Naval Battle Simulation System:
A Case Study in Software Engineering

LinFang Wang

A Major Report
in
The Department
of
Computer Science

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

March, 2002

© Linfang Wang, 2002
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 ........................................................................................................................................... 10  
   1.1 PURPOSE ........................................................................................................................................ 10  
   1.2 SCOPE ........................................................................................................................................... 10  
   1.3 DEFINITIONS, ACRONYMS, ABBREVIATIONS ........................................................................... 11  
   1.4 OVERVIEW ....................................................................................................................................... 12  
   1.5 REFERENCES .................................................................................................................................... 12  

2. GENERAL DESCRIPTION .............................................................................................................................. 13  
   2.1 PRODUCT PERSPECTIVE ........................................................................................................... 13  
   2.2 PRODUCT FUNCTIONS .............................................................................................................. 13  
   2.3 USER CHARACTERISTICS ............................................................................................................ 14  
   2.4 GENERAL CONSTRAINTS ............................................................................................................... 14  
   2.5 ASSUMPTIONS AND DEPENDENCIES ............................................................................................ 15  

3. SPECIFIC REQUIREMENTS ........................................................................................................................... 16  
   3.1 REQUIREMENT IDENTIFICATION ............................................................................................... 16  
   3.2 HIGH LEVEL USE CASE DESCRIPTION .................................................................................... 17  
   3.3 FUNCTIONAL REQUIREMENTS DESCRIPTION ................................................................................... 23  
      3.3.1 Simulation Controller Requirements ...................................................................................... 24  
      3.3.2 Communication/Detection Requirements ........................................................................ 40  
      3.3.3 Aircraft Carrier Requirements .......................................................................................... 58  
      3.3.4 Aircraft Requirements ........................................................................................................ 71  
      3.3.5 Destroyer Requirements ..................................................................................................... 86  
      3.3.6 Cruiser Requirements .......................................................................................................... 98  
      3.3.7 Battleship Requirements .................................................................................................... 110  
      3.3.8 Submarine Requirements ................................................................................................. 122  
      3.3.9 Weapons Requirements .................................................................................................... 134  
   3.4 EXTERNAL INTERFACE REQUIREMENTS ..................................................................................... 139  
      3.4.1 User Interface ....................................................................................................................... 139  
      3.4.2 Hardware Interface ............................................................................................................. 139  
      3.4.3 Software Interface .............................................................................................................. 139  
      3.4.4 Communication Interface ................................................................................................. 140  
   3.5 PERFORMANCE REQUIREMENTS ............................................................................................... 140  
   3.6 DESIGN CONSTRAINTS ..................................................................................................................... 141  
   3.7 QUALITY ATTRIBUTES ................................................................................................................... 141  
   3.8 OTHER REQUIREMENTS ................................................................................................................... 141  

4. SOFTWARE DESIGN ...................................................................................................................................... 142  
   4.1 DECOMPOSITION DESCRIPTION ............................................................................................... 142  
      4.1.1 Module Decomposition ......................................................................................................... 142  
      4.1.2 Concurrent Process Decomposition .................................................................................... 147  
      4.1.3 Data Decomposition ............................................................................................................. 147
4.2 DEPENDENCY DESCRIPTION .................................................................148
  4.2.1 Internal Module Dependency .......................................................148
  4.2.2 Internal Process Dependency ......................................................149
  4.2.3 Data Dependency .........................................................................149
4.3 INTERFACE DESCRIPTION ..........................................................150
  4.3.1 Module Interface ..........................................................................150
  4.3.2 Process Interface ..........................................................................156
4.4 SYSTEM DETAILED DESIGN .........................................................157
  4.4.1 Simulation Controller Detailed Design ...........................................157
  4.4.2 Communication/Detection Detailed Design ...............................171
  4.4.3 Ship and Aircraft Detailed Design .................................................190
  4.4.4 Weapon Detailed Design ..............................................................219

