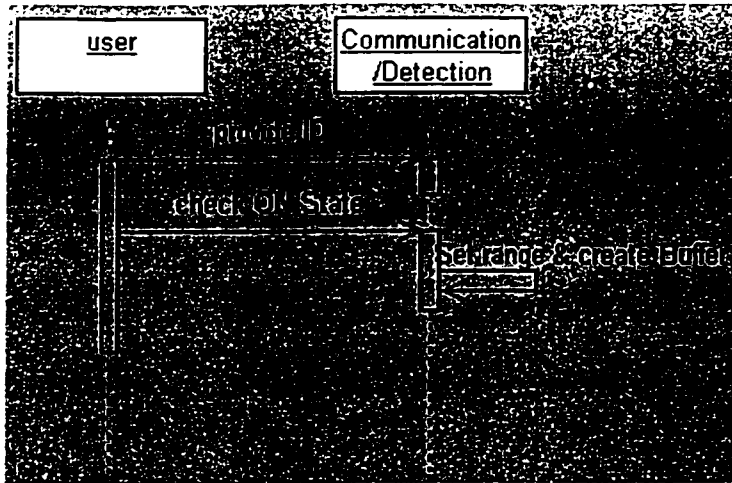


Figure 3-8 Sequence Diagram for Use Case Turn on Communication/Detection

Turn off Communication/Detection Device

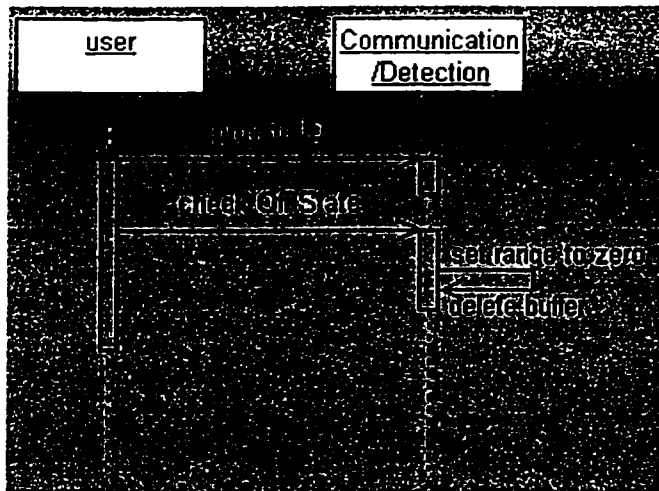


Figure 3-9 Sequence Diagram for Use Case Turn off Communication/Detection

Detection Emit Wave

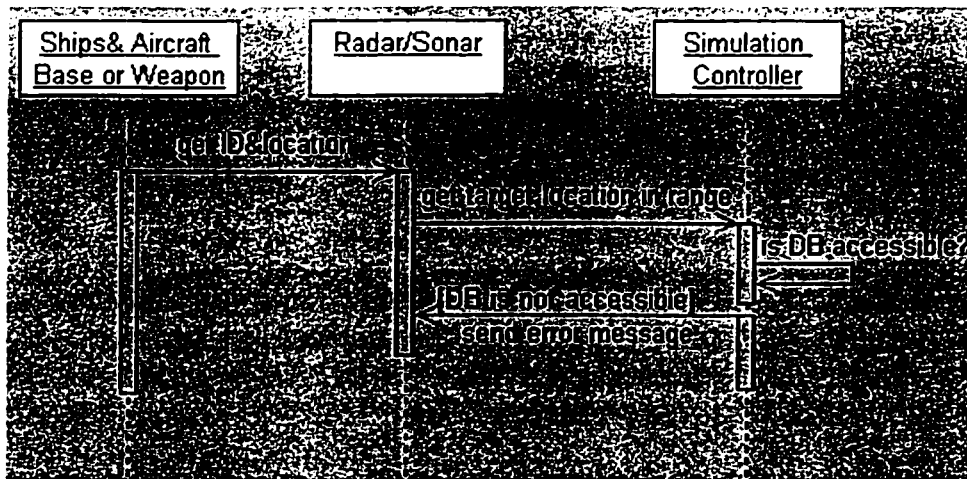


Figure 3-10 Sequence Diagram for Use Case Detection Emit Wave

Detection Receive Wave

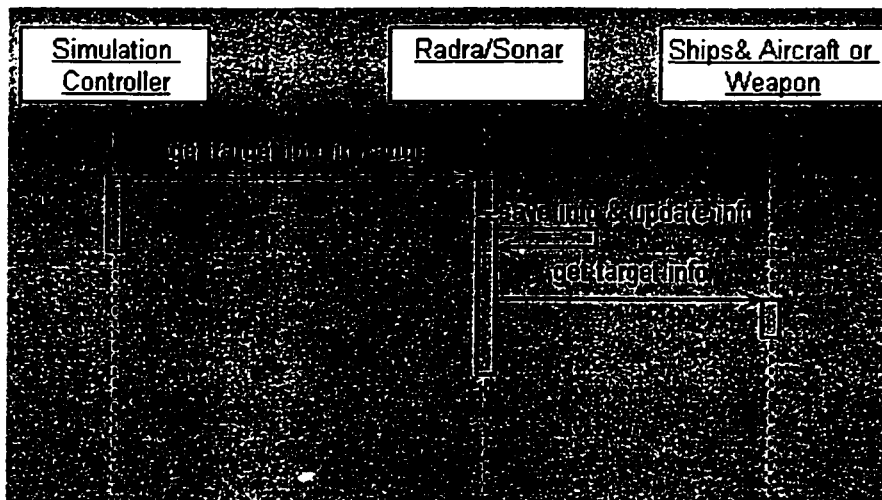


Figure 3-11 Sequence Diagram for Use Case Detection Receive Wave

3.3 Functional Requirements Description

The system requirement descriptions are based on the subsystem classification. Each subsystem is described from the aspects of use case diagram, requirements breakdown and use case description. Use case description refers to the standard [4].

Attacker list

Attacker	Possible target
Aircraft Carrier	No attack ability
Aircraft	Aircraft Carrier, Battleship, Cruiser, Destroyer, Aircraft
Battleship	Aircraft, Aircraft Carrier, Cruiser, Destroyer, Battleship
Cruiser	Aircraft
Destroyer	Submarine
Submarine	Battleship, Cruiser, Destroyer, Submarine

Table 3-2 Attacker List

Weapon list

Attacker	Possible Weapons
Aircraft Carrier	No attack ability.
Aircraft	Air-Air Missile, Air-Sea Missile
Battleship	Sea-Sea Missile, Sea-Air Missile, Heavy Cannon Shell, Torpedo
Cruiser	Sea-Air Missile
Destroyer	Sea-Sub Missile
Submarine	Sub-Sea Torpedo, Torpedo

Table 3-3 Weapon List

Communication/Detection list

Communication/Detection	Ship/Aircraft and Weapon
Radio	Aircraft, Aircraft Carrier, Destroyer, Cruiser, Submarine, Battleship
Radar	Battleship, Cruiser, Aircraft, Sea-Sea Missile, Air-Air Missile, Sea-Air Missile
Soanr	Submarine, Destroyer, Torpedo,

Table 3-4 Communication/Detection List

3.3.1 Simulation Controller Requirements

The Simulation Controller subsystem has the following seven sub modules:

- CMainframe
- SetUpDialog
- Controller
- Base Supplier
- Vehicle Info
- Position Vector
- Simulation Control

3.3.1.1 Use Case Diagram

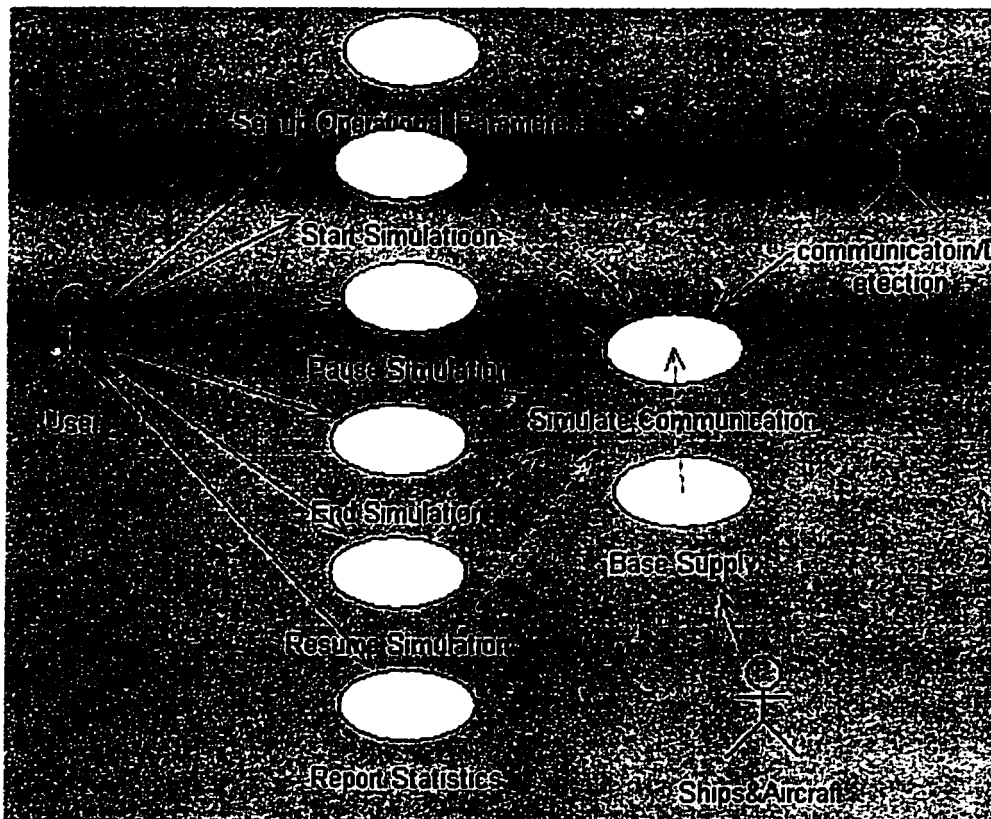


Figure 3-12 Use Case Diagram for Simulation Controller

3.3.1.2 Requirement Breakdown

Use Case: Set Up Operational Parameters

SC-001 Initialize Agents

The Simulation Controller shall create the agents for both friend and enemy sides.

The agents include Aircraft Carrier, Battleship, Cruiser, Destroyer, and Submarine.

SC-002 Add Agents

The Simulation Controller shall allow the user to add new agents to NBSS.

The new agents will be added from an agent list by name.

SC-003 Initialization Weapon

The Simulation Controller shall allow the user to set the used Weapons.

The used Weapons will be selected from a Weapon list by name.

SC-004 Set the Production Rate

The Simulation Controller shall allow the user to set the production rate for producing all kinds of agents, producing fuel and creating Weapons.

These rates will be used when simulation is running by both sides.

SC-005 Set the Limit for Supplying Base

The Simulation Controller shall allow the user to set the maximum stock for supplying all kinds of agents, fuel and Weapons.

No comments.

SC-006 Provide Set up User Interface

The UI shall provide the user to initialize and set the parameters to start the simulation.

GUI of Simulation system shall allow the user to perform SC-001 ~ SC-005.

Use Case: Start Simulation

SC-007 Display Environment

The UI shall display the air, water surface, and underwater environment.

No comments

SC-008 Act as Medium for Communication System

SC-008-01 Act as Water Medium

The Simulation Controller shall act as water medium to transfer the sound waves used by the Sonar.

No comments.

SC-008-02 Act as Air Medium

The Simulation Controller shall act as air medium to transfer the electromagnetic waves used by the Radar and Radio.

No comments.

SC-009 Animate Agents Movement on Screen

The UI shall display and animate the movement of the agents.

No comments

SC-010 Animate Attack and Communication

The UI shall animate the scenario when agents shot Weapon and agents communicate with each other.

No comments

SC-011 Global Time Clock

When the simulation is starting, one global time clock shall be created to provide a time scale for agents to update their status (position, alive/dead, etc.)

No comments

SC-012 Provide Start up User Interface

The UI shall allow the user to start the simulation.

No comments

Use Case: Simulate Communication

SC-013 Provide Agent Information to Communication System

The Simulation Controller shall provide agent's information to the Communication subsystems within the range of Radar and Sonar.

No comments.

SC-013-01 Provide Agent Location

The Simulation Controller shall provide agent's location to the Communication subsystem.

No comments.

- SC-013-02 Provide Agent Status**
 The Simulation Controller shall provide agent's status (alive/dead) to the Communication subsystem.
No comments.
- SC-013-03 Provide Agent Representative**
 The Simulation Controller shall provide an agent's representative (friend/enemy) and identification to the Communication subsystem
No comments.
- SC-014 Control Status of Communication/Detection system**
 The UI shall allow the user to turn on/off the status of Radar, Sonar and Radio for all the objects when the simulation is running.
No comments

Use Case: Base Supply

- SC-015 Provide Regenerate Function**
- SC-015-01 Produce Ships**
 The base supplier shall generate all kinds of ships based on the initialization setting for both sides and depending on the production rate.
No comments.
- SC-015-02 Produce Fuel**
 The base supplier shall produce the specific amount of fuel and depending on production rate.
No comments.
- SC-015-03 Create Weapon**
 The base supplier shall create all kinds of Weapons based on the initialization settings.
No comments
- SC-015-04 Transfer Fuel and Weapon**
 The base supplier shall transfer the fuel and Weapons to agents upon request from agents.
No comments
- SC-015-05 Update Stock**
 The base supplier shall update its stock for ships; also updates stock for fuel and Weapons and respond to agents' queries.
No comments.

Use Case: Pause Simulation

SC-016 Provide Pause Function

The UI shall allow the user to pause the simulation when the simulation is running.

No comments

Use Case: Resume Simulation

SC-017 Provide Resume Function

The UI shall allow the user to resume the simulation when the simulation is paused.

No comments

Use Case: End Simulation

SC-018 Provide Exit Function

The UI shall allow the user to stop the simulation when the simulation is running or paused.

No comments

Use Case: Report Statistics

SC-019 Provide Report Function

The UI shall allow the user to view the log file after the simulation has been started.

No comments

3.3.1.3 Use Case Description

3.3.1.3.1 Use Case: Set up Operational Parameters

Description		Provide the service to allow the user to initialize all the objects
Priority		Must have this use case in order to start the simulation
Status		Detailed description and completed scenario
Actor		NBSS User
Pre-Conditions		Simulation is not in running state or in pause state.
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. The user presses "Setup" button, the system displays a setup dialog window; 2. The user either can press the "Add" button, the vehicle configuration window is displayed and ask user to add a new vehicle, or can select Weapon and input the parameters, then click "OK", the dialog window is closed.
	Alternate Path	If the configuration exceeds the limitation or dissatisfies required conditions, the warning message window will pop up.
Post-Condition		<ol style="list-style-type: none"> 1. The valid input data are saved; 2. Set up window is closed.
Related Use Case	Used Use Case	Simulate Communication
Case	Extending Use Case	NA
Other Requirements		NA

Table 3-5 Use Case Set up Operational Parameters

Sequence Diagram

See next page.

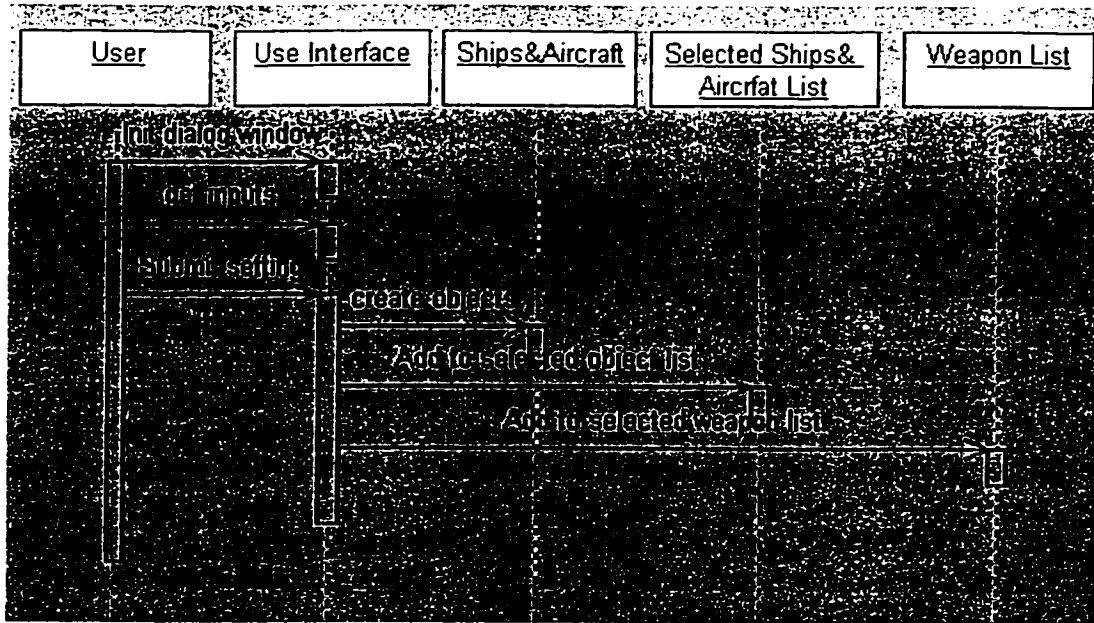


Figure 3-13 Sequence Diagram for Use Case: Set up Operational Parameters

3.3.1.3.2 Use Case: Start Simulation

Description:		Provide the service to start the simulation
Priority:		Must have this use case
Status:		Detailed description and completed scenario
Actor:		NBSS User
Pre-Conditions:		The user has set up the parameters
Flow of Events:	Base Path:	1. The user presses the "Start" button. 2. The system initializes the map, media, creates agents 3. Simulation begins.
	Alternate Path:	NA
Post-Condition:		Simulation successfully started
Related Use Case:	Used Use Case:	Simulate Communication
	Extending Use Case:	NA
Other Requirements:		NA

Table 3-6 Use Case Description for: Start Simulation

Sequence Diagram

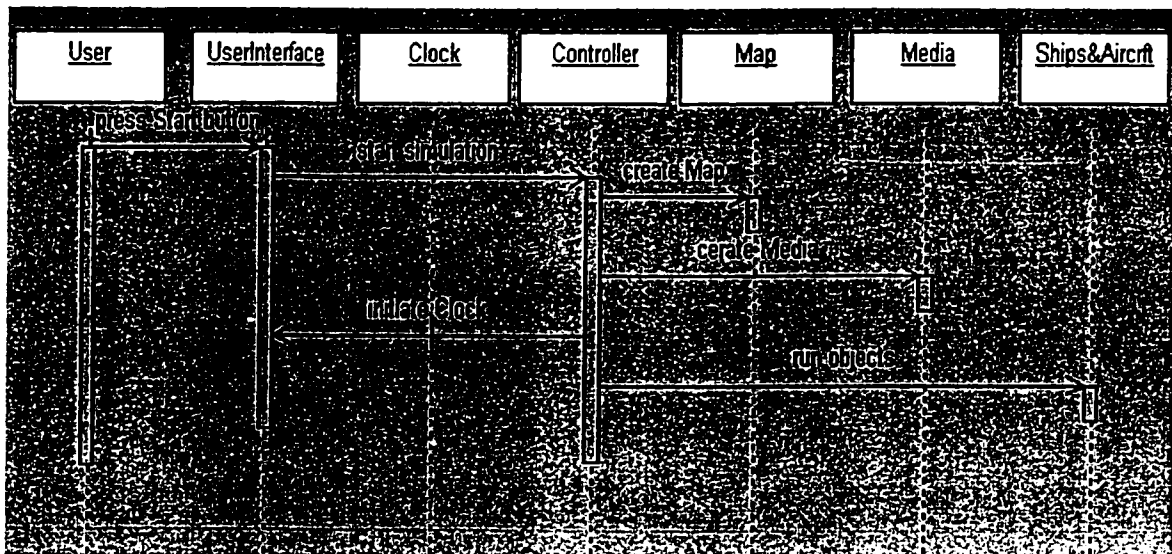


Figure 3-14 Sequence Diagram for Use Case: Start Simulation

3.3.1.3.3 Use Case: Simulate Communication

Description		Provide the service to allow SC and vehicles to communicate with each other, and allow to turn on/off the Radar/Sonar and Radio.
Priority		Must have this use case
Status		High level description
ACID		Communication/Detection
Pre-Conditions		Simulation is in running state
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. All the agents inform their status to the SC periodically 2. The SC transfers the information to the Communication and Detection system 3. Click "Turn on/off" button to change the status of Radar, Sonar, and Radio for selected ship or Aircraft.
	Alternate Path	NA
Post-Conditions		The SC know the status of agents, and all the agents are aware of the presence of other agents within their Communication/Detection range
Related Use Case	Used Use Case	NA
	Extending Use Case	NA
Other Requirements		NA

Table 3-7 Use Case Description for Simulate Communication

Sequence Diagram

See next page.

3.3.1.3.4 Use Case: Base Supply

Description		Provide the service to allow the SC to provide supplies (Weapons, fuel, ships) to both sides when the simulation is running.
Entity		Must have this use case
Status		Detailed description and completed scenario
Agents		NBSS Ships and Aircraft
Pre-Conditions		Simulation is in running state
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. The base supplier will check the stock and transfer the fuel or Weapon to the agents upon request. 2. The base supplier will produce the ships according to the productivity settings periodically.
	Alternate Path	NA.
Post-Condition		<ol style="list-style-type: none"> 1. The ships are generated when the simulation is running 2. The ships get rearmed and refueled.
Related Use Case	Use Case	Simulate Communication
Base Case	Extending Use Case	NA
Other Requirements		NA

Table 3-8 Use Case Description for Base Supply

Sequence Diagram

See next page.

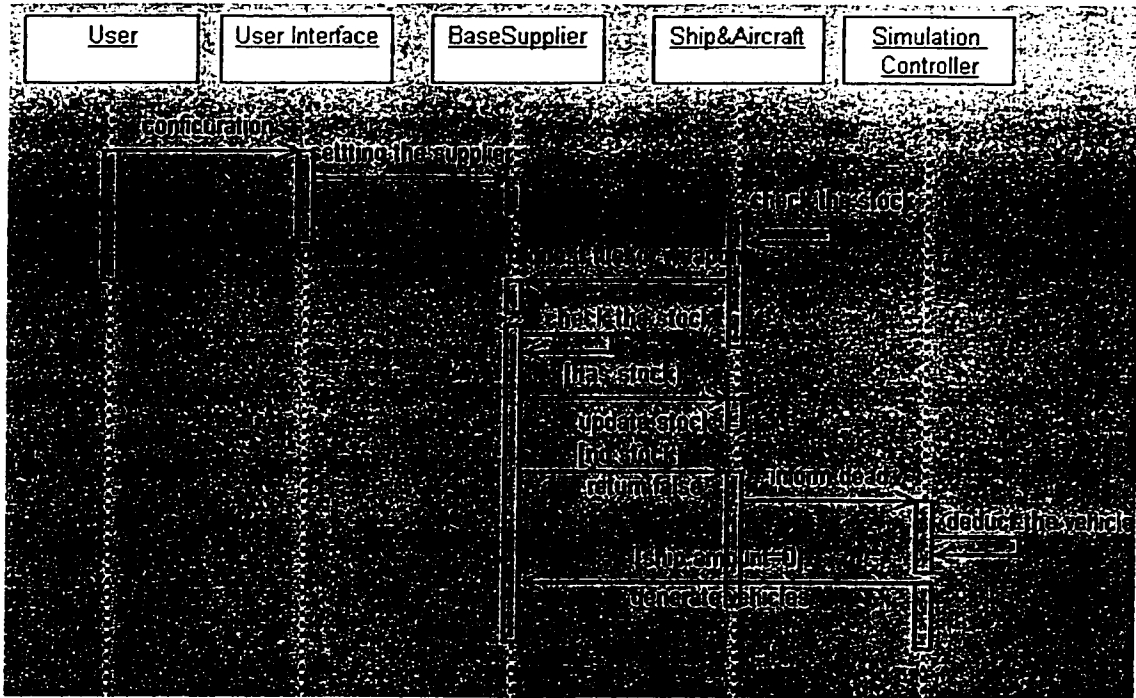


Figure 3-16 Sequence Diagram for Use Case: Base Supplier

3.3.1.3.5 Use Case: Pause Simulation

Description		Provide the service to allow the user to pause the simulation
Priority		Must have this use case
Staus		Detailed description and completed scenario
Actor		NBSS User
Pre-Conditions		Simulation is in running state
Flow of Events	Base Path	1. The user presses the "Pause" button 2. The system pauses the clock and suspends the simulation
	Alternate Path	NA.
Post-Condition		The system saved the current status of all agents and SC also.
Related Use Case	Used Use Case	NA
	Extending Use Cases	NA
Other Requirements		NA

Table 3-9 Use Case Description for Pause Simulation

Sequence Diagram

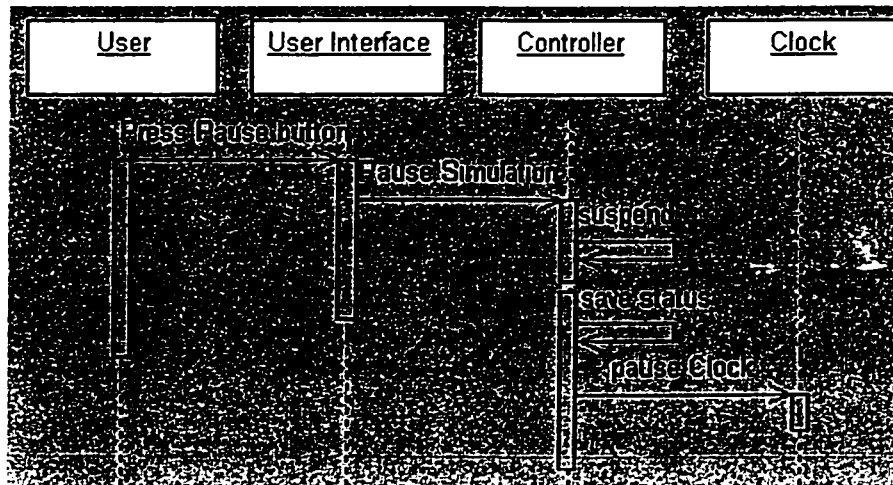


Figure 3-17 Sequence Diagram for Use Case: Pause Simulation

3.3.1.3.6 Use Case: Resume Simulation

Description:		Provide the service to allow the user to resume the simulation
Priority:		Must have this use case
Status:		Detailed description and completed scenario
Actor:		NBSS User
Pre-Conditions:		Simulation is in pause state.
Flow of Events:	Base Path:	1. The user presses the "Resume" button. 2. The system start the clock and resumes the simulation.
	Alternate Path:	NA
PostCondition:		Simulation resumes execution.
Related Use Case:	Used Use Case:	NA
	Extending Use Case:	NA
Other Requirements:		NA

Table 3-10 Use Case Description for Resume Simulation

Sequence Diagram

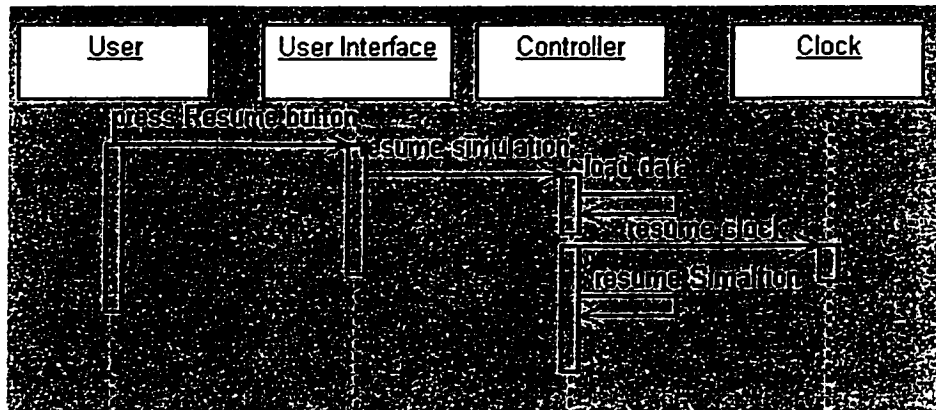


Figure 3-18 Sequence Diagram for Use Case: Resume Simulation

3.3.1.3.7 Use Case: End Simulation

Description:		Provide the service to allow the user to stop the simulation
Priority:		Must have this use case
Status:		Detailed description and completed scenario
Actor:		NBSS User
Pre-Conditions:		Simulation is in running state or in pause state.
Flow of Events:	Base Path:	1. The user presses the "End" button 2. The system terminates the simulation
	Alternate Path:	NA
Post-Condition:		Clean up all the agents, and ready for next simulation.
Related Use Case:	Used Use Case:	NA
	Extending Use Case:	NA
Other Requirements:		NA

Table 3-11 Use Case Description for End Simulation

Sequence Diagram

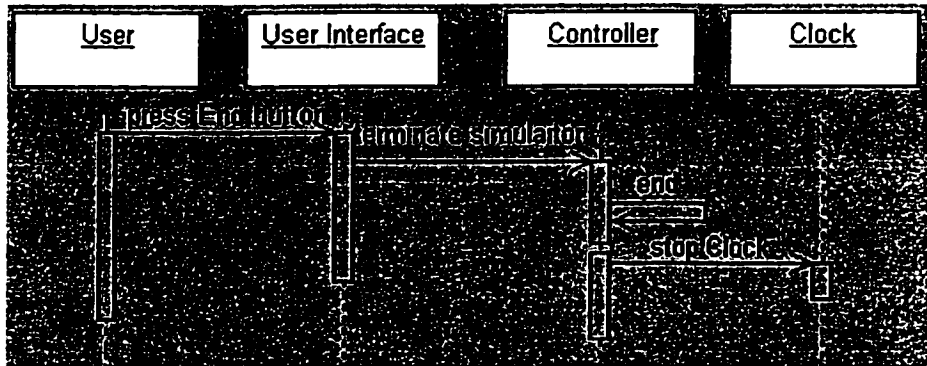


Figure 3-19 Sequence Diagram for Use Case: End Simulation

3.3.1.3.8 Use Case: Report Statistics

Description		Provide the service to allow the user to view an execution report of the running simulation.
Priority		Would like have this use case
Status		Detailed description and completed scenario
Actor		NBSS User
Pre-Conditions		Simulation is in running state or in pause state, or ended successfully.
Flow of Events	Base Path	1. The user presses the "Report" button. 2. The system displays a statistics window.
	Alternate Path	If the simulation terminated erroneously, the statistics window will show nothing.
Post-Condition		Save the valid statistics data, and close the statistics window.
Related Use Case	Used Use Case	NA
	Extending Use Case	NA
	Other Requirements	NA

Table 3-12 Use Case Description for Report Statistics

Sequence Diagram

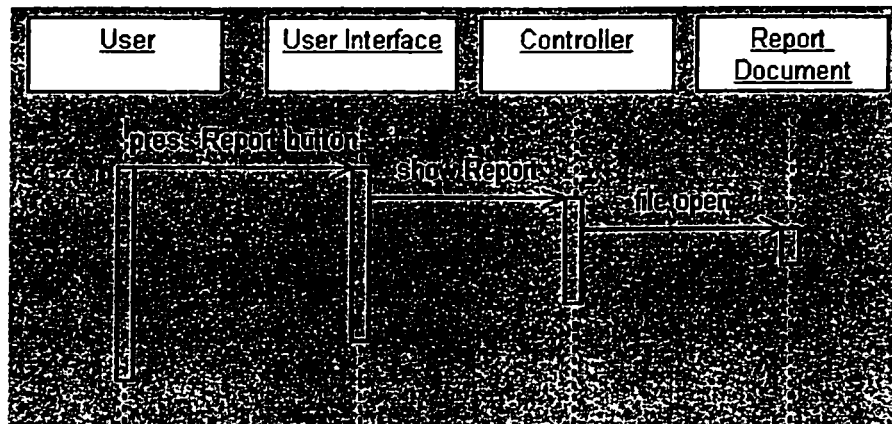


Figure 3-20 Sequence Diagram for Use Case: Report Statistics

3.3.2 Communication/Detection Requirements

The Communication/Detection subsystem has the following four modules:

- Radar system
- Sonar system
- Radio system
- Message Database
- Detected Database

3.3.2.1 Use Case Diagram

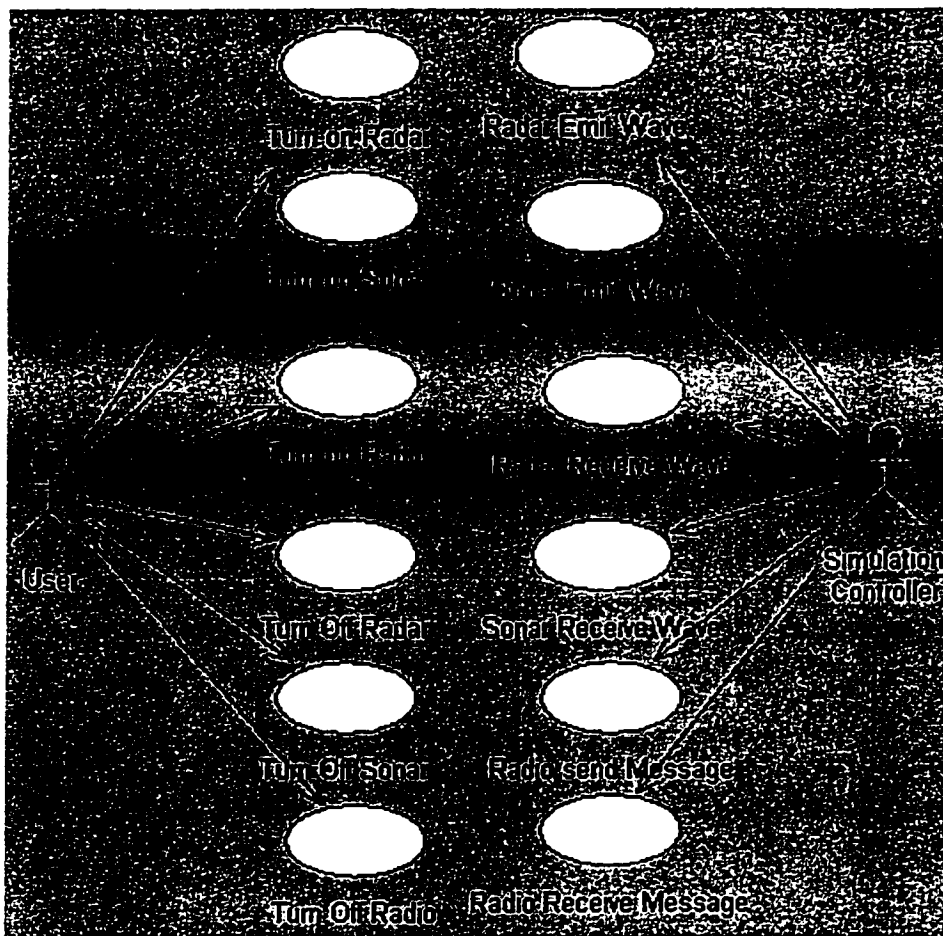


Figure 3-21 Use Case Diagram for Communication/Detection

3.3.2.2 Requirement Breakdown

Use Case: Turn on Radar

CD-001 Turn on Radar

The Radar can be turned on by its owner when it is in the "off" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Turn off Radar

CD-002 Turn off Radar

The Radar can be turned off by the user when it is in "on" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Radar Emit Wave

CD-003 Radar Send Information to SC

The Radar shall provide its owner's ID to the Simulation Controller.

No comments.

Use Case: Radar Receive Wave

CD-004 Radar Get Information from SC

The Radar shall get the information about surrounding objects, both on or above the surface of the water.

The objects refer to Ships, Aircrafts and Missiles.

CD- 004-01 Radar Get Status for Surrounding Objects

The Radar shall get all the position, status and ID information of surrounding objects within the Radar's range.

No comments.

CD-004-02 Radar Update Information

The Radar shall save all the information in its data buffer and update all the information periodically.

No comments.

Use Case: Turn on Sonar

CD-005 Turn on Sonar

The Sonar can be turned on by its owner when it in the "off" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Turn off Sonar

CD-006 Turn off Sonar

The Sonar can be turned off its owner when it is in the "on" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Sonar Emit Wave

CD-007 Send Information to SC

The Sonar shall provide its owner's ID to Simulation Controller.

No comments.

Use Case: Sonar Receive Wave

CD-008 Sonar Get Information from SC

The Sonar shall get the information about surrounding objects in the water. The objects refer to Ships and Torpedoes.

No comments.

CD- 008-01 Sonar Get Status for Surrounding Objects

The Sonar shall get all the position, status and ID information of surrounding objects on or under the surface of the water within the Sonar's range.

No comments.

CD-008-02 Sonar Update Information

The Sonar shall save all the information in its data buffer and update all the information.

No comments.

Use Case: Turn on Radio

CD-009 Turn on Radio

The Radio can be turned on by its owner when Radio is in the "off" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Turn off Radio

CD-010 Turn off Radio

The Radio can be turned off by its owner when Radio is in the "on" state during the simulation is undergoing initialization or running.

No comments.

Use Case: Radio Send Message

CD-011 Radio Send Message

The objects can send a message to its allies via its Radio system and within Radio's range.

The objects refer to all Ships and Aircrafts.

Use Case: Radio Receive Message

CD-012 Radio Receive Message

The objects can receive a message from its allies via its Radio system that communicates with emitting Radio objects within its Radio's range.

The objects refer to all Ships and Aircrafts.

3.3.2.3 Use Case Description

3.3.2.3.1 Use Case: Turn on Radar

Description		Provide a service to allow the user to turn on the Radar
Priority		Should have this use case
Status		Detailed description and completed scenario
Actor		<ol style="list-style-type: none"> 1. User 2. Simulation Controller 3. Battleship, Cruiser, Aircraft 4. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile
Pre-Condition		Radar is in the "off" state
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. User clicks on the "Set Radar" button, the system display Radar setting window. 2. User selects the object from object list. 3. User set state on for Radar, and close the window.
	Alternate Path	NA
Post-Condition		Radar is in the "on" state
Related Use Cases	Used Use Case	NA
	Extending Use Case	Turn on Communication/Detection
Other Requirement		NA

Table 3-13 Use Case Description for Turn on Radar

Sequence Diagram

Refer to Figure 3-8. The object list(ID list) is provided to the Radar owner only for Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer and Weapons (except the Heavy Cannon Shell, Sea-Sub Missile when under the water, Torpedo and Sub-Sea Torpedo).

3.3.2.3.2 Use Case: Turn off Radar

Description		Provide a service to allow the user to turn off the Radar
Priority		Should have this use case
Status		Detailed description and completed scenario
ABO		<ol style="list-style-type: none"> 1. User 2. Simulation Controller 3. Battleship, Cruiser, Aircraft 4. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile
Pre-Condition		Radar is in on state
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. User click "Set Radar" button, the system display Radar setting window. 2. User select the object from object list; 3. User set state off for Radar, and close the window.
	Alternate Path	NA
Post-Condition		Radar is in off state
Related Use Cases	Used Use Case	NA
Cases	Extending Use Case	Turn off Communication/Detection
Other Requirements		NA

Table 3-14 Use Case Description for Turn off Radar

Sequence Diagram

Refer to Figure 3-9. The object list(ID list) is provided to user only for Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer and Weapons (except the Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo).

3.3.2.3.3 Use Case: Radar Emit Wave

Description		Provide a service for objects to send info to the SC in order to detect the surrounding enemies by using Radar.
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		<ol style="list-style-type: none"> 1. Simulation Controller 2. Battleship, Cruiser, Aircraft 3. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile
Pre-Condition		<ol style="list-style-type: none"> 1. Object exist and Radar is created; 2. Object know its position, ID and flag; 3. The DB of SC is accessible.
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Radar gets its owner's ID, position and flag; 2. Radar sends its owner's information to SC;
	Alternate Path	If position DB is not accessible, SC return an error to the object,
Post-Condition		Radar send its owner's info to SC
Related Use Cases	Use Case	NA
	Use Case	Detection Emit Wave
Other Requirements		NA

Table 3-15 Use Case Description for Radar Emit Wave

Sequence Diagram

Refer to Figure 3-10. This use case is only applicable for objects Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer and Weapons (except the Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo).

3.3.2.3.4 Use Case: Radar Receive Wave

Description		Provide a service to allow the objects to receive the information from SC in order to detect the surrounding enemies by using a Radar.
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		<ol style="list-style-type: none"> 1. Simulation Controller 2. Battleship, Cruiser, Aircraft 3. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile
Pre-Condition		<ol style="list-style-type: none"> 1. Object exist and Radar is created; 2. Object know its position, ID and flag; 3. The DB of SC is accessible; 4. Radar's data buffer is available.
How to Events	Base Path	<ol style="list-style-type: none"> 1. Radar get the record of all the surrounding enemy objects within Radar's range from SC's status DB; 2. Radar save the info to its data buffer and update the info. 3. Radar gives the info to its owner.
	Alternate Path	If position DB is not accessible, SC return an error to the object.
Post-Condition		The Radar's owner gets the info about the surrounding enemy objects.
Related Use Cases	Used Use Case	NA
	Extending Use Case	Detection Receive Wave
Other Requirement		NA

Table 3-16 Use Case Description for Radar Receive Wave

Sequence Diagram

Refer to Figure 3-11 Sequence Diagram for Use Case Detection Receive Wave. This use case is only applicable for objects Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer and Weapons (except the Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo).

3.3.2.3.5 Use Case: Turn on Sonar

Description		Provide a service to allow the user to turn on the Sonar
Priority		Should have this use case
Status AGB		Detailed description and completed scenario
Pre-Condition		1. User 2. Simulation Controller 3. Destroyer, Submarine, and Torpedo Sonar is in off state
Flow of Events	Base Path	1. User click "Set Sonar" button, the system display Radar setting window. 2. User select the object from object list; 3. User sets state on for Sonar, and close the window.
	Alternate Path	NA
Post-Condition		Sonar is in on state
Related Use Cases	Used Use Case	NA
	Extending Use Case	Turn on Communication/Detection
Other Requirement		NA

Table 3-17 Use Case Description for Turn on Sonar

Sequence Diagram

Refer to Figure 3-8. The object list (ID list) is provided to user only for Submarine and Weapons (including Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo)

3.3.2.3.6 Use Case: Turn off Sonar

Description		Provide a service to allow the user to turn off the Sonar
Priority		Should have this use case
Status		Detailed description and completed scenario
Actor		1. User 2. Simulation Controller 3. Destroyer, Submarine, and Torpedo
Pre-Condition		Sonar is in on state
Flow of Events	Base Path	1. User click "Set Sonar" button, the system display Radar setting window. 2. User select the object from object list; 3. User sets state off for Sonar, and close the window.
	Alternate Path	NA
Post Condition		Sonar is in off state
Related Use Cases	Used Use Case	NA
	Extending Use Case	Turn off Communication/Detection
Other Requirement		NA

Table 3-18 Use Case Description for Turn off Sonar

Sequence Diagram

Refer to Figure 3-9. The object list (ID list) is provided to user only for Submarine and Weapons (including Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo

3.3.2.3.7 Use Case: Sonar Emit Wave

Descriptor		Provide a service for objects to send info to SC in order to detect the surrounding enemies using a Sonar.
Priority Status		Must have this use case
Actor		1. Simulation Controller 2. Destroyer, Submarine, and Torpedo
Pre-Condition		1. Object exists, Radar is created and in on state; 2. Object knows its position, ID and flag; 3. The DB of SC is accessible.
Flow of Events	Base Path	1. Sonar gets its owner's ID, position and flag; 2. Sonar sends its owner's information to SC;
	Alternate Path	NA
Post-Condition		Sonar send its owner's info to SC
Related Use Cases	Used Use Case	NA
	Depending Use Case	Detection Emit Wave
Other Requirements		NA

Table 3-19 Use Case Description for Sonar Emit Wave

Sequence Diagram

Refer to Figure 3-10. This use case is only applicable for objects Submarine and Weapons (including Heavy Cannon Shell, Sea-Sub Missile, Torpedo and Sub-Sea Torpedo).