5. SYSTEM TESTING .................................................................................251
  5.1 UNIT TESTING ................................................................................251
    5.1.1 Unit Testing for Simulation Controller .................................252
    5.1.2 Unit Testing for Communication/Detection .........................255
    5.1.3 Unit Testing for All Vehicles ....................................................261
    5.1.4 Unit Testing for Weapons .......................................................269
  5.2 SUBSYSTEM TESTING .................................................................275
    5.2.1 Simulation Controller Subsystem Testing ..............................275
    5.2.2 Communication/Detection Subsystem Testing ......................278
    5.2.3 Ship/Aircraft Subsystem Testing .............................................279
    5.2.4 Weapon Subsystem Testing .....................................................280
  5.3 SYSTEM INTEGRATION TESTING ..................................................281
    5.3.1 Integration scheme .................................................................281
    5.3.2 Test Cases and Results ..............................................................281
    5.3.3 Error Reports ............................................................................283
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-1</td>
<td>Requirement Identifiers</td>
<td>16</td>
</tr>
<tr>
<td>3-2</td>
<td>Attacker List</td>
<td>23</td>
</tr>
<tr>
<td>3-3</td>
<td>Weapon List</td>
<td>23</td>
</tr>
<tr>
<td>3-4</td>
<td>Use Case Set up Operational Parameters</td>
<td>29</td>
</tr>
<tr>
<td>3-5</td>
<td>Sequence Diagram for Use Case: Start Simulation</td>
<td>31</td>
</tr>
<tr>
<td>3-6</td>
<td>Use Case Description for Simulate Communication</td>
<td>32</td>
</tr>
<tr>
<td>3-7</td>
<td>Use Case Description for Base Supply</td>
<td>34</td>
</tr>
<tr>
<td>3-8</td>
<td>Use Case Description for Pause Simulation</td>
<td>36</td>
</tr>
<tr>
<td>3-9</td>
<td>Use Case Description for Resume Simulation</td>
<td>37</td>
</tr>
<tr>
<td>3-10</td>
<td>Use Case Description for End Simulation</td>
<td>38</td>
</tr>
<tr>
<td>3-11</td>
<td>Use Case Description for Report Statistics</td>
<td>39</td>
</tr>
<tr>
<td>3-12</td>
<td>Use Case Description for Turn on Radar</td>
<td>44</td>
</tr>
<tr>
<td>3-13</td>
<td>Use Case Description for Turn off Radar</td>
<td>45</td>
</tr>
<tr>
<td>3-14</td>
<td>Use Case Description for Radar Emit Wave</td>
<td>46</td>
</tr>
<tr>
<td>3-15</td>
<td>Use Case Description for Radar Receive Wave</td>
<td>47</td>
</tr>
<tr>
<td>3-16</td>
<td>Use Case Description for Turn on Sonar</td>
<td>48</td>
</tr>
<tr>
<td>3-17</td>
<td>Use Case Description for Turn off Sonar</td>
<td>49</td>
</tr>
<tr>
<td>3-18</td>
<td>Use Case Description for Sonar Emit Wave</td>
<td>50</td>
</tr>
<tr>
<td>3-19</td>
<td>Use Case Description for Sonar Receive Wave</td>
<td>51</td>
</tr>
<tr>
<td>3-20</td>
<td>Use Case Description for Turn on Radio</td>
<td>52</td>
</tr>
<tr>
<td>3-21</td>
<td>Use Case Description for Turn off Radio</td>
<td>53</td>
</tr>
<tr>
<td>3-22</td>
<td>Use Case Description for Radio Send Message</td>
<td>54</td>
</tr>
<tr>
<td>3-23</td>
<td>Use Case Description for Radio Receive Message</td>
<td>56</td>
</tr>
<tr>
<td>3-24</td>
<td>Use Case Description for Aircraft Carrier Navigation Control</td>
<td>63</td>
</tr>
<tr>
<td>3-25</td>
<td>Use Case Description for Aircraft Carrier Communicate with Allies</td>
<td>64</td>
</tr>
<tr>
<td>3-26</td>
<td>Use Case Description for Aircraft Carrier Make Decision</td>
<td>65</td>
</tr>
<tr>
<td>3-27</td>
<td>Use Case Description for Aircraft Control</td>
<td>67</td>
</tr>
<tr>
<td>3-28</td>
<td>Use Case Description for Aircraft Carrier Update Status</td>
<td>69</td>
</tr>
<tr>
<td>3-29</td>
<td>Use Case Description for Aircraft Carrier Refueling</td>
<td>70</td>
</tr>
<tr>
<td>3-30</td>
<td>Use Case Description for Aircraft Navigation Control</td>
<td>77</td>
</tr>
<tr>
<td>3-31</td>
<td>Use Case Description for Aircraft Detect Enemy</td>
<td>79</td>
</tr>
<tr>
<td>3-32</td>
<td>Use Case Description for Aircraft Communication with Allies</td>
<td>80</td>
</tr>
<tr>
<td>3-33</td>
<td>Use Case Description for Aircraft Make Decision</td>
<td>81</td>
</tr>
<tr>
<td>3-34</td>
<td>Use Case Description for Aircraft Weapon Control</td>
<td>83</td>
</tr>
<tr>
<td>3-35</td>
<td>Use Case Description for Aircraft Update Status</td>
<td>84</td>
</tr>
<tr>
<td>3-36</td>
<td>Use Case Description for Aircraft Refueling</td>
<td>85</td>
</tr>
<tr>
<td>3-37</td>
<td>Use Case Description for Destroyer Navigation Control</td>
<td>91</td>
</tr>
<tr>
<td>3-38</td>
<td>Use Case Description for Destroyer Navigation Control</td>
<td>92</td>
</tr>
<tr>
<td>3-39</td>
<td>Use Case Description for Destroyer Communication with Allies</td>
<td>93</td>
</tr>
<tr>
<td>3-40</td>
<td>Use Case Description for Destroyer Make Decision</td>
<td>94</td>
</tr>
<tr>
<td>3-41</td>
<td>Use Case Description for Destroyer Weapon Control</td>
<td>95</td>
</tr>
<tr>
<td>3-42</td>
<td>Use Case Description for Destroyer Update Status</td>
<td>96</td>
</tr>
<tr>
<td>3-43</td>
<td>Use Case Description for Destroyer Rearm and Refueling</td>
<td>97</td>
</tr>
<tr>
<td>3-44</td>
<td>Use Case Description for Cruiser Navigation Control</td>
<td>103</td>
</tr>
<tr>
<td>3-45</td>
<td>Use Case Description for Cruiser Navigation Control</td>
<td>104</td>
</tr>
<tr>
<td>3-46</td>
<td>Use Case Description for Aircraft Carrier Communication with Allies</td>
<td>105</td>
</tr>
<tr>
<td>3-47</td>
<td>Use Case Description for Cruiser Make Decision</td>
<td>106</td>
</tr>
<tr>
<td>3-48</td>
<td>Use Case Description for Cruiser Weapon Control</td>
<td>107</td>
</tr>
<tr>
<td>3-49</td>
<td>Use Case Description for Cruiser Update Status</td>
<td>108</td>
</tr>
<tr>
<td>3-50</td>
<td>Use Case Description for Cruiser Rearm and Refueling</td>
<td>109</td>
</tr>
<tr>
<td>3-51</td>
<td>Use Case Description for Battleship Navigation Control</td>
<td>115</td>
</tr>
<tr>
<td>TABLE 5-38</td>
<td>UNIT TEST CASE FOR CWActiveStateController initial position function ..........270</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-39</td>
<td>UNIT TEST CASE FOR CWAutoAimController trace target function ......................270</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-40</td>
<td>UNIT TEST CASE FOR CWAutoAimController trace target (AIRCRAFT) .................271</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-41</td>
<td>UNIT TEST CASE FOR CWAutoAimController trace target (SUBMARINE) ...............271</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-42</td>
<td>UNIT TEST CASE FOR CWChargeController HitDetect function ..........................272</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-43</td>
<td>UNIT TEST CASE FOR CWChargeController HitDetect(AIRCRAFT) function ............272</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-44</td>
<td>UNIT TEST CASE FOR CWChargeController HitDetect(SUBMARINE) .....................273</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-45</td>
<td>UNIT TEST CASE CWCharge detonate Target function .....................................273</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-46</td>
<td>UNIT TEST CASE CWRudder change Velocity function ......................................274</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-47</td>
<td>TEST CASE FOR SIMULATION CONTROLLER(SetUpDLG) SUBSYSTEM .....................275</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-48</td>
<td>TEST CASE FOR SIMULATION CONTROLLER(Vector) SUBSYSTEM ..........................275</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-49</td>
<td>TEST CASE FOR SIMULATION CONTROLLER (SC) SUBSYSTEM ...........................276</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-50</td>
<td>TEST CASE FOR SIMULATION CONTROLLER (VEHICLE FACTORY) SUBSYSTEM ..........276</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-51</td>
<td>TEST CASE FOR SIMULATION CONTROLLER (CONTROLLER) SUBSYSTEM ................277</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-52</td>
<td>TEST CASE FOR COMMUNICATION/DETECTION SUBSYSTEM ..................................278</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-53</td>
<td>TEST CASE FOR SHIP/AIRCRAFT SUBSYSTEM ..................................................279</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-54</td>
<td>TEST CASE FOR WEAPON(WTORPEDO) SUBSYSTEM ............................................280</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-55</td>
<td>TEST CASE FOR WEAPON (WCANNON SHELL) SUBSYSTEM ..................................280</td>
<td></td>
</tr>
<tr>
<td>TABLE 5-56</td>
<td>TEST CASES AND RESULTS ..............................................................................282</td>
<td></td>
</tr>
</tbody>
</table>
List of Figures

FIGURE 3-1  SEQUENCE DIAGRAM FOR USE CASE NAVIGATION CONTROL ..............................................17
FIGURE 3-2  SEQUENCE DIAGRAM FOR USE CASE DETECT ENEMY ..................................................18
FIGURE 3-3  SEQUENCE DIAGRAM FOR USE CASE COMMUNICATE WITH ALLIES .........................18
FIGURE 3-4  SEQUENCE DIAGRAM FOR USE CASE MAKE DECISION .............................................19
FIGURE 3-5  SEQUENCE DIAGRAM FOR USE CASE WEAPON CONTROL ...........................................19
FIGURE 3-6  SEQUENCE DIAGRAM FOR USE CASE UPDATE STATUS .............................................20
FIGURE 3-7  SEQUENCE DIAGRAM FOR USE CASE REARM AND REFUELING .................................20
FIGURE 3-8  SEQUENCE DIAGRAM FOR USE CASE TURN ON COMMUNICATION/DETECTION ..........21
FIGURE 3-9  SEQUENCE DIAGRAM FOR USE CASE TURN OFF COMMUNICATION/DETECTION ..........21
FIGURE 3-10 SEQUENCE DIAGRAM FOR USE CASE DETECTION EMIT WAVE ..................................22
FIGURE 3-11 SEQUENCE DIAGRAM FOR USE CASE DETECTION RECEIVE WAVE ..........................22
FIGURE 3-12 USE CASE DIAGRAM FOR SIMULATION CONTROLLER ..............................................24
FIGURE 3-13 SEQUENCE DIAGRAM FOR USE CASE: SET UP OPERATIONAL PARAMETERS ............30
FIGURE 3-14 SEQUENCE DIAGRAM FOR USE CASE: START SIMULATION .....................................31
FIGURE 3-15 SEQUENCE DIAGRAM FOR USE CASE: SIMULATE COMMUNICATION .....................33
FIGURE 3-16 SEQUENCE DIAGRAM FOR USE CASE: BASE SUPPLIER ...........................................35
FIGURE 3-17 SEQUENCE DIAGRAM FOR USE CASE: PAUSE SIMULATION .....................................36
FIGURE 3-18 SEQUENCE DIAGRAM FOR USE CASE: RESUME SIMULATION ...............................37
FIGURE 3-19 SEQUENCE DIAGRAM FOR USE CASE: END SIMULATION .......................................38
FIGURE 3-20 SEQUENCE DIAGRAM FOR USE CASE: REPORT STATISTICS ..................................39
FIGURE 3-21 USE CASE DIAGRAM FOR COMMUNICATION/DETECTION .........................................40
FIGURE 3-22 SEQUENCE DIAGRAM FOR USE CASE RADIO SEND MESSAGE ...............................55
FIGURE 3-23 SEQUENCE DIAGRAM FOR USE CASE RADIO RECEIVE MESSAGE .........................57
FIGURE 3-24 USE CASE DIAGRAM FOR AIRCRAFT CARRIER .........................................................58
FIGURE 3-25 SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT CARRIER MAKE DECISION ...........66
FIGURE 3-26 SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT CARRIER AIRCRAFT CONTROL ....68
FIGURE 3-27 USE CASE DIAGRAM FOR AIRCRAFT ........................................................................71
FIGURE 3-28 SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT NAVIGATION CONTROL ............78
FIGURE 3-29 SEQUENCE DIAGRAM FOR USE CASE AIRCRAFT MAKE DECISION ........................82
FIGURE 3-30 USE CASE DIAGRAM FOR DESTROYER .................................................................86
FIGURE 3-31 USE CASE DIAGRAM FOR CRUISER ........................................................................98
FIGURE 3-32 USE CASE DIAGRAM FOR BATTLESHIP ..................................................................110
FIGURE 3-33 USE CASE DIAGRAM FOR SUBMARINE ..................................................................122
FIGURE 3-34 USE CASE DIAGRAM FOR WEAPON .........................................................................134
FIGURE 3-35 SEQUENCE DIAGRAM FOR USE CASE WEAPON PROVIDE LOCATION .................136
FIGURE 3-36 SEQUENCE DIAGRAM FOR USE CASE WEAPON AIM TARGET ..............................137
FIGURE 3-37 SEQUENCE DIAGRAM FOR USE CASE WEAPON FIRE AND HIT TARGET ...............138
FIGURE 4-1  INTERACTION DIAGRAM FOR SUBSYSTEMS OF THE NBSS ....................................142
FIGURE 4-2  ARCHITECTURE OF THE NAVAL BATTLE SIMULATION SYSTEM .........................143
FIGURE 4-3  CLASS LEVEL INTERFACE DIAGRAM OF THE NAVAL BATTLE SIMULATION SYSTEM 144
FIGURE 4-4  SIMULATION CONTROLLER_FOR_COMMUNICATION/DETECTION ..............150
FIGURE 4-5  RADAR/SONAR_FOR_WEAPON ..................................................................................152
FIGURE 4-6  BASESHIP_FOR_SC ...............................................................................................154
FIGURE 4-7  BASEWEAPON_FOR_SIMULATION CONTROLLER ......................................................155
FIGURE 4-8  BASEWEAPON_FOR_SHIP AND AIRCRAFT ..............................................................156
FIGURE 4-9  CLASS DIAGRAM FOR SIMULATION CONTROLLER MODULE ...............................158
FIGURE 4-10 CLASS DIAGRAM FOR COMMUNICATION/DETECTION MODULE .......................171
FIGURE 4-11 CLASS DIAGRAM FOR BASESHIP (SHIP AND AIRCRAFT) MODULE ....................191
FIGURE 4-12 CLASS DIAGRAM FOR WEAPON MODULE ...........................................................219
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. We did our best 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