3.3.2.3.8 Use Case: Sonar Receive Wave

Description		Provide a service to allow the objects to receive the information from the SC in order to detect the surrounding enemies using a Sonar.
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		<ol style="list-style-type: none"> 1. Simulation Controller 2. Destroyer, Submarine, and Torpedo
Pre-Condition		<ol style="list-style-type: none"> 1. Object exists and Radar is created and in on state; 2. Object knows its position, ID and flag; 3. The DB of SC is accessible. 4. Sonar's data buffer id available.
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Sonar read the record of all the surrounding enemy objects within Radar's range; 2. Sonar save the info to its data buffer and update the info. 3. Sonar gives the info to its owner.
	Alternate Path	NA
Post-Condition		The Sonar's owner gets the info about the surrounding enemy objects.
Related Use Cases	Uses Use Case	NA
	Extending Use Case	Detection Receive Wave
Other Requirements		NA

Table 3-20 Use Case Description for Sonar Receive Wave

Sequence Diagram

Refer to Figure 3-10, This use case is only applicable for objects Submarine and Weapons (including Heavy Cannon Shell, Sea-Sub Missile (when under water), Torpedo and Sub-Sea Torpedo).

3.3.2.3.9 Use Case: Turn on Radio

Description		Provide a service to allow the user to turn on the Radio
Priority		Should have this use case
Status / AGO		Detailed description and completed scenario
		<ol style="list-style-type: none"> 1. User 2. Simulation Controller 3. Battleship, Cruiser, Aircraft, Destroyer, Submarine, Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile and Torpedo.
Pre-Condition		Radio is in off state
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. User click "Set Radio" button, the system display Radar setting window. 2. User select the object from object list; 3. User sets state on for Radio, and close the window.
	Alternate Path	NA
Post-Condition		Radio is in on state
Related Use Cases	Use Case	NA
Extending Use Case		Turn on Communication/Detection
Other Requirement		NA

Table 3-21 Use Case Description for Turn on Radio

Sequence Diagram

Refer to Figure 3-8. The object list (ID list) is provided to user for Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer, and Submarine.

3.3.2.3.10 Use Case: Turn off Radio

Description		Provide a service to allow the user to turn off the Radio
Priority		Should have this use case
Status		Detailed description and completed scenario
Actor		1. User 2. Simulation Controller 3. Battleship, Cruiser, Aircraft, Destroyer, Submarine,.
Pre-Condition		Radio is in on state
Flow of Events	Base Path	1. User clicks the "Set Radio" button, the system display Radar setting window. 2. User selects the object from object list; 3. User sets state off for Radio, and closes the window.
	Alternate Path	NA
Post-Condition		Radio is in off state
Related Use Cases	Uses Use Case	NA
	Extending Use Case	Turn off Communication/Detection
Other Requirements		NA

Table 3-22 Use Case Description for Turn off Radio

Sequence Diagram

Refer to Figure 3-8. The object list (ID list) is provided to user for Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer, and Submarine.

3.3.2.3.11 Use Case: Radio Send Message

Description		Provide a service for objects send the message to its allies via SC
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		<ol style="list-style-type: none"> 1. Simulation Controller 2. Battleship, Cruiser, Aircraft Carrier Aircraft, Destroyer, and Submarine.
Pre-Condition		<ol style="list-style-type: none"> 1. Object exists and Radio is created and in "on" state. 2. Object know its position, ID and flag; 3. Object know the receivers's IDs and message it want to send. 4. A data buffer for the message is available.
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Object sends a message to its Radio; 2. Radio passes the message to message DB; 3. Message DB check with SC to see if the receivers is within the Radio's range of sender; 4. Message DB keep the message in message list.
	Alternate Path	Step 4: if receiver is not within the range, message DB return an error message to the Radio, and Radio returns it to its owner.
Post-Condition		The message is available in the message DB for the receiver to retrieve them when needed.
Related Use Cases	Used Use Case	NA
	Extending Use Case	NA
Other Requirements		NA

Table 3-23 Use Case Description for Radio Send Message

Sequence Diagram

See next page.

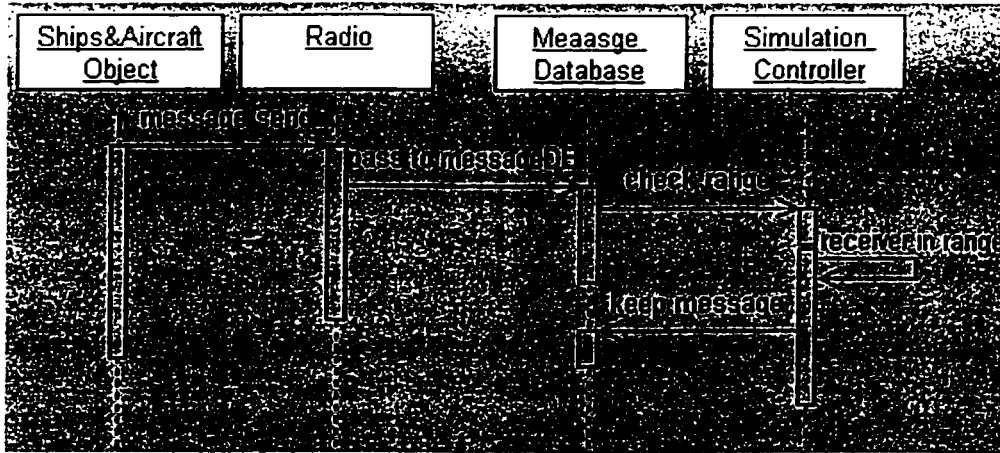


Figure 3-22 Sequence Diagram for Use Case Radio Send Message

3.3.2.3.12 Use Case: Radio Receive Message

Description		Provide a service for objects receive the message from its allies via the SC
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		1. Simulation Controller 2. Battleship, Cruiser, Aircraft Carrier Aircraft, Destroyer, and Submarine.
Pre-Condition		1. Object exists and Radio is created and in "on" state. 2. Object knows its ID; 3. A data buffer for the message list is available.
Flow of Events	Base Path	1. Object provides its ID to its Radio and ask Radio to get message; 2. Radio sends the ID with an empty message list to message DB; 3. Message DB checks the records and copies all the messages for this object ID to the message list; 4. Message DB deletes these copied records from the DB; 5. Message DB return the message list to the Radio; 6. Radio returns this list to its owner;
	Alternate Path	NA
Post-Condition		1. The messages are deleted from the DB; 2. Object receives a message list containing zero or more messages.
Related Use Cases	Used Use Case	NA
	Extending Use Case	NA
Other Requirement		NA

Table 3-24 Use Case Description for Radio Receive Message

Sequence Diagram

See next page.

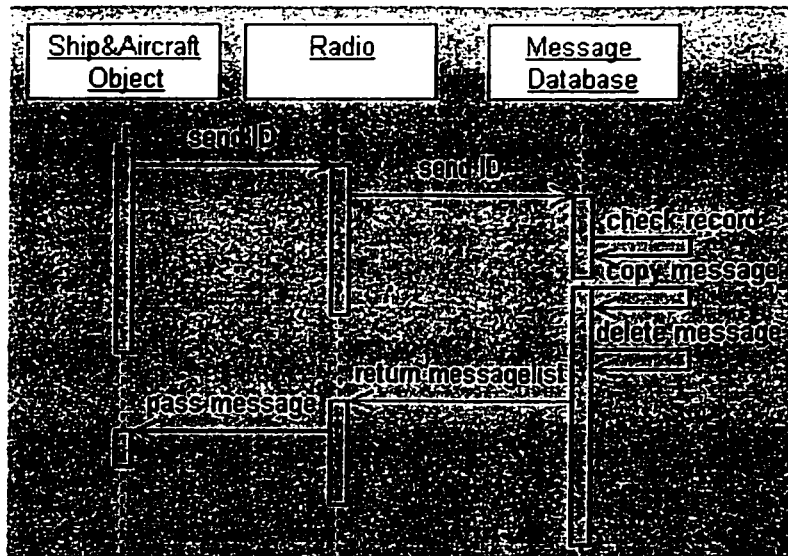


Figure 3-23 Sequence Diagram for Use Case Radio Receive Message

3.3.3 Aircraft Carrier Requirements

The Aircraft Carrier subsystem has the following four modules:

- Captain
- Communication Officer
- Navigation Officer
- Aircraft Launcher Officer

3.3.3.1 Use Case Diagram

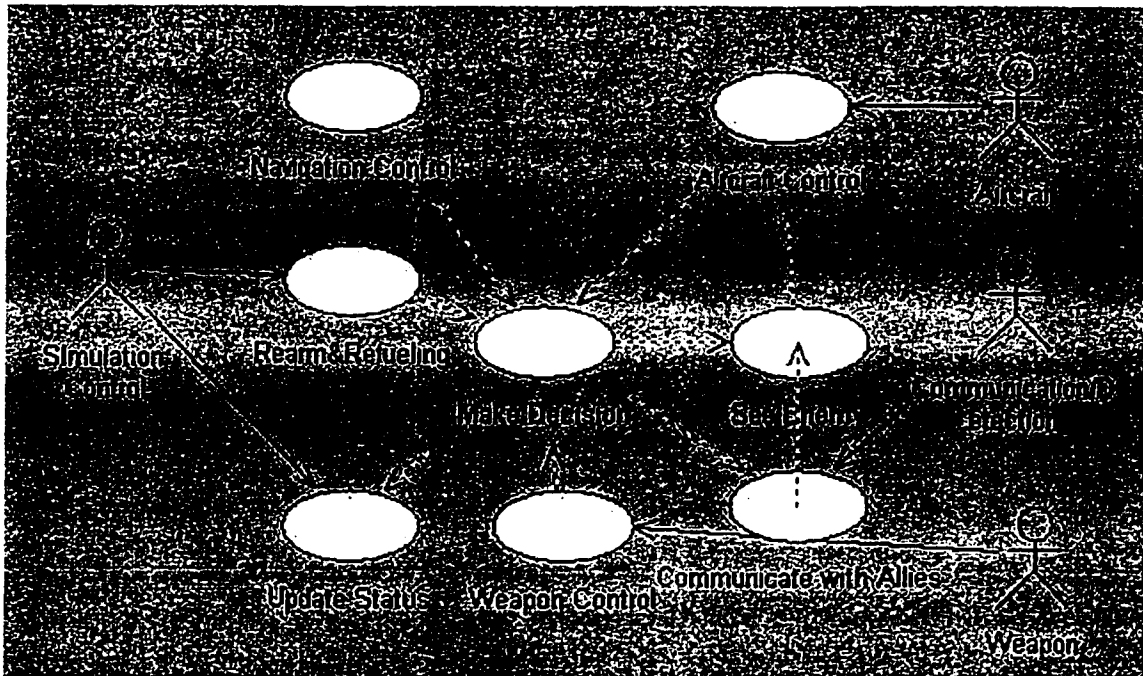


Figure 3-24 Use Case Diagram for Aircraft Carrier

3.3.3.2 Requirement Breakdown

Use Case: Aircraft Carrier Navigate Control

AC-001 Start/Stop Aircraft Carrier

AC-001-01 Start Aircraft Carrier

Aircraft Carrier shall start to move on the sea in a random direction after its initialization.

No comments.

AC-001-02 Stop Aircraft Carrier

Aircraft Carrier shall be stoppable by the user manually.

It is also stopped when its fuel is used up and base supplier has no more fuel.

AC-002 Accelerate/ Decelerate/ Rotate Aircraft Carrier

Aircraft Carrier shall accelerate, decelerate and rotate according to the Captain's command.

No comments.

AC-003 Control Steer Status

Aircraft Carrier shall turn on or turn off the steer in order to navigate on the sea.

No comments.

Use Case: Aircraft Carrier Communication with Allies

AC-004 Initialize Radio

When the Aircraft Carrier is created, a Radio object shall be initialized with location and range.

No comments.

AC-005 Updating Radio Location

Aircraft Carrier's Radio location shall be updated by Simulation Controller.

No comments

AC-006 Control Radio Status

The Aircraft Carrier shall be able to turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

AC-007 Receive Information from Radio

The Aircraft Carrier shall receive the report from its allies (including its Aircrafts) by Radio.

Radio needs to get all the information from Simulation Controller. The information about detected enemy is also sent by its allies (including its Aircrafts) from the Radio.

AC-008 Send Information to Allies

The Aircraft Carrier can send information to its allies (including its Aircrafts) by Radio.

The significant information include newly detected enemies, etc.

Use Case: Aircraft Carrier Make Decision

AC-009 Collect the Necessary Information from Radar and Radio.

This requirement is accomplished by AC-006, AC-011 and AC-011.

No comments.

AC-010 Analysis Information

Aircraft Carrier shall has the ability to analyze the received information to sort out the criticality of all the threats.

No comments.

AC-011 Decide Location to Conduct Ship

Captain shall take decision to steer, accelerate, decelerate the Aircraft Carrier based on the position of the enemies and the position of allied Aircrafts and Ships.

No comments.

AC-012 Decide Content of Sending Information

The Captain shall form the correct command and send them to the Navigation, Aircraft Launcher and Communication Officers.

No comments.

AC-013 Decide Time for Sending Information

The Captain shall decide the correct time to send commands to subsystems.

No comments.

Use Case: Aircraft Control

AC-014 Get Status of Aircraft

Aircraft Carrier receives the current position, speed, and resistance of allied Aircrafts.

No comments.

AC-015 Landing Control

Aircraft Carrier receives the landing request from its Aircrafts and sends the landing authorization to them.

No comments.

AC-016 Send Return Command

Aircraft Carrier shall send the return command to its Aircraft to ask the Aircraft come back.

No comments.

AC-017 Take off Aircraft

Aircraft Carrier shall issue the mission to its Aircraft and permit it to take off.

No comments.

Use Case: Aircraft Carrier Update Status

AC-018 Update Aircraft Carrier Location Periodically

Aircraft Carrier can update its location periodically and randomly if no threats are detected.

No comments.

AC-019 Calculate Aircraft Carrier Resistance

Aircraft Carrier shall calculate the resistance or hit points after each hit.

No comments.

AC-020 Aircraft Carrier Hit by Enemy Weapon

Aircraft Carrier shall know when it is hit by the enemy's Weapon.

No comments.

AC-021 Aircraft Carrier Recover Within Time Limit

Aircraft Carrier can determine if it can recover from the damage within the limited time.

No comments.

AC-022 Report Status to SC Periodically

Aircraft Carrier shall inform its status (location, alive/dead status) to the Simulation Controller periodically.

No comments.

AC-023 Aircraft Carrier Destroyed at Hit Points Limit

Aircraft Carrier shall determine to be destroyed when exceeding the hit points limit.

When hit points reaches zero the Aircraft Carrier is considered inactive.

AC-024 Aircraft Carrier Crashed with other object

Aircraft Carrier shall determine to be destroyed when crash with other object.

When Aircraft Carrier hit by another ship.

Use Case: Aircraft Carrier Refueling

AC-025 Update the Fuel Level

Aircraft Carrier shall reduce its fuel level according to the navigation time since its creation.

No comments.

AC-026 Refueling the Gas

Aircraft Carrier shall send request to its base supplying to refueling when its gas goes to the warning level.

No comments.

3.3.3.3 Use Case Description

3.3.3.3.1 Use Case: Aircraft Carrier Navigation Control

Description		Provide the service to navigate the Aircraft Carrier
Priority		Must have this use case in order to move on the sea
Status		Detailed description and completed scenario
Actor		NA
Pre-Conditions		1. Existing Aircraft Carrier object; 2. A command is received from the navigation officer
Flow Events	Base Path	Upon reception of the command from a navigation officer, the Aircraft Carrier may perform one of following operations: Start or Stop, Rotate, Accelerate, Decelerate
	Alternate Path	NA
Post-Condition		The Aircraft Carrier is moved
Related Use Cases	Use Case	Aircraft Carrier Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-25 Use Case Description for Aircraft Carrier Navigation Control

Sequence Diagram

Refer to Figure 3-1 Sequence Diagram for Use Case Navigation Control for Navigation Control.

3.3.3.3.2 Use Case: Aircraft Carrier Communicate with Allies

Description		Provide the communication service between Aircraft Carrier and its allies.
Priority		Must have this use case in order to pass information to the Aircraft Carrier's allies
Status		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		Existing Aircraft Carrier object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radio object with location and radius when Aircraft Carrier is created; 2. Update Radio location; 3. Turn on /off Radio; 4. Get object information around the Aircraft Carrier; 5. Send message to its allies
	Alternate Path	NA
Post-Conditions		The Aircraft Carrier received report from its allies, the allies received report from Aircraft Carrier
Related Use Case	Used Use Case	NA
	Extending Use Case	Communicate with Allies
Other Requirements		NA

Table 3-26 Use Case Description for Aircraft Carrier Communicate with Allies

Sequence Diagram

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.

3.3.3.3 Use Case: Aircraft Carrier Make Decision

Description		Provide the service to analyze the report, decide attack target, decide where to conduct the ship, decide to rearm and refuel
Priority		Must have this use case in order to know its next action
Status		Detailed description and completed scenario
Tag		Communication/Detection
Pre-Conditions		<ol style="list-style-type: none"> Existing Aircraft Carrier object; The Aircraft Carrier's status is updated; All the reports are received
Flow of Events	Base Path	<ol style="list-style-type: none"> Upon reception of reports, the Captain analyzes the threats and decides to attack a target; The Captain gives the order to the Navigation Officer for where to conduct the ship and at what speed; The Captain gives order to Aircraft Launch officer to prepare the attack; The Captain gives order to Communication Officer to send out the message about detected enemy; The Captain decide to rearm or refueling to send request to SC. The Aircraft Launcher Officer decide to launch the Aircraft.
	Alternate Path	NA
Post-Condition		<ol style="list-style-type: none"> The Navigation Officer executes captain's command The Weapon Officer executes captain's command The Communication Officer execute Captain's command; The Base Supplier perform the transaction task;
Related Use Case	Used Use Case	<ol style="list-style-type: none"> Aircraft Carrier Update Status; Aircraft Carrier Detect Enemy; Aircraft Carrier Communication with Allies;
	Extending Use Case	Make Decision
Other Requirements		NA

Table 3-27 Use Case Description for Aircraft Carrier Make Decision

Sequence Diagram

See next page

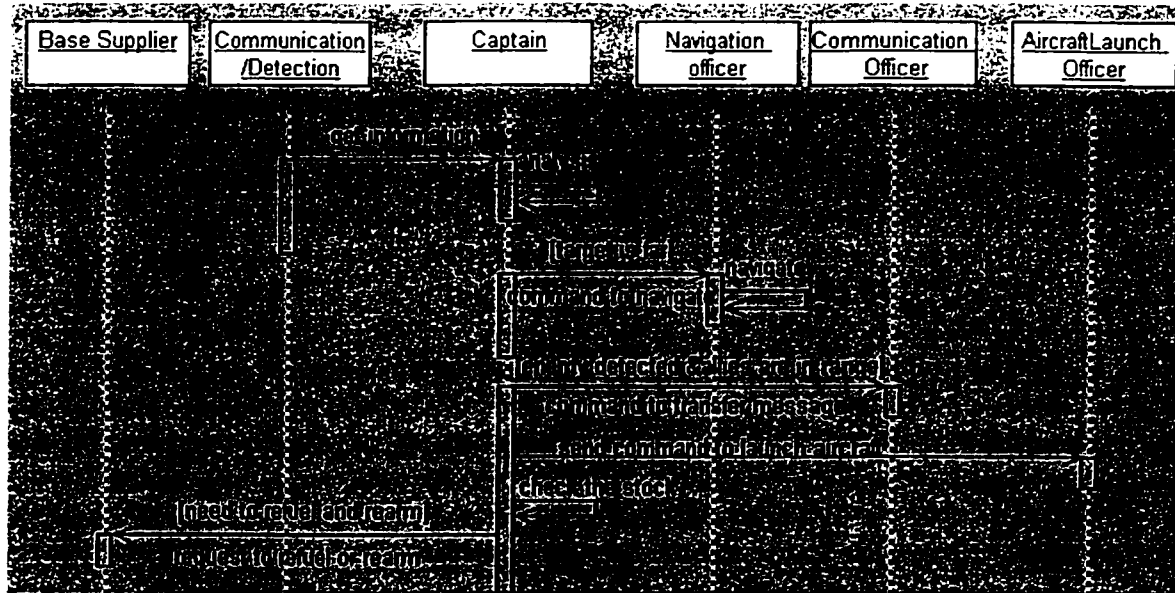


Figure 3-25 Sequence Diagram for Use Case Aircraft Carrier Make Decision

3.3.3.3.4 Use Case: Aircraft Control

Description		Provide the service to control the Aircraft
Priority		Must have this use case in order to control the Aircraft
Status		Detailed description and completed scenario
Actor		Aircraft
Pre-conditions		<ol style="list-style-type: none"> 1. The Aircraft Carrier object exist; 2. The Aircraft object exist; 3. The Aircraft need to be take off.
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. The Captain send request to launch the Aircraft; 2. The Captain allow the Aircraft to take off ; 3. The Aircraft Carrier receive information from its allies Aircraft. 4. Aircraft Carrier respond to the landing request and send command to return.
	Alternate Path	NA
Post Condition		The Aircraft Carrier launch the Aircraft, send command to Aircraft, and respond to Aircraft's request.
Related Use Case	Used Use Case	Make Decision
	Extending Use Case	NA
Other Requirements		NA

Table 3-28 Use Case Description for Aircraft Control

Sequence Diagram

See next page.

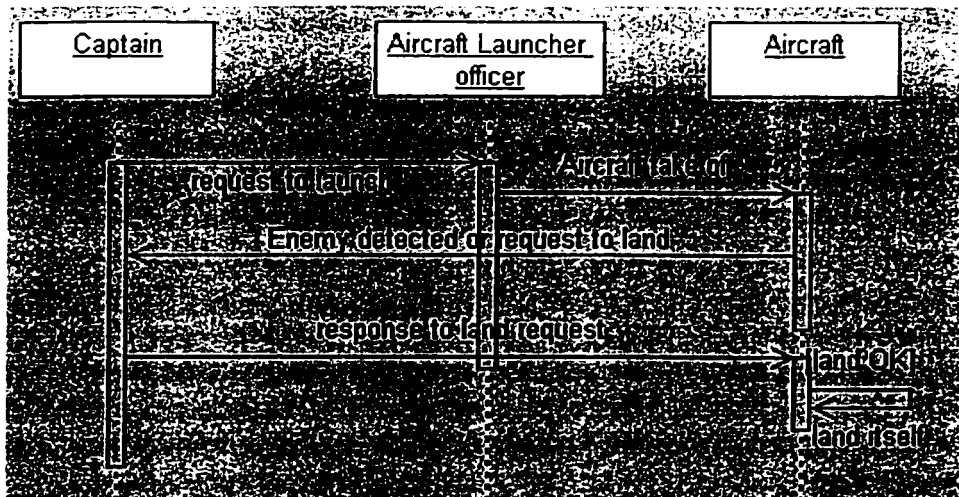


Figure 3-26 Sequence Diagram for Use Case Aircraft Carrier Aircraft Control

3.3.3.3.5 Use Case: Aircraft Carrier Update Status

Description		Provide the service to update Aircraft Carrier's location and other status (alive/dead)
Priority		Must have this use case
Status		Detailed description and completed scenario
Actor		Simulation Controller
Pre-Conditions		Exist a Aircraft Carrier object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Update the location of the Aircraft Carrier 2. Determine if the Aircraft Carrier is hit by Weapon 3. Get the hit points of the Aircraft Carrier 4. Determine if the Aircraft Carrier can recover from the hit points 5. Determine if the Aircraft Carrier is destroyed 6. Determine if the Aircraft Carrier crashes with other object
	Alternate Path	NA
Related Use Case	Used Use Case	The status of the Aircraft Carrier is updated
	Extending Use Case	Update Status
Other Requirements		NA

Table 3-29 Use Case Description for Aircraft Carrier Update Status

Sequence Diagram

Refer to Figure 3-6 Sequence Diagram for Use Case Update Status.

3.3.3.3.6 Use Case: Aircraft Carrier Refueling

Description:		Provide the service to refueling the Aircraft Carrier
Priority:		Would like to have this use case
Status:		Detailed description and completed scenario
Actors:		<ol style="list-style-type: none"> 1. Simulation Controller; 2. Aircraft Carrier; 3. Radio.
Pre-Conditions:		<ol style="list-style-type: none"> 1. The base supplier has enough fuel in stock; 2. The Radio is in "on" state.
Flow of Events:	Base Path:	<ol style="list-style-type: none"> 1. Navigation Officer sends information to ask captain to deduct the fuel; 2. Captain checks if the fuel is at limited level; 3. Captain sends request to SC to ask base supplier to refuel; 4. Base Supplier transfer the fuel to Aircraft Carrier;
	Alternate Path:	NA
Post-Condition:		The Aircraft Carrier gets refueled
Related Use Case:		Aircraft Carrier Make Decision
Extending Use Case:		NA
Other Requirements:		NA

Table 3-30 Use Case Description for Aircraft Carrier Refueling

Sequence Diagram

Refer to Figure 3-7 Sequence Diagram for Use Case Rearm and Refueling.

3.3.4 Aircraft Requirements

The Aircraft subsystem has the following five sub modules:

- Pilot
- Navigation Officer
- Communication Officer
- Weapon Officer
- Weapon Launcher

3.3.4.1 Use Case Diagram

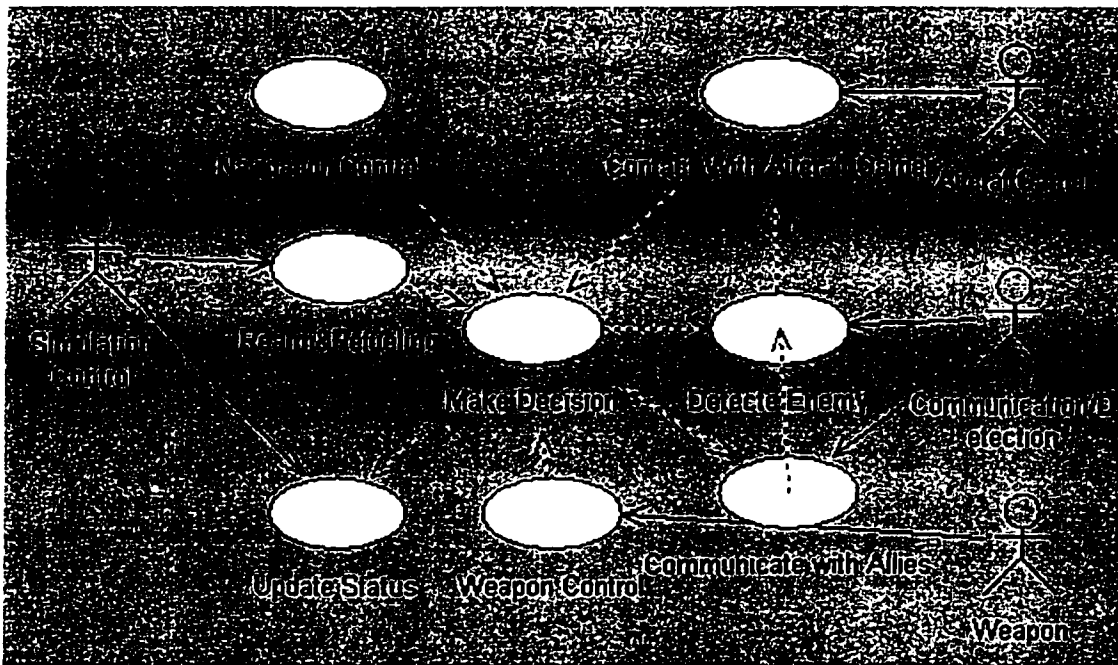


Figure 3-27 Use Case Diagram for Aircraft

3.3.4.2 Requirement Breakdown

Use Case: Aircraft Navigation Control

AT-001 Start/Stop Aircraft

AT-001-01 Start Aircraft

Aircraft shall start to move in the air in random direction after its initiation.

No comments.

AT-001-02 Stop Aircraft

Aircraft shall be stoppable by the user manually.

It is also stopped when its fuel is used up and base supplier has no more fuel.

AT-002 Accelerate/ Decelerate/ Rotate Aircraft

Aircraft shall accelerate, decelerate and rotate according to the Pilot's command.

No comments

AT-003 Control Steer Status

Aircraft shall turn on or turn off the steering in order to navigate.

No comments

Use Case: Aircraft Detect Enemy

AT-004 Initialize Radar

When the Aircraft is created, a Radar object shall be initialized with location and radius.

No comments.

AT-005 Updating Radar Location

Aircraft's Radar location shall be updated by Simulation Controller.

No comments

AT-006 Control Radar Status

The Aircraft shall turn on or turn off the Radar at any time after Radar initialization.

Default status after Radar initialization is turn on.

AT-007 Receive Information from Radar

The Aircraft shall get the information about the surrounding enemies from its Radar.

Radar needs to get all the information from Simulation Controller.

Use Case: Aircraft Communicate With Allies

AT-008 Initialize Radio

When the Aircraft is created, a Radio object shall be initialized with location and radius.

No comments.

AT-009 Updating Radio Location

Aircraft's Radio location shall be updated by Simulation Controller.

No comments

AT-010 Control Radio Status

The Aircraft shall turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

AT-011 Receive Information from Radio

The Aircraft shall receive the report from its allies (including its Aircraft Carrier) by its Radio.

Radio needs to get all the information from Simulation Controller.

AT-012 Send Information to Allies

The Aircraft can send information to its allies (including its Aircraft Carrier) by Radio.

The significant information include newly detected enemies, the target it will attack, etc.

Use Case: Aircraft Make Decision

AT-013 Collect the Necessary Information from Radar and Radio.

This requirement is accomplished by AT-006, AT-011 and AT-012.

No comments.

AT-014 Analysis Information

Aircraft shall has the ability to analyze the received information to decide all the threats.

No comments.

AT-015 Decide Attack Object

Decide attack objects among threats based on the analyzed threats

No comments.

AT-016 Decide Location to Conduct Ship

The Pilot shall take decision to steer, accelerate, decelerate the Aircraft based on position of allies and enemies.

No comments

AT-017 Decide Content of Sending Information

The Pilot shall form the correct command and send them to navigation officer, Weapon officer and communication officer.

No comments.

AT-018 Decide Time for Sending Information

The Pilot shall decide the correct time to send the command to subsystems.

No comments.

Use Case: Aircraft Weapon Control

AT-019 Select Number and Type of Weapon

Weapon Officer shall decide the type and quantity of Weapon to be used on the Aircraft.

No comments.

AT-020 Initialize Weapon

Weapon Officer will issue an order to Weapon launcher to create a Weapon.

No comments.

AT-021 Aim Object and Fire Weapon

Weapon object shall aim the target and fired by Weapon launcher.

Except the Heavy Cannon Shell, it is unguided after it is shot. It is also not for Aircraft.

AT-022 Update the Number of Weapon

Weapon Officer shall calculate and update the number of Weapons on board.

No comments.

AT-023 Recharge Weapon

When the Weapons are used up, the Aircraft shall go back to the base (just give some remind to show the Weapon is used up) and the Weapon officer can reload the Weapon as needed type and quantity.

No comments.

Use Case: Aircraft Update Status

AT-024 Update Aircraft Location Periodically

Aircraft can update its location periodically and randomly if no threats are detected.

No comments.

- AT-025 Calculate Aircraft Resistance**
Aircraft shall calculate the resistance or hit points after each hit.
When hit points reaches zero the Aircraft is considered inactive.
- AT-026 Aircraft Hit by Enemy Weapon**
Aircraft shall know when it is hit by the enemy's Weapon.
No comments.
- AT-027 Aircraft Recover Within Time Limit**
Aircraft can determine if it can recover from the hit points within the limited time.
No comments.
- AT-028 Report Status to SC Periodically**
Aircraft shall inform its status (location, alive/dead status) to Simulation Controller periodically.
No comments.
- AT-029 Report Status to Aircraft Carrier Periodically**
Aircraft shall inform its status (location, alive/dead status) to Aircraft Carrier periodically
No comments.
- AT-030 Aircraft Destroyed at Hit Points Limit**
Aircraft shall determine to be destroyed when exceed the hit points limit.
No comments.
- AT-031 Aircraft Crashed with other object**
Aircraft shall determine to be destroyed when crash with other object.
When Aircraft hit by other Aircraft.

Use Case: Aircraft Rearm and Refueling

- AT-032 Update the Fuel Level**
Aircraft shall reduce its fuel level according to the navigation time since its creation.
No comments.
- AT-033 Refueling the Gas**
Aircraft shall send request to its base supplying when its gas goes to the warning level.
No comments.

AT-034 Rearm the Weapon

Aircraft shall send the request to its base supplying once its Weapons are used up.

Actually, the Weapon are created by Aircraft when they are launched, only after the fired Weapon exceed the limits, the base supplying will create Weapon for Aircraft and transfer them to Aircraft.

3.3.4.3 Use Case Description

3.3.4.3.1 Use Case: Aircraft Navigation Control

Description		Provide the service to navigate the Aircraft
Priority		Must have this use case in order to move
Status		Detailed description and completed scenario
ACOP		NA
Pre-Conditions		<ol style="list-style-type: none"> Existing Aircraft object; A command is received from the navigation officer
Flow of Events	Base Path	<ol style="list-style-type: none"> Upon reception of the command from a navigation officer, the Aircraft may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate; Upon received return command from Aircraft Carrier, the Aircraft shall go back to its Aircraft Carrier.
	Alternate Path	NA
Post-Condition		The Aircraft is moved
Related Use Case	Used Use Case	Aircraft Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-31 Use Case Description for Aircraft Navigation Control

Sequence Diagram

See next page.

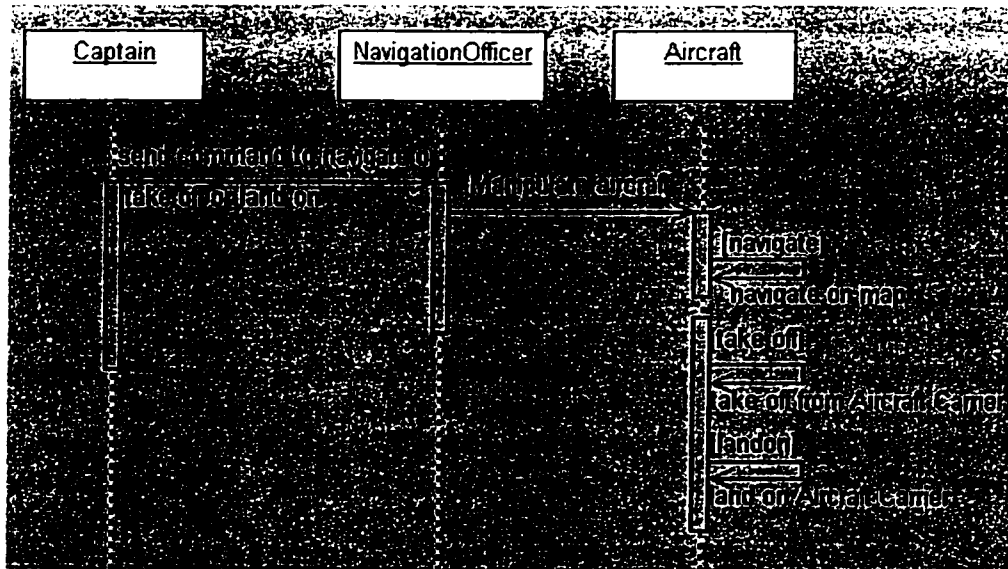


Figure 3-28 Sequence Diagram for Use Case Aircraft Navigation Control

3.3.4.3.2 Use Case: Aircraft Detect Enemy

Description		Provide the service to locate the enemy using Radar
Priority		Must have this use case in order to detect the enemy
Status		Detailed description and completed scenario
Acc.		Communication/Detection
Pre-Conditions		Existing Aircraft object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radar object with location and radius when Aircraft Carrier is created; 2. Update Radar location; 3. Turn on /off Radar; 4. Get enemy information around the Aircraft
	Alternate Path	NA
Post-Condition		Any enemy in the range are detected
Related Use Case	Used Use Case	NA
	Extending Use Case	Detect Enemy
Other Requirements		NA

Table 3-32 Use Case Description for Aircraft Detect Enemy

Sequence Diagram

Refer to Figure 3-2.

3.3.4.3.3 Use Case: Aircraft Communicate with Allies

Description		Provide the communication service among Aircraft, its allies , and its Aircraft Carrier.
Priority		Must have this use case in order to pass information to the Aircraft 's allies and its Aircraft Carrier
Status		Detailed description and completed scenario
Acft		Communication/Detection
Pre-Conditions		Exist a Aircraft object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radio object with location and radius when Aircraft is created; 2. Update Radio location; 3. Turn on /off Radio; 4. Get enemy object information around the Aircraft; 5. Send message to its allies and its Aircraft Carrier.
	Alternate Path	NA
Post-conditions		The Aircraft received report from its allies and Aircraft Carrier; the allies and Aircraft Carrier received report from Aircraft.
Related Use Case	Used Use Case	NA
	Extending Use Case	Communicate with Allies
Other Requirements		NA

Table 3-33 Use Case Description for Aircraft Communication with allies

Sequence Diagram

Refer to Figure 3-3.

3.3.4.3.4 Use Case: Aircraft Make Decision

Description		Provide the service to analyze the report, decide attack target, decide where to conduct the Aircraft, decide rearm and refueling
Priority		Must have this use case in order to know its next action
Status		Detailed description and completed scenario
Addr		Communication/Detection
Pre-Conditions		<ol style="list-style-type: none"> Existing Aircraft object; The Aircraft status is updated; All the reports are received
Flow of Events	Base Path	<ol style="list-style-type: none"> Upon reception of reports, the captain analyze the threats and decide attack target; The captain gives the order to navigation officer for where to conduct the Aircraft and at what speed; The captain gives order to Weapon officer to prepare the attack; The captain gives order to communication officer to send out the message about detected enemy; The Captain decide to rearm or refueling to send request to SC. The Pilot decide to land on the Aircraft Carrier.
	Alternate Path	NA
Post-condition		<ol style="list-style-type: none"> The navigation officer executes captain's command The Weapon office executes captain's command The communication officer execute captain's command; The Base Supplier perform the transaction task; Aircraft send request to land on.
Related Use Case	Used Use Case	<ol style="list-style-type: none"> Aircraft Update Status; Aircraft Detect Enemy; Aircraft Communication with Allies;
	Extending Use Case	Make Decision
Other Requirements		NA

Table 3-34 Use Case Description for Aircraft Make Decision

Sequence Diagram

See next page.

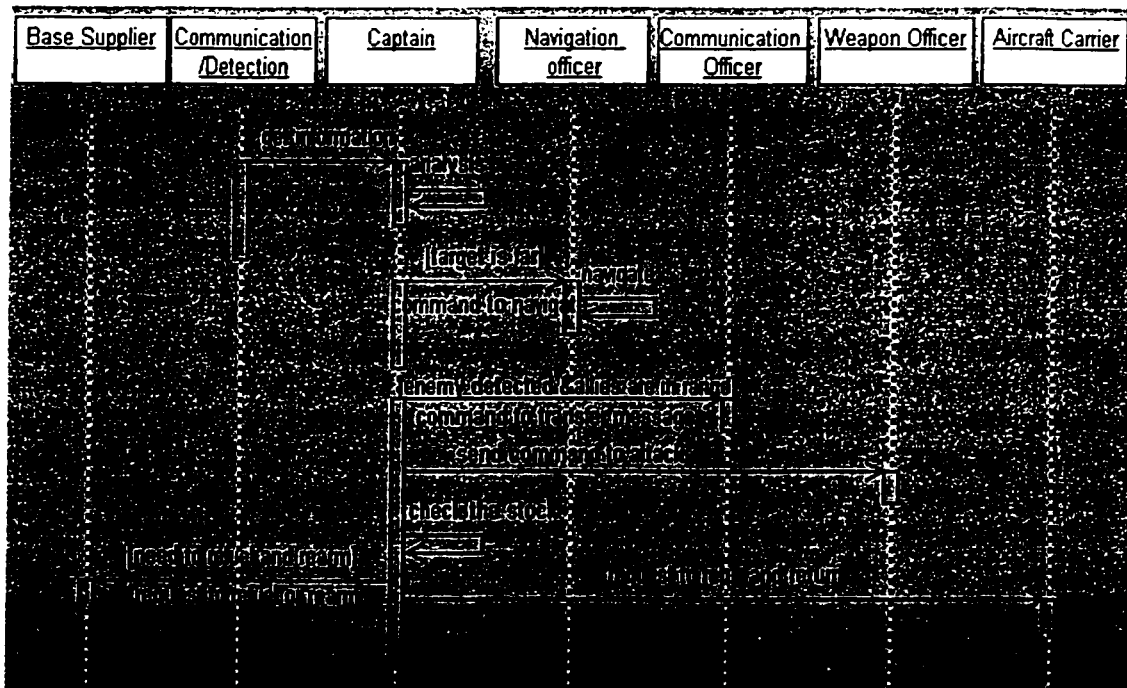


Figure 3-29 Sequence Diagram for Use Case Aircraft Make Decision

3.3.4.3.5 Use Case: Aircraft Weapon Control

Description		Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed
Priority		Must have this use case in order to attack the enemy
Status		Detailed description and completed scenario
Actor		Weapon
Pre-Conditions		An attacking command is received
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Decide the type and quantity of Weapon to be used; 2. Calculate and update the Weapon quantity on board 3. Issue an order to Weapon launcher 4. A Weapon object will be created and fired by Weapon launcher 5. Weapon launcher will aim and fire Weapon 6. Deduct the quantity of Weapon on board
	Alternate Path	If the Weapon is Sea-Sea Missile, it will return a message stating whether the target is destroyed or not.
Post-Condition		Weapon is fired and exploded
Related Use Case	Used Use Case	Aircraft Make Decision
	Extending Use Case	Weapon Control
Other Requirements		NA

Table 3-35 Use Case Description for Aircraft Weapon Control

Sequence Diagram

Refer to Figure 3-5.

3.3.4.3.6 Use Case: Aircraft Update Status

Description		Provide the service to update Aircraft's location and other status (alive/dead)
Entity		Must have this use case in order to report status to SC
Status		Detailed description and completed scenario
Actor		Simulation Controller
Pre-Conditions		Exist a Aircraft object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Update the location of the Aircraft 2. Determine if the Aircraft is hit by Weapon 3. Get the hit points of the Aircraft 4. Determine if the Aircraft can recover from the hit points 5. Determine if the Aircraft is destroyed 6. Determine if the Aircraft crashes with other object
	Alternate Path	NA
Post-Condition		The status of the Aircraft is updated
Related Use Case		NA
Case Extending Use Case		NA
Other Requirements		NA

Table 3-36 Use Case Description for Aircraft Update Status

Sequence Diagram

Refer to Figure 3-6.

3.3.4.3.7 Use Case: Aircraft Rearm and Refueling

Description:		Provide the service to refueling the Aircraft Carrier
Priority:		Would like to have this use case in order to continue moving on the sea
Status:		Detailed description and completed scenario
Actors:		1. Simulation Controller; 2. Aircraft; 3. Radio.
Pre-Conditions:		1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.
Flow of Events	Base Path	1. Navigation Officer send information to ask captain to deduct the fuel; 2. Pilot checks if the fuel is at limited level; 3. Pilot send request to SC to ask base supplier to refuel; 4. Base Supplier transfer the fuel to Aircraft;
	Alternate Path	NA
Post Related Use Case	Used Use Case	The Aircraft get refueling Aircraft Make Decision
	Extending Use Case	NA
	Other Requirements	NA

Table 3-37 Use Case Description for Aircraft Refueling

Sequence Diagram

Refer to Figure 3-7.