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

This document is organized in six major sections and data dictionary in appendix. 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

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. Acts as the communication media.
- **Communication/Detection**: responsible for detecting enemies and communicating with allies, also simulates aiming system for Weapons.
- **Aircraft Carrier**: cooperate with Aircraft to locate and destroy enemy ships and Aircraft.
- **Aircraft**: cooperate 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. Provide an interactive user interface
2. Simulate the communication media
3. Generate the vehicles for both sides
4. Animate the movements of vehicles
5. Generate the fuel and Weapon upon request

**Vehicles (Battleship, Cruiser, Destroyer, Submarine, Aircraft)**
1. Navigate on the map
2. Detect the enemy
3. Communicate with allies
4. Launch Weapon to attack targets
5. Make strategic decisions
**Aircraft Carrier**
1. Navigate on the map
2. Manage Aircraft take-offs
3. Manage Aircraft landings
4. Assign missions to Aircrafts
5. Communicate with allies
6. Make strategic decisions

**Communication/Detection:**
1. Pass information to the Simulation Controller
2. Detect vehicles
3. Enable communication between vehicles
4. Simulate the detecting system for Weapons

**Weapon:**
1. Aiming at a target
2. Fire at a target
3. Hit a target
4. Inflict damage to a vehicle

For the product functions 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.
- 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.

<table>
<thead>
<tr>
<th>Subsystem</th>
<th>Prefix</th>
<th>Maximal #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simulation Controller</td>
<td>SC</td>
<td>019</td>
</tr>
<tr>
<td>Communication/Detection</td>
<td>CD</td>
<td>012</td>
</tr>
<tr>
<td>Aircraft Carrier</td>
<td>AC</td>
<td>026</td>
</tr>
<tr>
<td>Aircraft</td>
<td>AT</td>
<td>034</td>
</tr>
<tr>
<td>Destroyer</td>
<td>DT</td>
<td>034</td>
</tr>
<tr>
<td>Cruiser</td>
<td>CS</td>
<td>034</td>
</tr>
<tr>
<td>Battleship</td>
<td>BS</td>
<td>030</td>
</tr>
<tr>
<td>Submarine</td>
<td>SM</td>
<td>034</td>
</tr>
<tr>
<td>Weapons</td>
<td>WP</td>
<td>008</td>
</tr>
</tbody>
</table>

Table 3-1  Requirement Identifiers

The “Last Used #” is the last number that was assigned to a requirement in a particular subsystem. 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 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**

![Sequence Diagram for Use Case Navigation Control](image)

**Figure 3-1** Sequence Diagram for Use Case Navigation Control
Detect Enemy

![Sequence Diagram for Use Case Detect Enemy](image)

Figure 3-2 Sequence Diagram for Use Case Detect Enemy

Communicate with Allies

![Sequence Diagram for Use Case Communicate with Allies](image)

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**

![Sequence Diagram for Use Case Turn on Communication/Detection](image)

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

**Turn off Communication/Detection Device**

![Sequence Diagram for Use Case Turn off Communication/Detection](image)