3.3.5 Destroyer Requirements

The Destroyer subsystem has the following five sub modules:

- Captain
- Navigation Officer
- Communication Officer
- Weapon Officer
- Weapon Launcher

3.3.5.1 Use Case Diagram

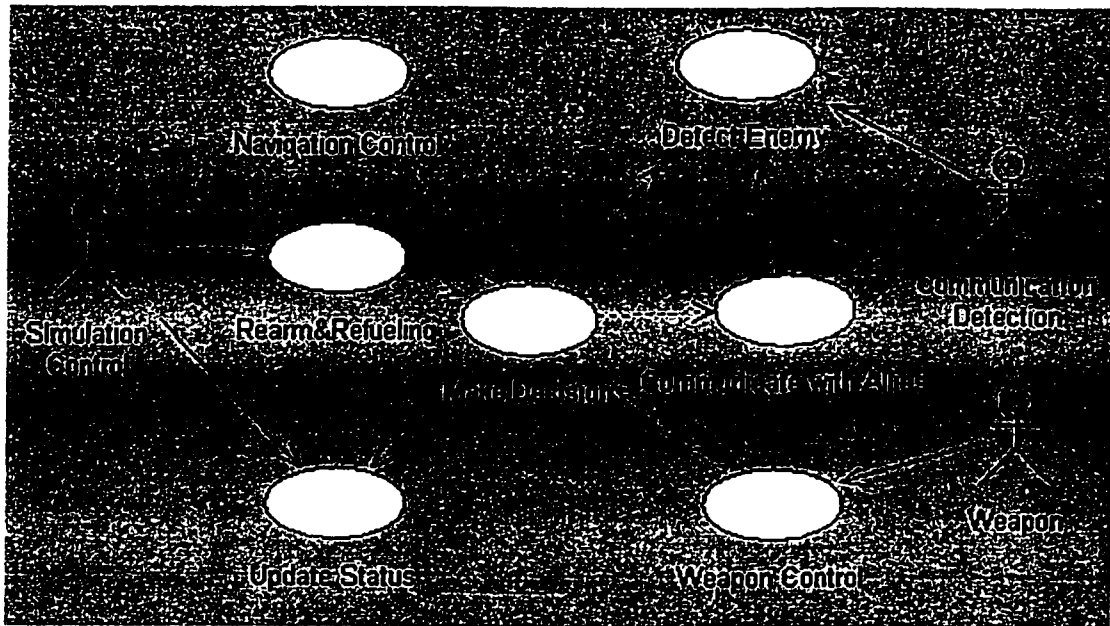


Figure 3-30 Use Case Diagram for Destroyer

3.3.5.2 Requirement Breakdown

Use Case: Destroyer Navigation Control

DT-001 Start/Stop Destroyer

DT-001-01 Start Destroyer

Destroyer shall start to move on the sea in random direction after its initiation.

No comments.

DT-001-02 Stop Destroyer

Destroyer shall be stoppable by the user manually.

It is also stopped when its fuel is used up and base supplier has no more Fuel.

DT-002 Accelerate/ Decelerate/ Rotate Destroyer

Destroyer shall accelerate, decelerate and rotate according to the Captain's command.

No comments.

DT-003 Control Steer Status

Destroyer shall turn on or turn off the steer in order to navigate on the sea.

No comments.

Use Case: Destroyer Detect Enemy

DT-004 Initialize Radar

When the Destroyer is created, a Radar object shall be initialized with location and radius.

No comments.

DT-005 Updating Radar Location

Destroyer's Radar location shall be updated by Simulation Controller.

No comments

DT-006 Control Radar Status

The Destroyer shall be able to turn on or turn off the Radar at any time after Radar initialization.

Default status after Radar initialization is turn on.

DT-007 Receive Information from Sonar

The Destroyer shall get the information about the near Submarine from its Sonar

Radar needs to get all the information from Simulation Controller.

Use Case: Destroyer Communication with Allies

DT-008 Initialize Radio

When the Destroyer is created, a Radio object shall be initialized with location and radius.

No comments

DT-009 Updating Radio Location

Destroyer's Radio location shall be updated by Simulation Controller.

No comments

DT-010 Control Radio Status

The Destroyer shall turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

DT-011 Receive Information from Radio

The Destroyer shall receive the report from its allies by its Radio.

Radio needs to get all the information from Simulation Controller.

DT-012 Send Information to Allies

The Destroyer can send information to its allies.

The significant information include newly detected enemies, the target it will attack, etc

Use Case: Destroyer Make Decision

DT-013 Collect Information from Radar and Radio.

This requirement is accomplished by DT-006, DT-011 and DT-012.

No comments.

DT-014 Analysis Information

Destroyer shall has the ability to analyze the received information to decide all the threats.

No comments.

DT-015 Decide Attack Object

Decide attack objects among threats based on the analyzed threats.

No comments

DT-016 Decide Location to Conduct Ship

DT-017 Decide Content of Sending Information

The Captain shall form the correct command and send them to the Navigation Officer, Weapon officer and communication officer.

No comments.

DT-018 Decide Time for Sending Information

The Captain shall decide the correct time to send the command to sub system.

No comments.

Use Case: Destroyer Weapon Control

DT-019 Select Number and Type of Weapon

Weapon officer shall decide the type and quantity of Weapon to be used on the Destroyer.

No comments.

DT-020 Initialize Weapon

Weapon Officer will issue an order to Weapon launcher to create a Weapon.

No comments.

DT-021 Aim Object and Fire Weapon

Weapon object shall aim the target and fired by Weapon launcher.

Except the Heavy Cannon Shell, it is unguided after it is shot. it is only for Battleship.

DT-022 Update the Number of Weapon

Weapon officer shall calculate and update the Weapon on board.

No comments

DT-023 Recharge Weapon

When the Weapons are used up, the Destroyer shall go back to the battle base, and the Weapon office can reload the Weapon as needed type and quantity.

No comments.

Use Case: Destroyer Update Status

DT-024 Update Destroyer Location Periodically

Destroyer can update its location periodically and randomly if no threats are detected.

No comments.

DT-025 Calculate Destroyer Resistance
Destroyer shall calculate the resistance or hit points after each hit.
When hit points reaches zero the Destroyer is considered inactive.

DT-026 Destroyer Hit by Enemy Weapon
Destroyer shall know when it is hit by the enemy's Weapon.
No comments.

DT-027 Destroyer Recover Within Time Limit
Destroyer can determine if it can recover from the hit points within the limited time.
No comments.

DT-029 Report Status to SC Periodically
Destroyer shall inform its status (location, alive/dead status) to Simulation Controller periodically.
No comments.

DT-030 Destroyer Destroyed at Hit Points Limit
Destroyer shall determine to be destroyed when exceed the hit points limit.
No comments.

DT-031 Destroyer Crashed with other object
Destroyer shall determine to be destroyed when crash with other object.
When Destroyer hit by another Ship.

Use Case: Destroyer Rearm and Refueling

DT-032 Update the Fuel Level
Destroyer shall reduce its fuel level according to the navigation time since its creation.
No comments.

DT-033 Refueling the Gas
Destroyer shall send request to its base supplying to refueling when its gas goes to the warning level.
No comments.

DT-034 Rearm the Weapon
Destroyer shall send the request to its base supplying once its Weapons are used up.
Actually, the Weapon are created by Destroyer when they are launched, only after the fired Weapon exceed the limits, the base supplying will create Weapon for Destroyer and transfer them to Destroyer .

3.3.5.3 Use Case Description

3.3.5.3.1 Use Case: Destroyer Navigation Control

Description		Provide the service to navigate the Destroyer
Priority		Must have this use case in order to move on the sea
Status		Detailed description and completed scenario
Actor		NA
Pre-Conditions		1. Exist a Destroyer object; 2. A command is received from the navigation officer
Flow of Events	Base Path	Upon reception of the command from a navigation officer, the Destroyer may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate
	Alternate Path	NA
Post Condition		The Destroyer is moved
Related Use Case	Used Use Case	Destroyer Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-38 Use Case Description for Destroyer Navigation Control

Sequence Diagram

Refer to Figure 3-1.

3.3.5.3.2 Use Case: Destroyer Detect Enemy

Description		Provide the service to locate the enemy using Radar
Priority		Must have this use case in order to detect the enemy
Status		Detailed description and completed scenario
Action		Communication/Detection
Pre-Conditions		Exist a Destroyer object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radar object with location and radius when Destroyer is created; 2. Update Radar location; 3. Turn on /off Radar; 4. Get enemy object information around the Destroyer
	Alternate Path	NA
Post-Condition		Any enemy in the range are detected
Related Use Case	Used Use Case	NA
Given	Extending Use Case	Detect Enemy
Other Requirements		NA

Table 3-39 Use Case Description for Destroyer Navigation Control

Sequence Diagram

Refer to Figure 3-2.

3.3.5.3.3 Use Case: Destroyer Communication with Allies

Description		Provide the communication service between Destroyer and its allies
Priority		Communication/Detect must have this use case in order to pass information to the Destroyer 's allies
Status		Detailed description and completed scenario
Acco		Communication/Detection
Pre-Conditions		Exist a Destroyer object
Flow of Events	Base Path	1. Initialize a Radio object with location and radius when Destroyer is created; 2. Update Radio location; 3. 3. Turn on /off Radio; 4. 4. Get object information around the Destroyer ; 5. Send message to its allies
	Alternate Path	NA
	Post-Condition	The Destroyer received report from its allies, the Allies received report from Destroyer
Related Use Case	Used Use Case	NA
	Extending Use Case	Communicate with Allies
Other Requirements		NA

Table 3-40 Use Case Description for Destroyer Communication with Allies

Sequence Diagram

Refer to Figure 3-3.

3.3.5.3.4 Use Case: Destroyer Make Decision

Description:		Provide the service to analyze the report, decide attack target, and decide where to conduct the ship
Priority:		Must have this use case in order to know its next action
Status:		Detailed description and completed scenario
ACOP:		Communication/Detection
Pre-Conditions:		<ol style="list-style-type: none"> 1. Exist a Destroyer object; 2. The Destroyer 's status is updated; 3. All the reports are received
Flow of Events:	Base Path:	<ol style="list-style-type: none"> 1. Upon reception of reports, the captain analyze the threats and decide attack target; 2. The captain gives the order to navigation officer for where to conduct the ship and at what speed; 3. The captain gives order to Weapon officer to prepare the attack; 4. The captain gives order to communication officer to send out the message 5. The Captain decide to rearm or refueling to send request to SC
	Alternate Path:	NA
Post Condition:		<ol style="list-style-type: none"> 1. The navigation officer executes captain's command 2. The Weapon office executes captain's command 3. The communication officer execute captain's command; 4. The Base Supplier perform the transaction task;
Related Use Case:	Used Use Case:	<ol style="list-style-type: none"> 1. Destroyer Update Status; 2. Destroyer Detect Enemy; 3. Destroyer Communication with Allies;
	Extending Use Case:	Make Decision
Other Requirements:		NA

Table 3-41 Use Case Description for Destroyer Make Decision

Sequence Diagram

Refer to Figure 3-4.

3.3.5.3.5 Use Case: Destroyer Weapon Control

Description		Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed
Priority		Must have this use case in order to attack the enemy
Status		Detailed description and completed scenario
Actor		Weapon
Pre-Conditions		An attacking command is received
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Decide the type and quantity of Weapon to be used; 2. Calculate and update the Weapon quantity on board 3. Issue an order to Weapon launcher 4. A Weapon object will be created and fired by Weapon launcher 5. Weapon launcher will aim and fire Weapon 6. When Weapons are used up, recharge the Weapon on board
	Alternate Path	If the Weapon is Sea-Sea Missile, it will return a message stating whether the target is destroyed or not
Post-Conditions		Weapon is fired and exploded.
Related Use Case	Used Use Case	Destroyer Make Decision
	Extending Use Case	Weapon Control
Other Requirements		NA

Table 3-42 Use Case Description for Destroyer Weapon Control

Sequence Diagram

Refer to Figure 3-5.

3.3.5.3.6 Use Case: Destroyer Update Status

Description		Provide the service to update Destroyer 's location and other status (alive/dead)
Priority		Must have this use case in order to report status to SC
Status		Detailed description and completed scenario
Actor		Simulation Controller
Pre-Conditions		Exist a Destroyer object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Update the location of the Destroyer 2. Determine if the Destroyer is hit by Weapon 3. Get the hit points of the Destroyer 4. Determine if the Destroyer can recover from the hit points 5. Determine if the Destroyer is destroyed 6. Determine if the Destroyer crashes with other object
	Alternate Path	NA
Post-Condition		The status of the Destroyer is updated
Related Use Case	Use Case	NA
Extending Use Case	Use Case	Update Status
Other Requirements		NA

Table 3-43 Use Case Description for Destroyer Update Status

Sequence Diagram

Refer to Figure 3-6.

3.3.5.3.7 Use Case: Destroyer Rearm and Refueling

Description:		Provide the service to rearm and refueling the Destroyer
Priority:		Would like to have this use case in order to continue moving on the sea
Status:		Detailed description and completed scenario
Actor:		<ol style="list-style-type: none"> 1. Simulation Controller; 2. Destroyer; 3. Radio.
Pre-Conditions:		<ol style="list-style-type: none"> 1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.
Flow or Events:	Base Path:	<ol style="list-style-type: none"> 1. Navigation Officer send information to ask captain to deduct the fuel; Weapon Officer send information to Captain to deduct the Weapon; 2. Captain check if the fuel is at limited level; Captain check if the Weapon is used up ; 3. Captain send request to SC to ask base supplier to refuel; Captain send request to SC to ask base supplier to create Weapon; 4. Base Supplier transfer the fuel or Weapon to Aircraft Carrier.
	Alternate Path:	NA
Post-Condition:		The Destroyer get rearm and refueling
Related Use Case:	Used Use Case:	NA
	Extending Use Case:	Rearm and Refueling
Other Requirements:		NA

Table 3-44 Use Case Description for Destroyer Rearm and Refueling

Sequence Diagram

Refer to Figure 3-7.

3.3.6 Cruiser Requirements

The Cruiser subsystem has the following five sub modules:

- Captain
- Navigation Officer
- Communication Officer
- Weapon Officer
- Weapon Launcher

3.3.6.1 Use Case Diagram

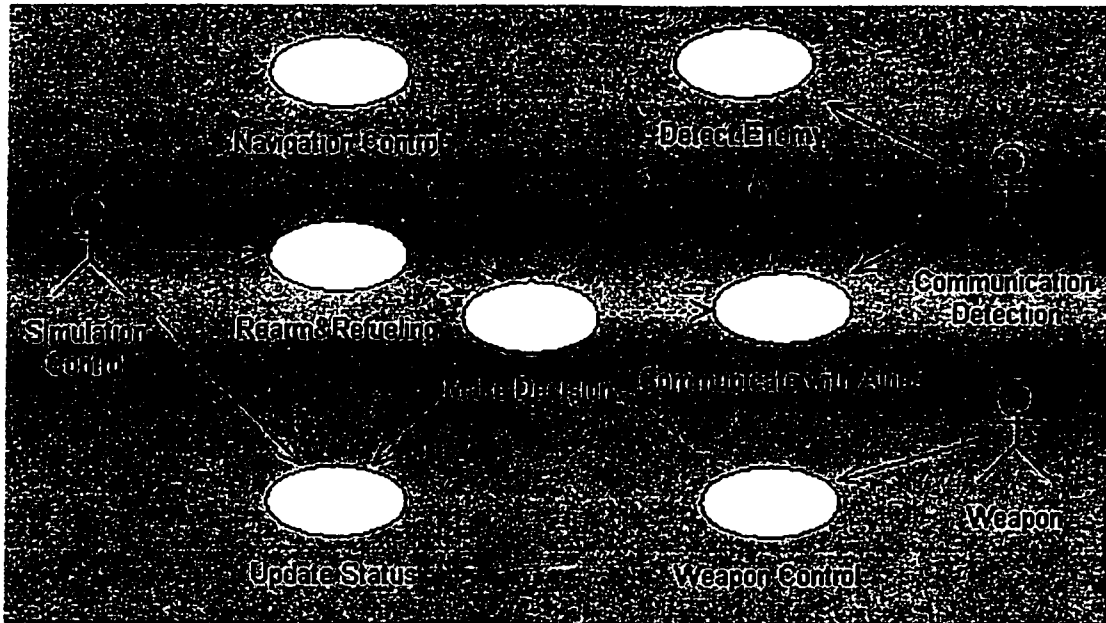


Figure 3-31 Use Case Diagram for Cruiser

3.3.6.2 Requirement Breakdown

Use Case: Cruiser Navigation Control

CS-001 Start/Stop Cruiser

CS-001-01 Start Cruiser

Cruiser shall start to move on the sea in random direction after its initiation.

No comments.

CS-001-02 Stop Cruiser

Cruiser shall be stopped by the user manually.

It is also stopped when its fuel is used up and base supplier has no more fuel.

CS-002 Accelerate/ Decelerate/ Rotate Cruiser

Cruiser shall accelerate, decelerate and rotate according to the Captain's command.

No comments.

CS-003 Control Steer Status

Cruiser shall turn on or turn off the steer in order to navigate on the sea.

No comments.

Use Case: Cruiser Detect Enemy

CS-004 Initialize Radar

When the Cruiser is created, a Radar object shall be initialized with location and radius.

No comments.

CS-005 Updating Radar Location

Cruiser's Radar location shall be updated by the Simulation Controller.

No comments.

CS-006 Control Radar Status

The Cruiser shall turn on or turn off the Radar at any time after Radar initialization.

Default status after Radar initialization is turn on.

CS-007 Receive Information from Radar

The Cruiser shall get the information about the nearing Aircrafts from its Radar.

Radar needs to get all the information from Simulation Controller.

Use Case: Cruiser Communication with Allies

CS-008 Initialize Radio

When the Cruiser is created, a Radio object shall be initialized with location and radius.

No comments

CS-009 Updating Radio Location

Cruiser's Radio location shall be updated by Simulation Controller.

No comments

CS-010 Control Radio Status

The Cruiser shall turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

CS-011 Receive Information from Radio

The Cruiser shall receive the report from its allies by its Radio.

Radio needs to get all the information from Simulation Controller.

CS-012 Send Information to Allies

The Cruiser can send information to its allies.

The significant information include newly detected enemies, the target it will attack, etc.

Use Case: Cruiser Make Decision

CS-013 Collect the Necessary Information from Radar and Radio.

This requirement is accomplished by CS-006, CS-011 and CS-012.

No comments.

CS-014 Analysis Information

Cruiser shall has the ability to analyze the received information to decide all the threats.

No comments.

CS-015 Decide Attack Object

Decide attack objects among threats based on the analyzed threats.

No comments.

CS-016 Decide Location to Conduct Ship

CS-017 Decide Content of Sending Information

The captain shall form the correct command and send them to navigation officer, Weapon officer and communication officer.

No comments.

CS-018 Decide Time for Sending Information

The captain shall decide the correct time to send the command to sub system.

No comments.

Use Case: Cruiser Weapon Control

CS-019 Select Number and Type of Weapon

Weapon officer shall decide the type and quantity of Weapon to be used on the Cruiser.

No comments.

CS-020 Initialize Weapon

Weapon officer will issue an order to Weapon launcher to create a Weapon.

No comments.

CS-021 Aim Object and Fire Weapon

Weapon object shall aim the target and fired by Weapon launcher.

Except the Heavy Cannon Shell, it is unguided after it is shot. It is not for Cruiser.

CS-022 Update the Number of Weapon

Weapon officer shall calculate and update the Weapon on board.

No comments.

CS-023 Recharge Weapon

When the Weapons are used up, the Cruiser shall go back to the battle base, and the Weapon office can reload the Weapon as needed type and quantity.

No comments.

Use Case: Cruiser Update Status

CS-024 Update Cruiser Location Periodically

Cruiser can update its location periodically and randomly if no threats are detected.

No comments.

- CS-025 Calculate Cruiser Resistance**
Cruiser shall calculate the resistance or hit points after each hit.
When hit points reaches zero the destroyer is considered inactive.
- CS-026 Cruiser Hit by Enemy Weapon**
Cruiser shall know when it is hit by the enemy's Weapon.
No comments.
- CS-027 Cruiser Recover Within Time Limit**
Cruiser can determine if it can recover from the hit points within the limited time.
No comments.
- CS-029 Report Status to SC Periodically**
Cruiser shall inform its status (location, alive/dead status) to Simulation Controller periodically.
No comments.
- CS-030 Cruiser Destroyed at Hit Points Limit**
Cruiser shall determine to be destroyed when exceed the hit points limit.
No comments.
- CS-031 Cruiser Crashed with other object**
Cruiser shall determine to self-destroyed when crash with other object.
When Cruiser hit by another ship.

Use Case: Cruiser Rearm and Refueling

- CS-032 Update the Fuel Level**
Cruiser shall reduce its fuel level according to the navigation time since its creation.
No comments.
- CS-033 Refueling the Gas**
Cruiser shall send request to its base supplying to refueling when its gas goes to the warning level.
No comments.
- CS-034 Rearm the Weapon**
Cruiser shall send the request to its base supplying once its Weapons are used up.
No comments.

3.3.6.3 Use Case Description

3.3.6.3.1 Use Case: Cruiser Navigation Control

Description		Provide the service to navigate the Cruiser
Priority		Must have this use case in order to move on the sea
Status		Detailed description and completed scenario
Action		NA
Pre-Conditions		1. Exist a Cruiser object; 2. A command is received from the navigation officer
Flow of Events	Base Path	Upon reception of the command from a navigation officer, the Cruiser may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate
	Alternate Path	NA
Post-Condition		The Cruiser is moved
Related Use Case	Used Use Case	Cruiser Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-45 Use Case Description for Cruiser Navigation Control

Sequence Diagram

Refer to Figure 3-1.

3.3.6.3.2 Use Case: Cruiser Detect Enemy

Description		Provide the service to locate the enemy using Radar
Priority		Must have this use case in order to detect the enemy
Status		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		Exist a Cruiser object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radar object with location and radius when Cruiser is created; 2. Update Radar location; 3. Turn on /off Radar; 4. Get enemy object information around the Cruiser
	Alternate Path	NA
Post-Condition		Any enemy in the range are detected
Related Use Case	Used Use Case	NA
	Extending Use Case	Detect Enemy
Other Requirements		NA

Table 3-46 Use Case Description for Cruiser Navigation Control

Sequence Diagram

Refer to Figure 3-2.

3.3.6.3.3 Use Case: Cruiser Communication with Allies

Description		Provide the communication service between Cruiser and its allies
Priority		Communication/Detect must have this use case in order to pass information to the Cruiser's allies
Status		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		Exist a Cruiser object
Flow of Events	Base Path	1. Initialize a Radio object with location and radius when Cruiser is created; 2. Update Radio location; 3. 3. Turn on /off Radio; 4. 4. Get object information around the Cruiser ; 5. Send message to its allies
	Alternate Path	NA
Post-Condition		The Cruiser received report from its allies, the Allies received report from Cruiser.
Relates Use Case	Uses Use Case	NA
Case	Extending Use Case	Communicate with Allies
Other Requirements		NA

Table 3-47 Use Case Description for Aircraft Carrier Communication with Allies

Sequence Diagram

Refer to Figure 3-3.

3.3.6.3.4 Use Case: Cruiser Make Decision

Description		Provide the service to analyze the report, decide attack target, and decide where to conduct the ship
Priority		Must have this use case in order to know its next action
Status		Detailed description and completed scenario
Action		Communication/Detection
Pre-Conditions		<ol style="list-style-type: none"> 1. Exist a Cruiser object; 2. The Cruiser's status is updated; 3. All the reports are received
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Upon reception of reports, the captain analyze the threats and decide attack target; 2. The captain gives the order to navigation officer for where to conduct the ship and at what speed; 3. The captain gives order to Weapon officer to prepare the attack; 4. The captain gives order to communication officer to send out the message ; 5. The Captain decide to rearm or refueling to send request to SC
	Alternate Paths	NA
Post-Condition		<ol style="list-style-type: none"> 1. The navigation officer executes captain's command 2. The Weapon office executes captain's command 3. The communication officer execute captain's command; 4. The Base Supplier perform the transaction task.
Related Use Case	Used Use Case	<ol style="list-style-type: none"> 1. Cruiser Update Status; 2. Cruiser Detect Enemy; 3. Cruiser Communication with Allies
	Extending Use Case	Make Decision
Other Requirements		NA

Table 3-48 Use Case Description for Cruiser Make Decision

Sequence Diagram

Refer to Figure 3-4.

3.3.6.3.5 Use Case: Cruiser Weapon Control

Description		Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed
Priority		Must have this use case in order to attack the enemy
Status		Detailed description and completed scenario
Actor		Weapon
Pre-conditions		An attacking command is received
Flow/Events	Base Path	<ol style="list-style-type: none"> 1. Decide the type and quantity of Weapon to be used; 2. Calculate and update the Weapon quantity on board 3. Issue an order to Weapon launcher 4. A Weapon object will be created by Weapon launcher 5. Weapon launcher will aim and fire Weapon 6. Deduct the Weapon on board
	Alternate Path	If the Weapon is Sea-Sea Missile, it will return a message stating whether the target is destroyed or not
Post-Condition		Weapon is fired and exploded.
Related Use Case	Uses Use Case	Cruiser Make Decision
	Extending Use Case	Weapon Control
Other Requirements		NA

Table 3-49 Use Case Description for Cruiser Weapon Control

Sequence Diagram

Refer to Figure 3-5.

3.3.6.3.6 Use Case: Cruiser Update Status

Description		Provide the service to update Cruiser's location and other status (alive/dead)
Priority		Must have this use case in order to report status to SC
Status		Detailed description and completed scenario
Actor		Simulation Controller
Pre-Conditions		Exist a Cruiser object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Update the location of the Cruiser 2. Determine if the Cruiser is hit by Weapon 3. Get the hit points of the Cruiser 4. Determine if the Cruiser can recover from the hit points 5. Determine if the Cruiser is destroyed 6. Determine if the Cruiser crashes with other object
	Alternate Path	NA
Post-Condition		The status of the Cruiser is updated
Related Use Case	Used Use Case	NA
	Depending Use Case	Update Status
Other Requirements		NA

Table 3-50 Use Case Description for Cruiser Update Status

Sequence Diagram

Refer to Figure 3-6.

3.3.6.3.7 Use Case: Cruiser Rearm and Refueling

Description		Provide the service to rearm and refueling the Cruiser
Priority		Would like to have this use case in order to continue moving on the sea
Status		Detailed description and completed scenario
Actor		1. Simulation Controller; 2. Cruiser; 3. Radio.
Pre-Conditions		1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.
Flow of Events	Base Path	1. Navigation Officer send information to ask captain to deduct the fuel; Weapon Officer send information to Captain to deduct the Weapon; 2. Captain check if the fuel is at limited level; Captain check if the Weapon is used up ; 3. Captain send request to SC to ask base supplier to refuel; Captain send request to SC to ask base supplier to create Weapon; 4. Base Supplier transfer the fuel or Weapon to Aircraft Carrier.
	Alternate Path	NA
Post Condition		The Cruiser get rearm and refueling
Related Use Case	Used Use Case	NA
	Extending Use Case	Rearm and Refueling
Other Requirements		NA

Table 3-51 Use Case Description for Cruiser Rearm and Refueling

Sequence Diagram

Refer to Figure 3-7.

3.3.7 Battleship Requirements

The Battleship subsystem has the following five sub modules:

- Captain
- Navigation Officer
- Communication Officer
- Weapon Officer
- Weapon Launcher

3.3.7.1 Use Case Diagram

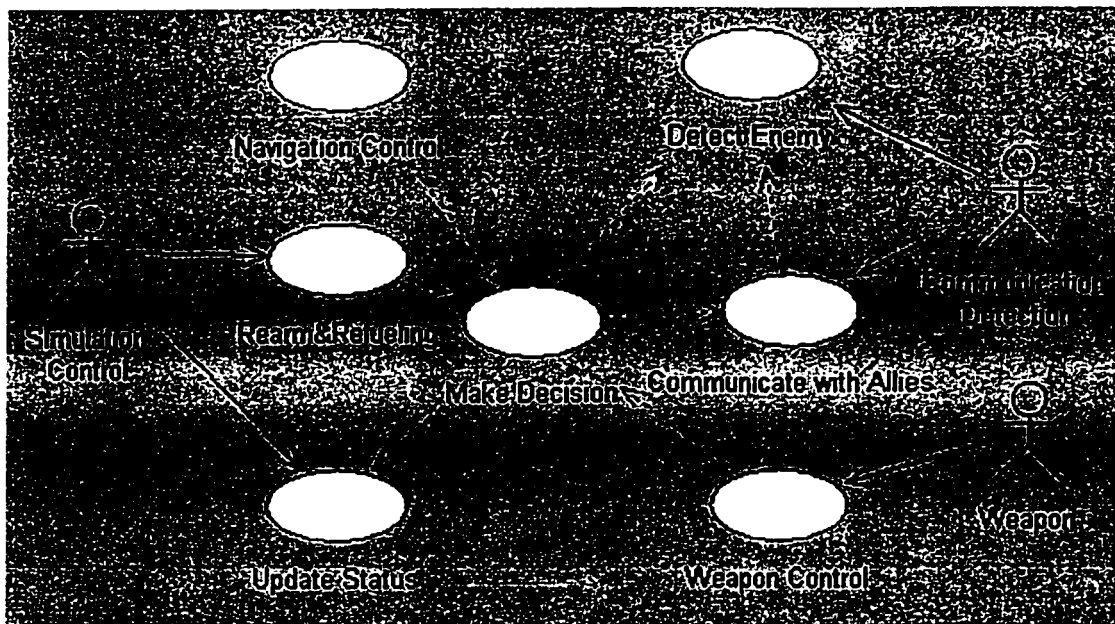


Figure 3-32 Use Case Diagram for Battleship

3.3.7.2 Requirement Breakdown

Use Case: Battleship Navigation Control

BS-001 Start/Stop Battleship

BS-001-01 Start Battleship

Battleship shall start to move on the sea in random direction after its initiation.

No comments.

BS-001-02 Stop Battleship

Battleship shall be stopped by the user manually.

It is also stopped when its fuel is used up and base supplier has no more fuel.

BS-002 Accelerate/ Decelerate/ Rotate Battleship

Battleship shall accelerate, decelerate and rotate according to the Captain's command.

No comments.

BS-003 Control Steer Status

Battleship shall turn on or turn off the steer in order to navigate on the sea.

No comments.

Use Case: Battleship Detect Enemy

BS-004 Initialize Radar

when the Battleship is created, a Radar object shall be initialized with location and radius.

No comments.

BS-005 Updating Radar Location

Battleship's Radar location shall be updated by Simulation Controller.

No comments.

BS-006 Control Radar Status

The Battleship shall turn on or turn off the Radar at any time after Radar initialization.

Default status after Radar initialization is turn on.

BS-007 Receive Information from Radar

The Battleship shall get the information about the surrounding objects from its Radar.

Radar needs to get all the information from Simulation Controller.

Use Case: Battleship Communication with Allies

BS-008 Initialize Radio

when the Battleship is created, a Radio object shall be initialized with location and radius.

No comments

BS-009 Updating Radio Location

Battleship's Radio location shall be updated by Simulation Controller.

No comments

BS-010 Control Radio Status

The Battleship shall turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

BS-011 Receive Information from Radio

The Battleship shall receive the report from its allies by its Radio.

Radio needs to get all the information from Simulation Controller.

BS-012 Send Information to Allies

The Battleship can send information to its allies.

The significant information include newly detected enemies, the target it will attack, etc.

Use Case: Battleship Make Decision

BS-013 Collect the Necessary Information from Radar and Radio.

This requirement is accomplished by BS-006, BS-011 and BS-012.

No comments.

BS-014 Analysis Information

Battleship shall has the ability to analyze the received information to decide all the threats.

No comments.

BS-015 Decide Attack Object

Decide attack objects among threats based on the analyzed threats.

No comments.

BS-016 Decide Location to Conduct Ship

BS-017 Decide Content of Sending Information

The captain shall form the correct command and send them to navigation officer, Weapon officer and communication officer.

No comments.

BS-018 Decide Time for Sending Information

The captain shall decide the correct time to send the command to sub system.

No comments.

Use Case: Battleship Weapon Control

BS-019 Select Number and Type of Weapon

Weapon officer shall decide the type and quantity of Weapon to be used on the Battleship.

No comments.

BS-020 Initialize Weapon

Weapon officer will issue an order to Weapon launcher to create a Weapon.

No comments.

BS-021 Aim Object and Fire Weapon

Weapon object shall aim the target and fired by Weapon launcher.
Except the Heavy Cannon Shell, it is unguided after it is shot. It is for Battleship.

BS-022 Update the Number of Weapon

Weapon officer shall calculate and update the Weapon on board.

No comments.

BS-023 Recharge Weapon

When the Weapons are used up, the Battleship shall go back to the battle base, and the Weapon office can reload the Weapon as needed type and quantity.

No comments.

Use Case: Battleship Update Status

BS-024 Update Battleship Location Periodically

Battleship can update its location periodically and randomly if no threats are detected.

No comments.

- BS-025 Calculate Battleship Resistance**
Battleship shall calculate the resistance or hit points after each hit.
No comments.
- BS-026 Battleship Hit by Enemy Weapon**
Battleship shall know when it is hit by the enemy's Weapon.
No comments.
- BS-027 Battleship Recover Within Time Limit**
Battleship can determine if it can recover from the hit points within the limited time.
No comments.
- BS-029 Report Status to SC Periodically**
Battleship shall inform its status (location, alive/dead status) to Simulation Controller periodically.
No comments.
- BS-030 Battleship Destroyed at Hit Points Limit**
Battleship shall determine to be destroyed when exceed the hit points limit.
No comments.
- BS-031 Battleship Crashed with other object**
Battleship shall determine to be destroyed when crash with other object.
No comments.

Use Case: Battleship Rearm and Refueling

- BS-032 Update the Fuel Level**
Battleship shall reduce its fuel level according to the navigation time since its creation.
No comments.
- BS-033 Refueling the Gas**
Battleship shall send request to its base supplying to refueling when its gas goes to the warning level.
- BS-034 Rearm the Weapon**
Battleship shall send the request to its base supplying once its Weapons are used up.
Actually, the Weapon are created by Battleship when they are launched, only after the fired Weapon exceed the limits, the base supplying will create Weapon for Battleship and transfer them to Battleship.

3.3.7.3 Use Case Description

3.3.7.3.1 Use Case: Battleship Navigation Control

Description		Provide the service to navigate the Battleship
Priority		Must have this use case in order to move on the sea
Status		Detailed description and completed scenario
Actor		NA
Pre-Conditions		1. Exist a Battleship object; 2. A command is received from the navigation officer
Flow of Events	Base Path	Upon reception of the command from a navigation officer, the battle ship may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate
	Alternate Path	NA
Post-Condition		The Battleship is moved
Related Use Case	Used Use Case	Battleship Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-52 Use Case Description for Battleship Navigation Control

Sequence Diagram

Refer to Figure 3-1.

3.3.7.3.2 Use Case: Battleship Detect Enemy

Description		Provide the service to locate the enemy using Radar
Priority		Must have this use case in order to detect the enemy
Status		Detailed description and completed scenario
Asio		Communication/Detection
Pre-Conditions		Exist a Battleship object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radar object with location and radius when Battleship is created; 2. Update Radar location; 3. Turn on /off Radar; 4. Get object information around the Battleship
	Alternate Path	NA
Post-Condition		Any enemy in the range are detected
Related Use Case	Used Use Case	NA
	Extending Use Case	Detect Enemy
Other Requirements		NA

Table 3-53 Use Case Description for Battleship Navigation Control

Sequence Diagram

Refer to Figure 3-2.

3.3.7.3.3 Use Case: Battleship Communication with Allies

Description		Provide the communication service between Battleship and its allies
Priority		Communication/Detect must have this use case in order to pass information to the Battleship's allies
Status		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		Exist a Battleship object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radio object with location and radius when Battleship is created; 2. Update Radio location; 3. Turn on /off Radio; 4. Get object information around the Battleship; 5. Send message to its allies.
	Alternate Path	NA
Post-Condition		The Battleship received report from its allies, the Allies received report from Battleship
Related Use Case	Base Use Case	NA
Case	Extending Use Case	Communication with Allies
Other Requirements		NA

Table 3-54 Use Case Description for Battleship Communication with Allies

Sequence Diagram

Refer to Figure 3-3.

3.3.7.3.4 Use Case: Battleship Make Decision

Description		Provide the service to analyze the report, decide attack target, and decide where to conduct the ship
Priority		Must have this use case in order to know its next action
Starts		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		<ol style="list-style-type: none"> 1. Exist a Battleship object; 2. The Battleship's status is updated; 3. All the reports are received
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Upon reception of reports, the captain analyze the threats and decide attack target; 2. The captain gives the order to navigation officer for where to conduct the ship and at what speed; 3. The captain gives order to Weapon officer to prepare the attack; 4. The captain gives order to communication officer to send out the message ; 5. The Captain decide to rearm or refueling to send request to SC.
	Alternate Path	NA
Post-Condition		<ol style="list-style-type: none"> 1. The navigation officer executes captain's command; 2. The Weapon office executes captain's command; 3. The communication officer execute captain's command; 4. The Base Supplier perform the transaction task.
Related Use Case	Used Use Case	<ol style="list-style-type: none"> 1. Battleship Update Status; 2. Battleship Detect Enemy; 3. Battleship Communication with Allies
	Extending Use Case	Make Decision
Other Requirements		NA

Table 3-55 Use Case Description for Battleship Make Decision

Sequence Diagram

Refer to Figure 3-4.

3.3.7.3.5 Use Case: Battleship Weapon Control

Description		Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed
Priority		Must have this use case in order to attack the enemy
State		Detailed description and completed scenario
Actor		Weapon
Pre-Conditions		An attacking command is received
Flow Events	Base Path	<ol style="list-style-type: none"> 1. Decide the type and quantity of Weapon to be used; 2. Calculate and update the Weapon quantity on board; 3. Issue an order to Weapon launcher; 4. A Weapon object will be created and fired by Weapon launcher; 5. Weapon launcher will aim and fire Weapon; 6. When Weapons are used up, recharge the Weapon on board.
	Alternate Path	If the Weapon is Sea-Sea Missile, it will return a message stating whether the target is destroyed or not
Post-Conditions		Weapon is fired and exploded.
Related Use Case	Use Case	Battleship Make Decision
Case	Extending Use Case	Weapon Control
Other Requirements		NA

Table 3-56 Use Case for Weapon Control

Sequence Diagram

Refer to Figure 3-5.

3.3.7.3.6 Use Case: Battleship Update Status

Description:		Provide the service to update Battleship's location and other status (alive/dead)
Priority:		Must have this use case in order to report status to SC
Status:		Detailed description and completed scenario
Actor:		Simulation Controller
Pre-Conditions:		Exist a Battleship object
Flow of Events:	Base Path:	<ol style="list-style-type: none"> 1. Update the location of the Battleship 2. Determine if the Battleship is hit by Weapon 3. Get the hit points of the Battleship 4. Determine if the Battleship can recover from the hit points 5. Determine if the Battleship is destroyed 6. Determine if the Battleship crashes with other object
	Alternate Path:	NA
Post-Condition:		The status of the Battleship is updated
Related Use Case:	Used Use Case:	NA
	Extending Use Case:	Update Status
Other Requirements:		NA

Table 3-57 Use Case Description for Battleship Update Status

Sequence Diagram

Refer to Figure 3-6.

3.3.7.3.7 Use Case: Battleship Rearm and Refueling

Description		Provide the service to rearm and refueling the Battleship
Priority		Would like to have this use case in order to continue moving on the sea
Status		Detailed description and completed scenario
Actor		1. Simulation Controller; 2. Battleship; 3. Radio.
Pre-Conditions		1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.
Flow of Events	Base Path	1. Navigation Officer send information to ask captain to deduct the fuel; Weapon Officer send information to Captain to deduct the Weapon; 2. Captain check if the fuel is at limited level; Captain check if the Weapon is used up ; 3. Captain send request to SC to ask base supplier to refuel; Captain send request to SC to ask base supplier to create Weapon; 4. Base Supplier transfer the fuel or Weapon to Aircraft Carrier.
	Alternate Path	NA
Post Condition		The Battleship get rearm and refueling
Related Use Case	Use Case	NA
	Extending Use Case	Rearm and Refueling
Other Requirements		NA

Table 3-58 Use Case Description for Battleship Rearm and Refueling

Sequence Diagram

Refer to Figure 3-7.

3.3.8 Submarine Requirements

The Submarine subsystem has the following five sub modules:

- Captain
- Navigation Officer
- Communication Officer
- Weapon Officer
- Weapon Launcher

3.3.8.1 Use Case Diagram

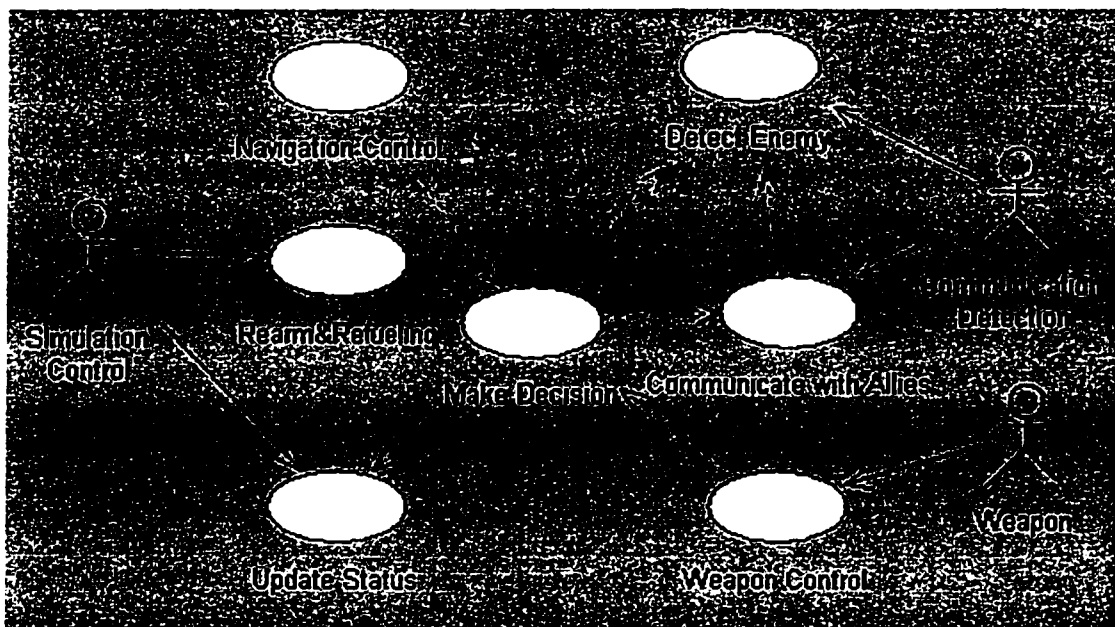


Figure 3-33 Use Case Diagram for Submarine

3.3.8.2 Requirement Breakdown

Use Case: Submarine Navigation Control

SM-001 Start/Stop Submarine

SM-001-01 Start Submarine

Submarine shall start to move(float\sink) in the sea with random direction after its initiation.

No comments.

SM-001-02 Stop Submarine

Submarine shall be stoppable by the user manually.

It is also stopped when its fuel is used up and base supplier has no more fuel.

SM-002 Accelerate/ Decelerate/ Rotate Submarine

Submarine shall accelerate, decelerate and rotate according to the Captain's command.

No comments.

SM-003 Control Steer Status

Submarine shall turn on or turn off the steer in order to navigate on the sea.

No comments.

Use Case: Submarine Detect Enemy

SM-004 Initialize Radar

when the Submarine is created, a Radar object shall be initialized with location and radius.

No comments.

SM-005 Updating Radar Location

Submarine's Radar location shall be updated by Simulation Controller.

No comments.

SM-006 Control Radar Status

The Submarine shall turn on or turn off the Radar at any time after Radar initialization.

Default status after Radar initialization is turn on.

SM-007 Receive Information from Sonar

The Submarine shall get the information about the surrounding enemy ships and Submarines from its Sonar.

Radar needs to get all the information from Simulation Controller.

Use Case: Submarine Communication with Allies

SM-008 Initialize Radio

When the Submarine is created, a Radio object shall be initialized with location and radius.

No comments

SM-009 Updating Radio Location

Submarine's Radio location shall be updated by Simulation Controller.

No comments

SM-010 Control Radio Status

The Submarine shall turn on or turn off the Radio at any time after Radio initialization.

Default status after Radio initialization is turn on.

SM-011 Receive Information from Radio

The Submarine shall receive the report from its allies by its Radio.

Radio needs to get all the information from Simulation Controller.

SM-012 Send Information to Allies

The Submarine can send information to its allies.

The significant information include newly detected enemies, the target it will attack, etc.

Use Case: Submarine Make Decision

SM-013 Collect the Necessary Information from Radar and Radio.

Refer to requirements SM-006, SM-011 and SM-012.

No comments.

SM-014 Analysis Information

Submarine shall has the ability to analyze the received information to decide all the threats.

No comments.

SM-015 Decide Attack Object

Decide attack objects among threats based on the analyzed threats.

No comments.

SM-016 Decide Location to Conduct Ship

SM-017 Decide Content of Sending Information

The captain shall form the correct command and send them to navigation officer, Weapon officer and communication officer.

No comments.

SM-018 Decide Time for Sending Information

The captain shall decide the correct time to send the command to sub system.

No comments.

Use Case: Submarine Weapon Control

SM-019 Select Number and Type of Weapon

Weapon officer shall decide the type and quantity of Weapon to be used on the Submarine.

No comments.

SM-020 Initialize Weapon

Weapon officer will issue an order to Weapon launcher to create a Weapon.

No comments.

SM-021 Aim Object and Fire Weapon

Weapon object shall aim the target and fired by Weapon launcher.

Except the Heavy Cannon Shell, it is unguided after it is shot. It is not for Submarine.

SM-022 Update the Number of Weapon

Weapon officer shall calculate and update the Weapon on board.

No comments.

SM-023 Recharge Weapon

When the Weapons are used up, the Submarine shall go back to the battle base, and the Weapon office can reload the Weapon as needed type and quantity.

No comments.

Use Case: Submarine Update Status

SM-024 Update Submarine Location Periodically

Submarine can update its location periodically and randomly if no threats are detected.

No comments.

SM-025 Calculate Submarine Resistance
Submarine shall calculate the resistance or hit points after each hit.
No comments.

SM-026 Submarine Hit by Enemy Weapon
Submarine shall know when it is hit by the enemy's Weapon.
No comments.

SM-027 Submarine Recover Within Time Limit
Submarine can determine if it can recover from the hit points within the limited time.
No comments.

SM-029 Report Status to SM Periodically
Submarine shall inform its status (location, alive/dead status) to Simulation Controller periodically.
No comments.

SM-030 Submarine Destroyed at Hit Points Limit
Submarine shall determine to be destroyed when exceed the hit points limit.
No comments.

SM-031 Submarine Crashed with other object
Submarine shall determine to be destroyed when crash with other object.
When Submarine hit by another ship.

Use Case: Submarine Rearm and Refueling

SM-032 Update the Fuel Level
Submarine shall reduce its fuel level according to the navigation time since its creation.
No comments.

SM-033 Refueling the Gas
Submarine shall send request to its base supplying to refueling when its gas goes to the warning level.
No comments.

SM-034 Rearm the Weapon
Submarine shall send the request to its base supplying once its Weapons are used up.
No comments.

3.3.8.3 Use Case Description

3.3.8.3.1 Use Case: Submarine Navigation Control

Description		Provide the service to navigate the Submarine
Priority		Must have this use case in order to move on the sea
Status		Detailed description and completed scenario
Action		NA
Pre-Conditions		1. Exist a Submarine object; 2. A command is received from the navigation officer
Flow of Events	Base Path	Upon reception of the command from a navigation officer, the battle ship may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate
	Alternate Path	NA
Post-condition		The Submarine is moved
Related Use Case	Used Use Case	Submarine Make Decision
	Extending Use Case	Navigation Control
Other Requirements		NA

Table 3-59 Use Case Description for Submarine Navigation Control

Sequence Diagram

Refer to Figure 3-1.

3.3.8.3.2 Use Case: Submarine Detect Enemy

Description		Provide the service to locate the enemy using Radar
Priority		Must have this use case in order to detect the enemy
Status		Detailed description and completed scenario
Use Case		Communication/Detection
Preconditions		Exist a Submarine object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radar object with location and radius when Submarine is created; 2. Update Radar location; 3. Turn on /off Radar; 4. Get object information around the Submarine
	Alternate Path	NA
Post Condition		Any enemy in the range are detected
Related Use Case	Used Use Case	NA
	Extending Use Case	Detect Enemy
Other Requirements		NA

Table 3-60 Use Case Description for Submarine Detect Enemy

Sequence Diagram

Refer to Figure 3-2.

3.3.8.3.3 Use Case: Submarine Communicate with Allies

Description		Provide the communication service between Submarine and its allies
Priority		Communication/Detect must have this use case in order to pass information to the Submarine's allies
Status		Detailed description and completed scenario
Actor		Communication/Detection
Pre-Conditions		Exist a Submarine object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Initialize a Radio object with location and radius when Submarine is created; 2. Update Radio location; 3. Turn on /off Radio; 4. Get object information around the Submarine; 5. Send message to its allies
	Alternate Path	NA
PostCondition		The Submarine received report from its allies, the Allies received report from Submarine
Related Use Case	Used Use Case	NA
	Extending Use Case	Communication with Allies
Other Requirements		NA

Table 3-61 Use Case Description for Submarine Communicate with Allies

Sequence Diagram

Refer to Figure 3-3.

3.3.8.3.4 Use Case: Submarine Make Decision

Description		Provide the service to analyze the report, decide attack target, and decide where to conduct the ship
Priority		Must have this use case in order to know its next action
Status		Detailed description and completed scenario
Tag		Communication/Detection
Pre-conditions		<ol style="list-style-type: none"> 1. Exist a Submarine object; 2. The Submarine's status is updated; 3. All the reports are received
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Upon reception of reports, the captain analyze the threats and decide attack target; 2. The captain gives the order to navigation officer for where to conduct the ship and at what speed; 3. The captain gives order to Weapon officer to prepare the attack; 4. The captain gives order to communication officer to send out the message ; 5. The Captain decide to rearm or refueling to send request to SC.
	Alternate Path	NA
Post-Condition		<ol style="list-style-type: none"> 1. The navigation officer executes captain's command; 2. The Weapon office executes captain's command; 3. The communication officer execute captain's command; 4. The Base Supplier perform the transaction task.
Related Use Case	Used Use Case	<ol style="list-style-type: none"> 1. Submarine Update Status; 2. Submarine Detect Enemy; 3. Submarine Communication with Allies
	Extending Use Case	Make Decision
Other Requirements		NA

Table 3-62 Use Case Description for Submarine Make Decision

Sequence Diagram

Refer to Figure 3-4.

3.3.8.3.5 Use Case: Submarine Weapon Control

Description:		Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed
Priority Status:		Must have this use case in order to attack the enemy
ACG:		Weapon
Pre-Conditions:		An attacking command is received
Flow of Events:	Base Path:	<ol style="list-style-type: none"> 1. Decide the type and quantity of Weapon to be used; 2. Calculate and update the Weapon quantity on board 3. Issue an order to Weapon launcher 4. A Weapon object will be created and fired by Weapon launcher 5. Weapon launcher will aim and fire Weapon 6. When Weapons are used up, recharge the Weapon on board
	Alternate Path:	If the Weapon is Sea-Sea Missile, it will return a message stating whether the target is destroyed or not
Post-Condition:		Weapon is fired and exploded.
Related Use Case:		Submarine Make Decision
Use Case:	Extending Use Case:	Weapon Control
Other Requirements:		NA

Table 3-63 Use Case description for Submarine Weapon Control

Sequence Diagram

Refer to Figure 3-5.

3.3.8.3.6 Use Case: Submarine Update Status

Description		Provide the service to update Submarine's location and other status (alive/dead)
Priority		Must have this use case in order to report status to SC
Status		Detailed description and completed scenario
Actor		Simulation Controller
Pre-Conditions		Exist a Submarine object
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. Update the location of the Submarine 2. Determine if the Submarine is hit by Weapon 3. Get the hit points of the Submarine 4. Determine if the Submarine can recover from the hit points 5. Determine if the Submarine is destroyed 6. Determine if the Submarine crashes with other object
	Alternate Path	NA
Post-Condition		The status of the Submarine is updated
Related Use Case	Used Use Case	NA
	Extending Use Case	Update Status
Other Requirements		NA

Table 3-64 Use Case Description for Submarine Update Status

Sequence Diagram

Refer to Figure 3-6.

3.3.8.3.7 Use Case: Submarine Rearm and Refueling

Description:		Provide the service to rearm and refueling the Submarine
Priority:		Would like to have this use case in order to continue moving on the sea
Status:		Detailed description and completed scenario
Actor:		<ol style="list-style-type: none"> 1. Simulation Controller; 2. Submarine; 3. Radio.
Pre-Conditions:		<ol style="list-style-type: none"> 1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.
Flow of Events:	Base Path:	<ol style="list-style-type: none"> 1. Navigation Officer send information to ask captain to deduct the fuel; Weapon Officer send information to Captain to deduct the Weapon; 2. Captain check if the fuel is at limited level; Captain check if the Weapon is used up ; 3. Captain send request to SC to ask base supplier to refuel; Captain send request to SC to ask base supplier to create Weapon; 4. Base Supplier transfer the fuel or Weapon to Submarine
	Alternate Path:	NA
Post-Condition:		The Submarine get rearm and refueling
Related Use Case:	Used Use Case:	NA
	Extending Use Case:	NA
Other Requirements:		NA

Table 3-65 Use Case Description for Submarine Rearm and Refueling

Sequence Diagram

Refer to Figure 3-7.

3.3.9 Weapons Requirements

The Weapons subsystem has the following four sub modules:

- Weapon (Carried Weapon)
- Controller
- Rudder
- Charger

Weapons can be classified into the following eight types:

- Sea-Sub Missile (Carrying Torpedo)
- Sea-Air Missile
- Heavy Cannon Shell
- Sea-Sea Missile
- Torpedo
- Sub-Sea Torpedo (Carrying Missile)
- Air-Sea Missile
- Air-Air Missile

More Weapon types may be added when the NBSS6 needs to extend its functionality.

3.3.9.1 Use Case Diagram

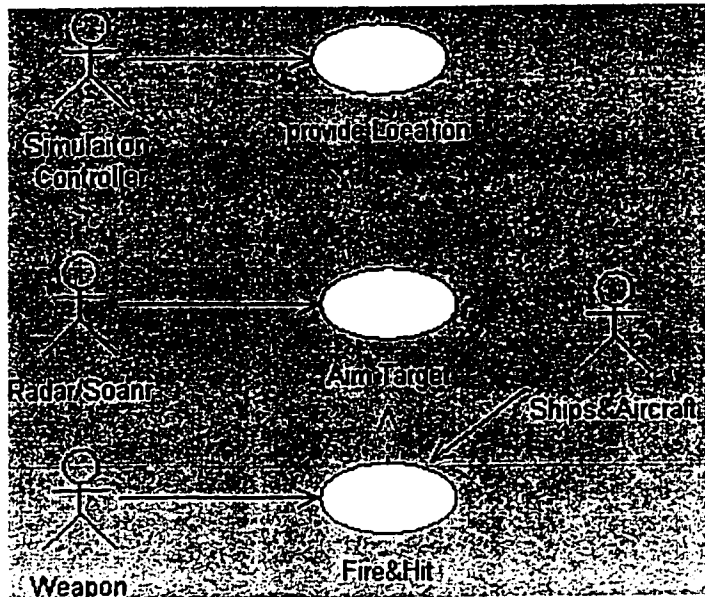


Figure 3-34 Use Case Diagram for Weapon

3.3.9.2 Requirement Breakdown

Use Case: Provide Location

WP-001 Report Position to SC

The Weapon shall report its position to SC periodically.

No comments.

Use Case: Aim Target

WP-002 Target Tracing via Radar or Sonar

The Weapon except the Cannon Shell, shall aim and trace the target by its Radar or Sonar.

The Radar and Sonar act as simulation for Weapon detection device.

WP-003 Trajectory Control

The Cannon Shells shall be shot according to ballistic.

No comments.

WP-004 Steering Weapon

The Weapon except the Cannon Shells can be steered after shot.

No comments.

Use Case: Fire and Hit target

WP-005 Fire Itself

The Weapon shall fire itself after receiving a command from Weapon launcher.

No comments.

WP-006 Detonate

The Weapon should signal and transfer the power to the target when the target is hit.

No comments.

WP-007 Inform the Hit Target

The Weapon shall inform the target that has been hit by it.

No comments.

WP-008 Inform the Owner

Once the Weapon detonated itself, the Weapon will send a message to its owner it has been exploded.

3.3.9.3 Use Case Description

3.3.9.3.1 Use Case: Provide Location

Description		Provide the Weapon location to SC
Priority		Must have this use case in order to aim the target.
Status		Detailed description and completed scenario
Actor		1. All the Weapon; 2. Simulation Controller.
Pre-Conditions		Weapon knows its location
Flow of Events	Base Path	1. When Weapon will be launched, report its location to SC; 2. When Weapon is fired, provide the updated location to SC periodically.
	Alternate Path	NA
Post-Condition		SC get the Weapon 's location
Related Use Case	Used Use Case	NA
	Extending Use Case	NA
Other Requirements		NA

Table 3-66 Use Case Description for Provide Location

Sequence Diagram

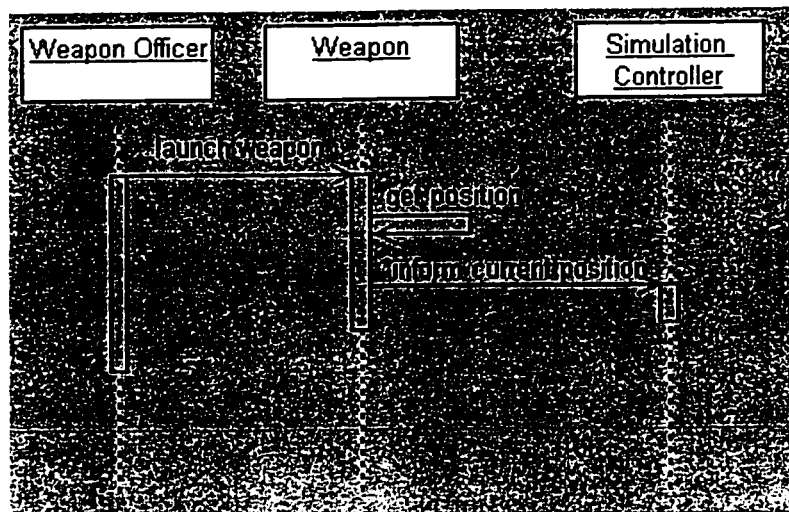


Figure 3-35 Sequence Diagram for Use Case Weapon Provide Location

3.3.9.3.2 Use Case: Aim Target

Description		Provide a service to trace the target location
Priority		Must have this use case in order to aim the target
Status		Detailed description and completed scenario
Accept		<ol style="list-style-type: none"> 1. All the Weapon (Heavy Cannon Shell, Sea-Sub Missile and Sub-Sea Missile will based on ballistic to aim the target); 2. Communication/Detection; 3. Target Objet.
Pre-Conditions		Detected targets are within the Radar or Sonar' s range
Flow of Events	Base Path	<ol style="list-style-type: none"> 1. The Weapon's owner detect the target and launch the Weapon; 2. Weapon use its detection system to trace the location of the nearest target;
	Alternate Path	NA
Post-Condition		The location of target has been traced by Weapon
Related Use Case	User Use Case	Provide Location
	Extending Use Case	NA
Other Requirements		NA

Table 3-67 Use Case Description for Aim Target

Sequence Diagram

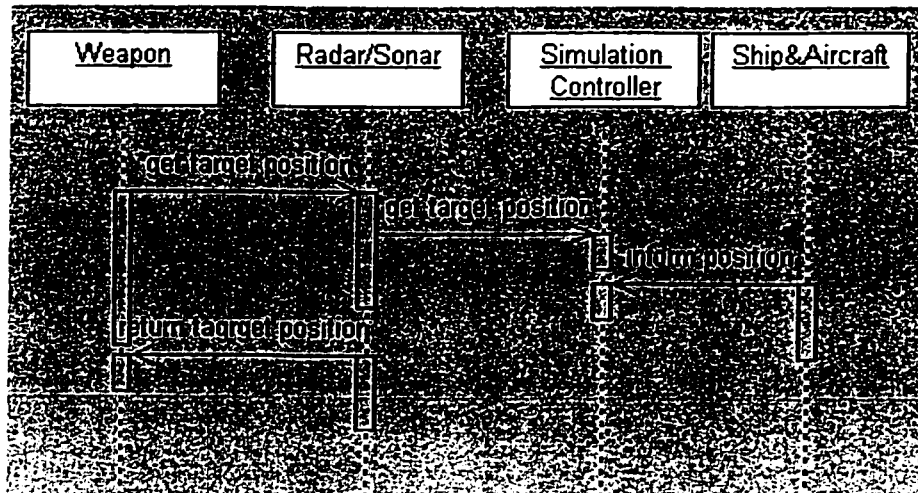


Figure 3-36 Sequence Diagram for Use Case Weapon Aim Target

3.3.9.3.3 Use Case: Fire and Hit Target

Description		Provide the service for Weapon to be fired and hit the target.
Priority		Must have this use case in order to hit the target
Scope		Detailed description and completed scenario
Actor		1. All the Weapon; 2. Target Object.
Pre-Conditions		Weapon is launched and prepared to fire.
Flow of Events	Base Path	1. Weapon is launched by the Weapon's owner; 2. Weapon is fired and detonated when it hit the target; 3. Weapon inform the hit target to reduce its resistance
	Alternate Path	NA
Post-Condition		The Weapon is fired and target has been hit.
Related Use Case	Used Use Case	Aim Target
	Extending Use Case	NA
Other Requirements		NA

Table 3-68 Use Case Description for Fire and Hit Target

Sequence Diagram

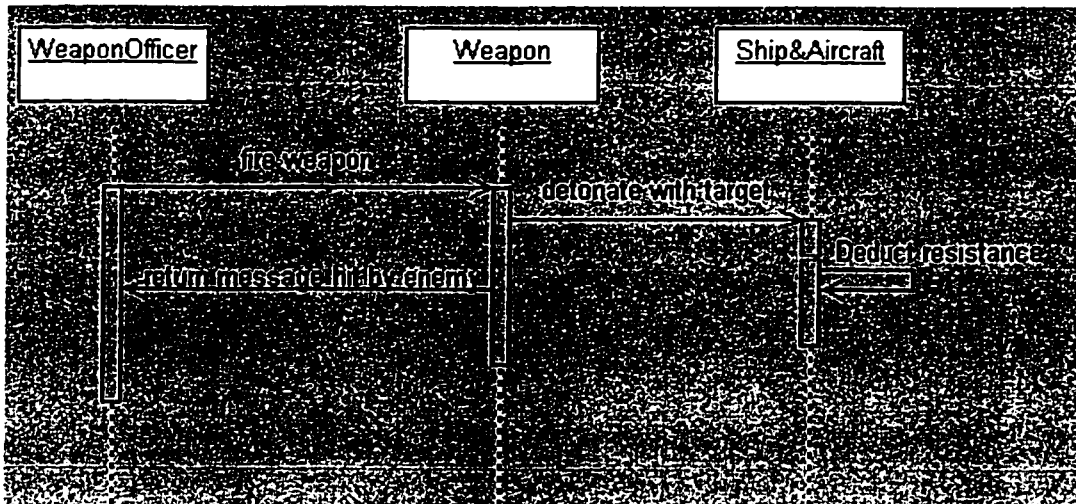


Figure 3-37 Sequence Diagram for Use Case Weapon Fire And Hit Target

3.4 External Interface Requirements

Each subsystem need the external interface to provide the service and use the services provided by other subsystem. The detailed interface requirements are listed in this section to accomplish a successful design goal.

3.4.1 User Interface

The only user interface is provided by the Simulation Controller subsystem. All the other subsystem has no direct user interface.

3.4.2 Hardware Interface

The software is supported by personal computer equipped with a keyboard and a mouse.

3.4.3 Software Interface

The software interface is outlined as following. The detailed software interface will be addressed in the software design section.

Ship and Aircraft vs. Simulation Controller

A) Ship and Aircraft provide to Simulation Controller

- Constructor to create the ship or Aircraft object
- Ship or Aircraft current location and alive/dead status

B) Simulation Controller provide to Ship and Aircraft for initialize ship and Aircraft

- Initial location, direction
- Initial speed
- Initial quantity of fuel
- Blue/Red flag
- Object ID
- On board quantity of Weapons(Sea-Sea Missile and heave cannons)
- For Aircraft Carrier, on board quantity of Aircraft.
- Base supplier responds to fuel and Weapon request during the simulation is running.

Ship and Aircraft vs. Communication/Detection

A) Ship and Aircraft provide to Communication/Detection

- Create and initialize Radar/Sonar object
- Update Radar/Sonar location
- Create and initialize Radio object
- Update Radio location
- Prepare information to be sent

B) Communication/Detection provide to Ship and Aircraft

- Radar provide the location, speed, direction of all objects detected around the ship
- Distinguish the enemy or friend
- Emit and receive wave function
- Radio send report to friends
- Receive report from friends

Ship and Aircraft vs. Weapon

A) Ship and Aircraft provide to Weapon

- Initialize the Weapon object
- Target location
- Initialize location, speed and direction of heavy cannon.

B) Weapon provide to Ship and Aircraft

- Fire Weapon function
- Inform the ship and Aircraft when they are hit
- Trace the target(Sea-Sea Missile)

3.4.4 Communication Interface

NA

3.5 Performance Requirements

This software is designed for single user and single terminal. Simulation controller will set up a time limits and a terminated condition. User starts the simulation program and input all the parameters required, simulation will start

and run by itself. When the simulation reach its time limit or the terminate condition is satisfied, this simulation will be terminated.

3.6 Design Constraints

The design is based on personal computer with Microsoft Windows 95/98/NT2000. The language to implement this design is Visual C++. Since each subsystem of NBSS need to co-operate with each other to accomplish the whole system function, it is extremely important that the connection between the interfaces of subsystems is well designed.

3.7 Quality Attributes

All the functional requirement will be tested to insure the quality of the software. Software documentation will be supplied to insure the good learnability and maintainability.

3.8 Other Requirements

NA

4. Software Design

4.1 Decomposition Description

This section describes partition of the system into design entities, the way the system has been structured, the purpose and the function of each entity. The main criteria and methods for entity decomposition is information hiding, which means the module's interface of definition was chosen to reveal as little as possible about its inner workings. [1]&[8].

4.1.1 Module Decomposition

The Naval Battle Simulation System consists of nine subsystems: Simulation Controller, Communication/Detection, Weapons, Aircraft Carrier, Aircraft, Destroyer, Submarine, Cruiser and Battleship. In the following figure, MFC and OpenGL are external library of system.

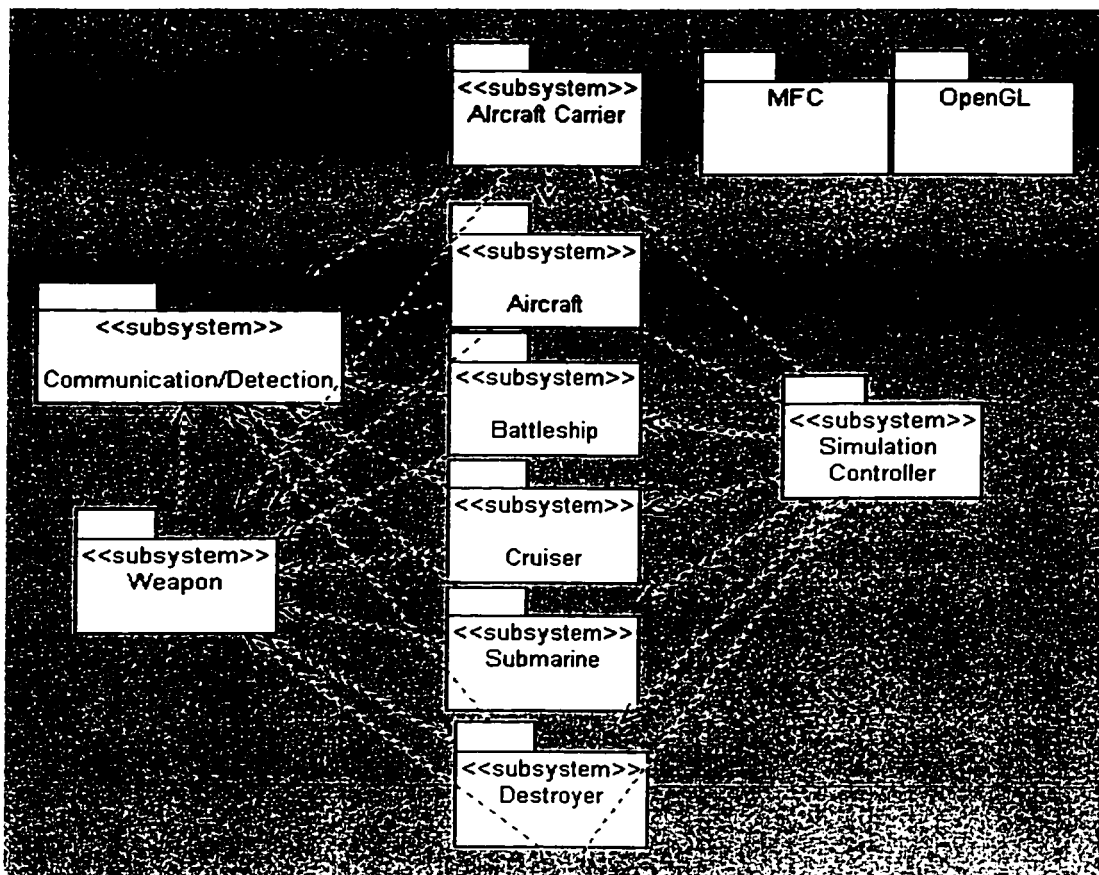


Figure 4-1 Interaction diagram between subsystems

The following figure describes the architecture of the system:

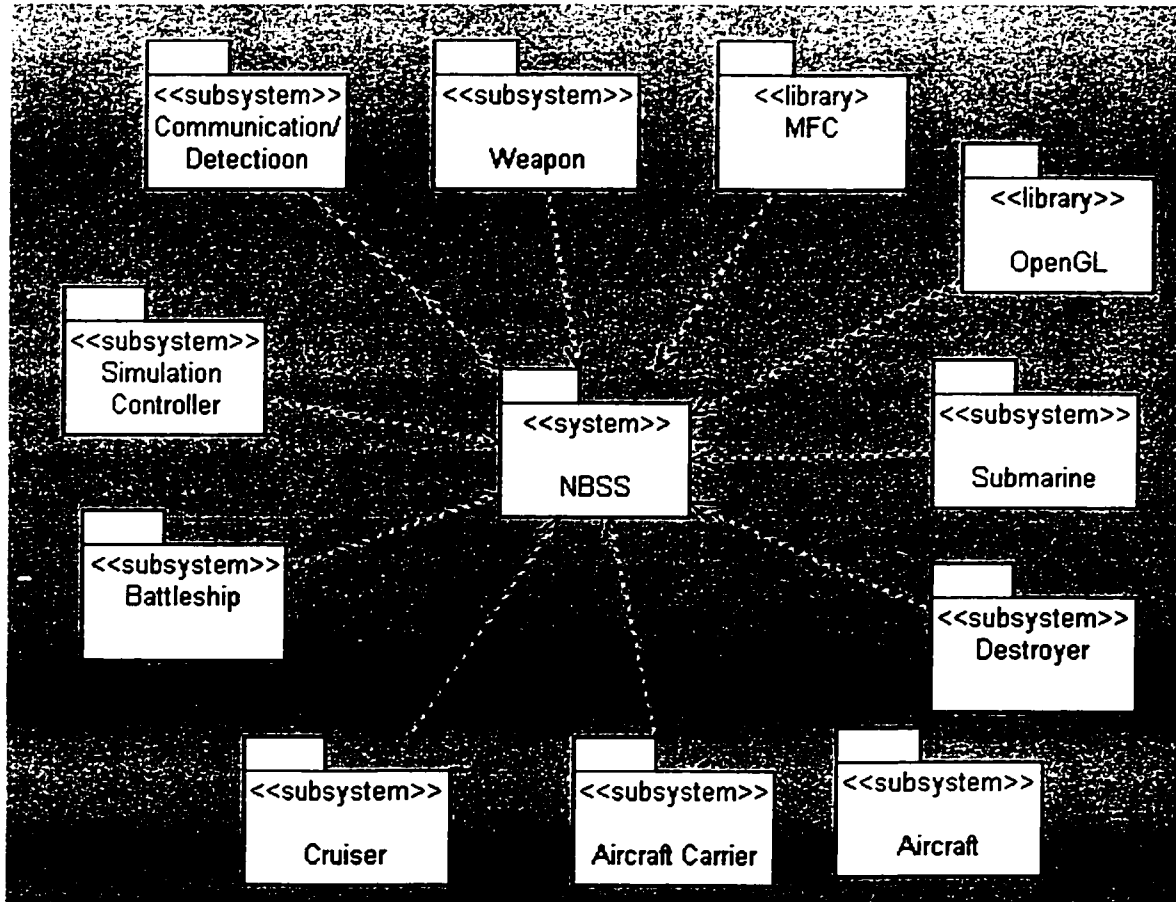


Figure 4-2 Architecture of the Naval Battle Simulation System

The following diagram describes the subsystem interface diagram at the class level:

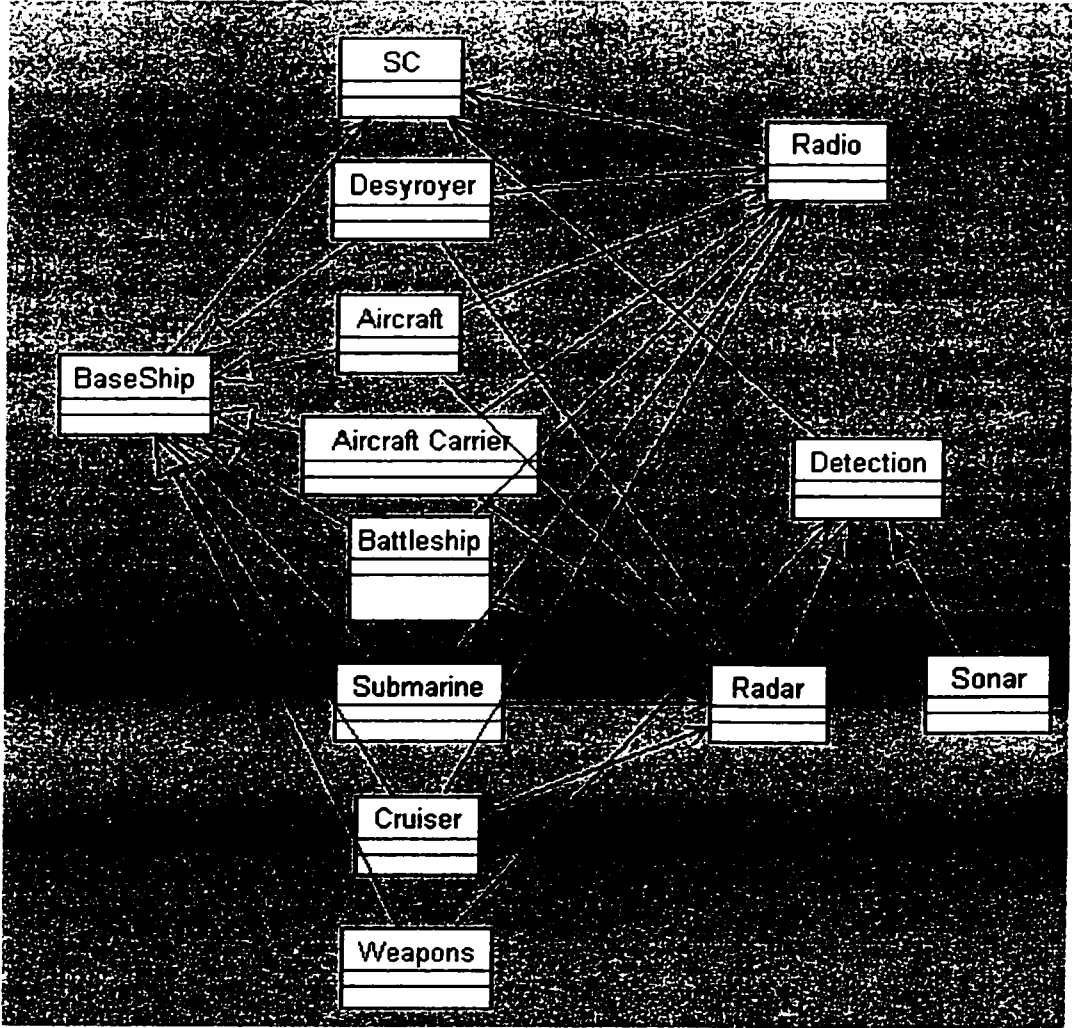


Figure 4-3 Class Level Interface diagram

4.1.1.1 Simulation Controller

The Simulation Controller is the heart of the simulation. It provides a user interface to view the objects navigating on the map. Consequently, threats are generated to provoke offensive and defensive maneuvers at the beginning and periodically after running as well.

To perform the simulation, Simulation Controller allows every object to have a time slice to update its data information. By tracking the positions and status of all Vehicles and Weapons objects periodically, it generates an animated view of the naval battlefield. The Simulation Controller knows exactly where each agent is at any time and draws the agents on the screen.

For any agent, the only way to know the position of another agent is done by interrogating the Simulation Controller through Communication/Detection. Communication (Radio) and Detection (Radar and Sonar) can interact with the Simulation Controller to detect the enemies and exchange information among allies. The Simulation Controller depends on all other subsystems except Communication/Detection subsystem.

4.1.1.2 Communication/Detection Description

Enemies can only be detected using a Radar for Aircraft and Ships or Sonar for Submarines and Destroyers. Radars and Sonars are on board ships and Aircrafts. If an enemy is not detected using a Radar or Sonar (i.e. it is outside its range), it is virtually non-existent in the simulation, as far as other Ships and Aircrafts are concerned.

Allies also have to communicate with one another to share some information about the location of enemies. Aircraft Carriers also need to communicate their orders to Aircrafts. In the simulation, Communication/Detection acquire agent position information by interrogating the Simulation Controller. It depends on the Simulation Controller and all other subsystems depend on it, except the Simulation Controller.

4.1.1.3 Aircraft Carrier Description

The Aircraft Carrier gives long-range capacities to the fleet by launching Aircrafts to locate and destroy enemy Ships and Aircrafts. The Aircraft Carrier itself is "blind". It can only "see" enemies by the information it gets from its patrolling Aircrafts and its allied Ships using its Radio (Communication).

Much of the job done by the Aircraft Carrier itself is communication with its Aircrafts to gather threat information and react to it as fast as possible to eliminate threats while they are as far as possible from the fleet. The Aircraft Carrier can transmit its updated position to the Simulation Controller. It depends on the Communication/Detection and its Aircrafts.

4.1.1.4 Aircraft Description

The Aircraft is used by the Aircraft Carrier to provide a long-range detection by patrolling using its Radar (Detection). It is also able to intercept far enemy Aircrafts and Ships by firing Weapons (Air-Sea Missile and Air-Air Missile). It communicates using its Radio (Communication) to the Aircraft Carrier the position of any enemy Aircraft and Ship it encounters during a patrol. An Aircraft can transmit its updated position to the Simulation Controller. It depends on the Communication/Detection and Weapon subsystems.

4.1.1.5 Destroyer Description

The Destroyer locates underwater threats with its Sonar (Detection) and attempts to intercept them with its torpedoes and sea-sub Missiles (Weapons). It cooperates with Submarines teammates by sending them the coordinates of all detected enemy Submarines using their Radio (Communication). The Destroyer can transmit its updated position to the Simulation Controller. It depends on the Communication/Detection and Weapon subsystems.

4.1.1.6 Cruiser Description

The Cruiser locates airborne threats with its Radar (Detection) and gives the information about far threats to its allies using its Radio (Communication). It also attempts to intercept close airborne threats with its sea-air Missiles (Weapons). It also receives information using its Radio (Communication) about far enemy Aircrafts detected by allies. The Cruiser can transmit its updated position to the Simulation Controller. It depends on the Communication/Detection and Weapon subsystems.

4.1.1.7 Battleship Description

With its Radar (Detection), the Battleship scans the surrounding water surface for enemy ships. It also receives information from its allies about far seaborne threats by Radio (Communication). The Battleship attempts to eliminate the nearest threats using its Weapons (Sea-Sea Missiles and Heavy Cannons). Battleship can transmit its updated position to Simulation Controller. It depends on the Communication/Detection and Weapon subsystems.

4.1.1.8 Submarine Description

The Submarine cruises underwater and attempts to detect enemies in the water using its Sonar (Detection) and to destroy enemy ships and Submarines using its torpedoes and Sub-Sea Missiles (Weapon). It can use its Radio (Communication) to communicate to its allies all the enemies it detected with its Sonar. The Submarine has a unique advantage: it is invisible to all Ships and Aircrafts, except to Destroyers and to other Submarines, which can detect them underwater with their Sonar. The Submarine can transmit its updated position to Simulation Controller. It depends on the Communication/Detection and Weapon subsystems.

4.1.1.9 Weapon Description

The Weapons are used by Ships and Aircrafts to eliminate threats. They have limited functionalities, but there are different kinds of Weapons, such as the various Missiles, Torpedoes and Cannon Shells. Most Weapons are auto-aiming, relying on their own Radar or Sonar (Detection) to aim at their assigned target. Some others (e.g. Cannon Shells) follow a ballistic trajectory and are unguided after they are shot. The Weapon can transmit the object's position to the Simulation Controller from time to time. It depends on the Communication/Detection subsystem only.

4.1.2 Concurrent Process Decomposition

NA.

4.1.3 Data Decomposition

4.1.3.1 Data entity description

- Each object has a position of the Vector type. A position comprises three float numbers, representing the object's tridimensional coordinates.
- Each object also has a status, which represent it is alive or dead.
- Each object has a type represented as follows: 1-Aircraft Carrier; 2-Aircraft; 3-Destroyer; 4-Cruiser; 5-Battleship; 6-Submarine; 7-Missile/Torpedo; 8-Heavy Shell Cannon.
- Each object has a flag of the Character type to indicate its side.

4.2 Dependency Description

This section describes the dependency relationships among all the subsystems i.e. what subsystem uses or requires from other subsystems. The main purpose of designing emphasizes low module coupling and high module cohesion in terms of subsystem dependency. [10]

4.2.1 Internal Module Dependency

4.2.1.1 Simulation Controller dependency on BaseShip Subsystem

SC depends on BaseShip to create/destroy itself, update its position and status, get type, and get flag etc. SC needs all these functions to control the BaseShip activity during simulation process.

4.2.1.2 Simulation Controller dependency on BaseWeapon subsystem

SC depends on BaseWeapon to get position, update its position and status, get type, get flag, and execute fire behavior etc. SC needs all these BaseWeapon functions to simulate the BaseWeapon activity when Weapon are fired and hit the target.

4.2.1.3 Communication/Detection dependency on Simulation Controller

Communication/Detection depends on SC to get the object list within range of Radar/Sonar. Radio also depends on SC to communicate with its allies ship.

4.2.1.4 Communication/Detection dependency on BaseShip

Communication/Detection depends on BaseShip to get its ID, type, position and status when Radar/Sonar detects the ship or aircraft. Same dependency is between Radio and BaseShip when Radio needs to send/receive the message.

4.2.1.5 BaseShip Subsystem dependency on Communication/Detection

BaseShip depends on Communication/Detection to create/destroy itself (radar/soanr, radio), get detected objects information, go through the objects information, send/receive information, get sender/receiver ID and type, and get sender/receiver position etc. By the above dependency, BaseShip can detect enemy and pass the information to allies.

4.2.1.6 BaseShip (except Aircraft Carrier) dependency on BaseWeapon

BaseShip depends on BaseWeapon to create/destroy itself, get attributes, get speed, get position, get type as well. BaseWeapon also provides its status, velocity to BaseShip. Especially, BaseWeapon can fire itself and hit the target by listen to the BaseShip command.

4.2.1.7 BaseWeapon dependency on BaseShip

BaseWeapon depends on BaseShip to deduce its resistance when it is hit by Weapon.

4.2.1.8 BaseWeapon dependency on Communication/Detection

BaseWeapon including all the Weapons except Heavy Cannon Shell, Sub-Sea Torpedo and Sea-Sub Missile depend on Communication/Detection (Radar and Sonar, not Radio) to simulate the Weapon detection device. BaseWeapon need to get the detected object information and go through the detected information to aim the target.

4.2.2 Internal Process Dependency

NA

4.2.3 Data Dependency

NA

4.3 Interface Description

This section describes the details of external and internal interfaces not provided in the software requirement specification. It provides the information for the developer to know how to correctly use the functions provided by each entity. It contains everything another designer needs to know on how to interact with a specific entity. It also specifies the type of relations in terms of shared information, prescribed order of execution, or parameters interfaces. [10]

4.3.1 Module Interface

The whole system working well needs all subsystems to cooperate with each other. Besides using functions of other subsystems, each subsystem also provides some service for some other subsystems. This section described the interface of each subsystem in interface interaction diagram and detailed function description as well.

4.3.1.1 Simulation Controller

Simulation Controller provides the services to Communication/Detection subsystem and Weapon subsystem as described following.

4.3.1.1.1 Simulation Controller for Weapon

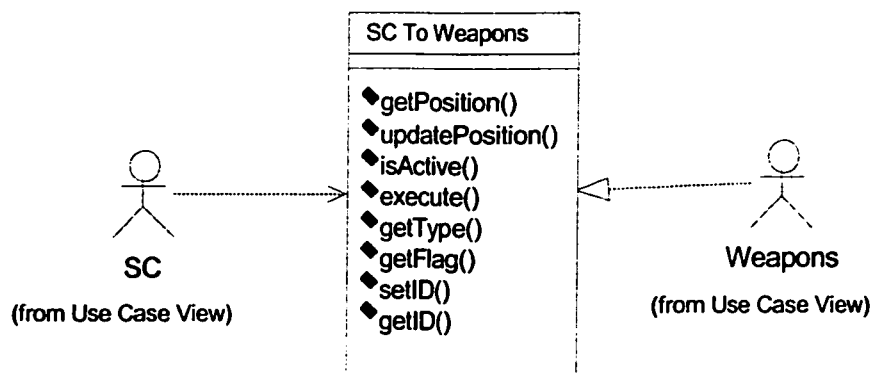


Figure 4-4 Simulation Controller for Weapon

4.3.1.1.2 Simulation Controller for Communication/Detection

- **getVehicleList()** takes no parameters, and return a pointer to the array of base ship class. When the ships or Aircraft need to get the information about the other objects, the Radar/Sonar needs to call this function of SC to get the object information within its range. The Radio also needs this function to know the allies position to communicate with each other.