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**

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Possible target</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircraft Carrier</td>
<td>No attack ability</td>
</tr>
<tr>
<td>Aircraft</td>
<td>Aircraft Carrier, Battleship, Cruiser, Destroyer, Aircraft</td>
</tr>
<tr>
<td>Battleship</td>
<td>Aircraft, Aircraft Carrier, Cruiser, Destroyer, Battleship</td>
</tr>
<tr>
<td>Cruiser</td>
<td>Aircraft</td>
</tr>
<tr>
<td>Destroyer</td>
<td>Submarine</td>
</tr>
<tr>
<td>Submarine</td>
<td>Battleship, Cruiser, Destroyer, Submarine</td>
</tr>
</tbody>
</table>

Table 3-2  Attacker List

**Weapon list**

<table>
<thead>
<tr>
<th>Attacker</th>
<th>Possible Weapon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aircraft Carrier</td>
<td>No attack ability.</td>
</tr>
<tr>
<td>Aircraft</td>
<td>Air-Air Missile, Air-Sea Missile</td>
</tr>
<tr>
<td>Battleship</td>
<td>Sea-Sea Missile, Sea-Air Missile, Heavy Cannon Shell, Torpedo</td>
</tr>
<tr>
<td>Cruiser</td>
<td>Sea-Air Missile</td>
</tr>
<tr>
<td>Destroyer</td>
<td>Sea-Sub Missile</td>
</tr>
<tr>
<td>Submarine</td>
<td>Sub-Sea Torpedo, Torpedo</td>
</tr>
</tbody>
</table>

Table 3-3  Weapon 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

![Use Case Diagram for Simulation Controller](image)
### 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. *No comments.*

#### 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 depending on the production rate.
No comments.

SC-015-02  Produce Fuel
The base supplier shall produce the specific amount of fuel 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the user to initialize all the objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to start the simulation</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is not in running state or in pause state.</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>If the configuration exceeds the limitation or dissatisfies required conditions, the warning massage window will pop up.</td>
</tr>
</tbody>
</table>

**Post-Condition**

| 1. The valid input data are saved; 2. Set up window is closed. |

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>Simulate Communication</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Other Requirements**

| NA |

Table 3-4  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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to start the simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>The user has set up the parameters</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>1. The user presses the “Start” button.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. The system initializes the map, media, creates agents</td>
</tr>
<tr>
<td></td>
<td>3. Simulation begins.</td>
</tr>
</tbody>
</table>

**Alternate Path**

| NA |

**Post-Condition**

| Simulation successfully started |

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>Simulate Communication</th>
</tr>
</thead>
</table>

**Extending Use Case**

| NA |

**Other Requirements**

| NA |

Table 3-5  Sequence Diagram for Use Case: Start Simulation

**Sequence Diagram**

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow SC and vehicles to communicate with each other, and allow to turn on/off the Radar/Sonar and Radio.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>High level description</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in running state</td>
</tr>
</tbody>
</table>

**Flow of Events**

| Base Path | 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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Post-Condition**
The SC know the status of agents, and all the agents are aware of the presence of other agents within their Communication/Detection range

**Table 3-6 Use Case Description for Simulate Communication**

**Sequence Diagram**

See next page.
Figure 3-15  Sequence Diagram for Use Case: Simulate Communication
### 3.3.1.3.4 Use Case: Base Supply

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the SC to provide supplies (Weapons, fuel, ships) to both sides when the simulation is running.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS Ships and Aircraft</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in running state</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>base path</td>
<td>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.</td>
</tr>
<tr>
<td>alternate path</td>
<td>NA.</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>1. The ships are generated when the simulation is running 2. The ships get rearmed and refueed.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td></td>
</tr>
<tr>
<td>used use case</td>
<td>Simulate Communication</td>
</tr>
<tr>
<td>extending use case</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-7  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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the user to pause the simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in running state</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. The user presses the “Pause” button</td>
</tr>
<tr>
<td></td>
<td>2. The system pauses the clock and suspends the simulation</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The system saved the current status of all agents and SC also.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>Extending Use Case</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-8 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the user to resume the simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in pause state.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. The user presses the “Resume” button.</td>
</tr>
<tr>
<td></td>
<td>2. The system resumes the simulation.</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Simulation resumes execution.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>Extending Use Case</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-9  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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the user to stop the simulation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to stop the simulation system</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in running state or in pause state.</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>Alternate Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. The user presses the “End” button</td>
<td>NA</td>
</tr>
<tr>
<td>2. The system terminates the simulation</td>
<td></td>
</tr>
</tbody>
</table>

**Post-Condition**

Clean up all the agents, and ready for next simulation.

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>Extending Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Other Requirements</th>
<th>NA</th>
</tr>
</thead>
</table>

Table 3-10 Use Case Description for End Simulation

**Sequence Diagram**

![Sequence Diagram](image)

Figure 3-19 Sequence Diagram for Use Case: End Simulation
### 3.3.1.3.8 Use Case: Report Statistics

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to allow the user to view an execution report of the running simulation.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NBSS User</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Simulation is in running state or in pause state, or ended successfully.</td>
</tr>
</tbody>
</table>

#### Flow of Events

<table>
<thead>
<tr>
<th>Base Path</th>
<th>1. The user presses the “Report” button.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. The system displays a statistics window.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>If the simulation terminated erroneously, the statistics window will show nothing.</td>
</tr>
</tbody>
</table>

#### Post-Condition

Save the valid statistics data, and close the statistics window.

### Related Use Case

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
</tr>
</tbody>
</table>

### Other Requirements

NA

**Table 3-11 Use Case Description for Report Statistics**

**Sequence Diagram**

![Sequence Diagram](image)

**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

![Use Case Diagram for Communication/Detection](image)

**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 by 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service to allow the user to turn on the Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Should have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. User</td>
</tr>
<tr>
<td></td>
<td>2. Simulation Controller</td>
</tr>
<tr>
<td></td>
<td>3. Battleship, Cruiser, Aircraft</td>
</tr>
<tr>
<td></td>
<td>4. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile</td>
</tr>
<tr>
<td>Pre-Condition</td>
<td>Radar is in the “off” state</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>1. User clicks on the “Set Radar” button, the system display Radar setting window.</td>
</tr>
<tr>
<td></td>
<td>2. User selects the object from object list.</td>
</tr>
<tr>
<td></td>
<td>3. User set state on for Radar, and close the window.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Radar is in the “on” state</td>
</tr>
<tr>
<td>Related Use Cases</td>
<td>Used Use Case NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Turn on Communication/Detection</td>
</tr>
<tr>
<td>Other Requirement</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-12 Use Case Description for Turn on Radar

Sequence Diagram

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service to allow the user to turn off the Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Should have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. User</td>
</tr>
<tr>
<td></td>
<td>2. Simulation Controller</td>
</tr>
<tr>
<td></td>
<td>3. Battleship, Cruiser, Aircraft</td>
</tr>
<tr>
<td></td>
<td>4. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile</td>
</tr>
<tr>
<td>Pre-Condition</td>
<td>Radar is in on state</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>1. User click “Set Radar” button, the system display</td>
</tr>
<tr>
<td></td>
<td>Radar setting window.</td>
</tr>
<tr>
<td></td>
<td>2. User select the object from object list;</td>
</tr>
<tr>
<td></td>
<td>3. User set state off for Radar, and close the window.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Radar is in off state</td>
</tr>
<tr>
<td>Related Use Cases</td>
<td></td>
</tr>
<tr>
<td>Used Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Turn off Communication/Detection</td>
</tr>
<tr>
<td>Other Requirement</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-13 Use Case Description for Turn off Radar

Sequence Diagram

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service for objects to send info to the SC in order to detect the surrounding enemies by using Radar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. Simulation Controller\n2. Battleship, Cruiser, Aircraft\n3. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile</td>
</tr>
<tr>
<td>Pre-Condition</td>
<td>1. Object exist and Radar is created;\n2. Object know its position, ID and flag;\n3. The DB of SC is accessible.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>1. Radar gets its owner’s ID, position and flag;\n2. Radar sends its owner’s information to SC;</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>If position DB is not accessible, SC return an error to the object,</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Radar send its owner’s info to SC</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Related Use Cases</th>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>Detection Emit Wave</td>
<td></td>
</tr>
<tr>
<td>Other Requirement</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-14  Use Case Description for Radar Emit Wave

Sequence Diagram

Refer to Figure 3-10  Sequence Diagram for Use Case Detection Emit 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).
### 3.3.2.3.4 Use Case: Radar Receive Wave

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service to allow the objects to receive the information from SC in order to detect the surrounding enemies by using a Radar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>
| Actor       | 1. Simulation Controller  
2. Battleship, Cruiser, Aircraft  
3. Sea-Sea Missile, Sea-Air Missile, Air-Air Missile, Air-Sea Missile |
| Pre-Condition | 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. |
| Flow of Events | 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 3-15  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

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

Table 3-16 Use Case Description for Turn on Sonar

Sequence Diagram

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

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

**Table 3-17 Use Case Description for Turn off Sonar**

**Sequence Diagram**

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service for objects to send info to SC in order to detect the surrounding enemies using a Sonar.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>
| Actor       | 1. Simulation Controller  
             | 2. Destroyer, Submarine, and Torpedo                                                              |
| Pre-Condition | 1. Object exists, Radar is created and in on state;  
             | 2. Object know 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                                                                                                  |
|             | Extending Use Case  
             | Detection Emit Wave                                                                                 |
| Other Requirement | NA                                                                                                  |

| Table 3-18 Use Case Description for Sonar Emit Wave |

**Sequence Diagram**

Refer to Figure 3-10 Sequence Diagram for Use Case Detection Emit Wave, 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**

1. Simulation Controller
2. Destroyer, Submarine, and Torpedo

**Pre-Condition**

1. Object exist and Radar is created and in on state;
2. Object know its position, ID and flag;
3. The DB of SC is accessible.
4. Sonar’s data buffer id available.

**Flow of Events**

**Base Path**

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**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
</table>

**Extending Use Case**

Detection Receive Wave

**Other Requirement**

NA

---

**Table 3-19 Use Case Description for Sonar Receive Wave**

### Sequence Diagram

Refer to Figure 3-10 Sequence Diagram for Use Case Detection Emit Wave for Detection Receive Wave, 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

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

**Table 3-20  Use Case Description for Turn on Radio**

**Sequence Diagram**

Refer to Figure 3-8 Sequence Diagram for Use Case Turn on Communication/Detection. The object list (ID list) is provided to user for Aircraft Carrier, Aircraft, Battleship, Cruiser, Destroyer, and Submarine.
### Use Case: Turn off Radio

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service to allow the user to turn off the Radio</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Should have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. User</td>
</tr>
<tr>
<td></td>
<td>2. Simulation Controller</td>
</tr>
<tr>
<td>Pre-Condition</td>
<td>Radio is in on state</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>1. User clicks the “Set Radio” button, the system display Radar setting window.</td>
</tr>
<tr>
<td></td>
<td>2. User selects the object from object list;</td>
</tr>
<tr>
<td></td>
<td>3. User sets state off for Radio, and closes the window.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Radio is in off state</td>
</tr>
<tr>
<td>Related Use Cases</td>
<td></td>
</tr>
<tr>
<td>Used Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Turn off Communication/Detection</td>
</tr>
<tr>
<td>Other Requirement</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-21  Use Case Description for Turn off Radio**