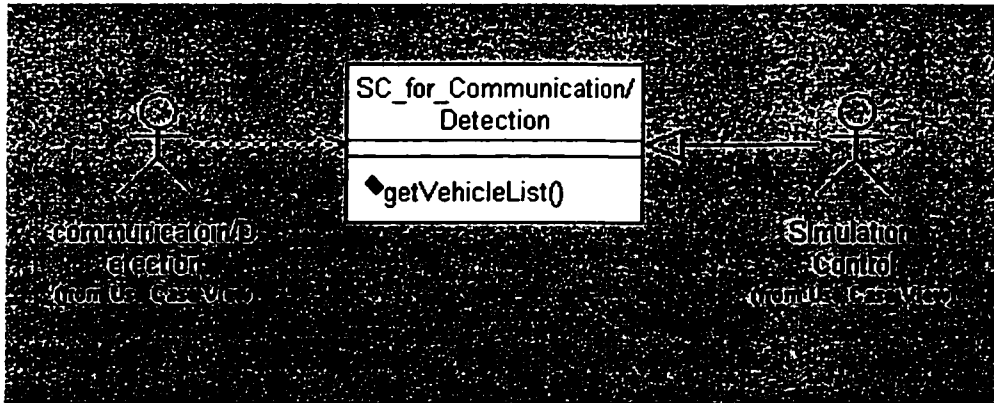


Figure 4-5 Simulation Controller for Communication/Detection

4.3.1.2 Communication/Detection

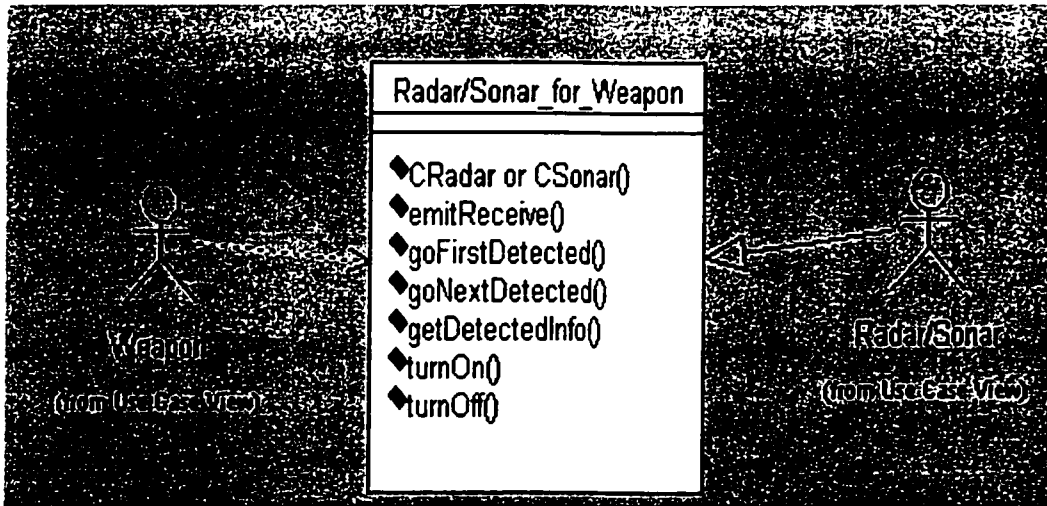
Communication/Detection subsystem provides the service to all the ships (Aircraft) and Weapon subsystem. The Communication/Detection is the simulation system of detection for Weapon.

4.3.1.2.1 Communication/Detection for Ships and Aircraft

- **emitReceive()** takes vector type for its position as parameter, returns no value to ensure that all the object information is updated. This function is called before getting position info to ensure that all position info are up to date when the Ship or Aircraft need to know the position of other Ship or Aircraft.
- **goFirstDetected()** takes no parameters, returns the first detected object. This function is called when the ship want to know the first detected object info.
- **goNextDetected()** takes no parameters, returns the next detected object. This function is called when the ship want to know the next detected object info.
- **getDetectedInfo()** takes no parameters, returns the Detected type of object information.

4.3.1.2.2 Communication/Detection for Weapon

- **emitReceive()** takes vector type for its position as parameter, returns no



value to ensure that all the object information is updated. This function is called before getting position info to ensure that all position info are up to date when the Weapon need to know the position of target Ship or Aircraft.

- **goFirstDetected()** takes no parameters, returns the first detected object. This function is called when the Weapon wants to know the first detected object info.
- **goNextDetected()** takes no parameters, returns the next detected object. This function is called when the Weapon wants to know the next detected object info.
- **getDetectedInfo()** takes no parameters, returns the Detected type of object information. Then call a derived object of Ship Base Object functions **getId()**, **getFlag()**, **getPosition()**, **getSpeed()** and **getPowerSwitch()** to get the information of the detected object. This function is called when the Weapon wants to know the detailed detected object info.

Figure4-6 Radar/Sonar_for_Weapon

4.3.1.3 BaseShip Class

Base ship class provides the services to the Simulation Controller, Communication/Detection and Weapon. If the derived ship class has additional services, they will be described in each derived class section.

4.3.1.3.1 BaseShip Class for Simulation Controller

- **SBaseConstructor()** takes different parameters to create the different kinds of ships and Aircraft for both sides respectively when simulation is started.
- **getPosition()** takes no parameter. Returns a vector type position of a derived object of Ship.
- **updatePosition()** takes no parameter, and returns no value. It updates the Ship's position from the last time slice to the present time slice.
- **isActive()** takes no parameter, and return value is Boolean type. It indicates if a Ship object is still alive. TRUE means alive and FALSE means sunk.
- **execute(int)** takes an integer type time slice as a parameter, and no return value. It is called by the Simulation Controller to allow a derived object of Ship to undertake its all computation at the latest time slice.
- **getType()** takes no parameter, and return value is an integer. The different return value indicates the different type of a Ship.
- **getFlag()** takes no parameter, and returns a char. The return value 'R' indicates a Ship belongs to "RED" side and 'B' to "BLUE" side.
- **setID()** takes integer as a parameter, and no return value. This function sets a unique ID to a Ship as soon as it is created.
- **getID()** takes no parameters, and returns an integer. The return value indicates the unique ID of a derived object of a Ship.
- **setFuelAmount()** takes one float parameter as the fuel amount at the initial setting, and another integer to indicate ID of a Ship, returns no value.
- **setFuelLimit()** takes one float parameters as the fuel limit at initial setting, and another integer to indicate ID of a Ship, returns no value.
- **requestFuel()** takes one float parameters as the requested fuel amount, and another integer to indicate ID of a Ship, returns Boolean value to indicate if the refilling fuel is success or fail.
- **requestWeapon()** takes no parameters and returns Boolean value to indicate if the Weapon request is success or fail.
- **setWeaponType()** takes one integer parameter as the Weapon type at the initialize setting, and another integer to indicate ID of a Ship, returns no value.
- **setWeaponAmount()** takes one float parameters as the weapon amount at the initialize setting, and another integer to indicate ID of a Ship, returns no value.

- **SetWeaponLimit()** takes one parameter as the Weapon limit at the initialize setting, and another integer to indicate ID of a derived object of Ship, returns no value.

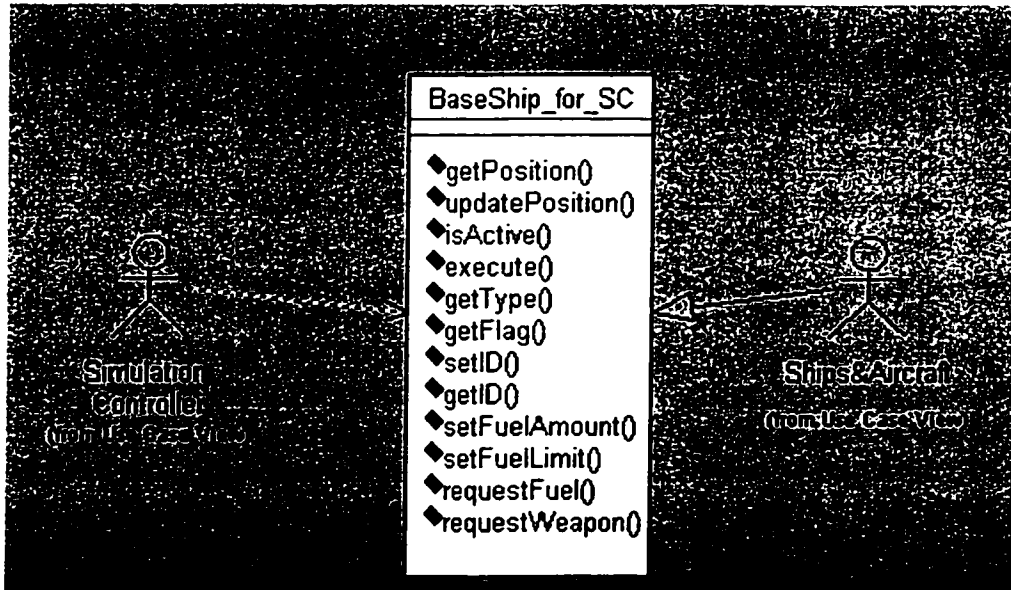


Figure 4-7 BaseShip_for_SC

4.3.1.3.2 Base Ship Class for Communication/Detection

- **getPosition()** takes no parameter. Returns a vector type position of a derived object of Ship.
- **updatePosition()** takes no parameter, and returns no value. It updates the Ship's position from the last time slice to the present time slice.
- **isActive()** takes no parameter, and return value is Boolean type. It indicates if a Ship object is still alive. TRUE means alive and FALSE means sunk.
- **getType()** takes no parameter, and return value is an integer. The different return value indicates the different type of a Ship.
- **getFlag()** takes no parameter, and returns a char. The return value 'R' indicates a Ship belongs to "RED" side and 'B' to "BLUE" side.
- **getID()** takes no parameters, and returns an integer. The return value indicates the unique ID of a derived object of a Ship.

4.3.1.3.3 Base Ship Class for Weapon

- **hitObject()** takes one integer type of parameter for firepower and returns void. The function is called when Weapon is hit with ship or Aircraft. The ship will update its resistance according to firepower.

4.3.1.4 BaseWeapon

Weapon subsystem provides the service to Simulation controller and all ships and Aircraft except the Aircraft Carrier.

4.3.1.4.1 BaseWeapon for Simulation Controller

- **execute()** takes vector as position for parameters, and returns void to execute all necessary real-time function when it is fired on the map of SC.
- **updatePosition()** takes one integer type for Weapon ID as parameter and returns void. When the Weapon is launched, the SC need this function to know the Weapon updated position for aiming and firing the object.

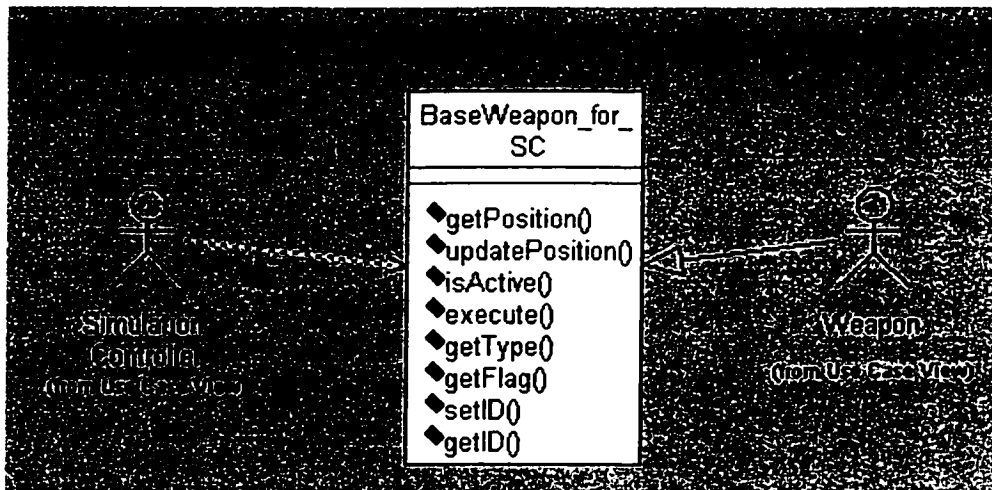


Figure 4-8 BaseWeapon_for_Simulation Controller

4.3.1.4.2 BaseWeapon for Ship and Aircraft

- **WBaseConstructor()** takes different parameters to create the different types of Weapon objects respectively when the Weapon are launched by the ship s and Aircraft.
- **getAttribute()** takes one integer value as Weapon ID , returns vector value to indicate the Weapon attribute;
- **getSpeed()** takes one integer value as Weapon ID, returns float value to indicate the Weapon speed;
- **getType()** takes one integer value as Weapon ID, returns integer value to indicate the Weapon type.
- **getFalg()** takes one integer type as Weapon ID, returns char type as flag of Weapon. The ships and Aircraft need to use this function to know the Weapon belongs to which side.
- **getID()** takes no parameters and return the ID of a Weapon. The ships need this function to know the Weapon ID.
- **getPosition()** takes one integer type for Weapon ID as parameter and return the Position type of position of Weapon. The ships need this function to know the Weapon current position.
- **getType()** takes one integer type for Weapon ID as parameter and return integer type for Weapon type. The ships need this function to know the Weapon type.
- **isActive()** takes one integer value as Weapon ID , returns Boolean value to indicate the Weapon is active or not. The ships need this function to know the Weapon state.
- **fire()** takes two Position type for start position of launcher and destination position of target as parameters, and return void. The ships need this function to fire the Weapon.
- **getVelocity()** takes one integer type for Weapon ID and return Velocity type
- for velocity of Weapon.
- **getStatus()** takes one integer type for Weapon ID and return integer type for status of Weapon.(Moving or static)

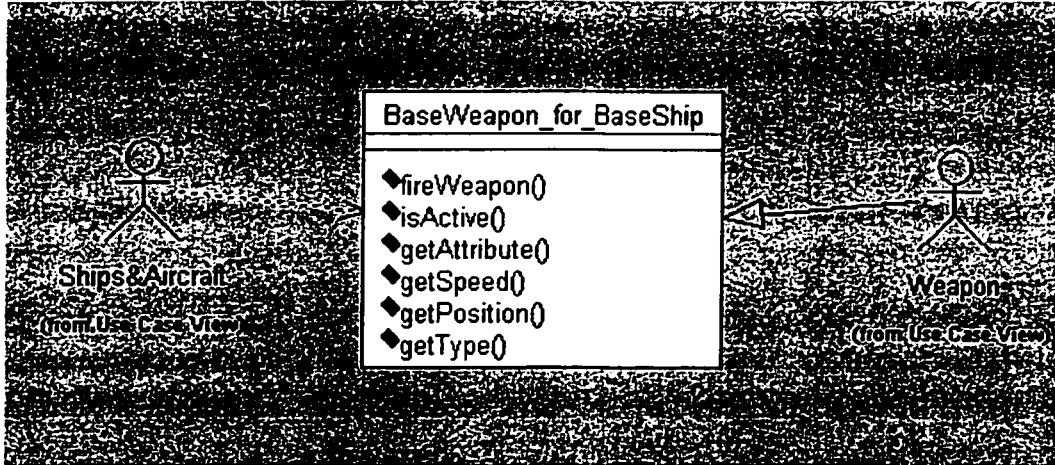


Figure 4-9 BaseWeapon_for_Ship and Aircraft

4.3.2 Process Interface

NA.

4.4 System Detailed Design

This section describes the internal design detail of each subsystem. It includes the attribute descriptions for identification, processing and data. Each subsystem is described in the aspects of module detailed design, class definition and description of class data members and member functions.

In Class Definition sub section, the traceability of the class design to SRS requirement is listed for each class. The constants and private data member of class are described in the Constant table and Private(Protected or public)data member table.

In the description of functions, when one function need to use a function in another class, we use sign \rightarrow . The left side of sign \rightarrow is the class name and the right side is the function type. This applies to all class descriptions in section 5.4.

4.4.1 Simulation Controller Detailed Design

This section describes all the classes of the SC module of the NBSS and the functions they contain. In the module detailed design section, the modules of this subsystem are diagrammed in UML and designed in such a way that this module can be implemented easily in MFC and OpenGL. We employed MFC's View/Document architecture to describe the core structure of the SC module as shown in the following figure.

4.4.1.1 Module Detailed Design

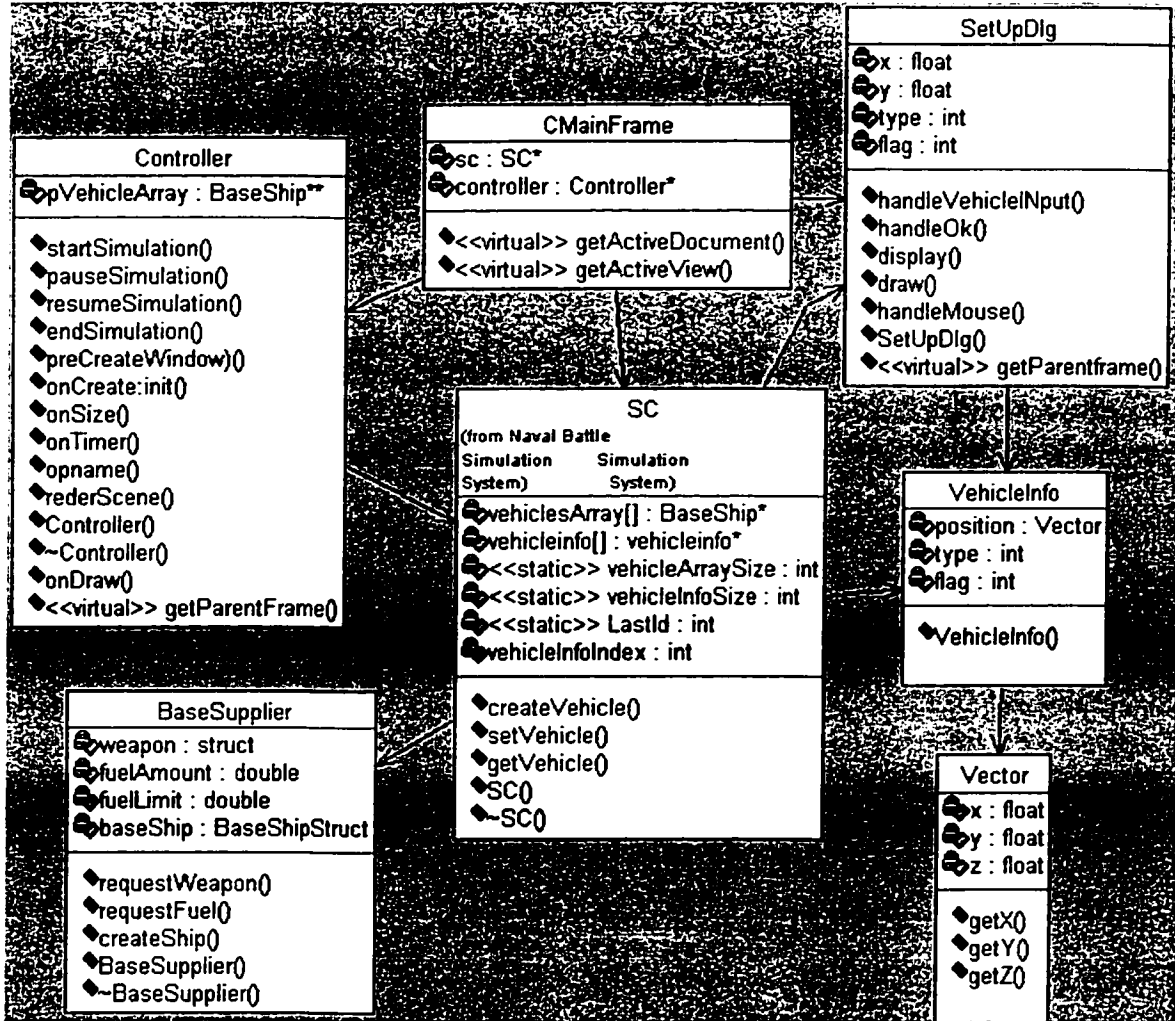


Figure 4-10 Class Diagram for Simulation Controller Module

4.4.1.2 Class Definition

4.4.1.2.1 CMainFrame Class

Traceability to SRS

SC-006, SC-012, SC-014, SC-017, SC-018, SC-019

Constants

NA

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
sc	SC*	Handle to Document Class
controller	Controller*	Handle to View Class

Public functions

Name: CMainFrame

Input: none

Output: none

Description: default constructor, inherit CFrameWnd class of MFC

Pseudo-code:

 Begin:

 End

Name: getActiveDocument

Input: none

Output: CDocument*

Description:

Pseudo-code:

 Begin:

 Return a handle of the active Document

 End

Name: getActiveView

Input: none

Output: CView*

Description:

Pseudo-code:

 Begin:

 Return a handle of the active View.

 End

Name: -CMainFrame

Input: none

Output: none

Description: virtual destructor

Pseudo-code:

Begin:

End

4.4.1.2.2 SetUpDlg Class

Traceability to SRS

SC-001, SC-002, SC-003, SC-004, SC-005, SC-007, SC-008, SC-008-01, SC-008-02

Constants

NA

Private data members

Name	Type	Description
X	Float	Condinate of x_axiel of position
Y	Float	Condinate of y_axiel of position
Type	char	Ship type
Flag	char	Side flag
DrawInfo	Struct	Structure of draw information about object
m_typebutton	Integer	Ship type button flag
vInfo[15][15];	VehicleInfo	Ship info 2-D array

Public functions

Name: SetUpDlg

Input: pParent CWnd*

Output: none

Description: constructor, inherit from CDialog class of MFC

Pseudo-code:

Begin:

m_typeButton=-1

End

Name: Draw

Input: wBmp WORD,x1 int,y1 int

Output: none

Description: constructor, inherit from CDialog class of MFC

Pseudo-code: draw the ship object on the map

Begin:

Select image symbol according to type

Copy the bitmap to screen

End

Name: OnInitDialog

Input: none

Output: none

Description: initialize the draw info array and ship info array

Pseudo-code:

```
Begin:
  Loop to initialize the draw info array
  Loop to initialize the ship info array
  vInfo[i][j] = NULL; //i and j are index for ship objects array
End
```

Name: OnLButtonDown

Input: nFlags UINT, point CPoint

Output: none

Description:

Pseudo-code: draw the ship object on the map

```
Begin:
  Select image symbol according to type
  Copy the bitmap to screen
End
```

Name: OnUndo

Input: none

Output: none

Description: undo the drawing object on map

Pseudo-code:

```
Begin:
  take the top element of undoStack;
  delete vInfo[r][c]; //r and c are index of ship element in the top of
  undoStack
  set vInfo[r][c] = NULL;
  set drawInfo array to default value
End
```

Name: OnClearall

Input: none

Output: none

Description: clear all the ship image on the map

Pseudo-code:

```
Begin:
  For all the ship on the map
  delete vInfo[r][c]; //r and c are index of ship element on the map
  set vInfo[r][c] = NULL;
  set drawInfo array to default value
End
```

4.4.1.2.3 SC class

Traceability to SRS

SC-011, SC-013, SC-013-01, SC-013-02, SC-013-03, SC-013

Constants

Name	Type	Value	Description
PARA	double	1.3	
RADAR_RANGE	double	150.0	Radar range
SONAR_RANGE	double	100.0	Sonar range
RADIO_RANGE	double	1000.0	Radio range
WEAPON_RANGE	double	140.0	Weapon range for all Weapon
Time	double	0.07	Time slice for each ship or Aircraft

Private data members

Name	Type	Description
vpVehicles	static VPtr	vector of pointers to Vehivles
Fac	VehicleFactory	
vehicleInfo[15][15] ;	VehicleInfo*	2-D Array of ship information
Mdir	CMap<int,int, float, float>	Simulation Map
Anim	bool	Indicate animation is started or not
Time	static double	Time slice for each ship or Aircraft
lastID	static Integer	The lastID of new created ship object

Public functions

Name: SC

Input: none

Output: none

Description: constructor, inherit from CDocument class of MFC

Pseudo-code:

```
Begin:
    Set animation to false;
    Loop to set vehicleInfo[i][j]=NULL;
End
```

Name: calVelocity

Input: b1 Vector, v0 Vector, speed double

Output: vector

Description: calculate the velocity

Pseudo-code:

```
Begin:
    Generate the random number as V1 and V0,
    Use V1 and v0 and speed to get the vector of next position;
End
```

Name: iterator findNearest

Input: *vptr vector<ShipClass*>, *vpPtr Vector, t1 int, t2 int, t3 int, t4 int, t5 int

Output: vector<baseClass*>

Description: find the pointer of an object which is nearest to the current position

Pseudo-code:

```
Begin:
    LOOP to get the nearest position
    If((the target position minus current position < minimum length) and the
    target is type 1 to 6 except itself)
    Update the minimum length;
    Return pointer of the nearest object
End
```

Name: getVehicleList

Input: none

Output: VPtr*

Description:

Pseudo-code:

```
Begin:
    return & vpVehicles
End
```

Name: OnStartSetup

Input: none

Output: none

Description: start the animation

Pseudo-code:

```
Begin:
    Loop to set the vehicleInfo[row][col]//index are position on the map
    VehicleFactory->createVehicle();
    Set anim to true
End
```

Name: freeVehicleList

Input: none

Output: none

Pseudo-code:

```
Begin:
    Set the pointer to the first of pVehicles;
    Loop until the end of the vector to erase the element of vector}
End
```

Name: getTimeSlice

Input: none

Output: double

Description:

Pseudo-code:

```
Begin:
    return time
End
```

Name: incrLastID

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    lastID++;
End
```

Name: -SC

Input: none

Output: none

Description: distructor

Pseudo-code:

```
Begin:
    Call freeupVehicleList();
End
```

4.4.1.2.4 Controller Class

Traceability to SRS

SC-009, SC-010, SC-013, SC-014

Constants

Name	DTG	Value	Description
PI	GLfloat	3.1415926f	

Private data members

Name	Type	Description
counterActive	integer	Counter of system
start1	Clock_t	Start system time

Protected data members

Name	Type	Description
percent	integer	Test variable

Public data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
Fhor	float	Time variable
fVer	float	Velocity variable
zoom	float	Image zoom variable
TextureImage	Struct	Image struture
textures[18]	TextureImage	TestureImage array
offset	GLfloat	Offset of image

Public functions

Name: Controller**Input:** none**Output:** none**Description:** constructor, inherit from CView class of MFC**Pseudo-code:**

```
Begin:
    Initialize the member data;
End
```

Name: OnDrawc**Input:** pDC CDC***Output:** none**Description:****Pseudo-code:**

```
Begin:
    Test one loop time;
    Clear out the color & depth buffers;
    Draw picture by using OpenGL function
    Get the object size of ships by calling VPtr *ptr = SC::getVehicleList();
    clearing dead Weapons;
    Tell OpenGL to flush its pipeline;
    Swap the buffers;
    If the simulation is over, Swap the buffer;
End
```

Name: InitializeOpenGL**Input:** none**Output:** bool**Description:****Pseudo-code:**

```
Begin:
    Get a DC for the Client Area; if fail, return false;
    Create Rendering Context by calling ::wglCreateContext (m_pDC-
    >GetSafeHdc()); if fail, return false;
    Make the Rendering Context Current; if fail, return false;
    Otherwise, return true;
End
```

Name: calDir

Input: Vo Vector, V1 Vector

Output: GLfloat

Pseudo-code:

Begin:

Calculate the direction according to the vo and v1

End

Name: OnStartSetup

Input: none

Output: none

Description: start the animation

Pseudo-code:

Begin:

Loop to set the vehicleInfo[row][col]

VehicleFactory->createVehicle();

Set anim to true

End

Name: OnCreate

Input: lpCreateStruct LPCREATESTRUCT

Output: integer

Description: start the animation

Pseudo-code:

Begin:

get rid of the default title;

Call InitializeOpenGL();

Return -1 if can not load images;

Call OpenGL function to set the background and Enable blending

Return 0;

End

Name: OnSize

Input: nType UINT, cx int, cy int

Output: none

Description:

Pseudo-code:

Begin:

Handle paints of graphical ships when window size is changing

End

Name: OnTimer

Input: nIDEvent UINT

Output: none

Pseudo-code:

Begin:

For each element of vehicleArray

Do execute function in a time slice

Do Update position;

End

Name: LoadTGA

Input: texture TextureImage *, filename char *

Output: bool

Description: Loads A TGA File Into Memory

Pseudo-code:

Begin:

Open The TGA File by calling FILE *file = fopen(filename, "rb");

Read file Bytes;

Loop the image data to swap the data;

If (texture Building of OpenGL function) success Return True;

End

Name: drawVehicles

Input: TextureImage *tex, float posX, float posY, float w, float h,
float angle

Output: none

Pseudo-code:

Begin:

Call OpenGL function to draw the ship or Aircraft objects;

Flush the buffer for openGL;

End

4.4.1.2.5 VehicleInfo Class

Traceability to SRS

SC-001, SC-002, SC-013, SC-015, SC-015-01

Constants

NA

Public data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
position	vector	vector of Vehicle position
type	integer	Type of Vehicle
flag	char	Flag of Vehicle

Public functions

Name: VehicleInfo

Input: Vector pos, int aType, char aFlag

Output: none

Description: constructor

Pseudo-code:

Begin:

Initialize the member data

End

4.4.1.2.6 VehicleFactory Class

Traceability to SRS

SC-001, SC-002, SC-013, SC-015, SC-015-01

Constants

NA

Public data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
pDoc	Cdocument*	Handle for document object

Public functions

Name: VehicleFactory

Input: CDocument* pDoc

Output: none

Description: constructor

Pseudo-code:

```
Begin:
    this->pDoc = pDoc;
End
```

Name: createVehicle

Input: none

Output: bool

Description: create the ship or Aircraft according to the user setting

Pseudo-code:

```
Begin:
    Create SC object;
    Switch(SC->VehicleInfo[I][j]->Type)
    Case AircraftCarrier:
    Case Aircraft:
        Create new object;
        Initialize flag, position and ID for this object;
        Increase the object number counter;
    Case://for all the other ship object
    :
    :
    :
    If (counter>0)
    Return true;
    Else return flase;
End
```

Name: virtual -VehicleFactory

Input: none

Output: none
Description: virtual destructor
Pseudo-code:
 Begin:
 End

4.4.1.2.7 BaseSupplier Class

Traceability to SRS

SC-015, SC-015-01, SC-015-02, SC-015-03, SC-015-04, SC-015-05,

Constants

NA

Public data members

Name	Type	Description
bship	BaseShipStructure	New created ship object
Fuelamount	double	Total fuel amount of base supplier
Weapon	struct	Total Weapon amount and Weapon type structure

Public functions

Name: BaseSupplier

Input: none

Output: none

Description: default constructor

Pseudo-code:

```
Begin:
    Fuelamount=0;
    Weapon.type=-1;
    Weapon.amount=0;
    Ship.type=-1;
    Ship.amount=0;
End
```

Name: BaseSupplier

Input: double fue, Weapon wep, BaseShipStructure ship

Output: none

Description: constructor

Pseudo-code:

```
Begin:
    Furamount=fue;
    Weapon.type=wep.amount;
    Weapon.amount=wep.type;
    Bship.type=ship.type;
    Bship.amount=ship.amount
End
```

Name: requestFuel

Input: double fuel

Output: bool

Description:

Pseudo-code:

Begin:

Check the fuel is enough or not;

Deduct the fuel amount;

Return true;

Else return false;

End

Name: requestWeapon

Input: Weapon wep

Output: bool

Description:

Pseudo-code:

Begin:

Check the wepaon amount and type;

Create Weapon;

Deduct the Weapon amount o fth etype;

Return true;

Else return false;

End

Name: createShip

Input: none

Output: bool

Description:

Pseudo-code:

Begin:

Check the ship object amount;

If amount<=limits

Create the ship fo setting type and amount.

Deduct the ship amount of the type;

Retrun true;

Else

return false;

End

Name: ~BaseSupplier

Input: none

Output: none

Description: destructor

Pseudo-code:

Begin:

End

4.4.2 Communication/Detection Detailed Design

This section describes all the classes of Communication/Detection subsystem of the NBSS and the functions they contain. In module detailed design section, the modules of this subsystem are diagrammed in UML and designed in such a way that this module can be implemented easily using MFC. The architecture of this subsystem is shown in Figure 4-11.

4.4.2.1 Module Detailed Design

The class operation and attribute are not list in the class diagram for all the classes in Communication/Detection module. Refer to the section of **Description of Class Members and Members Functions** for each class.

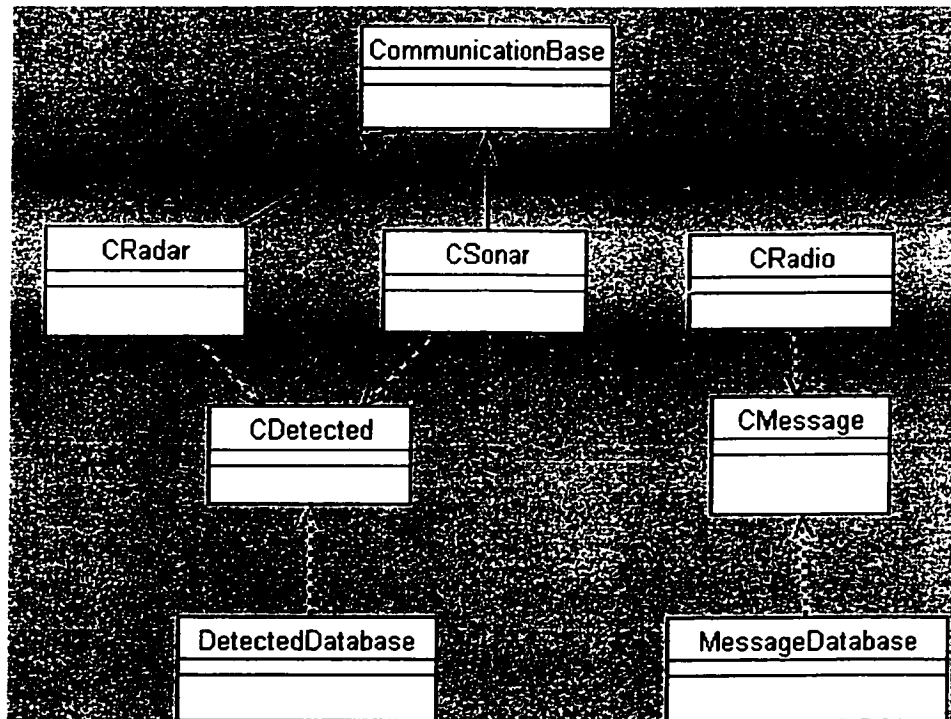


Figure 4-11 Class Diagram for Communication/Detection Module

4.4.2.2 Class Definition

4.4.2.2.1 CommunicaitonBase Class

Traceability to SRS

CD-001, CD-002, CD-003, CD-004, CD-004-01, CD-004-02, CD-005, CD-006, CD-007, CD-008, CD-008-01, CD-008-02

Constants

NA.

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
type	integer	1 is Radar, 2 is Sonar
ID	integer	Radar/Sonar object ID
ddb	CDetectedDatabase	All the detected information class
state	integer	Radar/Sonar on/off state ("on" for object creation)
range	double	Radar/Sonar radius of detection

Public functions

Name: CommunicationBase

Input: integer ty

Output: none

Description: default constructor

Pseudo-code:

```
Begin:
  Initialize the member data;
  ID=0;
  Range=0;
  type=ty;
End
```

Name: CommunicationBase

Input: nId integer, nRange integer, ty integer

Output: none

Description: constructor

Pseudo-code:

```
Begin:
  Id= nId;           //initialize id
  Range=nRange;     //initialize Range
  Type=ty;
End
```

Name: emitReceive

Input: Vector pos

Output: integer

Description:

Pseudo-code:

```
Begin:
  //refresh the detection list  ddb.deleteAll();
  difference=0.0 ;           // distance between two positions.
  i=0;                       // indicator for static gloable array from SC
  for (int i=0; i < SCarraylength; i++)
  {
    length =0;                //length of detected object list.
    point= new detected;      // a pointer point to a detected object.
    detected  dobject;        // instance of detected object.
    Dpoint = SCarray[i] ;     //this pointer point to a object.

    if (SCarray[i]-> active()) //pointer access in BaseShip class.
    {
      p1 = dpoint->getPosition();
      p2 = pos;
      p3 = p1-p2 ;            // difference between two vectors.
      p3.length();
      if (difference < range) and (difference >0.0))
      {
        //set data members for detected object dobject;
        dobject.setDetData(SCarray[I]);
        //insert detected object b to container DetectedDatabase ddb
        ddb.addOneDetIntheList(dobject);
        increment length by 1;
      }
    }
  }
  return length;
End
```

Name: getDetected

Input: none

Output: baseClass*

Description:

Pseudo-code:

```
Begin:
  get detected object by calling getDetectedFromList() in DetectedDatabase
End
```

Name: getFirstDetected

Input: none

Output: baseClass*

Description:

Pseudo-code:

```
Begin:
  set pointer points to the first object by calling setFirstDetected() in
  DetectedDatabase;
End
```

Name: getNextDetected

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    set pointer points to the next object by calling getNextDetected() in
    DetectedDatabase;
End
```

Name: turnOff

Input: ty integer, id integer

Output: integer

Description:

Pseudo-code:

```
Begin:
    assign 0 to State for object.ID=id for Radar 1, for Sonar 2.
    return State;
End
```

Name: turnOn

Input: ty integer, id integer

Output: integer

Description:

Pseudo-code:

```
Begin:
    assign 1 to State for object.ID=id for Radar 1, for Sonar 2.
    return State;
End
```

Name: - communicationBase

Input: none

Output: none

Description: virtual destructor

Pseudo-code:

```
Begin:
End
```

4.4.2.2 CDetected Class

Traceability to SRS

CD-004, CD-004-01, CD-004-02, CD-007, CD-008, CD-008-01, CD-008-02

Constants

NA.

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
ID	integer	Detected object ID
flag	integer	Detected object flag
type	integer	Detected object type
powerswitch	integer	Detected object power switch
pos	vector	Detected object position
velocity	vector	Detected object velocity

Public functions

Name: CDetected

Input: none

Output: none

Description: default constructor

Pseudo-code:

```
Begin:
  ID=0;
  flag = 0;
  type =0;
  powerswitch = 0;
End
```

Name: CDetected

Input: de CDetected &

Output: none

Description: constructor

Pseudo-code:

```
Begin:
  ID = de.ID;
  flag = de.flag;
  type = de.type;
  powerswitch = de.powerswitch;
  pos = de.pos;
  velocity = de.velocity;
End
```

Name: CDetected

Input: int i1, int f1, int t1, int ps1,
Vector p1, Vector s1

Output: none

Description: constructor

Pseudo-code:

```
Begin:
  ID=i1;
  flag=f1;
  type=t1;
  powerswitch=ps1;
  pos=p1;
  velocity=s1;
End
```

Name: getID
Input: none
Output: integer
Description:
Pseudo-code:
 Begin:
 return ID;
 End

Name: getFlag
Input: none
Output: integer
Description:
Pseudo-code:
 Begin:
 return flag;
 End

Name: getPosition
Input: none
Output: vector
Description:
Pseudo-code:
 Begin:
 return pos;
 End

Name: getVelocity
Input: none
Output: vector
Description:
Pseudo-code:
 Begin:
 return velocity
 End

Name: getPowerSwitch
Input: none
Output: integer
Description:
Pseudo-code:
 Begin:
 return powerswitch;
 End

Name: setDetData
Input: vehicle BaseShip*
Output: none
Description:
Pseudo-code:

```
Begin:
  set position, type;
  set ID, flag, velocity
  Switch on ship type to call their setPowerswitch() function;
End
```

Name: setID
Input: id Integer
Output: none
Description:
Pseudo-code:
Begin:
 ID = id;
End

Name: setFlag
Input: fl Integer
Output: none
Description:
Pseudo-code:
Begin:
 flag = fl
End

Name: setPos
Input: posit Vector
Output: none
Description:
Pseudo-code:
Begin:
 pos = posit
End

Name: setPowerSwitch
Input: ps Integer
Output: none
Description:
Pseudo-code:
Begin:
 powerswitch = ps;
End

Name: setType
Input: ty Integer
Output: none
Description:
Pseudo-code:
Begin:
 type = ty;
End

Name: setVelocity
Input: ve Vector
Output: none
Description:
Pseudo-code:
 Begin:
 velocity = ve;
 End

Name: -CDetected
Input: none
Output: none
Description: distructor
Pseudo-code:
 Begin:
 End

4.4.2.2.3 DetectedDatabase Class

Traceability to SRS
CD-004, CD-008

Constants

NA.

Private data members

list	DetList	typedef vector<CDetected*> DetList;
itCurrDetected;	DetList::iterator	Iterator to the vector of DetList

Public functions

Name: CdetectedDatabase
Input: none
Output: none
Description: default constructor
Pseudo-code:
 Begin:
 End

Name: -CDetectedDatabase
Input: none
Output: none
Description: distructor
Pseudo-code:
 Begin:
 End

Name: addDetected

Input: det CDetected*

Output: none

Description:

Pseudo-code:

```
Begin:
    //Call Vector push function
    list.push_back( det );
End
```

Name: getDetected

Input: none

Output: CDetected

Description:

Pseudo-code:

```
Begin:
    CDetected det ;           //create new pointer.
    if( itDetList < list.end() ) //get detected object pointed by iterator
    Det.getDetData( *itCurDetect )//Remove the det from the database of
    messages
    delete current iterator which is list.begin() by calling erase() in
    vector;
    return det;
End
```

Name: setFirstDetected

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    set pointer to the first element of database be calling list.begin();
End
```

Name: setNextDetected

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    set pointer to the next element of database by increment iterator;
End
```

Name: deleteAll

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    empty list using predefined vector function;
End
```

Name: singleton

Input: none

Output: CDetectedDatabase

Description:

Pseudo-code:

```
Begin:
    static CDetectedDatabase instance;
    return instance;
End
```

4.4.2.2.4 CRadar Class

Traceability to SRS

CD-001, CD-002, AT-004, DT-004, CS-004, BS-004.

Constants

NA.

Private data members

N/A.

Public functions

Name: CRadar

Input: none

Output: none

Description: default constructor, inherit from the CommunicationBase Class

Pseudo-code:

```
Begin:
    Type=1;
End
```

4.4.2.2.5 CSonar Class

Traceability to SRS

CD-005, CD-006, SM-004

Constants

NA.

Private data members

N/A.

Public functions

Name: Csonar

Input: none

Output: none

Description: default constructor, inherit from the CommunicationBase Class

Pseudo-code:

```
Begin:
  Type=2;
End
```

4.4.2.2.6 CRadio Class

Traceability to SRS

CD-009, CD-010, CD-011, CD-012, AC-004, AT-008, DT-008, CS-008, BS-008, SM-008.

Constants

NA.

Private data members

Member Name	Type	Description
myRadioId	integer	Radio object ID
range	float	Range of Radio radius

Private functions

Name: SetRadioId

Input: int RadioId

Output: none

Description:

Pseudo-code:

```
Begin:
  myRadioId = RadioId;
End
```

Public functions

Name: CRadio

Input: none

Output: none

Description: default constructor

Pseudo-code:

```
Begin:
End
```

Name: CRadio
Input: RadioId Integer
Output: none
Description: constructor
Pseudo-code:

```
Begin:  
    SetRadioId( RadioId );  
    range=1000.0;  
End
```

Name: SendMessage
Input: CMessage & Msg
Output: none
Description:
Pseudo-code:

```
Begin:  
    Msg.updateSenderInfo();  
    MESSAGE_DB.AddOneMsgIntheList(Msg);  
End
```

Name: ReceiveMessage
Input: none
Output: none
Description:
Pseudo-code:

```
Begin:  
    CMessage *msg = MESSAGE_DB.GetMyMsg( myRadioId );  
    return *msg;  
End
```

Name: turnOff
Input: none
Output: State integer
Description:
Pseudo-code:

```
Begin:  
    assgin 0 to State;  
    return State;  
End
```

Name: turnOn
Input: none
Output: none
Description:
Pseudo-code:

```
Begin:  
    assgin 1 to State;  
    return State;  
End
```

Name: DeleteMessages

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    MESSAGE_DB.DeleteMyMessages(myRadioId);
End
```

Name: -CRadio

Input: none

Output: none

Description: virtual destructor

Pseudo-code:

```
Begin:
End
```

4.4.2.2.7 CMessage Class

Traceability to SRS

CD-011, CD-012

Constants

NA.

Private data members

Member	Type	Description
Msg	Message	struct define Message include senderID, receiverID, senderType, command, sender Position, destination position and enemyInfo of CDetected type.
pVehicle	BaseShip*	Pointer variable of BaseShip type to indicate the ship information.