**Sequence Diagram**

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service for objects send the message to its allies via SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. Simulation Controller</td>
</tr>
<tr>
<td>Pre-Condition</td>
<td>1. Object exists and Radio is created and in “on” state.</td>
</tr>
<tr>
<td></td>
<td>2. Object know its position, ID and flag;</td>
</tr>
<tr>
<td></td>
<td>3. Object know the receivers's IDs and message it want to send.</td>
</tr>
<tr>
<td></td>
<td>4. A data buffer for the message is available.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. Object sends a message to its Radio;</td>
</tr>
<tr>
<td></td>
<td>2. Radio passes the message to message DB;</td>
</tr>
<tr>
<td></td>
<td>3. Message DB check with SC to see if the receivers is within the Radio’s range of sender;</td>
</tr>
<tr>
<td></td>
<td>4. Message DB keep the message in message list.</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The message is available in the message DB for the receiver to retrieve them when needed.</td>
</tr>
<tr>
<td>Related Use Cases</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>Extending Use Case</td>
</tr>
<tr>
<td>Other Requirement</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-22 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service for objects receive the message from its allies via the SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>
| 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; |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

| Post-Condition | 1. The messages are deleted from the DB;  
2. Object receives a message list containing zero or more messages. |

<table>
<thead>
<tr>
<th>Related Use Cases</th>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

| Other Requirement | NA |

Table 3-23 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

![Use Case Diagram for Aircraft Carrier](image-url)

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.
No comments.
AC-024 Aircraft Carrier Crashed with other object
Aircraft Carrier shall determine to be destroyed when crash with other object.
No comments.

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Aircraft Carrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Existing Aircraft Carrier object;</td>
</tr>
<tr>
<td></td>
<td>2. A command is received from the navigation officer</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>Alternate Path: NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Aircraft Carrier is moved</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case: Aircraft Carrier Make Decision</td>
</tr>
<tr>
<td></td>
<td>Extending Use Case: Navigation Control</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-24 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the communication service between Aircraft Carrier and its allies.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to pass information to the Aircraft Carrier's allies</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Existing Aircraft Carrier object</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
</tbody>
</table>
| Base Path | 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 massage to its allies |
| Alternate Path | NA |

**Post-Condition**  
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 3-25 | Use Case Description for Aircraft Carrier Communicate with Allies |

**Sequence Diagram**

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
### 3.3.3.3.3 Use Case: Aircraft Carrier Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, decide where to conduct the ship, decide to rearm and refuel</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
</tbody>
</table>

#### Pre-Conditions
1. Existing Aircraft Carrier object;
2. The Aircraft Carrier’s status is updated;
3. All the reports are received

#### Flow of Events
- **Base Path**
  1. Upon reception of reports, the Captain analyzes the threats and decides to attack a target;
  2. The Captain gives the order to the Navigation Officer for where to conduct the ship and at what speed;
  3. The Captain gives order to Aircraft Launch officer to prepare the attack;
  4. The Captain gives order to Communication Officer to send out the message about detected enemy;
  5. The Captain decide to rearm or refueling to send request to SC.
  6. The Aircraft Launcher Officer decide to launch the Aircraft.

- **Alternate Path**
  NA

#### Post-Condition
1. The Navigation Officer executes captain’s command
2. The Weapon Officer executes captain’s command
3. The Communication Officer execute Captain’s command;
4. The Base Supplier perform the transaction task;

#### Related Use Case
1. Aircraft Carrier Update Status;
2. Aircraft Carrier Detect Enemy;
3. Aircraft Carrier Communication with Allies;

#### Extending Use Case
Make Decision

#### Other Requirements
NA

| Table 3-26 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to control the Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to control the Aircraft</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Aircraft</td>
</tr>
</tbody>
</table>
| Pre-Conditions       | 1. The Aircraft Carrier object exist;  
                      | 2. The Aircraft object exist;  
                      | 3. The Aircraft need to be take off. |
| Flow of Events       | Base Path 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 3-27 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Aircraft Carrier’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Aircraft Carrier object</td>
</tr>
</tbody>
</table>

#### Flow of Events

**Base Path**

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

#### Post-Condition

The status of the Aircraft Carrier is updated

#### Related Use Case

- Used Use Case: NA
- Extending Use Case: Update Status

#### Other Requirements

- NA

<table>
<thead>
<tr>
<th>Table 3-28</th>
<th>Use Case Description for Aircraft Carrier Update Status</th>
</tr>
</thead>
</table>

### Sequence Diagram

Refer to Figure 3-6  Sequence Diagram for Use Case Update Status.
3.3.3.3.6 Use Case: Aircraft Carrier Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to refueling the Aircraft Carrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. The base supplier has enough fuel in stock; 2. The Radio is in “on” state.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>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;</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Aircraft Carrier gets refueled</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>Aircraft Carrier Make Decision</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-29 Use Case Description for Aircraft Carrier Refueling

Sequence Diagram

Refer to Figure 3-7 Sequence Diagram for Use Case Rarm 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

![Use Case Diagram for Aircraft](image-url)

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 steer 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. 
*No comments.*

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.  
*No comments.*

**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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
</tbody>
</table>

Pre-Conditions:  
1. Existing Aircraft object;  
2. A command is received from the navigation officer

Flow of Events
- Base Path
  1. Upon reception of the command from a navigation officer, the Aircraft may perform one of following operations: Start or stop, Rotate, Accelerate, Decelerate;
  2. 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 3-30 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to locate the enemy using Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to detect the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Existing Aircraft object</td>
</tr>
</tbody>
</table>

#### Flow of Events

**Base Path**

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

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>Extending Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
<td>Detect Enemy</td>
</tr>
</tbody>
</table>