Public functions

Name: CMessage

Input: none

Output: none

Description: default constructor

Pseudo-code:

```
Begin:
    pVehicle=0;
    Msg.sPos = Position(0,0,0);
    Msg.dPos = Position(0,0,0);
    Msg.senderID = 0;
    Msg.senderType = 0;
    Msg.receiverID = 0;
    Msg.command = 0;
End
```

Name: CMessage

Input: baseClass *aVehicle

Output: none

Description: constructor

Pseudo-code:

```
Begin:
    pVehicle=aVehicle;
    Msg.sPos = pVehicle->getPosition();
    Msg.dPos = Position(0,0,0);
    Msg.senderID = pVehicle->getID();
    Msg.senderType = pVehicle->getType();
    Msg.receiverID = 0;
    Msg.command = 0;
End
```

Name: validToSend

Input: none

Output: bool

Description:

Pseudo-code:

```
Begin:
    return (pVehicle!=0);
End
```

Name: SetMsgData

Input: Message *outMsg

Output: none

Description:

Pseudo-code:

```
Begin:
    set enemyInfo to outMsg
End
```

Name: GetMsgData

Input: Message inMsg

Output: none

Description:

Pseudo-code:

```
Begin:
    Put the inMsg to Msg struct;
End
```

Name: updateSenderInfo

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    Read the pVehicle info to Msg struct;
End
```

Name: SetSenderId

Input: int psId

Output: none

Description:

Pseudo-code:

```
Begin:
    Msg.senderID = psId;
End
```

Name: getSenderFlag

Input: none

Output: char

Description:

Pseudo-code:

```
Begin:
    if (pVehicle!=0) return pVehicle->getFlag();
    else return 'f';
End
```

Name: GetSenderId

Input: none

Output: integer

Description:

Pseudo-code:

```
Begin:
    return Msg.senderID;
End
```

Name: SetSenderType

Input: int psType

Output:

Description:

Pseudo-code:

```
Begin:
    Msg.senderType = psType;
End
```

Name: GetSenderType

Input: Message inMsg

Output: integer

Description:

Pseudo-code:

```
Begin:
    return Msg.senderType;
End
```

Name: SetReceiverId

Input: int prId

Output:

Description:

Pseudo-code:

```
Begin:
    Msg.receiverID = prId;
End
```

Name: GetReceiverId
Input: Message inMsg
Output: Integer
Description:
Pseudo-code:
 Begin:
 return Msg.receiverID;
 End

Name: SetCommand
Input: int pCommand
Output: none
Description:
Pseudo-code:
 Begin:
 Msg.command = pCommand;
 End

Name: GetCommandId
Input: none
Output: Integer
Description:
Pseudo-code:
 Begin:
 return Msg.command;
 End

Name: SetSenderPosition
Input: Vector Pos
Output: Integer
Description:
Pseudo-code:
 Begin:
 Msg.sPos[1] = Pos[1];
 Msg.sPos[2] = Pos[2];
 Msg.sPos[3] = Pos[3];
 End

Name: GetSenderPosition
Input: none
Output: Vector
Description:
Pseudo-code:
 Begin:
 return Msg.sPos;
 End

Name: SetDestinationPosition

Input: Vector Pos

Output: none

Description:

Pseudo-code:

```
Begin:
    Msg.dPos = Pos;
End
```

Name: GetDestinationPosition

Input: none

Output: Vestor

Description:

Pseudo-code:

```
Begin:
    return Msg.dPos;
End
```

Name: SetDetectedInfo

Input: CDetected Det

Output: none

Description:

Pseudo-code:

```
Begin:
    Msg.enemyInfo = Det;
End
```

Name: GetDetectedInfo

Input: none

Output: CDetected

Description:

Pseudo-code:

```
Begin:
    return Msg.enemyInfo;
End
```

4.4.2.2.8 MessageDatabase Class

Traceability to SRS

CD-011, CD-012

Constants

NA.

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
list	MsgList	Typedef std::vector<Message*> MsgList

Private functions

Name: MessageDatabase

Input: none

Output: none

Description: constructor

Pseudo-code:

```
Begin:
    Message      *msgData= new Message();//member data
    list.push_back(msgData);
End
```

Public functions

Name: ~MessageDatabase

Input: none

Output: none

Description: virtual destructor

Pseudo-code:

```
Begin:
End
```

Name: DeleteAllMsg

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    list.clear();
End
```

Name: singleton

Input: none

Output: CMessageDatabase&

Description:

Pseudo-code:

```
Begin:
    static CMessageDatabase instance;
    return instance;
End
```

Name: AddOneMsgIntheList

Input: CMessage & Msg

Output: none

Description:

Pseudo-code:

```
Begin:
    check if this Msg is valid;
    keep this message to the list;
End
```

Name: GetMyMsg

Input: int pRadioId

Output: CMessage

Description: Get the message from the database

Pseudo-code:

Begin:

return the first message in the list with receiverID equal to pRadioId;

return NULL if no message with this receiverID.

delete this message

End

Name: DeleteMyMessages

Input: int pRadioId

Output: none

Description:

Pseudo-code:

Begin:

delete the all message in the list with receiverID equal to pRadioId;

End

4.4.3 Ship and Aircraft Detailed Design

The Ship and Aircraft subsystem is composed of Aircraft Carrier, Aircraft, Destroyer, Cruiser, Battleship and Submarine. All of them are derived from the base ship and Aircraft class. The derived class feature is described in each sub section of this part. In module detailed design section, the modules of this subsystem are diagrammed in UML and designed in such a way that this module can be implemented easily using MFC. The architecture of this subsystem is shown in Figure 4-12.

4.4.3.1 Module Detailed Design

The class operation and attribute are not listed in the class diagram for class Captain, WeaponLauncher, WeaponOfficer, RadioOfficer, RadarOfficer, SonarOfficer, NavigationOfficer and BaseShip class. Refer to the section of **Description of Class Members and Members Functions** for each class. See Figure 4-12 for a diagram representing the detailed design.

4.4.3.2 Class Definition

4.4.3.3 Description of Class Members and Member Functions

The traceability of the class design to SRS requirement is listed for each class. The constants and private data member of class are described in the Constant table and Private data member table. In the description of function, when one function needs to use another function of other class, we use sign →. The left side of sign → is the class name and the right side is the function type. This applies to all class descriptions in section 5.5

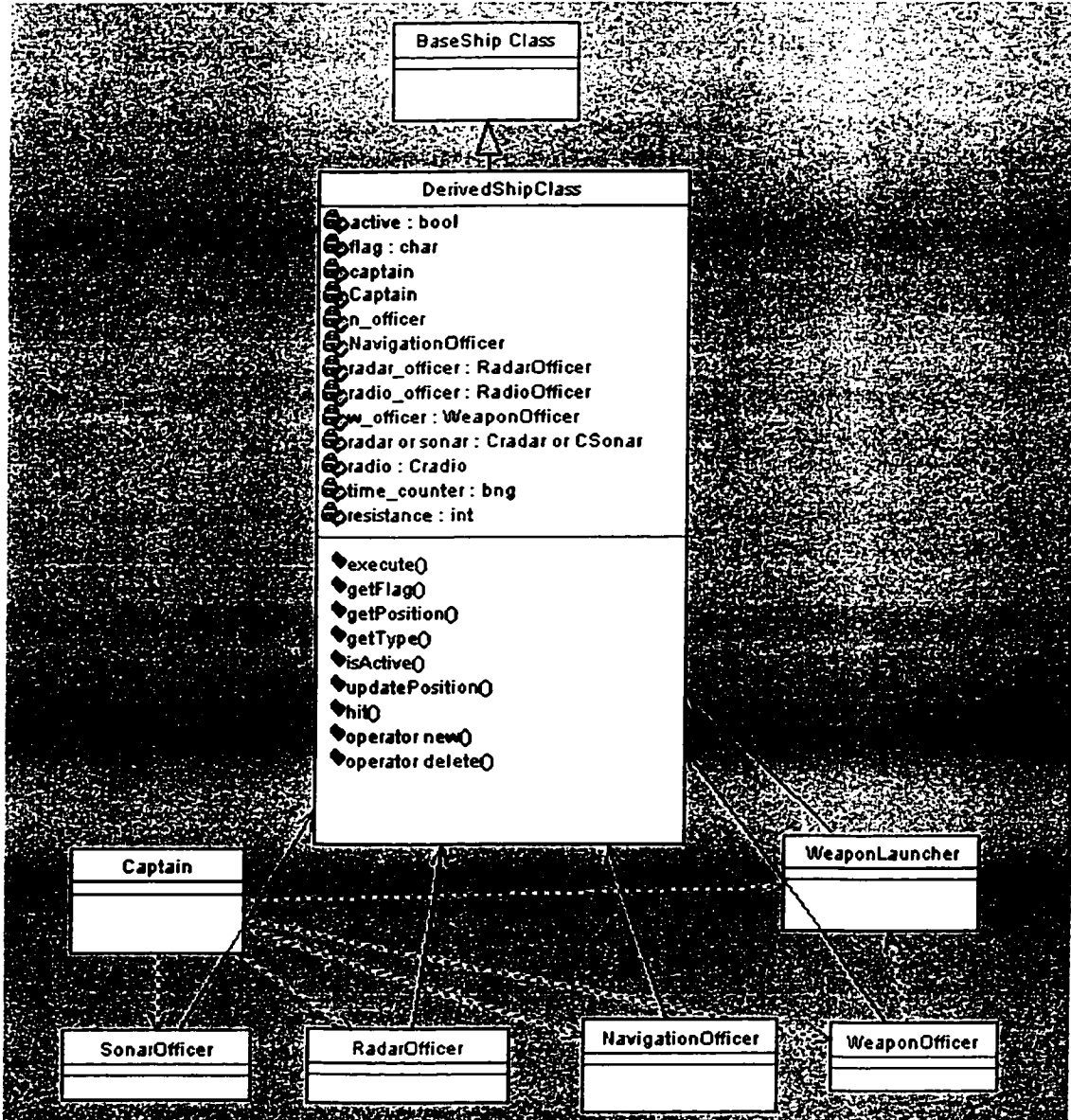


Figure 4-12 Class Diagram for BaseShip (ship and Aircraft) Module

4.4.3.3.1 BaseShip Class

Traceability to SRS

SC-001, SC-002

Constants (Defined in the derived class if different constant is used)

Name	Type	Value	Description
MAX_RESISTANCE	integer	Depends on ship	resistance value when ship and Aircraft first created
RECOVERABLE_RESISTANCE	integer	Depends on ship	minimum resistance that the can make reparation
MAX_REPAIR_TIME	integer	Depends on ship	Maximum time the ship and Aircraft needs to restore the resistance

Protected data members

Name	Type	Description
ID	integer	Ship and Aircraft ID
Check	int	used to indicate if the ship object is selected or not

Public functions

Name: BaseShip

Input: none

Output: none

Description: default constructor

Pseudo-code:

```
Begin:  
    baseClass(){ check = 0; }  
End
```

Name: getPosition

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:  
    virtual Vector getPosition() = 0;  
End
```

Name: updatePosition

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual void updatePosition() = 0;
End
```

Name: isActive

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual bool isActive() = 0;
End
```

Name: execute

Input: a double type as time to recover

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual void execute(double) = 0;
End
```

Name: getType

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual int getType() = 0;
End
```

Name: getFlag

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual char getFlag() = 0;
End
```

Name: setID

Input: none

Output: none

Description: pure virtual function

Pseudo-code:

```
Begin:
  virtual char getFlag() = 0;
End
```

Name: setID

Input: an integer type as ID

Output: none

Description: to set the object ID when it is creation

Pseudo-code:

```
Begin:
    ID = id
End
```

Name: getID

Input: none

Output: an integer type as ID

Description: to get the object ID when it is creation

Pseudo-code:

```
Begin:
    Return id
End
```

Name: setCheck

Input: an integer type as Check is true or false

Output: none

Description: to set the object Check is true or false

Pseudo-code:

```
Begin:
    check = ck
End
```

Name: getCheck

Input: none

Output: an integer type as Check is true or false

Description: to get the object Check is true or false

Pseudo-code:

```
Begin:
    return check
End
```

Name: ~BaseShip

Input: none

Output: none

Description: virtual distructor

Pseudo-code:

```
Begin:
    virtual ~baseClass(){}
End
```

4.4.3.3.2 Derived Class

The derived class includes Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer, and Submarine. Because the most of function of derived class are

same, the general function will be described for all the derived class in one pseudo code section, only the different and additional functions will be addressed with **bold style**; otherwise, the Battleship is taken as the example IN pseudo code dextrption. Radar/Sonar represents the Radar class for all the applicable ships and Sonar class for all the applicable ship in different class implementation.

Traceability to SRS
SC-001, SS-002

Constants(Redefined in Different Derived Ship Class if applicable)

Name	Type	Description
MAX_RESISTANCE	integer	resistance value when ship and Aircraft first created
RECOVERABLE_RESISTANCE	integer	minimum resistance that the can make reparation
MAX_REPAIR_TIME	integer	Maximum time the ship and Aircraft needs to restore the resistance

Private data members

ID	integer	Ship and Aircraft ID
active	bool	used to distinguish between alive and dead
flag	char	used to distinguish between allies and enemies
type	integer	used to distinguish among different ships and Aircraft
fuelamount	integer	fuel amount at ship creation
fuellimit	integer	Fuel limit when need to send request
Weaponamount	integer	Amount of on board Weapon when ship is created
captain	Captain	an object of the class Captain
n_officer	NavigationOfficer	An instance of class NavigationOfficer
Radar_officer	DetectionOffice	An instance of class DetectionOffice
Radio_officer	RadioOfficer	An instance of class RadioOfficer
w_officer	WeaponOfficer	An instance of class WeaponOfficer
w_launcher	WeaponLauncher	An instance of class WeaponLauncher
S Radar	Radar	An instance of class Radar
s Radio	Radio	An instance of class Radio
time_counter	long	records the simulation time
resistance	integer	The value stands for the status of the ship and Aircraft, i.e. how serious the ship is damaged

Public functions

Name: AircraftCarrier, Aircraft, Battleship, Cruiser, Destroyer, Submarine

Input: none

Output: none

Description: default constructor

Pseudo-code:

Begin:

```
create n_officer using default constructor
create captain
call getID() function which is in the base class to obtain the continued
ID for this object
create Radar, pass ID and sea Radar radius as parameter
create Radar_officer
create Radio_officer
create Radio, pass ID as parameters for derived object
create w_officer
create w_launcher
set flag and type for this object
resistance = MAX_RESISTANCE;
active = true;
time_counter = 0;
```

End

Name: AircraftCarrier, Aircraft, Battleship, Cruiser, Destroyer, Submarine

Input: fl: char, cPos: Vector, dPos: Vector

Output: none

Description: constructor

Pseudo-code:

Begin:

```
create n_officer, pass cPos, dPos as parameters
create captain
call getID() function which is in the base class to obtain the ID of this
object
create Radar, pass ID and sea Radar radius as parameter
create Radar_officer
create Radio_officer
create Radio, pass ID as parameters
create w_officer (Not for AircraftCarrier Class)
create w_launcher (Not for AircraftCarrier Class)
flag = fl;
type = 1 to 6; //SC assign integer 1 for AircraftCarrier, 2 for
Aircraft,3 for Cruiser, 4 for Destroyer,5 for the type Battleship and 6
for Submarine.
resistance = MAX_RESISTANCE;
active = true;
time_counter = 0;
```

End

Name: -AircraftCarrier, -Aircraft, -Battleship, -Cruiser, -Destroyer, -Submarine

Input: none
Output: none
Description: destructor
Pseudo-code:
 Begin:
 End

Name: execute
Input: t: integer
Output: void
Description: update the ship or Aircraft status
Pseudo-code:
 Begin:
 time_counter + 1;
 w_launcher → deleteWeapon(); (Not for AircraftCarrier Class)
 captain → updateCaptain(t, Radar_officer, Radio_officer, n_officer,
 w_officer, w_launcher, Radar, Radio, time_counter);
 (Not for AircraftCarrier Class)
 captain → updateCaptain(t, Radar_officer, Radio_officer, n_officer,
 Radar, Radio, time_counter); (for AircraftCarrier Class)
 updateStatus(t);
 End

Name: getFlag
Input: none
Output: char
Description: get the flag of the ship or Aircraft, 'B' OR 'R'
Pseudo-code:
 Begin:
 return flag;
 End

Name: getType
Input: none
Output: integer
Description: get the ship or Aircraft type
Pseudo-code:
 Begin:
 return type;
 End

Name: isActive
Input: none
Output: bool
Description: check if the Battleship is alive or dead
Pseudo-code:
 Begin:
 return active;
 End

Name: getPosition

Input: none
Output: Vector
Description: get position of the ship or Aircraft
Pseudo-code:
 Begin:
 return n_Officer → getPosition();
 End

Name: updatePosition
Input: none
Output: void
Description: update position from last snapshot to this snapshot
Pseudo-code:
 Begin:
 n_Officer → updatePosition()
 End

Name: hit
Input: firePower: integer
Output: void
Description: used to decrease resistance points when ship or Aircraft is hit
Pseudo-code:
 Begin:
 resistance = resistance - power;
 End

Name: * operator new
Input: size_t s
Output: void
Description: overloading operator: create an object, register this object to the Simulation Controller and return this object. Simulation Controller will provide code.
Pseudo-code:
 Begin:
 create an object and register this object to the Simulation Controller;
 return this object;
 End

Name: operator delete
Input: void * mem
Output: void
Description: overloading operator: delete this object; remove the object. registration from the Simulation Controller. Simulation Controller will provide code
Pseudo-code:
 Begin:
 delete this object;
 remove the object registration from Simulation Controller;
 End

Private functions

Name: updateStatus

Input: t: integer

Output: void

Description: update the status(alive or dead)

Pseudo-code:

```
Begin:
    if resistance <= 0 or captain → isCrash() = true, set active = false
    if resistance > RECOVERABLE_RESISTANCE and < MAX_RESISTANCE
        call resistanceRecover(t)
End
```

Name: resistanceRecover

Input: t: integer

Output: void

Description: used to recover resistance point

Pseudo-code:

```
Begin:
    resistance = resistance + (MAX_RESISTANCE - RECOVERABLE_RESISTANCE) * t
    /    MAX_REPAIR_TIME;
    if resistance > MAX_RESISTANCE, resistance = MAX_RESISTANCE;
End
```

Name: getResistance

Input: none

Output: integer

Description: get resistance point

Pseudo-code:

```
Begin
    return resistance;
End
```

Name: fuelRequest

Input: Integer

Output: bool

Description: if true, the ship or Aircraft get the fuel filling from the SC base supplier

Pseudo-code:

```
Begin:
    If(fuelamount of base supplier >=fuelamount request)
    {
        Basesupplier->deductFuel(fuelamount);
        return true;
    }
    else return false;
End
```

Name WeaponRequest (Not for AircraftCarrier class)

Input: Integer

Output: bool

Description: if true, the ship or Aircraft get the Weapon needed from the SC base supplier

Pseudo-code:

```
Begin:
  If(Weaponamount of base supplier >=Weaponamount request)and
  Weapontype==ship's Weapon type)
  {
    Basesupplier->createWeapon();
    Return true;
  }
  else return false;
End
```

4.4.3.3 Captain Class

Traceability to SRS

AC-001, AC-001-01, AC-001-02, AC-003, AC-009, AC-010, AC-011, AC-012, AC-013, AC-025, AC-026. AC-018 to AC-024. AT-001-01,AT-001-02, AT-002, AT-003, AT-013 to AT-018,AT-032 to AT034, AT-024 to AT-031. DT-001-01,DT-001-02, DT-002, DT-003, DT-013 to DT-018,DT-032 to DT034, DT-024 to DT-031. CS-001-01,CS-001-02, CS-002, CS-003, CS-013 to CS-018,CS-032 to CS034, CS-024 to CS-031. BS-001-01,BS-001-02, BS-002, BS-003, BS-013 to BS-018,BS-032 to BS034, BS-024 to BS-031. SM-001-01,SM-001-02, SM-002, SM-003, SM-013 to SM-018,SM-032 to SM034, SM-024 to SM-031.

Constants

NA

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
friend_list	ObjectList	the node of the ObjectList will contain information of id, position, flag, speed of object. This list contains friends info.
enemy_list	ObjectList	list after update
previous_enemy_list	ObjectList	list before update
crash	bool	if true, the ship or Aircraft collides with another object
attack_target	Detected	Target object the ship and Aircraft will attack

Public member functions

Name: Captain

Input: none

Output: none

Description: constructor

Pseudo-code:

```
Begin:
    initial friend_List ,enemy_List, and previous_enemy_list as empty list
    crash = false;
    attack_target = NULL;    //no attack target
End
```

Name: -Captain

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

Name: updateCaptain

Input: t: int, Radar: RadarOfficer&, Radio: RadioOfficer&, n_officer: navigationOfficer&, w_officer: WeaponOfficer&, w_launcher: WeaponLauncher&, Radar: Radar&, Radio: CRadio&,counter: long
(Not for AircraftCarrier Class)

Input: t: int, Radar: RadarOfficer&, Radio: RadioOfficer&, n_officer: navigationOfficer&, Radar: Radar&, Radio: CRadio&,counter: long
(for AircraftCarrier Class)

Output: void

Description: execute every time slice, to update all decisions made by captain

Pseudo-code:

```
Begin:
First step:
    update friend_list and enemy_list
    remove all elements in the friend_list;
    remove all elements in the previous_enemy_list;
    copy enemyList to previous_enemy_list;
    remove all elements in the enemyList;
Second step:
    decide if the ship collides with another object, no matter friend or
    enemy by checking both the friend-list and enemy-list. If there is one
    object is too close to the ship or Aircraft, which means that the
    distance between two object is less than one tolerant value,we think it
    collides with the ship, then the ship will sink.
    crash = true;
Third step:
    If there are any new enemies detected, send message to allies
```

```

loop the friend_list
{
  compare previous-enemy-list with friend-list, whenever find an object
  that is in friend-list and not in previous-enemy-list
  RadioOfficer/SonarOffice → sendDetectMessage (bRadio, Detected, 0)
}
Fourth step:
  if(ifAttack()==true), attack the enemy
  get current position of the ship from NavigationOfficer
  get target position, speed, ID from object attackTarget
  WeaponOfficer→prepareAttack(currPos,targetPos,targetSpeed,targetId,
  count,launcher)
  (Not for AircraftCarrier Class)
  (for AircraftCarrier Class)
Fifth step:
  adjust navigation: adjustNavigation();
End

```

Name isCrash

Input: none

Output: bool

Description: if true, the ship and Aircraft collides with other object

Pseudo-code:

```

Begin:
  return crash;
End

```

Private member functions

Name: ifAttack

Input: none

Output: bool

Description: if true, there is a specific target to attack

Pseudo-code:

```

Begin:
  case 1: there is no enemy around, return false
    if(the enemyList is empty) return false
  case 2: there are only enemies which can not be target for this object,
    return false
    --check all elements in the enemy_list from the first one to the
  case 3: there is at least one enemy for this object, for example, sea-
  borne enemy for Battleship
    //the following code take Battleship as example, it is also
    applicable for other ship or Aircraft object
    (not for AircraftCarrier Class)
    if (the number of the sea-borne is equal to one)
    then it is the intended target
    if (this object position is within the Missile range)
    {
      int wtype = WeaponOfficer → selectWeapon();
    }

```

```

int cQty = WeaponOfficer → getCannonQty();
int mQty = WeaponOfficer → getMissileQty();
if(wtype is cannon and (cQty or mQty >= 1) or wtype is Missile
and mQty >= 1))
{
attack_target = this object
return true.
}else
{
can not attack the target,
return false;
}
}
if (the target position is out of the Missile range)
return false;
if (the number of the sea-borne is more than one)
{
Compute the distance between each enemy and the Battleship
Choose the nearest one to the sea-brone as the target.
Following the same procedure as the case of having only one sea-
borne enemy
}
}
End

```

Name: adjustNavigation

Input: none

Output: void

Description: adjust navigation, speed and direction

Pseudo-code:

Begin:

```

case 1: there is no enemy within range in enemy_list at this
moment, for example, sea-borne enemy for Battleship
if(found enemies' Submarine(s) (z value of the positon is less than
0))
{
calculate the distances from enemies' Submarine(s), steer to a
direction which has angle  $\theta$  with current direction to get away
from enemy.
NavigationOfficer → steer( $\theta$ );
double accl= 525; //525km/hr2 for Battleship
NavigationOfficer → adjustSpeed(accl, MAX_SPEED);
}
if(no friend on the heading direction and |speed|<Max)
{
find a direction which has angle  $\theta$  with current direction where
there is no friends and object on the way;
NavigationOfficer → steer( $\theta$ );
}
}
if(friends or object on the way)
{
find a direction which has angle  $\theta$  with current direction where
there is no friends and object on the way;
NavigationOfficer → steer( $\theta$ );
double deceleration = -700; // -700km/hr2
}
}

```

```

        for Battleship
        NavigationOfficer → adjustSpeed(deceleration, 0);
    }

    case 2:
    if (ifAttack() = true)
    {
        find a closest target direction on which there is no friend;
        NavigationOfficer → cruise(t, attack_target.position);
        double deceleration = -700; // -700km/hr2 for Battleship
        NavigationOfficer → adjustSpeed(deceleration, 0);
    }
}
End

```

Name: addToFriendList

Input: Detected

Output: void

Description: add new detected or received friend info to friend_list

Pseudo-code:

```

Begin:
    add Detected to friend_list
End

```

Name: addToEnemyList

Input: Detected

Output: void

Description: add new detected or received enemy info to enemy_list

Pseudo-code:

```

Begin:
    add Detected to enemy_list
End

```

Name iffuelEmpty

Input: none

Output: bool

Description: if true, the ship or Aircraft has no fuel any more

Pseudo-code:

```

Begin:
    If(fuelamount==0) Return true;
    else return false;
End

```

4.4.3.3.4 Radar/Sonar Officer

Traceability to SRS

AT-004 to AT-007, CS-004 to CS-007, DT-004 to DT-007,BS-004 to BS-007, SM-004 to SM-007.

Constants

NA

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
det	Detected	a Detected object, store object information
Radar on/Sonar on	bool	Radar/Sonar is on if true

Public member functions

Name: RadarOfficer**Input:** none**Output:** none**Description:** constructor**Pseudo-code:**

```
Begin:
    det = Detected ();
    Radar_On = true; or Sonar_on=true;
End
```

Name: -RadarOfficer**Input:** none**Output:** none**Description:** destructor**Pseudo-code:**

```
Begin:
End
```

Name: turnOffRadar/turnOffSonar**Input:** Radar/Soanr: Radar&/Sonar&**Output:** void**Description:** turn off Radar/Sonar**Pseudo-code:**

```
Begin:
    Radar/Sonar → turnoff();
End
```

Name: turnOnRadar/turnOnSonar**Input:** Radar/Soanr: Radar&/Sonar&**Output:** void**Description:** turn on Radar/Sonar**Pseudo-code:**

```
Begin:
    Radar/Sonar → turnon();
End
```

Name: getNumOfDetected**Input:** Radar/Sonar: Radar&/Sonar&, pos: Vector**Output:** integer

Description: the function pass the ship position in order to know the center of the Radar/Sonar. It is used to get number of detected objects

Pseudo-code:

```
Begin:
    return Radar/Sonar → emitReceive(pos);
End
```

Name: getFirstDetected

Input: Radar/Sonar: Radar&/Sonar&,

Output: Detected

Description: get the first detected object information

Pseudo-code:

```
Begin:
    Radar/Sonar → goFirstDetected();
    return Radar/Sonar → getDetectedInfo();
End
```

Name: getNextDetected

Input: Radar/Sonar: Radar&/Sonar&,

Output: Detected

Description: get the next detected object information

Pseudo-code:

```
Begin:
    Radar/Sonar → goNextDetected();
    return Radar/Soanr → getDetectedInfo();
End
```

4.4.3.3.5 RadioOfficer Class

Traceability to SRS

AC-004 to AC-008, AT-008, AT-012, CS-008 to CS-012, DT-008 to DT-012, BS-008 to BS-012, SM-008 to SM-012.

Constants

NA

Private data members

Name	Type	Description
message	Cmessage	an instance of CMessage, store message info

Public member functions

Name: RadioOfficer

Input: object: BaseShip

Output: none

Description: constructor

Pseudo-code:

Begin:

```
message = CMessage(object); //communication group ask for this
```

End

Name: RadioOfficer

Input: object: BaseShip

Output: none

Description: default constructor

Pseudo-code:

Begin:

```
message = CMessage();
```

End

Name: -RadioOfficer

Input: none

Output: none

Description: destructor

Pseudo-code:

Begin:

End

Name: sendDetectMessage

Input: Radio: CRadio&, det: Detected, id: integer

Output: void

Description: send the detected message to a specific object or broadcast

Pseudo-code:

Begin:

```
message → setReceiverId(id); // set 0 for message broadcast
```

```
message → setDetectedInfo(det);
```

```
Radio → sendMessage(message);
```

End

Name: sendDesPosMessage

Input: Radio: CRadio&, pos: Vector, id: integer

Output: void

Description: send the destination position to a specific object or broadcast

Pseudo-code:

Begin:

```
message → setReceiverId(id); // set 0 for message broadcast
```

```
message → setDestinationPosition(pos);
```

```
Radio → sendMessage(message);
```

End

Name: receiveMessage

Input: Radio: Radio&

Output: CMessage

Description: receive message by using Radio

Pseudo-code:

Begin:

```

    return message = Radio → receiveMessage();
End

```

Name: getMessage

Input: none

Output: CMessage

Description: get the value of data member message

Pseudo-code:

```

Begin:
    return message;
End

```

4.4.3.3.6 NavigationOffice Class

Traceability to SRS

AC-001, AC-001-01, AC-001-02, AC-003. AT-001-01, AT-001-02, AT-002, AT-003. DT-001-01, DT-001-02, DT-002, DT-003. CS-001-01, CS-001-02, CS-002, CS-003. BS-001-01, BS-001-02, BS-002, BS-003. SM-001-01, SM-001-02, SM-002, SM-003.

Constants

Constant Name	Type	Description
MAX_SPEED	integer	Maximum speed of the Battleship

Private data members

Member Name	Type	Description
Curr_position	Vector	Current position
Temp_position	Vector	Temporarily position before update
velocity	Vector	including direction

Public member functions

Name: NavigationOfficer

Input: curPos: Vector, desPos: Vector, spd: Vector

Output: none

Description: constructor

Pseudo-code:

```

Begin:
    curr_position = curPos;
    temp_position = curPos;
    velocity = (desPos - curPos)*MaxSpeed;
End

```

Name: -NavigationOfficer

Input: none

Output: none

Description: destructor

Pseudo-code:

 Begin:

 End

Name: cruise

Input: Δt : integer, decPos: Vector

Output: void

Description: navigate the ship or Aircraft from current position to the destination position

Pseudo-code:

 Begin:

 //ship decelerate at the original Velocity (Vector), and adjust direction of Velocity accordingly every t interval. See the figure below to understand the algorithm.

 //calculate direction needed to get to target position.

 Vector direction = targetPos - curr_position;

 //calculate Velocity on original direction after Δt .

 Vector velocity_ori = velocity-a Δt ;

 //calculate Vector Velocity on target direction.

 Vector velocity_des= direction/length()*|velocity|; //target Velocity

 //calculate the actual Velocity at this time slot and update velocity of

 //ship or aircraft.

 velocity = velocity_des - velocity_ori;

 //calculate the position after Δt and update position of ship or aircraft.

 curr_position = curr_position + Velocity Δt ;

 End

Name: getPosition

Input: none

Output: Vector

Description: get current position

Pseudo-code:

 Begin:

 return curr_position;

 End

Name: getVelocity

Input: none

Output: Vector

Description: get current velocity

Pseudo-code:

 Begin:

 return velocity;

 End

Name: setPosition

Input: pos: Vector

Output: void

Description:

Pseudo-code:

```
Begin:
  curr_position = pos;
End
```

Name: setVelocity

Input: vel: Vector

Output: void

Description: set velocity

Pseudo-code:

```
Begin:
  Velocity = vel;
End
```

Name: adjustSpeed

Input: accl: double, targetSpeed: double

Output: void

Description: adjust the velocity with certain acceleration to the target velocity.

Pseudo-code:

```
Begin:
  //accelerate to a Velocity bigger than original one.
  if ((accl>0) and (targetVelocity>velocity))
    velocity = velocity + acclΔt;
  //decelerate to a velocity smaller than original
  else if (accl<0 & (targetVelocity<Velocity)&(targetVelocity>=0))
    {temp_Velocity = velocity + acclΔt;
     if (temp_Velocity<0) velocity =0;
     else velocity = velocity + acclΔt;
  }
End
```

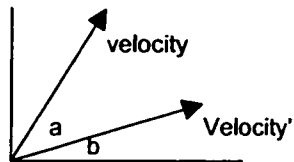
Name: steer

Input: angle: float

Output: void

Description: changes the navigation direction of the ship or Aircraft by angle with the current direction.

Pseudo-code:



Begin:

```
tan(b)=velocity.y/velocity.x;
tan(a+b) = velocity'.y/velocity'.x;
```

End

Name: updatePosition

Input: none

Output: void

Description: updates the current position of the ship or Aircraft with temp_position

Pseudo-code:

```
Begin:
    curr_position = temp_position;
End
```

4.4.3.3.7 Weapon Officer Class

Traceability to SRS

AT-019 to AT-023. CS-019 to CS-023. DT-019 to DT-023. BS-019 to BS-023. SM-019 to SM-023

Constants

Name	Type	Description
CANNON_QTY	integer	The quantity of cannon (Battleship eg.)
MISSILE_QTY	integer	The quantity of sea-sea Missile (Battleship eg.)

Private data members

Name	Type	Description
cannon_qty	integer	contain the quantity of cannon(Battleship eg.)
Missile_qty	integer	contain the quantity of Missile(Battleship eg.)
is_cannon	integer	record the selected Weapon: 1 denotes cannon, 0 denotes Missile(Battleship eg.)
target_id	integer	record target id
first_aim_time	Long integer	record the first aim time
last_fire_time	Long integer	record the last fire time

Public member functions

Name: WeaponOfficer

Input: none

Output: none

Description: Constructor initializes attributes

Pseudo-code:

```
Begin:
    is_cannon = 0;
    target_id = 0;
    first_aim_time = 0;
    last_fire_time = 0;
    //For Battleship
    cannon_qty = CANNON_QTY;
    Missile_qty = MISSILE_QTY;
End
```

Name: -WeaponOfficer

Input: none

Output: none

Description: Destructor

Pseudo-code:

Begin:

End

Name: prepareAttack

Input: cp:Vector, tp:Vector, ts:Vector, tid:int, ct:long, launcher: WeaponLauncher

Output: void

Description: directly or indirectly do every prepare work for attack enemy: select Weapon, check if the target id has been changed and the selected Weapon has been changed, consider aim latency time and fire latency time, call the function of launcher to create Weapon and fire it, and finally update the quantity of Weapon.

Pseudo-code:

```
Begin:
  // check if the target Id has been changed.
  if(target_id isn't equal to tid, i.e. the target Id has been changed
  comparing with the last target Id)
    {Record target Id, first aim time, last fire time and the choosed
    Weapon at this snapshot:
    target_id = tid;
    first_aim_time = ct;
    last_fire_time = 0;
    is_cannon = selectWeapon(cp, tp);
    }
  if(target_id = tid, i.e. the target Id hasn't been changed)
    {//choose Weapon and record it at this snapshot:
    int n = selectWeapon(cp, tp);
    // check if the selected Weapon has been changed. For example, the
    Battleship has two types of Weapon as canon and Missile:
    if((is_cannon isn't equal to n, i.e. the selected Weapon has been
changed)
      {record first aim time, last fire time and the chosen Weapon again
      at this snapshot:
      first_aim_time = ct;
      last_fire_time = 0;
      is_cannon = n;}
    if(is_cannon = n, i.e. the selected Weapon hasn't been changed)
      {if(the choosed Weapon is cannon and aim time >= latency time and
      fire time >= fire interval for continually firing cannon)
        {compute the intended destination of cannon:
        launcher-> aimByBallistic(cp, cs, tp, ts),
        return destination Vector: dp;
        Create and fire cannon shell:
        launcher->fireCannonShell(cp, dp);
        Record last fire time: last_fire_time = ct;
        Update the quantity of cannon: updateCannonQty();}
      if(the choosed Weapon is Missile and aim time >= latency time and
      fire time >= fire interval for continually firing Missile)
        {Create and fire Missile:
        launcher->fireMissile(cp, tp);
        Record last fire time: last_fire_time = ct;
```



```
        Update the quantity of Missile: updateMissileQty();}
    }
}
End
```

Name: cancelAttack

Input: none

Output: void

Description: cancel this attack

Pseudo-code:

```
Begin:
  //Cancel attack and initialize attributes:
  target_id = 0;
  first_aim_time = 0;
  last_fire_time = 0;
End
```

Name: selectWeapon

Input: tp: Vector, cp: Vector

Output: integer

Description: select Weapon: for example, cannon or Missile according to the distance between Battleship and target. If choose cannon, return 1; if choose Missile, return 0. Suppose that before this function is called, the quantity of Weapon has been checked.

Pseudo-code:

```
Begin:
  Suppose that before this function is called, the quantity of Weapon has
  been checked.
  Compute the distance between Battleship and target;
  if(this distance <= the range of cannon){
    if(the quantity of cannon >= 3)
    {
      Choose cannon:
      return 1;
    }
    otherwise
    {
      Choose Missile:
      return 0;
    }
  }
  if(this distance > the range of cannon)
  {
    choose Missile:
    return 0;
  }
End
```

Name: updateCannonQty (for Battleship)

Input: none

Output: void

Description: update the quantity of cannon

Pseudo-code:

```

Begin:
  Update cannon quantity (suppose that three cannon shell will be fired
  every time): cannon_qty = cannon_qty - 3
End

```

Name: updateMissileQty (for Battleship)

Input: none

Output: void

Description: update the quantity of Missile

Pseudo-code:

```

Begin:
  Update Missile quantity: Missile_qty = Missile_qty - 1
End

```

Name: getCannonQty (for Battleship)

Input: none

Output: integer

Description: return the quantity of cannon

Pseudo-code:

```

Begin:
  return the quantity of cannon;
End

```

Name: getMissileQty (for Battleship)

Input: none

Output: integer

Description: return the quantity of Missile

Pseudo-code:

```

Begin:
  return the quantity of Missile;
End

```

4.4.3.3.8 WeaponLauncher Class

Traceability to SRS

AT-021, CS-021, DT-021, BS-021, SM-021

Constants

Name	Type	Description
GRAVITY ACCELERATION	double	Physic constant (Battleship)

Private data members

Name	Type	Description
cannon_attribute	WAttribute	contain the attributes of cannon (Battleship)
Missile_attribute	WAttribute	contain the attributes of Missile (Battleship)
cannon_list	List	keep the created cannon shell until it is detonated (Battleship)

Missile_list	List	keep the created Missile until it is detonated (Battleship)
--------------	------	---

Public member functions

Name: WeaponLauncher

Input: none

Output: none

Description: constructor initializes the attributes

Pseudo-code:

 Begin:

 End

Name: -WeaponLauncher

Input: none

Output: none

Description: destructor

Pseudo-code:

 Begin:

 End

Name: aimByBallistic (for Battleship)

Input: cp: Vector, tp: Vector, ts: Vector

Output: Vector

Description: For example, Battleship compute initial velocity of cannon shell and intended destination by using ballistic trajectory formular based on some assumption

Pseudo-code:

 Begin:

 use the ballistic equation to calculate the fire angles and fire speeds of cannon shells so that they can hit the targeted ship precisely.

 The equations used here are:

$$(1) \quad V \cos \gamma t = (g t^2) / 2$$

$$V \cos \beta t = (Y_m - Y_e) - V_x t$$

$$V \cos \alpha t = (X_m - X_e) - V_y t$$

$$(\cos \alpha)^2 + (\cos \beta)^2 + (\cos \gamma)^2 = 1$$

 Note: V is the magnitude of the cannon shell speed.

α , β , γ are the fire angles of the cannon with x, y, z coordinate directions respectively

X_m , Y_m are the positions of my ship in x and y coordinates respectively

X_e , Y_e are the positions of enemy ship in x, y coordinate respectively

V_x , V_y are the speeds of enemy ship in x and y directions respectively

 From the above four functions we can derive the following equation:

$$V^2 t^2 = ((g t^2) / 2)^2 + ((Y_m - Y_e) - V_x t)^2 + ((X_m - X_e) - V_y t)^2$$

 Return the destination Vector of cannon shells;

 End

Name: fireCannonShell (for Battleship)
Input: cp: Vector, dp: Vector
Output: void
Description: create cannon shell and fire it
Pseudo-code:

```
Begin:
    create cannon_shell of WCannonShell;
    add cannon_shell to cannon_list;
    fire cannon: cannon_shell → fire(cp, dp);
End
```

Name: fireMissile (for Battleship)
Input: cp: Vector, tp: Vector
Output: void
Description: create Missile and fire it
Pseudo-code:

```
Begin:
    create sea_Missile of WMissileSeaSea;
    add Missile_list to Missile_list;
    fire Missile: sea_Missile → fire(cp, tp);
End
```

Name: deleteWeapon (for Battleship)
Input: none
Output: void
Description: delete cannons or Missiles if them have been detonated
Pseudo-code:

```
Begin:
    while(cannon_list is not empty)
    {
        if(cannon_shell → isActive() = false, i.e. the cannon has been
detonated)
            delete cannon_shell;
    }
    while(Missile_list is not empty)
    {
        if(sea_Missile → isActive() = false, i.e. the Missile has been
detonated)
            delete sea_Missile;
    }
End
```

Name: getCannonAttribute (for Battleship)
Input: none
Output: WAttribute
Description: return the attributes of cannon
Pseudo-code:

```
Begin:
    return attributes of cannon;
```

End

Name: getMissileAttribute (for Battleship)
Input: none
Output: WAttribute
Description: return the attributes of Missile
Pseudo-code:
 Begin:
 return attributes of Missile;
 End

4.4.4 Weapon Detailed Design

This section describes all the classes of Weapon subsystem of the NBSS and the functions they contain. In module detailed design section, the modules of this subsystem are diagrammed in UML and designed in such a way that this module can be implemented easily in MFC . The architecture of this subsystem is shown in Figure 4-13.

4.4.4.1 Module Detailed Design

The class operation and attribute are not listed in the class diagram for all the classes in Weapon module. Refer to the section of **Description of Class Members and Members Functions** for each class.

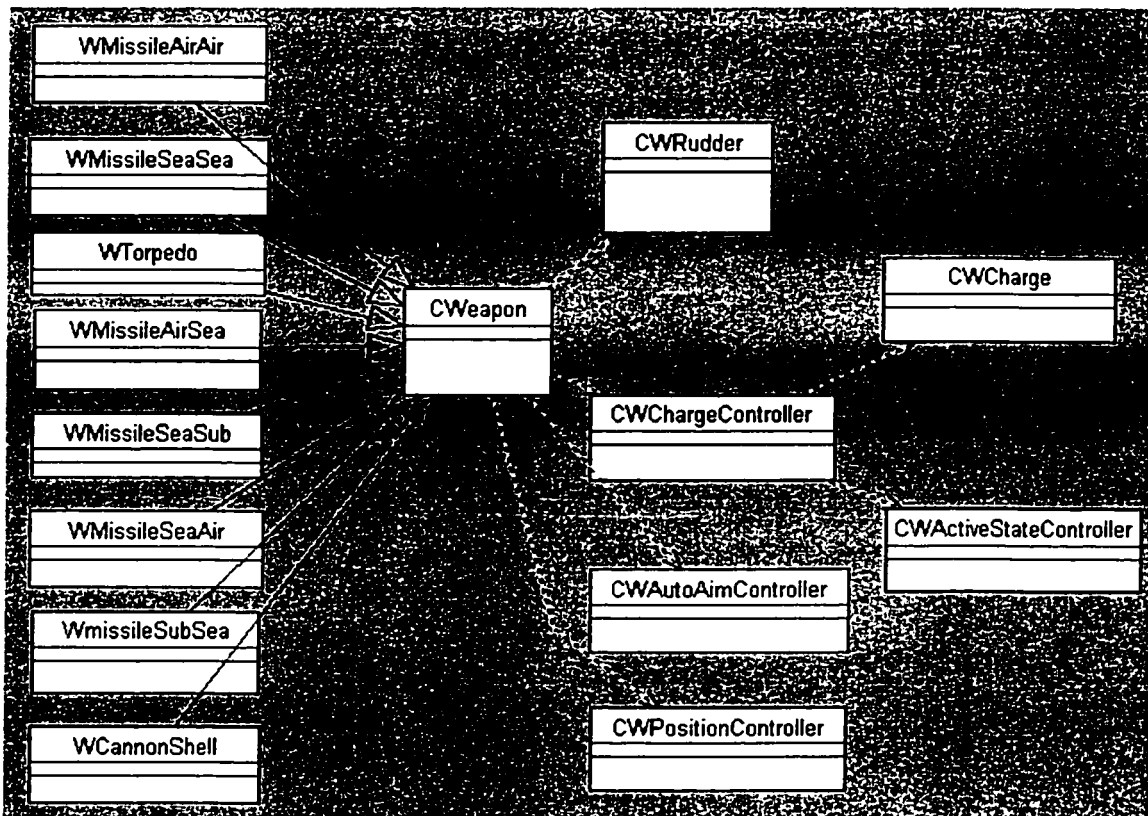


Figure 4-13 Class Diagram for Weapon Module

4.4.4.2 Class Definition

4.4.4.2.1 CWeapon

Traceability to SRS

WP-001

Constants

Name	Type	Value	Description
CHARGE_RANGE	float	Depends on Weapon	Take from structure wAttr.wMaxSpeed/15

Private data members

Name	Type	Description
wFlag	integer	friend and enemy
time_len	double	record time length for each loop
wPosContr	CWPositionController	
wAimContr	CWAutoAimController	
wChgContr	CWChargeController	
wStaContr	CWActiveStateController	

Protected data members

Name	Type	Description
wAttr	struct WAttribute	Weapon Attribute Structure

Private member functions

Name: checkValidPosition

Input: none

Output: integer

Description:

Pseudo-code:

```
Begin:
    call checkValidPosition position controller
    For Cannon Shell: detonate()
    For Carrier Weapons: launched()
    For AutoAiming Weapons: detonate()
End
```

Protected member functions

Name: Initialize

Input: TYPE_WEAPON id, int flag, CWCharge *charge

Output: none

Description: function overloading for different type of Weapon

Pseudo-code:

```
Begin:
    initialize Weapon instead of constructor function
    // differ three kinds of Weapons to implement
    // Cannon Shell, carrier Weapons, auto aiming Weapons.
    // Cannon Shell:    only Charge
    // Carrier Weapons:    only carried Weapon pointer
    // Auto Aiming Weapons: Rudder, Charge, Radar/Sonar.
    For Auto Aiming Weapons.Rudder, Charge, Radar/Sonar use function
    Initialize(TYPE_WEAPON id, int flag, CWRudder *rud, CWCharge *charge,
    void *RSpt)
End
```

Public member functions

Name: CWeapon

Input: none

Output: none

Description: Default Constructor to initializes attributes

Pseudo-code:

```
Begin:
    wFlag(0),
    wCarriedWeapon((CWeapon *)NULL)
    //Initialize(WeaponType);
End
```

Name: getFlag

Input: none

Output: char

Description:

Pseudo-code:

```
Begin:
    return (char) wFlag
End
```

Name: setFlag

Input: char flag

Output: none

Description:

Pseudo-code:

```
Begin:
    if( wFlag == flag ) return;
    wFlag = flag;
    wAimContr.setFlag(flag);
    wChgContr.setFlag(flag);
End
```

Name: getPosition

Input: none

Output: Position

Description: return current position from PositionController

Pseudo-code:

```
Begin:
    return wPosContr.getPosition();
End
```

Name: getType

Input: none

Output: integer

Description:

Pseudo-code:

```
Begin:
    return wAttr.wType
End
```

Name: isActive()

Input: none

Output: bool

Description: return state from StateController

Pseudo-code:

```
Begin:
    return wStaContr.getState();
End
```

Name: updatePosition

Input: none

Output: none

Description:

Pseudo-code:

```
Begin:
    wPosContr.updatePosition();
End
```

Name: getAttribute

Input: none

Output: Wattribute

Description:

Pseudo-code:

```
Begin:
    return wAttr;
End
```

Name: locateTargetPosition

Input: Position curPos

Output: integer

Description: Only for carried Weapon: SeaSeaMissile and Torpedo

Pseudo-code:

```
Begin:
    set target position for carried Weapon
    return 0 for successful; return 1 for fail;
End
```

Name: setInitTargetPosition

Input: Position targetPos

Output: none

Description: Only for carried Weapon: SeaSeaMissile and Torpedo

Pseudo-code:

```
Begin:
  Call wAimContr.setInitTargetPosition(targetPos);
  set target position for carried Weapon by calling
  wCarriedWeapon->setInitTargetPosition(targetPos);
End
```

Name: fire

Input: Position curPos, Position destPos

Output: none

Description:

Pseudo-code:

```
Begin:
  calls ActiveStateController.setState(ACTIVE) to set active state.
  calls PositionController.setInitPosition(init) to set initial position.
  calls AutoAimController.setInitTargetPosition(target) to set target
position.
  calls PositionController.setDestinationPosition() to set destination
position.
  But for carrier Weapon, this function create Weapon object that will be
  launched by carrier Weapon.
  Call ActiveStateController.setState(ACTIVE)
  Call PositionController.setInitPosition(initial position)
  Call PositionController.setDestinationPosition(destination)
  If Weapon type is Carrier Weapon like Sub-Sea Missile and Sea-Sub Missile
  Then
  Create launchedWeapon
  Call launchedWeapon.setInitTargetPosition
  // launchedWeapon is a Weapon carried by this carrier Weapon
  else
  call AutoAimController.setInitTargetPosition
  endif
End
```

Name: execute

Input: double time

Output: none

Description: main function to control all modules in controller

Pseudo-code:

```
Begin:
  If Weapon type is not carrier type Weapon like Sub-Sea Torpedo/Missile
  and Sea-Sub Missile/Torpedo
  Then
  Call chargecont.checkDetonateRange
  Endif
  If Weapon type is auto aim Weapon
  Then
  Call AutoAimController.locateTargetPosition
  Call AutoAimController.updateVelocity
  updateVelocity is called in locateTargetPosition()
  Endif
  If Weapon type is Carrier Weapon like Sub-Sea Torpedo/Missile and
  Sea-Sub Missile/Torpedo
```

```

    Then
        Call launchedWeapon.locateTargetPosition
        //launchedWeapon is a Weapon carried by this carrier Weapon
    endif
    Generate a random value ram which is between 0 to 1;
    if (ram > precision) // The Weapon failed to hit the target.
        return false;
    else
        return true;          // The target was hit
    End

```

Name: checkValidPosition

Input: none

Output: integer

Description:

Pseudo-code:

```

    Begin:
        call checkValidPosition position controller
        For Cannon Shell: detonate()
        For Carrier Weapons: launched()
        For AutoAimming Weapons: detonate()
    End

```

Name: operator delete

Input: void * mem

Output: none

Description:

Pseudo-code:

```

    Begin:
        vector<baseClass*>::iterator first = SC::vpVehicles.begin(), last =
        SC::vpVehicles.end(), it;
        it = find(first, last, (baseClass*)mem);
        if(it != last)
        {
            ::delete mem;
            *it = NULL; // set mem = NULL
            SC::setDelete();
        }
        else cerr<<"Nothing can be deleted\n"; //This command is for command user
interface.
    End

```

Name: operator new

Input: size_t

Output: none

Description:

Pseudo-code:

```

    Begin:
        int id=SC::getLastID(); // assign a new index to the new object
        SC::vpVehicles.push_back(::new WMissileAirSea());
        int sz = SC::vpVehicles.size();
        SC::vpVehicles[sz-1]->setID(id);
        SC::vpVehicles[sz-1]->setCheck(0);
        SC::incrLastID();
        SC::setNew();
    End

```

```

return SC::vpVehicles[sz-1];
End

```

Name: -CWeapon

Input: none

Output: none

Description: distructor

Pseudo-code:

Begin:

End

4.4.4.2 WCommon Class

Traceability to SRS

WP-002, WP-003

Constants

Name	Type	Value	Description
DOUBLE_MAX	double	(1.0e+60)	Maximum double
INVALID_VEC	Vector	(Vector(- DOUBLE_MAX, DOUBLE_MAX, DOUBLE_MAX))	Invalid Vector for speed
W RADAR RANG	integer	50	50000 meters
MAX TARGET DIST	double	DOUBLE_MAX	Maximum target distance
WeaponTypeStart	integer	7	the begin type of Weapon
WRadar_Type	integer	0	//aiming device no.
Ballistic	integer	2	//aiming device no.
DOUBLE_PREC	double	0.00001	Precise of double
AircraftCarrier_Type	integer	1	Ship type
Aircraft_Type	integer	2	Ship type
Destroyer_Type	integer	3	Ship type
Cruiser_Type	integer	4	Ship type
Battleship_Type	integer	5	Ship type
Submarine_Type	integer	6	Ship type
HeavyCannonShell	integer	WeaponTypeStart	Weapon Type
AirAirMissile	integer	WeaponTypeStart+1	Weapon Type
AirSeaMissile	integer	WeaponTypeStart+2	Weapon Type
SeaSeaMissile	integer	WeaponTypeStart+3	Weapon Type
SeaAirMissile	integer	WeaponTypeStart+4	Weapon Type
SeaSubMissile	integer	WeaponTypeStart+5	Weapon Type
Torpedo	integer	WeaponTypeStart+6	Weapon Type
SubSeaTorpedo	integer	WeaponTypeStart+7	Weapon Type

Private data members

Name	Type	Description
struct WAttribute	struct	Weapon Attribute

Public functions

Name: IsTargetType

Input: int mytype, int targettype

Output: bool

Description:

Pseudo-code:

Begin:

Switch on the Weapon type, and check if the target can be hit by this type of Weapon;

End

Name: betweenTwoPosition

Input: Position destPos, Position start, Position end

Output: bool

Description:

Pseudo-code:

Begin:

Return Value: TRUE: destpos is on the line between two positions
FALSE: not on the line.

Cannon Shell should be detonated when destination position is on the line from current position to next time position.

how to check current position ??? two necessary conditions

1. the distance between destination and current position should be less than distance between current position and next time position
2. the unit of (destination - current position) should equal to the unit of (next time position - current position)

End

Name: calDestination

Input: int type ,Position curPos,Position targetPos,double range

Output: Position

Description:

Pseudo-code:

Begin:

get two project positions for current and target position
calculate maximum horizontal distance
calculate horizontal direction
convert to unit (length == 1)
calculate destination horizontal position
return position;

End

Name: IsSamePosition

Input: Position p1, Position p2

Output: bool

Description:

Pseudo-code:

Begin:

Compare the position value of x, y and z
return TRUE;//if same;
else return false;

End

Name: IsZeroDouble
Input: double db
Output: bool
Description:
Pseudo-code:
 Begin:
 If(abs(db) < DOUBLE_PREC) return TRUE;
 Else return FALSE;
 End

Name: IsSameDouble
Input: double db1, double db2
Output: bool
Description:
Pseudo-code:
 Begin:
 return ((db1 > db2)? ((db1 - db2) < DOUBLE_PREC)
 : ((db2 - db1) < DOUBLE_PREC));
 End

4.4.4.2.3 CWAutoAimController Class

Traceability to SRS

Constants

Name	Type	Value	Description
<i>doublePI</i>	const double	3.1415926;	radius of Circle

Private data members

Name	Type	Description
wTargetPosition	Position	Target position
wType	TYPE WEAPON	Weapon type
MyFlag	int	Friend and enemy
StaContrPt;	CWActiveStateController*	Weapon state controller
rudderPt	CWRudder*	Weapon Rudder
RSDetect	void*	convert void pointer in function according to Weapon type.

Public functions

Name: CWAutoAimController
Input: none
Output: none
Description: default constructor
Pseudo-code:
 Begin:
 wType(0), myFlag(0),
 staContrPt((CWActiveStateController *) NULL),
 rudderPt((CWRudder *)NULL),
 RSDetect(NULL),

```
wTargetPosition(INVALID_VEC)
End
```

Name: CWAutoAimController

Input: none

Output: none

Description: default constructor, Cannon Shell don't use this class For Carrier Weapons, no rudder and Radar/Sonar

Pseudo-code:

Begin:

```
(TYPE_WEAPON id,int flag, CWActiveStateController *state)
wType(id),
myFlag(flag),
staContrPt(state),
rudderPt( (CWRudder *)NULL ),
RSDetect(NULL),
wTargetPosition(INVALID_VEC)
```

End

Name: CWAutoAimController

Input: TYPE_WEAPON id,int flag,
CWActiveStateController *state,
CWRudder *rud, void *RSpt

Output: none

Description: For Auto Aiming Weapons: Rudder, Radar/Sonar system

Pseudo-code:

Begin:

```
wType(id),
myFlag(flag),
staContrPt(state),
rudderPt(rud),
RSDetect(RSpt),
wTargetPosition(INVALID_VEC)
```

End

Name: init

Input: TYPE_WEAPON id, int flag,
CWActiveStateControlle *state

Output: none

Description: for carrier Weapons, function overloading

Pseudo-code:

Begin:

```
wType = id;
myFlag = flag;
staContrPt = state;
rudderPt = (CWRudder *)NULL;
RSDetect = (void *)NULL;
//for Auto Aiming Weapons
wType = id;
myFlag = flag;
staContrPt = state;
rudderPt = rud;
RSDetect = RSpt;
```

End

Name: updateVelocity

Input: Position curPos, Position desPos

Output: integer

Description:

Pseudo-code:

Begin:

Call CWPositionController.getPosition() to get current position

Call Rudar.setCurrentPos() to set current position.

Call Rudar.setTargetPos() to set target position.

Call Rudar.calcVelocity() to get the change of Velocity.

Call Rudar.getVelocity() to get the Velocity and set wVelocity
to returned Velocity.

End

Name: locateTargetPosition

Input: Position curPos

Output: integer

Description: differ Radar and Sonar system

Pseudo-code:

Begin:

if (Radar/Sonar.EmitReceive()==0) return 0

else return # of objects in the Radar/Sonar range.

For each object

Gets target using Radar/Sonar.getFirstDetect()

Gets target using Radar/Sonar.getNextDetect()

if (isTargetType())=valid type

call Radar/Sonar.getPosition() to get position of object.

counts the distance between object position and the target position.

else go to second step for next object.

Compares this distance with saved distance, and keep distance
and position of the lesser distance object. If saved distance
is null, then keep this distance and position.

sets wTargetPosition to the position of the nearest object,
return 1.

End

Name: setInitTargetPosition

Input: Position targetPos

Output: none

Description: called in fire() function

Pseudo-code:

Begin:

if((wType == SeaSeaMissile) || (wType == SeaAirMissile)

|| (wType == Torpedo) || (wType == AirSeaMissile)

|| (wType == AirAirMissile))

wTargetPosition = targetPos;

End

Name: -CWAutoAimController

Input: none
Output: none
Description: destructor
Pseudo-code:
 Begin:
 End

4.4.4.2.4 CWCharge Class

Traceability to SRS

WP-005, WP-006, WP-007, WP-008

Constants

N/A

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
Firepower	integer	Fire power of Weapon
precision	double	Weapon precision

Public functions

Name: CWCharge
Input: none
Output: none
Description: default constructor
Pseudo-code:
 Begin:
 firepower(0),
 precision(0)
 End

Name: CWCharge
Input: int fp, double ps
Output: none
Description: constructor
Pseudo-code:
 Begin
 firepower = fp;
 precision = ps;
 End

Name: setFirepower
Input: int fp
Output: none
Description:
Pseudo-code:
 Begin:
 firepower = fp;
 End

Name: setPrecision
Input: double ps
Output: none
Description:
Pseudo-code:
 Begin:
 precision = ps;
 End

Name: chargeTarget
Input: none
Output: bool
Description: check if the target was hit
Pseudo-code:
 Begin:
 double ram = rand()/(RAND_MAX+1);
 if (ram > precision)
 return false; // The Weapon failed to hit the target
 else
 return true; // The target was hit
 End

Name: detonateTarget
Input: baseClass *target
Output: bool
Description:
Pseudo-code:
 Begin:
 Switch on Ship type
 Call hit() function of the BaseShip class;
 Return true;
 Default: return false;
 End

Name: -CWCharge
Input: none
Output: none
Description: destructor
Pseudo-code:
 Begin:
 End

4.4.4.2.5 CWChargeController Class

Traceability to SRS

WP-005, WP-006, WP-007, WP-008

Constants

N/A

Private data members

Name	Type	Description
hitDet	ChitDetect	Hit detected object
Ch	CWCharge*	Weapon charge object
Asc	CWActiveStateController *	Weapon state controller object
MyFlag	integer	Enemy or friend
FirePower	integer	Fire power of Weapon
HitRange	double	Hit range of Weapon
WeaponType	integer	Type of Weapon
pObject	baseClass*	Target object

Private functions

Name: detonate

Input: baseClass *pO

Output: none

Description:

Pseudo-code:

```

Begin:
    Detonate the Weapon;
End

```

Public functions

Name: CNChargeController

Input: none

Output: none

Description: default constructor

Pseudo-code:

```

Begin:
End

```

Name: init

Input: TYPE_WEAPON id, int flag,
CWActiveStateController *pAsc

Output: none

Description: overload function

Pseudo-code:

```

Begin
    WeaponType = id;
    myFlag = flag;
    asc = pAsc;
    ch = (CWCharge *)NULL;
    hitRange = 0;
    firePower = 0;
End

```

Name: checkDetonateRange

Input: double timeLen, Position curPos, Position nexPos

Output: integer

Description:

Pseudo-code:

```

Begin:
    number = Call Detect.EmitReceive

```

```

Loop index from zero until index = number
  If index is zero
  Then
    Call Detect.getFirstDetect
  Else
    Call Detect.getNextDetect
  Endif
Type = Call Detect.getType
  If IsTargetType(type) is false
  Then
    Goto loop
  Endif
  objectPoint = Call Detect.getObjectPoint()
  detonate( objectPoint )
End Loop
If the Weapon type of this controller is Cannon Shell
Then
  CWActiveStateController.setState(INACTIVE)
  return 1
Endif
if state is INACTIVE
  Return 1
else
  return 0;
End

```

Name: checkDetonateRange

Input: double timeLen, Position curPos, Position nexPos

Output: integer

Description:

Pseudo-code:

```

Begin:
  number = Call Detect.EmitReceive
  Loop index from zero until index = number
    If index is zero
      Call Detect.getFirstDetect
    Else
      Call Detect.getNextDetect
    Endif
    Type = Call Detect.getType
    If IsTargetType(type) is false
      Goto loop
    Endif
    objectPoint = Call Detect.getObjectPoint()
    detonate( objectPoint )
  End Loop
  If the Weapon type of this controller is Cannon Shell
  CWActiveStateController.setState(INACTIVE)
  return 1;
Endif
if state is INACTIVE Return 1;
else return 0;
End

```

Name: -CWChargeController

Input: none

Output: none

Description: distructor

Pseudo-code:
 Begin:
 End

4.4.4.2.6 CWPositionController Class

Traceability to SRS
 WP-001, WP-002, WP-003

Constants
 N/A

Private data members

Name	Type	Description
wCurrentPosition	Position	current position
wDestinationPosition	Position	Destination position
wNextPosition	Position	next time slice position
wRoute	double	Route for this Weapon
wVelocity	Velocity	Cannon Shell and carrier Weapons don't have Radar and Sonar system. other Weapons use Velocity from CWAutoAimController.getVelocity()
wType	TYPE_WEAPON	Type of Weapon
CWRudder *rudderPt	CWActiveStateController*	None of Cannon shell
StaContrPt	CWActiveStateController*	Weapon active state control object

Public functions

Name: CWPositionController
Input: none
Output: none
Description: default constructor
Pseudo-code:

```

Begin:
  wType(0),
  rudderPt((CWRudder *)NULL),
  staContrPt((CWActiveStateController *)NULL),
  wRoute(0),
  wCurrentPosition(INVALID_VEC),
  wDestinationPosition(INVALID_VEC),
  wNextPosition(INVALID_VEC)
End
  
```

Name: CWPositionController

Input: TYPE_WEAPON id,CWActiveStateController *state

Output: none

Description: constructor For Cannon Shell

Pseudo-code:

```
Begin
  wType(id),
  rudderPt((CWRudder *)NULL),
  staContrPt(state),
  wRoute(0),
  wCurrentPosition(INVALID_VEC),
  wDestinationPosition(INVALID_VEC),
  wNextPosition(INVALID_VEC)
End
```

Name: CWPositionController

Input: TYPE_WEAPON id,CWActiveStateController *state,
CWRudder *rud

Output: none

Description: constructor For Auto Aimming Weapons: CWRudder to getVelocity

Pseudo-code:

```
Begin:
  wType(id),
  rudderPt(rud),
  staContrPt(state),
  wRoute(0),
  wCurrentPosition(INVALID_VEC),
  wDestinationPosition(INVALID_VEC),
  wNextPosition(INVALID_VEC)
End
```

Name: init

Input: TYPE_WEAPON id

Output: none

Description: function overloading, init is for Cannon Shell and init is for carrier

Weapons

Pseudo-code:

```
Begin:
  wRoute = 0;
  wType = id;
  staContrPt = state;
  rudderPt = (CWRudder *)NULL;
  //init for Auto Aimming Weapons
  //Parameters TYPE_WEAPON id,CWActiveStateController *state,
  CWRudder *rud
  wRoute = 0;
  wType = id;
  staContrPt = state;
  rudderPt = rud;
End
```

Name: checkValidPosition

Input: none

Output: integer

Description:

Pseudo-code:

```

Begin:
  checks range for any Weapon. If it exceeds range, wActive is set to
  INACTIVE.
  checks condition for height
  return 1;
  else return 0;
End

```

Name: updateNextPosition

nput: double newtime

Output: none

Pseudo-code:

```

Begin:
  Call RudarController.getVelocity to get current
  velocity.
  Count new position according to current position, velocity and time.
  Increase wRoute value.
End

```

Name: updatePosition

Input: none

Output: none

Pseudo-code:

```

Begin:
  if it is INACTIVE state, then don't change position. Next position is
  calculated in updateNextPosition() only when updatePosition() is called,
  currentPosition is updated by next position that is kept in
  wNextPosition. It also increase wRoute when current position is changed.
End

```

Name: ~CWPositionController

Input: none

Output: none

Description: destructor

Pseudo-code:

```

Begin:
End

```

4.4.4.2.7 CWActiveStateController Class

Traceability to SRS

WP-005, WP-006

Constants

N/A

Private data members

Name	Type	Description
wActive	bool	Weapon state (Alive or dead)

Public functions

Name: CWActiveStateController
Input: bool d_wActive
Output: none
Description: default constructor
Pseudo-code:
 Begin:
 wActive(d_wActive),
 End

Name: CWActiveStateController
Input: none
Output: none
Description: constructor
Pseudo-code:
 Begin
 wActive(false)
 End

Name: getState
Input: none
Output: bool
Description:
Pseudo-code:
 Begin:
 return wActive;
 End

Name: setState
Input: bool state
Output: integer
Pseudo-code:
 Begin:
 wActive = state;
 return 0;
 End

Name: ~CWActiveStateController
Input: none
Output: none
Description: destructor
Pseudo-code:
 Begin:
 End

4.4.4.2.8 CWRudder Class

Traceability to SRS

WP-002, WP-003, WP-004

Constants

Name	Type	Value	Description
doublePI	const double	3.1415926;	radius of Circle

Private data members

Name	Type	Description
wpSpeed	wSpeed	Weapon speed (velocity) wSpeed is structure of speed
currentPos	Vector	Weapon current position
TargetPos	Vector	target position
currentRad	double	current steering angle
NewRad	double	new steering angle
Steering	bool	steering on/off
maxSpeed	integer	maxium Weapon speed (speed)

Public functions

Name: CWRudder

Input: int d_maxSpeed, double d_currentRad

Output: none

Description: constructor

Pseudo-code:

```

Begin:
    maxSpeed(d_maxSpeed), currentRad(d_currentRad)
End

```

Name: CWRudder

Input: none

Output: none

Description: default constructor

Pseudo-code:

```

Begin:
    maxSpeed(0), currentRad(-1.0)
End

```

Name: calcSpeed

Input: none

Output: none

Description:

Pseudo-code:

```

Begin:
    set Weapon speed to 0 if targetpos equal to currentpos;
    according to the Weapon's current position and target position, get the
    new steering angle;
    before Weapons are finally fired, steering will not be turned on.
    especially for those topedos and Missiles lauched with carrier;
    calculate distance between target position and current position;
    calculate speed z;
    calculate speed x;
    calculate speed y;
End

```

Name: setCurrentPos

Input: Vector pos
Output: none
Description:
Pseudo-code:
 Begin:
 currentPos=pos;
 End

Name: setTargetPos
Input: Vector pos
Output: none
Description:
Pseudo-code:
 Begin:
 targetPos=pos;
 End

Name: getSpeed
Input: none
Output: Vector
Description:
Pseudo-code:
 Begin:
 initialize speed;
 End

Name: setSteering
Input: bool st
Output: none
Description:
Pseudo-code:
 Begin:
 steering=st;
 End

Name: setMaxSpeed
Input: int sp
Output: none
Description:
Pseudo-code:
 Begin:
 maxSpeed=sp;
 End

4.4.4.2.9 WMissileAirAir Class

Traceability to SRS

AT-019, AT-020, CS-019, CS-020, DT-019, DT-020, BS-019, BS-020, SM-019, SM-020.

Constants

N/A

Private data members

Name	Type	Description
Rudder	CWRudder	Weapon Rudder object
Charge	CWCharge	Weapon charge object
Radar	CRadar	Radar object

Public functions

Name: WMissileAirAir

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Description:

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Precession;
    initialize Radar;
End
```

Name: -WMissileAirAir

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.10 WMissileAirSea Class

Traceability to SRS

AT-019, AT-020

Constants

N/A

Private data members

	Type	Description
Rudder	CWRudder	Weapon Rudder object
Charge	CWCharge	Weapon charge object
Radar	CRadar	Radar object

Public functions

Name: WMissileAirSea

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Description:

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WMissileAirSea

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.11 WMissileSeaAir Class

Traceability to SRS

CS-019, CS-020, BS-019, BS-020.

Constants

N/A

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
Rudder	CWRudder	Weapon Rudder object
Charge	CWCharge	Weapon charge object
Radar	CRadar	Radar object

Public functions

Name: WMissileSeaAir

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Description:

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WMissileSeaAir

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.12 WMissileSeaSea Class

Traceability to SRS

BS-019, BS-020

Constants

N/A

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
Rudder	CWRudder	Weapon Rudder object
Charge	CWCharge	Weapon charge object
Radar	CRadar	Radar object

Public functions

Name: WMissileSeaSea

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Description:

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WMissileSeaSea

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.13 WMissileSeaSub Class

Traceability to SRS

DT-019, DT-020.

Constants

N/A

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
CarriedTorpedo	Wtorpedo*	Carriage Missile object

Public functions

Name: WMissileSeaSub

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WMissileSeaSub

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.14 WtorpedoSubSea Class

Traceability to SRS

SM-019, SM-020.

Constants

N/A

Private data members

Member Name	Type	Description
carriedMissile	WMissileSeaSea*	Carriage Missile object

Public functions

Name: WtorpedoSubSea

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WtorpedoSubSea

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.15 WcannonShell Class

Traceability to SRS

BS-019, BS-020

Constants

N/A

Private data members

<u>Name</u>	<u>Type</u>	<u>Description</u>
charge	CWCharge*	Weapon Charge Object

Public functions

Name: WcannonShell

Input: none

Output: none

Description: constructor derived from CWeapon Class

Pseudo-code:

```
Begin:
    initInstance(DEFAULT_FLAG);
End
```

Name: initInstance

Input: int flag

Output: none

Description:

Pseudo-code:

```
Begin:
    initialize Rudder: MaxSpeed;
    initialize Charge: FirePower, Preceision;
    initialize Radar;
End
```

Name: -WcannonShell

Input: none

Output: none

Description: destructor

Pseudo-code:

```
Begin:
End
```

4.4.4.2.16 Wtorpedo Class

Traceability to SRS

SM-019, SM-020.

Constants

N/A

Private data members

NAME	TYPE	DESCRIPTION
Rudder	CWRudder	Weapon Rudder object
Charge	CWCharge	Weapon charge object
Sonar	CSonar	Sonar object

Public functions

Name: Wtorpedo
Input: none
Output: none
Description: constructor derived from CWeapon Class
Pseudo-code:
 Begin:
 initInstance(DEFAULT_FLAG);
 End

Name: initInstance
Input: int flag
Output: none
Pseudo-code:
 Begin:
 initialize Rudder: MaxSpeed;
 initialize Charge: FirePower, Preceision;
 initialize Radar;
 End

Name: -Wtorpedo
Input: none
Output: none
Description: destructor
Pseudo-code:
 Begin:
 End

5. System Testing

We use glass-box testing to test all the functions for all the subsystem, <<Test data>> is input of test cases, <<Expected result>> is expected output from <<Test data>>, which is shown on the screen. The <<traceability>> traces the test case specific requirements.

Test cases are derived based on major functions in each class Knowledge of algorithms used to implement functions is used to identify equivalence partition. Most of the cases, path testing is used. If test cases of a function are complex, the function will be listed separately from other simpler functions.

5.1 Unit Testing

The units in the project are defined as functional components within modules. All functional components should be verified individually. Unit tests are conducted on each individual functional component to ensure that it is as clean as possible before we move on to more complex, multi-component integration. The goals of these tests are to verify data integrity, proper hyperlink connection and database access.

Testing Tasks

- Test preparation: read the Detailed Design Document, SRD; Design the Module testing plan and test cases; design test design specifications, test procedures.
- Design test drivers for each bottom up testing. Isolate the testing Module from other modules, prepare the methods for recording data output.
- Execute test cases according to the specified test procedure, record the testing result, find the defects, and solve the problem, and then retest the suspended test.

Test Methods

Unit Static testing	
Objective	Identifying coding errors
Technique	Code inspection All inspection questions must be checked
Completion criteria	Every line of code has been inspected And each kind of error in the check list has been checked and corrected
Special considerations	None

Table 5-1 Unit Static testing

Unit Dynamic testing	
Objective	Ensure that the internal functions of each component appear to be working correctly Ensure that proper input processing and data integrity have followed the rules
Technique	Both white box testing and black box testing will be used For each data integrity and access rule, at least one test script should be created for testing
Completion criteria	All test cases must be executed No high priority or severity defects are found
Special considerations	None

Table 5-2 Unit Dynamic testing

Here, for every class, we choose some important functions to test and some simple functions are ignored. Testing is done on major functions in all the class by choosing some significant data as input and observing if the expected output results appear.

5.1.1 Unit Testing for Simulation Controller

These test case are mainly for test the class functions includes: SetUpDlg, Controller, and other classes.

5.1.1.1 Unit Test Case for SetUpDlg Class Functions

5.1.1.1.1 Unit Test Cases and Results

Function Name: Draw			
Objective: test the overlap and out of range			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-001	Bitmap:1 X:300 Y:300	No overlap	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
	Bitmap:1 X:300 Y:300		
TC_SC-002	Bitmap:1 X:500 Y:500	No overlap	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
	Bitmap:2 X:500 Y:500		
TC_SC-003	Bitmap:2 X:400 Y:400	<i>No overlap</i>	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
	Bitmap:2 X:400 Y:400		
TC_SC-004	Bitmap:1 X:729 Y:599	Within region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-005	Bitmap:1 X:729 Y:600	Out of region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-006	Bitmap:1	Out of region	SC-001, SC-002, SC-007, SC-008, SC-

	X:730 Y:600		008-01, SC-008-02
TC_SC-007	Bitmap:2 X:130 Y:0	Within region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-008	Bitmap:1 X:130 Y:599	Within region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-009	Bitmap:1 X:729 Y:0	Within region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-010	Bitmap:1 X:100 Y:100	out of region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-011	Bitmap:1 X:200 Y:700	Out of region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-012	Bitmap:2 X:100 Y:700	Out of region	SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02

Table 5-3 Unit Test Case for SetUpDlg Draw function

Function Name: Undo			
Objective: test the undo for the ship object			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-013	Destroyer	properly	SC-001
TC_SC-014	Submarine	properly	SC-001
TC_SC-015	Cruiser	properly	SC-001
TC_SC-016	Destroyer,Submarine,Cruiser,Battleship	Undo correctly	SC-001
TC_SC-017	Destroyer,Destroyer,Destroyer,Destroyer	Undo correctly	SC-001
TC_SC-018	No input	No undo	SC-001

Table 5-4 Unit Test Case for SetUpDlg Undo function

5.1.1.1.2 Error Reports

- Window is flashing when undo. We changed the called OnPaint() function by draw() function.
- The image is drawn overlap for test case 2. We construct a 15*15 matrix and trace each image sizing 40 by 40 pixels,
- Image out of map for test case 4. We set image position x, y into the top-left of each cell. It is solved problem.

5.1.1.2 Unit Test Case for Controller Class Functions

5.1.1.2.1 Unit Test Cases and Results

Function Name: LoadTGA			
Objective: load the image file			
Test Case #	Test Data	Expected Result	Trace-ability
TC_SC-019	image_tga/Weapon_red.tga	Output "image load successful" and display red Weapon on screen	SC-009, SC-010
TC_SC-020	image_tga/Weapon.tga	Output "Load image failure!"	SC-009,SC-010
TC_SC-021	image_tga/Weapon_red1.tga	Output "Load image failure!"	SC-009, SC-010

TC_SC-022	image_tga/Weapon_red1.bmp	Output "Load image failure!"	SC-009, SC-010
-----------	---------------------------	------------------------------	----------------

Table 5-5 Unit Test Case for Controller LoadTGA function

Function Name: calDir			
Objective: To test the calculation of direction vector			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-023	Vector(10.0, 10.0, 0), Vector(10.0, 10.0, 0)	9.0f (as flag)	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-024	Vector(10.0, 10.0, 0), Vector(10.0, 20.0, 0)	PI/2.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-025	Vector(10.0, 10.0, 0), Vector(10.0, 0.0, 0)	3.0*PI/2.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-026	Vector(10.0, 10.0, 0), Vector(0.0, 10.0, 0)	PI	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-027	Vector(10.0, 10.0, 0), Vector(20.0, 10.0, 0)	0.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-028	Vector(0.0, 0.0, 0), Vector(10.0, 10.0, 0)	PI/4.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-029	Vector(0.0, 0.0, 0), Vector(-10.0, 10.0, 0)	3*PI/4.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-030	Vector(0.0, 0.0, 0), Vector(-10.0, -10.0, 0)	5*PI/4.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014
TC_SC-031	Vector(0.0, 0.0, 0), Vector(10.0, -10.0, 0)	7*PI/4.0f	SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014

Table 5-6 Unit Test Case for Controller calDir function

Function Name: OnKeyDown			
Objective: To test the response key down function			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-032	Press key "F12"	Images get bigger	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-033	Press key "F11"	Images get smaller	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-034	Press key "←"	Images move left	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-035	Press key "→"	Images move right	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-036	Press key "↑"	Images move up	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-037	Press key "↓"	Images move down	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-038	Press key "space" and "a" and "1"	Image position no change	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-039	Vector(0.0, 0.0, 0), Vector(-10.0, -10.0, 0)	5*PI/4.0f	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02
TC_SC-040	Vector(0.0, 0.0, 0), Vector(10.0, -10.0, 0)	7*PI/4.0f	SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02

Table 5-7 Unit Test Case for Controller OnKeyDown function

5.1.1.2.2 Error Reports

None

5.1.1.3 Other Unit Test Through User Interaction

Other units related to UI and receivers, setters are tested through user interaction and execution of the program. Traceability for this test case are: SC-003, SC-004, SC-005, SC-006, SC-012, SC-016, SC-017, SC-018, SC-019.

Class Controller	Class SetUpDialog	Class SC
drawVehicles(); SetupPixelFormat(); OnRButtonUp(); pauseSimulation(); resumeSimulation(); startSimulation(); endSimulation();	OnLButtonDown(); OnClickAircraftcarrierB(); OnClickAircraftcarrierR(); OnClickBattleshipB(); OnClickBattleshipR(); OnClickCruiserB(); OnClickCruiserR(); OnClickDestroyerB(); OnClickDestroyerR(); OnClickSubmarineB(); OnClickSubmarineR(); OnPaint(); OnClearall();	OnStartSetup();

Table 5-8 Other Unit Test Through User Interaction

5.1.1.3.1 Error Reports

None

5.1.2 Unit Testing for Communication/Detection

Test cases for testing the class functions includes: CDetected, CRadar, CdetectedDatabase. CSonar, CMessage, CMessageDatabase, and CRadio.

5.1.2.1 Unit Test Case for CDetctcd Class Functions

5.1.2.1.1 Unit Test Cases and Results

Function Name: setUpDatabase			
Objective: To create database according to the type of detected object			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-001	Pointer to Aircraft Carrier, state=0 or state = 1	AircraftCarrier (ID, type, flag, Powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-002	Pointer to Aircraft, state=0	Aircraft (ID, type, flag, powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-003	Pointer to Aircraft,	Aircraft (ID, type, flag, Powerswitch=1, pos,velocity)	CD-004, CD-004-01, CD-004-02

	<i>state=1</i>		
TC_CD-004	Poiner to Destroyer, state=0	Destroyer (ID, type, flag, Powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-005	Poiner to Destroyer, state=1	Destroyer (ID, type, flag, Powerswitch=1, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-006	Poiner to Cruiser, state=0	Cruiser (ID, type, flag, powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-007	Poiner to Cruiser, state=1	Cruiser (ID, type, flag, Powerswitch=1, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-008	Poiner to Battleship, state=0	Battleship (ID, type, flag, powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-009	Poiner to Battleship, state=1	Battleship (ID, type, flag, powerswitch=1, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-010	Poiner to Submarine state=0	Cruiser (ID, type, flag, powerswitch=0, pos,velocity)	CD-008, CD-008-01, CD-008-02
TC_CD-011	Poiner to Submarine, state=1	Submarine (ID, type, flag, powerswitch=1, pos,velocity)	CD-008, CD-008-01, CD-008-02
TC_CD-012	Poiner to missile, state=0 or state=1	Missile (ID, type, flag, Powerswitch=1, pos,velocity)	CD-004, CD-004-01, CD-004-02
TC_CD-013	Poiner to any other state=0 or stat =1	Object (ID, type, flag, Powerswitch=0, pos,velocity)	CD-004, CD-004-01, CD-004-02

Table 5-9 Unit Test Case for Cdetcted setDetData function

Test Case #	Test Data	Expected Result	Traceability
TC_CD-014	CDetected Object	Output object's ID, flag, type, pos (x,y,z) and velocity (x,y,z)	CD-004, CD-008

Table 5-10 Unit Test Case for CDetcted operator <<overloading function

5.1.2.1.2 Error Reports

None

5.1.2.2 Unit Test Case for CDetectedDatabase Class Functions

5.1.2.2.1 Unit Test Cases and Results

Test Case #	Test Data	Expected Result	Traceability
TC_CD-015	Delare Radar (Sonar) object and call emitReceive two times	The number of detected object calling in the second time should be same with that calling in the first time	CD-004, CD-008

Table 5-11 Unit Test Case for CDetctedDatabase DeleteAll function

Function Name: addDeleted			
Objective: To add the object that is within the Radar (Sonar) range to Radar's (Sonar's) database list.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-016	CDetected object	There is an error message to indicate the input should be constant type	CD-004, CD-008
TC_CD-017	The pointer of CDetected object	Insert the pointer of CDetected object to database	CD-004, CD-008

Table 5-12 Unit Test Case for CDetectedDatabase addDeleted function

5.1.2.2.2 Error Reports

None

5.1.2.3 Unit Test Case for CRadar Class Functions

5.1.2.3.1 Unit Test Cases and Results

Function Name: EmitReceive			
Objective: To get detected objects within Radar's range and save them to the database.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-018	State = 0	Output message to user: " Radar is turned off, no object can be detected "	CD-004
TC_CD-019	State = 1 and Range = 0	Output error message to user " Radar's range can't be less or equal to zero "	CD-004
TC_CD-020	State = 1 and Range = -1	Output error message to user " Radar's range can't be less or equal to zero "	CD-004
TC_CD-021	State = 1 and Range =10 and type=3 or type =8	Output the number of detected object and a list of pointer to detected objects.	CD-004
TC_CD-022	State = 1 and Range =10 and type != 3 or type != 8	the number of detected object is zero and the list of pointer to detected objects is empty	CD-004

Table 5-13 Unit Test Case for CRadar EmitReceive function

5.1.2.3.2 Error Reports

None

5.1.2.4 Unit Test Case for CSonar Class Functions

5.1.2.4.1 Unit Test Cases and Results

Function Name: EmitReceive			
Objective: To get detected objects within Sonar's range and save them in the database			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-023	State = 0	Output message to user: " Sonar is turned off, no object can be detected "	CD-008
TC_CD-024	State = 1 and Range = 0	Output error message to user " Sonar's range can't be less or equal to zero "	CD-008
TC_CD-025	State = 1 and Range = -1	Output error message to user " Sonar's range can't be less or equal to zero "	CD-008
TC_CD-026	State = 1 and Range =10 and type=3 or type =8	Output the number of detected object and a list of pointer to detected objects.	CD-008
TC_CD-027	State = 1 and Range =10 and type != 3 or type != 8	the number of detected object is zero and the list of pointer to detected objects is empty	CD-008

Table 5-14 Unit Test Case for CSonar EmitReceive function

5.1.2.4.2 Error Reports

None

5.1.2.5 Unit Test Case for CMessage Class Functions

5.1.2.5.1 Unit Test Cases and Results

Function Name: validToSend			
Objective: To check that the message is valid to send to jpa			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-028	Pointer to the vehicle's base class set as parameter to the Cmessage object	True is returned	CD-011
TC_CD-029	Pointer to the vehicle's base class NOT set as parameter to the Cmessage object	False is returned	CD-011

Table 5-15 Unit Test Case for CMessage validToSend function

Function Name: UpdateSenderInfo			
Objective: To update the sender's Id, type and position.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-030	Pointer to the vehicle's base class set as parameter to the Cmessage object	Sender's Id is updated. Sender's Type is updated. Sender's Position is updated.	CD-011
TC_CD-031	Pointer to the vehicle's base class NOT set as parameter to the Cmessage object	Function is not called	CD-011

Table 5-16 Unit Test Case for CMessage validToSend function

5.1.2.5.2 Error Reports

None

5.1.2.6 Unit Test Case for CMessageDatabase Class Functions

5.1.2.6.1 Unit Test Cases and Results

Function Name: DeleteAllMsg			
Objective: To delete all the messages in the database of messages corresponding to the Receiver's Id.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-032	Receiver's Id is the Radio's Id and is passed as parameter	All messages belonging to the Radio's id are deleted from the message database.	CD-012
TC_CD-033	Receiver's Id IS NOT the Radio's Id and is passed as parameter	No messages are deleted.	CD-012

Table 5-17 Unit Test Case for CMessage validToSend function

Function Name: DeleteAllMsg			
Objective: To delete all the messages from the database of messages			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-034	Function Call	All messages from the message database are deleted.	CD-012

Table 5-18 Unit Test Case for CMessage DeleteAllMsg function

Function Name: AddOneMsgIntheIs			
Objective: To add one message in the message database			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-035	CMessage object sent for broadcast.	The message is added in the message database for all the corresponding receivers.	CD-012
TC_CD-036	CMessage object sent for a	The message is added	CD-012

	specific receiver.	in the message database for the specific receiver.	
--	--------------------	--	--

Table 5-19 Unit Test Case for CMessage AddOneMsgIntheList function

Function Name: GetMyMsg			
Objective: To retrieve all the messages from the message database.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-037	Radio id = receiver id and is different from sender id.	A message object is returned.	CD-012
TC_CD-038	Radio's id != receiver's id but is still different from sender's id.	A default message object with data set to default values (0) is returned	CD-012
TC_CD-039	Radio's id != receiver's id and is not different from sender's id.	A default message object with data set to default values (0) is returned	CD-012
TC_CD-040	Radio's id = id and is not different from the sender's id.	A default message object with data set to default values (0) is returned	CD-012

Table 5-20 Unit Test Case for CMessage GetMyMsg function

5.1.2.6.2 Error Reports

None

5.1.2.7 Unit Test Case for CRadio Class Functions

5.1.2.7.1 Unit Test Cases and Results

Function Name: DeleteMessages			
Objective: To delete all the messages from the database of messages corresponding to the Radio calling this function.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-041	Function call	Call to function DeleteMyMessages of CMessageDatabase class (refer to CMessageDatabase class).	CD-010 CD-012

Table 5-21 Unit Test Case for CRadio DeleteMessages function

Function Name: SendMessage			
Objective: To add one message in the message database.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-042	CMessage object is passed as parameter.	Call to function AddOneMsgIntheList of CMessageDatabase class (refer to CMessageDatabase class).	CD-011

Table 5-22 Unit Test Case for CRadio SendMessage function

Function Name: ReceiveMessages			
Objective: To verify that the ReceiveMessages function works as expected.			
Test Case #	Test Data	Expected Result	Traceability
TC_CD-043	Function call	Call to function GetMessage of CMessageDatabase class (refer to CMessageDatabase class).	CD-012

Table 5-23 Unit Test Case for CRadio ReceiveMessages function

5.1.2.7.2 Error Reports

None

5.1.3 Unit Testing for All Vehicles

Classes Ship or Aircraft are all derived from the class: BaseShip class, a class for all vehicles. It is responsible to initialize all classes used in the ship or Aircraft subsystem, including Captain, NavigationOfficer, RadioOfficer, WeaponOfficer, WeaponLauncher and onboard Radar/Sonar and Radio. All the derived class includes AircraftCarrier, Aircraft, Battleship, Cruiser, Destroyer, and Submarine. The general test case for these class are described in the table of test case, only the special test case scenario is described in bold for some subsystems.

5.1.3.1 Unit Test Case for Derived BaseShip Class Functions

5.1.3.1.1 Unit Test Cases and Results

Test Case ID	Test Data	Expected Results	Traceability
TC_BS-001	Battleship();	1)N_officer created. 2)Captain created : flag='B', type=5, resistance=300, active=true, time_counter=0. 3) Radar created : id=myID,radius=75. 4) BRadarOfficer created . 5) BRadioOfficer created : type=5. 6) Radio created : id=myID. 7) BWeaponOfficer created. 8) BWeaponLauncher created.	SC-001, SC-002, BS-001
TC_BS-002	Battleship('R', Vector(2,2,0),Vector(5,5,0))	1)N_officer created : curr_position=Vector(2,2,0), temp_position=Vector(5,5,0). 2)Captain created : flag='R', type=5, resistance=300, active=true, time_count=0. 3)Radar created : id=myID, radius=75. 4)BRadarOfficer created. 5)BRadioOfficer created : type=5. 6)Radio created : id=myID. 7)BWeaponOfficer created. 8)BWeaponLauncher created.	SC-001, SC-002, BS-001

Table 5-24 Unit Test Case for Derived BaseShip Constructor function

Function Name: updateStatus, resistanceRecovery			
Objective: When resistance < RECOVERABLE_RESISTANCE or resistance > RECOVERABLE_RESISTANCE, but time not being hit again is longer than minimum (5400) to check ship or Aircraft can or can not recover. (T = time not attacked by enemy)			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-003	Resistance= 190. T = 5600.	Resistance=190. IsActive = ture.	BS-024 to BS-031 BS-032 to BS-034
TC_BS-004	Resistance =201 T = 5399	Resistance=201. IsActive = ture.	BS-024 to BS-031 BS-032 to BS-034
TC_BS-005	Resistance = 201 T = 5401	Resistance=300 IsActive = ture.	BS-024 to BS-031 BS-032 to BS-034

Table 5-25 Unit Test Case for Derived BaseShip updateStatus and resistanceRecovery function

5.1.3.1.2 Error Reports

None

5.1.3.2 Unit Test Case for Captain Class Functions

5.1.3.2.1 Unit Test Cases and Results

Function Name: isOnTheWay			
Objective: analyzing the situation, and making decision whether and which enemy should be attacked.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-006	enemy_list = NULL or dist = 120000	return false	BS-015
TC_BS-007	sea_enemy_count = 0	return false	BS-015
TC_BS-008	sea_enemy_count = 2, dist = 90000, wtype = 0 , cQty = 50 OR sea_enemy_count = 2, dist = 90000, wtype = 1 , mQty = 10	return attack = true	BS-015
TC_BS-009	sea_enemy_count = 2, dist = 90000, mQty = 0, cQty = 0	Return attack = flase	BS-015

Table 5-26 Unit Test Case for Derived Captain ifAttack function

Function Name: isOnTheWay, adjustNavigator			
Objective: calculating the distance between this Battleship and object detected.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-010	pos = vector (50, 30, 0) my_pos = vector(40, 20, 0)	return dist = 80.62	BS-001 to BS-003

Table 5-27 Unit Test Case for Derived Captain : isOnTheWay, adjustNavigation function

5.1.3.2.2 Error Reports

None

5.1.3.3 Unit Test Case for NavigationOfficer Class Functions

5.1.3.3.1 Unit Test Cases and Results

Function Name: adjustSpeed			
Objective: The expected results are calculated according to the following data: curr_position=Vector(3,1,0), velocity=Vector(0,-3,0)			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-011	nofficer3. adjustSpeed(40,80,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,-139,0) velocity=Vector(0,-70,0)	BS-001 to BS-003
TC_BS-012	nofficer3. adjustSpeed(30,80,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,-125,0) velocity=Vector(0,-63,0)	BS-001 to BS-003
TC_BS-013	nofficer3. adjustSpeed(40,60,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,-119,0) velocity=Vector(0,-60,0)	BS-001 to BS-003
TC_BS-014	nofficer3. adjustSpeed(40,80,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,-85,0) velocity=Vector(0,-43,0)	BS-001 to BS-003
TC_BS-015	nofficer3. adjustSpeed(-2,-1,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,1,0) velocity=Vector(0,0,0)	BS-001 to BS-003
TC_BS-016	nofficer3. adjustSpeed(-2,-1,1)	curr_position=Vector(3,1,0) temp_position=Vector(3,0,0) velocity=Vector(0,-1,0)	BS-001 to BS-003
TC_BS-017	nofficer3. adjustSpeed(-2,-1,1)	curr_position=Vector(3,1,0) temp_position=Vector(3,-1,0) velocity=Vector(0,-2,0)	BS-001 to BS-003
TC_BS-018	nofficer3. adjustSpeed(-1.2,1,2)	curr_position=Vector(3,1,0) temp_position=Vector(3,-1,0) velocity=Vector(0,-1,0)	BS-001 to BS-003
TC_BS-019	nofficer3. adjustSpeed(-1.2,9,2)	error message temp_position=curr_position=Vector(3,1,0) velocity=Vector(0,-3,0)	BS-001 to BS-003
TC_BS-020	nofficer3. adjustSpeed(2,1,2)	error message temp_position=curr_position=Vector(3,1,0) velocity=Vector(0,-3,0)	BS-001 to BS-003

Table 5-28 Unit Test Case for NavigationOfficer adjustSpeed function

Function Name: other functions			
Objective: Test the other function (in bold font)			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-021	Bnavigation nofficer1() ; BNavigationOfficer() function	curr_position=Vector(0,0,0) temp_postiion=Vector(0,0,0) velocity=Vector(0,0,0)	BS-001 to BS-003
TC_BS-022	BNavigationOfficer nofficer2(Vector(2,2,0),Vector,(5,5,0),Vector(1,3,0)) BNavigationOfficer (Vector curPos,Vector desPos,Vector spd)	curr_position=Vector(2,2,0) temp_postiion=Vector(5,5,0) velocity=Vector(1,3,0)	BS-001 to BS-003
TC_BS-023	BNavigationOfficer nofficer2(Vector(2,2,0),Vector(5,5,0),Vector(70,70,0)) BNavigationOfficer (Vector curPos,Vector desPos,Vector spd)	curr_position=Vector(2,2,0) temp_postiion=Vector(5,5,0) velocity=Vector(49.4975,49.4975,0)	BS-001 to BS-003
TC_BS-024	BNavigationOfficer nofficer3(Vector(2,2,0), Vector(5,5,0)) BNavigationOfficer (Vector curPos, Vector desPos);	curr_position=Vector(2,2,0) temp_postiion=Vector(5,5,0) velocity=Vector(49.4975,49.4975,0)	BS-001 to BS-003
TC_BS-025	None ~ BNavigationOfficer();	main() runs without error.	BS-001 to BS-003
TC_BS-026	nofficer3.getPosition() ; getPosition()	Vector(2,2,0)	BS-013 to BS-018
TC_BS-027	nofficer3.getVelocity() ; getVelocity()	Vector(7.07107,7.07107,0)	BS-013 to BS-018
TC_BS-028	nofficer3.setPosition(Vector(3,1,0)) setPosition(Vector pos)	curr_position=Vector(3,1,0)	BS-013 to BS-018
TC_BS-029	nofficer3.setVelocity(Vector(60,80,0)) setVelocity(Vector spd)	Velocity=Vector(42,56,0)	BS-013 to BS-018
TC_BS-030	nofficer3.setVelocity(Vector(4,3,0)) setVelocity(Vector spd)	Velocity=Vector(4,3,0)	BS-013 to BS-018
TC_BS-031	nofficer3.cruise(Vector(3,1,0),1) cruise(Vector targetPos, double t) cruise(Vector targetPos, double t)	curr_position=Vector(3,1,0) temp_position=Vector(3,1,0) velocity=Vector(0,0,0)	BS-013 to BS-018
TC_BS-032	nofficer3.cruise(Vector(3,-3,0),1) cruise(Vector targetPos, double t)	curr_position=Vector(3,1,0) temp_position=Vector(3,-2,0) velocity=Vector(0,-3,0)	BS-013 to BS-018
TC_BS-033	nofficer3.steer(0.1) steer(a)	curr_position=Vector(3,1,0) temp_position=Vector(3,1,0) original velocity=Vector(0,-3,0) velocity=Vector(-2.98501,-0.2995,0)	BS-013 to BS-018
TC_BS-034	nofficer3.setVelocity (Vector (0, -3,0)) nofficer3.adjustSpeed(30,80,2) nofficer3.updatePosition() ; updatePosition()	curr_position=Vector(3,1,0) temp_position=Vector(3,-125,0) velocity=vector00,-63,0)	BS-013 to BS-018

Table 5-29 Unit Test Case for NavigationOfficer other function

5.1.3.4 Unit Test Case for RadioOfficer Class Functions

5.1.3.4.1 Unit Test Cases and Results

Because it is difficult to test this unit without simulating the communication class, this unit test will be done in subsystem testing case.

5.1.3.4.2 Error Reports

None

5.1.3.5 Unit Test Case for Radar/SonarOfficer Class Functions

5.1.3.5.1 Unit Test Cases and Results

Because it is difficult to test this unit without simulating the communication class, this unit test will be done in subsystem testing case.

5.1.3.5.2 Error Reports

None

5.1.3.6 Unit Test Case for WeaponOfficer Class Functions

5.1.3.6.1 Unit Test Cases and Results

Function Name: prepareAttack			
Objective: For each test case, some critical data are chosen as input and the output results are to be compared with the expected results.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-035	In every time i, cp=Vector(i,i,0), tp=Vector(26+i,26+i,0)ts=Vector(20,20,0) tid=1, ct=10+5*i	After 5 times call, the object of <i>WeaponLauncher</i> be called and the cannon shell sent, 8 times, sent again	BS-019 to BS-023
TC_BS-036	In every time i, cp=Vector(i,i,0), tp=Vector(27+i,27+i,0) ts=Vector(20,20,0), tid=1, ct=10+10*i	After 4 times call, the object of <i>WeaponLauncher</i> be called and a Missile sent, 7 time calls, sent again.	BS-019 to BS-023
TC_BS-037	In every time i, cp=Vector(i,i,0), tp=Vector(80+i,80+i,0) ts=Vector(20,20,0), tid=1, ct=10+10*i	After 4 times call, the object of <i>WeaponLauncher</i> be called and a Missile sent, 7 time calls, sent again.	BS-019 to BS-023
TC_BS-038	In every time i, cp=Vector(i,i,0),	The object of <i>WeaponLauncher</i> is not called, so	BS-019 to BS-023

	tp=Vector(85+i,85+i,0) ts=Vector(20,20,0), tid=1, ct=10+10*i	neither cannon shells nor misslies is launched.	
TC_BS-039	In every time i, cp=Vector(i,i,0), tp=Vector(26+i,26+i,0) ts=Vector(27+i,27+i,0) ts=Vector(20,20,0), tid=1, ct=10+10*i	The object of WeaponLauncher is not called, so neither cannon shells nor misslies is launched.	BS-019 to BS-023
TC_BS-040	In every time i, cp=Vector(i,i,0), tp=Vector(26+i,26+i,0) ts=Vector(20,20,0), tid=1, then tid=2, ct=10+10*i	The object of BWeaponLauncher is not called, so no cannon shells are launched.	BS-019 to BS-023

Table 5-30 Unit Test Case for WeaponOfficer prepareAttack function

Function Name: selectWeapon			
Objective: For each test case, some critical data are chosen as input and the output results are to be compared with the expected results.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-041	cp=Vector(0, 0, 0), tp=Vector(26, 26, 0) cannon_qty=4.	return value is 1.	BS-019
TC_BS-042	cp=Vector(0, 0, 0), tp=Vector(26, 26, 0), cannon_qty=2.	return value is -1.	BS-019
TC_BS-043	cp=Vector(0, 0, 0), tp=Vector(27, 27, 0), Missile_qty=1.	return value is 0.	BS-019
TC_BS-044	cp=Vector(0, 0, 0), tp=Vector(27, 27, 0), Missile_qty=0.	return value is -1.	BS-019
TC_BS-045	cp=Vector(0, 0, 0), tp=Vector(80, 80, 0), Missile_qty=1.	return value is 0.	BS-019
TC_BS-046	cp=Vector(0, 0, 0), tp=Vector(85, 85, 0), Missile_qty=1.	return value is 0	BS-019

Table 5-31 Unit Test Case for WeaponOfficer selectWeapon function

5.1.3.6.2 Error Reports

- a) In the test of the *prepareAttack* function, we observed that when enemy was the fire range of Missiles, after the latency time for launching Missiles was arrived, there were no Missile launched. After examining the code, we found that there was an error in calculating the latency time for Missile launching.
- b) In the test of the *selectWeapon* function, we found that it might cause confusion if using return value 0 to represent two cases when Missile was

selected and neither Missile nor cannon was selected. We add a return value -1 which represent the neither Missile nor cannon selection case.

5.1.3.7 Unit Test Case for WeaponLauncher Class Functions

5.1.3.7.1 Unit Test Cases and Results

Function Name: aimByBallistic			
Objective: For each case, some significant or critical data are inputted and the output results are to be compared with the expected results.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-047	cp=Vector(0, 0, 0), tp=Vector(12, 13, 0) , ts=Vector(10, 20,0),	dx=10/3600*t+12 dy=20/3600*t+13	CS-021, DT-021, SM-021, AT-021
TC_BS-048	cp=Vector(0, 0, 0), tp=Vector(18, 20, 0) , ts=Vector(20,15,0),	dx=20/3600*t+18 dy=15/3600*t+20	CS-021, DT-021, SM-021, AT-021
TC_BS-049	cp=Vector(0, 0, 0), tp=Vector(25, 28, 0) , ts=Vector(15, 20,0),	dx=15/3600*t+25 dy=20/3600*t+28	CS-021, DT-021, SM-021, AT-021

Table 5-32 Unit Test Case for WeaponLauncher aimByBallistic function

Function Name: fireCannonShell			
Objective: This function is used to create cannon shells, insert them in the cannon_list, and fire them when they are required.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-050	cp=Vector(0, 0, 0), tp=Vector(20, 20, 0) , b-flag=R.	A cannon shell is inserted into the cannon_list. The fire function in WMissileSeaSea is called	BS-021

Table 5-33 Unit Test Case for WeaponLauncher fireCannonShell function

Function Name: fireMissile			
Objective: This function is used to create Missiles, insert them in the Missile_list, and fire them when they are required.			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-051	cp=Vector(0, 0, 0), tp=Vector(50, 50, 0) , b-flag=R.	A Missile is inserted into the Missile_list. The fire function in WMissileSeaSea is called.	CS-021, DT-021,SM-021, AT-021

Table 5-34 Unit Test Case for WeaponLauncher fireMissile function

Function Name: deleteWeapon			
Objective: This function is used to delete cannon shells or Missiles when they have been detonated			
Test Case #	Test Data	Expected Result	Traceability
TC_BS-052	Create a Missile list with Missiles some marked active, some inactive	The Missiles marked active are deleted and those marked inactive remain	CS-022, DT-022, SM-022, AT-022
TC_BS-053	Create a cannon shell list with cannon shells some marked active, some inactive	The cannon shells marked active are deleted and those marked inactive remain	BS-022

Table 5-35 Unit Test Case for WeaponLauncher deleteWeapon function

5.1.3.7.2 Error Reports

In the test of *aimByBallistic* function, we found that the output results were too large, comparing with the expected results. After checking the code carefully, we found that there was some errors with the units used in some places in the function. After correcting the error, we get the expected results.

5.1.4 Unit Testing for Weapons

These test cases are mainly for testing the class functions includes: CWActiveStateController, CWPositionController, CWAutoAimController, CWChargeController, CWCharge, and CWRudder.

5.1.4.1 Unit Test Case for CWActiveStateController Class Functions

5.1.4.1.1 Unit Test Cases and Results

Function Name: Every Weapon has this class to indicate its state. There are only 2 functions one is getState() and another is setState().			
Objective: While constructing, it needs parameter of the state, which is true or false (active or inactive respectively).			
Test Case #	Test Data	Expected Result	Traceability
TC_WP-001	Create an active instance	getState return true	WP-005
TC_WP-002	SetState to inactive	getState() returns false.	WP-005
TC_WP-003	Create an inactive instance	getState return false	WP-005
TC_WP-004	SetState to active	getState return true	WP-005

Table 5-36 Unit Test Case for CWActiveStateController get/setState function

5.1.4.1.2 Error Reports

None

5.1.4.2 Unit Test Case for CWPositionController Class Functions

5.1.4.2.1 Unit Test Cases and Results

Aerial Weapons					
Function Name: Initial position					
Objective: To control Weapon's positions, such as initial position, destination position, update current position, and check if the position is valid. For self-guided Weapon, a mode will be created to update the velocity.					
Test Case #	Test Data		Expected Result		Traceability
	destination	state	velocity	position	
TC_WP-005	(100,0,0)	Active	(50,0,0)	(0.014,0,0)	WP-001
TC_WP-006	(100,0,100)	Active	(35.36,0,35.36)	(0.01,0,0.01)	WP-001
TC_WP-007	(100,0,-100)	Inactive	N/A	N/A	WP-001

Table 5-37 Unit Test Case for CWActiveStateController initialposition function

Submarine Weapons					
Function Name: initial position					
Objective: To control Weapon's positions, such as initial position, destination position, update current position, and check if the position is valid. For self-guided Weapon, a model will be created to update the velocity.					
Test Case #	Test Data		Expected Results		Traceability
	destination	state	velocity	position	
TC_WP-008	(100,0,0)	Active	(50,0,0)	(0.014,0,0)	WP-001
TC_WP-009	(100,0,100)	Inactive	N/A	N/A	WP-001
TC_WP-010	(100,0,-100)	Active	(35.36,0,35.36)	(0.01,0,0.01)	WP-001

Table 5-38 Unit Test Case for CWActiveStateController initial position function

5.1.4.2.2 Error Reports

None

5.1.4.3 Unit Test Case for CWAutoAimController Class Functions

5.1.4.3.1 Unit Test Cases and Results

Target: ship							
Function Name:							
Objective: self-guided Weapons to trace target, uses Radar/Sonar to detect all objects in Radar/Sonar's range, select the valid nearest target to the previous target position.							
Test Case #	Test Data					Expected Result	Traceability
	Initial Position	Detected Object 1	Detected Object 2	Detected Object 3	Detected Object 4		
TC_WP-011	(100,0,0)	Ship, (100,0,0)	Ship, (100,10,0)	Submarine, (100,0,-50)	Aircraft, (100,10,80)	Ship, (100,0,0) Return success	WP-002, WP-003 WP-004
TC_WP-012	(100,0,0)	Ship, (300,0,0)	Aircraft, (100,0,50)	Submarine (100,0,-50)	N/A	Ship, (300,0,0) Return success	WP-002, WP-003 Wp-004
TC_WP-013	(50,0,0)	Aircraft, (50,0,100)	Submarine (50,0,-50)	N/A	N/A	Return failure	WP-002, WP-003 Wp-004

Table 5-39 Unit Test Case for CWAutoAimController tracetarget function

Target: Aircraft							
Function Name:							
Objective: self-guided Weapons to trace target (uses Radar/Sonar to detect all objects in Radar/Sonar's range, select the valid nearest target to the previous target position)							
Test Case #	Test Data					Expected Result	Traceability
	Initial Position	Detected Object 1	Detected Object 2	Detected Object 3	Detected Object 4		
TC_WP-014	(100,0,100)	Ship, (100,0,0)	Aircraft, (100,0,100)	Submarine, (100,0,-50)	Aircraft, (100,10,80)	Aircraft, (100,0,100) Return success	WP-002, WP-004
TC_WP-015	(100,0,50)	Ship, (100,0,0)	Aircraft, (200,0,100)	Submarine (100,0,-50)	N/A	Aircraft, (200,0,100) Return success	WP-002, WP-004
TC_WP-016	(50,0,50)	Ship, (100,0,0)	Submarine (100,0,-50)	N/A	N/A	Return failure	WP-002, WP-004

Table 5-40 Unit Test Case for CWAutoAimController tracetarget (Aircraft) function

Target: Submarine							
Function Name:							
Objective: self-guided Weapons to trace target (uses Radar/Sonar to detect all objects in Radar/Sonar's range, select the valid nearest target to the previous target position)							
Test Case #	Test Data					Expected Result	Traceability
	Initial Position	Detected Object 1	Detected Object 2	Detected Object 3	Detected Object 4		
TC_WP-017	(100,0,-50)	Ship, (100,0,0)	Submarine, (80,0,-50)	Submarine, (100,0,-50)	Aircraft, (100,10,80)	Submarine, (100,0,-50) Return success	WP-002, WP-003, Wp-004
TC_WP-018	(100,0,-50)	Ship, (100,0,0)	Aircraft, (100,0,50)	Submarine (150,0,-50)	N/A	Submarine (150,0,-50) Return success	WP-002, WP-003, Wp-004
TC_WP-019	(50,0,-50)	Ship, (50,0,0)	Aircraft, (50,0,50)	N/A	N/A	Return failure	WP-002, WP-003, Wp-004

Table 5-41 Unit Test Case for CWAutoAimController tracetarget (Submarine) function

5.1.4.3.2 Error Reports

None

5.1.4.4 Unit Test Case for CWChargeController Class Functions

5.1.4.4.1 Unit Test Cases and Results

All ships							
Purpose/Use Case/Requirement							
Objective: This unit test verifies that the HitDetect function of the CWChargeController class returns the correct state for the Weapon's state (active and detonate) the target.							
Test Case #	Test Data					Expected Result	Traceability
	Weapon Position	Weapon Velocity	Detected Object 1	Detected Object 2	Detected Object 3		
TC_WP-020	(0,0,0)	(1000,0,0)	Ship1, (0,0,0)	Ship2, (0.2,0,0)	N/A	Return success (detonate 2 objects)	WP-005, Wp-006
TC_WP-021	(0,0,0)	(1000,0,0)	Ship1, (0.1,0,0)	N/A	N/A	Return success (detonate 1 objects)	WP-005, Wp-006
TC_WP-022	(0,0,0)	(1000,0,0)	N/A	N/A	N/A	Return failure	WP-005, Wp-006

Table 5-42 Unit Test Case for CWChargeController HitDetect function

Aircraft							
Purpose/Use Case/Requirement							
Objective: This unit test verifies that the HitDetect function of the CWChargeController class returns the correct state for the Weapon's state (active and detonate) the target.							
Test Case #	Test Data					Expected Result	Traceability
	Weapon Position	Weapon Velocity	Detected Object 1	Detected Object 2	Detected Object 3		
TC_WP-023	(0,0,100)	(1000,0,0)	Aircraft, (0,0,100)	Aircraft, (0,0,100.2)	N/A	Return success (detonate 2 objects)	WP-005, Wp-006
TC_WP-024	(0,0,100)	(1000,0,0)	Aircraft, (0,0,100.1)	N/A	N/A	Return success (detonate 1 objects)	WP-005, Wp-006
TC_WP-025	(0,0,100)	(1000,0,0)	N/A	N/A	N/A	Return failure	WP-005, Wp-006

Table 5-43 Unit Test Case for CWChargeController HitDetect(Aircraft) function

Submarine							
Function Name: HitDetect							
Objective: This unit uses HitDetect to detect if there is any valid target in the Weapon's detonate range all the time. If there is any, it set Weapon's state inactive and detonate the target.							
Test Case	Test Data					Expected Result	Traceability
	Weapon Position	Weapon Velocity	Detected Object 1	Detected Object 2	Detected Object 3		
TC_WP-026	(0,0,-50)	(1000,0,0)	Submarine, (0,0,0)	Submarine, (0.2,0,0)	N/A	Return success (detonate 2 objects)	WP-005, Wp-006
TC_WP-027	(0,0,-50)	(1000,0,0)	Submarine, (0.1,0,0)	N/A	N/A	Return success (detonate 1 objects)	WP-005, Wp-006
TC_WP-028	(0,0,-50)	(1000,0,0)	N/A	N/A	N/A	Return failure	WP-005, Wp-006

Table 5-44 Unit Test Case for CWChargeController HitDetect(Submarine) function

5.1.4.4.2 Error Reports

- a) The HitDetect returned a null pointer of target, but the target actually existed.

The return type of HitDetect is wrong. It has been fixed:

```
int CWChargeController::checkDetonateRange(double timeLen,
Position curPos, Position nexPos)
```

- b) Weapon attack any target no matter if its flag is opposite to itself.

Modified the following code:

```
int vehicleFlag = infoDet.getFlag();
if( ( vehicleFlag != myFlag ) && (
IsTargetType(WeaponType,vehicleType) == TRUE ) )
```

Old version:

```
if( IsTargetType(WeaponType,vehicleType) == TRUE
```

5.1.4.5 Unit Test Case for CWCharge Class Functions

5.1.4.5.1 Unit Test Cases and Results

Function Name: This unit is called when Weapons detonate targets.			
Objective: It determines whether hit or no according the Weapon's precision. If hit, it calls target's hit() function with the parameter fire power.			
Test Case #	Test Data	Expected Result	Traceability
TC_WP-029	baseClass * ship1, resistance 50, precision=80%, fire power=1 .	20 iterations are run and 18 times hit, 2 times miss, resistance=42.	WP-006, WP-0007, WP-008

Table 5-45 Unit Test Case CWCharge detonateTarget function

5.1.4.5.2 Error Reports

Hit function is not called. The reason is the type of the pointer is baseCalss, we have to convert it to the type of each vehicle respectively. It's fixed.

5.1.4.6 Unit Test Case for CWRudder Class Functions

5.1.4.6.1 Unit Test Cases and Results

Function Name: This unit will change Weapon's velocity according to Weapon's current velocity, current position and target position.
Objective: In each time slice it will not turn more than 90 degree

Test Case #	Test Data			Expected Result	Traceability
	Current Velocity	Current Position	Target Position		
TC WP-030	(1000,0,0)	(0,0,0)	(0,0,0)	(0,0,0)	WP-001
TC WP-031	(1000,0,0)	(0,0,0)	(100,0,0)	(1000,0,0)	WP-001
TC WP-032	(1000,0,0)	(0,0,0)	(0,100,0)	(866.03, 500,0)	WP-001
TC WP-033	(1000,0,0)	(0,0,0)	(0,100,100)	(612.37,353.55,707.11)	WP-001

Table 5-46 Unit Test Case CWRudder changeVelocity function

5.1.4.6.2 Error Reports

Vector::unit() will happen assert 0 Error in Vector Class, if speed is zero. So we can't return zero speed if speed doesn't have valid value. We offer a minimum speed.

5.2 Subsystem testing

After all the classes and functions has complete the unit testing. The subsystem testing must be done to ensure various components in the subsystem corporate correctly and fulfill all the functionality. Testing interface is also developed for effective and convenient testing.

5.2.1 Simulation Controller Subsystem Testing

5.2.1.1 Test Cases and Results

Objective: For class SetUpDlg test the user action on interface is complete.			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-041	Click icon, click map	Bitmap, position, type, flag	SC-001 to SC-006
TC_SC-042	Click icon, click map	Full of vehicles within the map	SC-001 to SC-006
TC_SC-043	Click clear all button	All vehicles disappear from the map	SC-001 to SC-006
TC_SC-044	Click Undo button	The most recent object is removed from the map	SC-001 to SC-006
TC_SC-045	Click Ok button	Setup dialogue window closed and main window display	SC-001 to SC-006
TC_SC-046	Click Cancel button	Setup dialogue will be closed.	SC-001 to SC-006

Table 5-47 Test Case for Simulaiton Controller(SetUpDlg) Subsystem

Objective: For class Vector test the vector operation.			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-047	V1(0.0, 0.0, 0.0) + V2(1.0,1.0,1.0)	V(1.0,1.0,1.0)	SC-013-01
TC_SC-048	V2(1.0,1.0,1.0) * 2.0	V(2.0, 2.0, 2.0)	SC-013-01
TC_SC-049	V2(1.0,1.0,1.0) / 2.0	V(0.5, 0.5, 0.5)	SC-013-01
TC_SC-050	V2(1.0,1.0,1.0) / 0.0	Error	SC-013-01
TC_SC-051	V1(1.0, 1.0, 1.0) -	V1 (0.0, 0.0, 0.0)	SC-013-01
TC_SC-052	Click Cancel button	Setup dialogue will be closed.	SC-013-01

Table 5-48 Test Case for Simulaiton Controller(Vector) Subsystem

Objective: For class SC, test the user action to interface is correct			
Test Case #	Test Data	Expected Result	Traceability
TC_SC-053	Click on SETUP button on the Toolbar or Set up item from Start menu of the main window	Set up dialog window is to be displayed, iconic buttons and a cyan rectangle shown	SC-012
TC_SC-054	Using mouse clicks to select vehicles, generate positions and create 1, 10, 225 VehicleInfo objects in separate tests as described in 3.2.1.1	Output the text info of the 2-D array to a text file out.txt via cout. The same number of VehicleInfo expected	SC-001
TC_SC-055	Click on OK button of the SetUpDlg dialog window after picking up a number of vehicles	Created objects will display in the simulated naval battle fields in the main window	SC-012

Table 5-49 Test Case for Simulaiton Controller (SC) Subsystem

Objective: For class VehicleFactory, test creation of ship object is correct					
Test Case #	Test Data			Expected Result	Traceability
	Type	#Blue	#Red		
TC_SC-056	Aircraft Carrier	1		Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-057	Aircraft Carrier		1	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-058	Aircraft Carrier	1	1	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-059	Aircraft Carrier Aircraft	1 10	1 10	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-060	Aircraft Carrier Aircraft Destroyer	1 10 2	1 10 2	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-061	Aircraft Carrier Aircraft Destroyer Cruiser	1 10 2 1	1 10 2 1	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-062	Aircraft Carrier Aircraft Destroyer Cruiser Battleship	1 10 2 1 1	1 10 2 1 1	Created objects will display in the simulated naval battle fields	SC-001, SC-012
TC_SC-063	Aircraft Carrier Aircraft Destroyer Cruiser Battleship Submarine	1 10 2 1 1 1	1 10 2 1 1 1	Created objects will display in the simulated naval battle fields	SC-001, SC-012

Table 5-50 Test Case for Simulaiton Controller (VehicleFactory) Subsystem

Objective: For class Controller, test the start, pause, stop, and resume functions.			
Test Case#	Test Data	Expected Result	Traceability
TC_SC-064	Click "start" with vehicles created	Animation starts. (Fig. 4)	SC-012
TC_SC-065	Click "start" without vehicles created	No action.	SC-012
TC_SC-066	Click "start" when animation is running	No effect.	SC-012
TC_SC-067	Click "Pause" when animation is running	Animation is paused.	SC-016
TC_SC-068	Click "Pause" when animation isn't running	No action.	SC-016
TC_SC-069	Click "Resume" when animation is paused	Animation is resumed.	SC-017
TC_SC-070	Click "Pause" when animation running	No action.	SC-016
TC_SC-071	Click "Stop" when animation is running	Animation is terminated and is reset for next simulation.	SC-018
TC_SC-072	Click "Stop" when animation isn't running	No action. System is set for new simulation if necessary.	SC-018

Table 5-51 Test Case for Simulaiton Controller (Controller) Subsystem

5.2.1.2 Error Reports

None

5.2.1.3 Untested Components

All the important components are tested.

5.2.2 Communication/Detection Subsystem Testing

5.2.2.1 Test Cases and Results

Objective: Vehicle can use its Radar/Sensor to get detailed information about objects that are within its range. For ground-based objects, it can get the ID, ID number, status, name, direction, distance, bearing, etc. For air-based objects, it can get the Radar/Sensor to detect other vehicles within Radar's range. It can also get the total number of detected vehicles and view attributes of every detected objects.

Test Case #	Test Data	Expected Result	Traceability
TC_CD-044	Vehicle declare Radar object r(ID, 200), state=1, and its position(4,5,6)	Get the number of detected object within Radar's range and a list of pointer to detected objects	CD-004 CD-008
TC_CD-045	Vehicle declare default Radar object r(), range=1000, state=1, position(4,5,6)	Get the number of detected object within Radar's range and a list of pointer to detected objects	CD-004 CD-008
TC_CD-046	the number of detected object within Radar's range.	Go through all pointer inside the detected List	CD-004 CD-008
TC_CD-047	vehicle detects the number of objects within Radar's range, and access detected objects by declaring a detected object.	Get each detected object pointed by pointer inside the detected list	CD-004 CD-008
TC_CD-048	Call turnoff	Nothing is detected.	CD-004 CD-008

Table 5-52 Test Case for Communication/Detection Subsystem

5.2.2.2 Error Reports

None.

5.2.2.3 Untested Components

All the important components are tested

5.2.3 Ship/Aircraft Subsystem Testing

5.2.3.1 Test Cases and Results

Test Case	Test Data	Expected Result	Traceability
TC_BS-054	Ship/Aircraft current position Vector(10,10,0) destination position Vector(100,10,0)	Ship/Aircraft move at fix speed towards to the destination	BS-001 to BS-002
TC_BS-055	1)An underwater object: Vector(10,10,-10)) is aimed at ship/Aircraft. 2)Message send by allies.	Ship/Aircraft changes its direction 180 ^o C at max speed.	BS-003
TC_BS-056	1)Ship/Aircraft position : Vector(0,0,0) 2)Object Vector(26,26,0).	Ship/Aircraft reduces its speed and fire cannon.	BS-001 to BS-002, BS-021
TC_BS-057	1)Ship/Aircraft position: Vector(0,0,0) 2)Object Vector(80,80,0)	Ship/Aircraft reduces its speed and fire Missile.	BS-002, BS-021
TC_BS-058	1)Ship/Aircraft position : Vector(0,0,0) 2)Object Vector(53,53,0)	Ship/Aircraft reduces its speed and fire Missile.	BS-002, BS-021
TC_BS-059	Fire Weapon to ship/Aircraft	The resistance points of ship/Aircraft reduces continuously without recovery and finally reduces to zero.	BS-025, BS-026,
TC_BS-060	Fire Weapon to ship/Aircraft	The resistance points reduced first and recovered to maximum 300 later on.	BS-025, BS-026, BS-027

Table 5-53 Test Case for Ship/Aircraft Subsystem

5.2.3.2 Error Reports

None

5.2.3.3 Untested Components

All the important components are tested.

5.2.4 Weapon Subsystem Testing

5.2.4.1 Test Cases and Results

Test Case #	Test Data	Expected Result	Trace-ability
TC_WP-034	Destroyer(0,0,0) Flag1 Destroyer(30,0,0) Flag0	Both Destroyer are hit	WP-005,WP-006 WP-007,WP-008
TC_WP-035	Destroyer (0,0,0) Flag1 Destroyer (50,0,0) Flag0	None of Destroyer is hit because out of range	WP-005,WP-006 WP-007,WP-008
TC_WP-036	Destroyer (0,0,0) Flag1 Submarine (30,0,-10) Flag0	Both vehicle are hit	WP-005,WP-006 WP-007,WP-008
TC_WP-037	Destroyer (0,0,0) Flag1 Aircraft(30,0,100) Flag0	Invalid position	WP-005,WP-006 WP-007,WP-008
TC_WP-038	Destroyer (0,0,0) Flag1 Destroyer(10,0,0) to(10,100,0) Flag0	Trace Target and hit	WP-005,WP-006 WP-007,WP-008

Table 5-54 Test Case for Weapon(Wtorpedo) Subsystem

Test Case #	Test Data	Expected Result	Trace-ability
TC_WP-039	Battle Ship(0,0,0) Flag1 Battle Ship(30,0,0) Flag0	Both Battle ships are hit.	WP-005,WP-006 WP-007,WP-008
TC_WP-040	Battle Ship(0,0,0) Flag1 Battle Ship(50,0,0) Flag0	None of battle ship is hit because out of range.	WP-005,WP-006 WP-007,WP-008
TC_WP-041	Battle Ship(0,0,0) Flag1 Battle Ship(30,0,-10) Flag0	Invalid Weapon position	WP-005,WP-006 WP-007,WP-008
TC_WP-042	Battle Ship(0,0,0) Flag1 Submarine(30,0,0) Flag0	Submarine cannot be detonate	WP-005,WP-006 WP-007,WP-008

Table 5-55 Test Case for Weapon (WcannonShell) Subsystem

5.2.4.2 Error Reports

None

5.2.4.3 Untested Components

All the important components are tested.

5.3 System Integration Testing

The Naval Battle Simulation System is composed of nine subsystems. All subsystems must be integrated and their interaction must be verified. In order to check if the whole nine subsystems can co-operate and undertake their functions well, integration testing must be performed.

5.3.1 Integration scheme

The Simulation Controller subsystem provides a user interface and affects the performance of the whole system, so it is the top-level of the whole system. The top-down strategy with incremental approach should be used for system testing. The Communication/Detection subsystem is responsible for detecting enemies and communicating with allies and the Weapons subsystem provides different kinds of Weapons that can be used by ships and Aircrafts to attack enemies, they have much interaction with each other and other subsystems. Therefore the successful integration and coordination of these three subsystems is the basis for the integration and coordination of the whole system. According to this analysis, these three subsystems should be integrated at first place. After they are successfully integrated, the other subsystems should be integrated one by one.

However, because there are some relationships between different subsystems, the integration should follow a sequence. The Aircraft Carrier subsystem should be integrated before Aircraft subsystem, because Aircraft Carrier will provide launching and landing base for Aircrafts. Then the Cruiser subsystem should be integrated because the Cruisers must have Aircrafts to fire at; the Submarine subsystem should be integrated before Destroyer subsystem because Destroyers must have Submarines to be destroyed, and etc. The Battleship subsystem should be integrated into the system later because Battleships must defense the Submarines and Aircrafts.

5.3.2 Test Cases and Results

The successful integration of the system is only one part of the success of the system. The more important part of the success is that each subsystem can work coordinately with each other and the whole system can operate well and achieve the anticipated goals.

The following test cases are designed to check if Battleship subsystem can work coordinately with other subsystems when it is put together with them. The method used in the test cases is black-box testing. Some crucial and critical

situations are chosen as input states and the output results are examined and compared with the expected results.

Objective: To test the system in a fully integrated state, as used, after finished.		
Test case #	Descriptive	Description
TC1	Test Description	This test case is to check the navigation aspect of ship/Aircraft subsystem when other ships and Aircrafts are present.
	Input states	Some other ships, Aircrafts on both sides are created and put in places relatively near, then relatively far away to the ship/Aircrafts.
	Expected results	Battle ships navigate properly and accordingly, meaning they adjust their directions and navigation speeds to avoid collision and for defense. If there is no enemy around, they navigate with constant speed towards the destination
TC2	Test Description	This test case is to check the interaction of ship/Aircraft subsystem with detection/communication and Weapon subsystems. Allies should exchange information about the presence of enemy with each other, and Missiles should be launched when enemy ships enter the fire area of Missile.
	Input states	Some other ships and Aircrafts on both sides are created and some allies are placed in the communication areas of ship/Aircraft, some enemy ships are placed in the Missile fire range, but out of the Radar detect range and the fire range of cannon of the ship/Aircrafts.
	Expected results	Ship/Aircrafts act accordingly with the presence of enemy and allies. When there are enemies in the fire range of Missiles (which is out of the detect range of Radar on battleships), Missiles, not cannon are launched.
TC3	Test Description	This test case is to check the interaction of ship/Aircraft subsystem with detection/communication and Weapon subsystems. Enemies should be detected by Radar on the ship/Aircrafts and Missiles, not cannon should be launched when enemy ships are out of the fire range of cannon on the battle ships.
	Input states	Some other ships and Aircrafts on both sides are created and allies are placed out of the communication areas of ship/Aircraft, some enemy ships are placed in the detect area of Radar (75km), but out of the cannon fire range (38km) of battleship.
	Expected results	Ship/Aircrafts act accordingly with the presence of enemy and allies. When the Missiles, not cannon shells are launched.
TC4	Test Description	This test case is to check the interaction of battle ship subsystem with detection/communication and Weapon subsystems. Enemies should be detected by the Radar on the ship/Aircrafts. Cannon shells should be launched when enemy ships enter the fire area of cannon.
	Input states	Some other ships on both sides are created and some enemy ships are placed within the detection range of Radar on the ship/Aircraft, which is also within the range of cannon.
	Expected results	Ship/Aircrafts act accordingly with the presence of enemy and allies. When there are enemies in the fire range of cannon (which is within the detect range of Radar on battle ships), cannon, not Missiles are launched.
TC5	Test Description	This test case is to check if Weapons fly in the right way and the targets should vanish when their resistance points reached.
	Input states	Some other ships of both sides and some enemy ships are placed in the fire range of Missile, some are placed in the fire range of cannon of ship/Aircrafts
	Expected results	Missiles, cannon shells fly towards enemies not allies and hit the enemies with certain precision. Enemies vanish when their resistance points reach.

Table 5-56 Test Cases and Results

5.3.3 Error Reports

The results found a “division by zero” error. After checking, the errors were found when using “unit()” of Vector class. Since unit() is actually calculated by deviding Vector by length, so length can not be zero. By adding the checking code to make sure the unit is not called when length is zero. Also other places where calculation includes division are checked.

6. Conclusion

This project is mainly focus on a case studying in software engineering. It uses use case driven approach and complies with the IEEE standards on Requirement Gathering, Specification and Software Design. By going through each phase (Analysis, Specification, Design, Implementation, Testing and Delivery/Maintenance) of NBSS, we prove that software development should be a systematical activity. The basic concerns of this project is improving the productivity of the programming/development process, improving the quality of software product at levels of Reliability, Efficiency, User-Friendly and Maintainability. The following list concludes the solutions and improvements for each phase of the project development:

6.1 Requirements Gathering and Specification

- Grouping the requirements based on each subsystem to make the requirement easy to be clarified;
- For requirements specification, describing the basic meaning with the extra comments to reduce the misunderstanding between customers and developers;
- Splitting multiple features requirement into sub-feature definitions and giving the sub-identification to them. Especially by giving each requirement an identification number, making the whole system to be easy traceable on user requirements;
- The Use-Case-Driven approach is used to describe the requirements. The UML diagrams used in this phase also are useful for system design.

6.2 Design and Implementing

- Defining the reasonable number of sub-systems and components for sub-systems;
- Forming components with low couple and high cohesion. Describing functionality for each module separately and describing their dependency to specify the relations among them;

- Defining the minimal while complete interface of each key class. By describing the interface, further to find out the detailed services of the class;
- Using Object-Oriented principles, explaining the role of each classes which designed as an abstraction;
- Implementation is combined with detailed design in the project, describing the detailed class information: data attributes, sequence of actions, functions, and local variables;
- Requirement ID is used with each class implementation to promise a good traceability.

6.3 Testing and Delivery/Maintenance

- Providing a integrated test design covering unit test , subsystem test and integration test for the system;
- Giving each test case a test case ID and matching them to Requirement ID to test on all the requirements;
- Specifying the input data and expected output clearly for each test, giving the pass criteria explicitly.

6.4 Further Improvements

The function prototype (pseudocode) needs to follow the well-defined syntax rules that shall be easy to learn and fast to use. The algorithm of class function also needs to be revisited in order to optimize the design. Some UML notations need to be verified to meet the requirement of the specific scenarios. For test case design, need to have the fail criteria for each test case.