**Other Requirements**

NA

---

**Table 3-31 Use Case Description for Aircraft Detect Enemy**

**Sequence Diagram**

Refer to Figure 3-2 Sequence Diagram for Use Case Detect Enemy.
### 3.3.4.3.3 Use Case: Aircraft Communicate with Allies

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the communication service among Aircraft, its allies, and its Aircraft Carrier.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to pass information to the Aircraft’s allies and its Aircraft Carrier</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Aircraft object</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. Initialize a Radio object with location and radius when Aircraft is created;</td>
</tr>
<tr>
<td></td>
<td>2. Update Radio location;</td>
</tr>
<tr>
<td></td>
<td>3. Turn on/off Radio;</td>
</tr>
<tr>
<td></td>
<td>4. Get enemy object information around the Aircraft;</td>
</tr>
<tr>
<td></td>
<td>5. Send message to its allies and its Aircraft Carrier.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Aircraft received report from its allies and Aircraft Carrier; the allies and Aircraft Carrier received report from Aircraft.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Extending Use Case</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Communicate with Allies</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-32 Use Case Description for Aircraft Communication with allies**

**Sequence Diagram**

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
### 3.3.4.3.4 Use Case: Aircraft Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, decide where to conduct the Aircraft, decide rearm and refueling</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
</tbody>
</table>
| Pre-Conditions | 1. Existing Aircraft object;  
                 2. The Aircraft status is updated;  
                 3. All the reports are received                  |

#### Flow of Events

<table>
<thead>
<tr>
<th>Base Path</th>
<th>Flow of Events</th>
</tr>
</thead>
</table>
| 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 Aircraft 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 about detected enemy;  
5. The Captain decide to rearm or refueling to send request to SC.  
6. The Pilot decide to land on the Aircraft Carrier. |
| Alternate Path | NA |

#### Post-Condition

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;  
5. Aircraft send request to land on.

#### Related Use Case

1. Aircraft Update Status;  
2. Aircraft Detect Enemy;  
3. Aircraft Communication with Allies;

#### Extending Use Case

Make Decision

#### Other Requirements

NA

**Table 3-33 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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to attack the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Weapon</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>An attacking command is received</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>If the Weapon is Sea-Sea Missile, it will return a massage stating whether the target is destroyed or not.</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Weapon is fired and exploded</td>
</tr>
</tbody>
</table>

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>Aircraft Make Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>Weapon Control</td>
</tr>
</tbody>
</table>

**Other Requirements**

| NA |

Table 3-34 Use Case Description for Aircraft Weapon Control

**Sequence Diagram**

Refer to Figure 3-5 Sequence Diagram for Use Case Weapon Control.
3.3.4.3.6 Use Case: Aircraft Update Status

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Aircraft’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Aircraft object</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>1. Update the location of the Aircraft</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Determine if the Aircraft is hit by Weapon</td>
</tr>
<tr>
<td></td>
<td>3. Get the hit points of the Aircraft</td>
</tr>
<tr>
<td></td>
<td>4. Determine if the Aircraft can recover from the hit points</td>
</tr>
<tr>
<td></td>
<td>5. Determine if the Aircraft is destroyed</td>
</tr>
<tr>
<td></td>
<td>6. Determine if the Aircraft crashes with other object</td>
</tr>
</tbody>
</table>

**Alternate Path**

| Alternate Path | NA |

**Post-Condition**

| The status of the Aircraft is updated |

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
<td></td>
</tr>
</tbody>
</table>

| Other Requirements | NA |

Table 3-35  Use Case Description for Aircraft Update Status

**Sequence Diagram**

Refer to Figure 3-6  Sequence Diagram for Use Case Update Status.
3.3.4.3.7 Use Case: Aircraft Rearm and Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to refueling the Aircraft Carrier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>

| Pre-Conditions | 1. The base supplier has enough fuel in stock; 2. The Radio is in ON state. |
| Flow of Events | 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-Condition | The Aircraft get refueling |

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
</tr>
</tbody>
</table>

| Other Requirements | NA |

Table 3-36  Use Case Description for Aircraft Refueling

Sequence Diagram

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

![Use Case Diagram for Destroyer](image)

*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 not for Destroyer.*

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.

*No comments.*

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.

*No comments.*

**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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Destroyer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
</tbody>
</table>
| 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 Used Use Case | Destroyer Make Decision |
| Extending Use Case | Navigation Control |
| Other Requirements | NA |

Table 3-37 Use Case Description for Destroyer Navigation Control

Sequence Diagram

Refer to Figure 3-1 Sequence Diagram for Use Case Navigation Control.
### 3.3.5.3.2 Use Case: Destroyer Detect Enemy

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to locate the enemy using Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to detect the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Destroyer object</td>
</tr>
</tbody>
</table>

#### Flow of Events

**Base Path**

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

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>Extending Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
<td>Detect Enemy</td>
</tr>
</tbody>
</table>

#### Other Requirements

NA

| Table 3-38 Use Case Description for Destroyer Navigation Control |

**Sequence Diagram**

Refer to Figure 3-2 Sequence Diagram for Use Case Detect Enemy.
### 3.3.5.3.3 Use Case: Destroyer Communication with Allies

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the communication service between Destroyer and its allies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Communication/Detect must have this use case in order to pass information to the Destroyer’s allies</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Destroyer object</td>
</tr>
</tbody>
</table>

#### 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
NA

#### Extending Use Case
Communicate with Allies

#### Other Requirements
NA

| Table 3-39 Use Case Description for Destroyer Communication with Allies |

#### Sequence Diagram

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
### 3.3.5.3.4 Use Case: Destroyer Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, and decide where to conduct the ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Exist a Destroyer object; 2. The Destroyer’s status is updated; 3. All the reports are received</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path: 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</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>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;</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>1. Destroyer Update Status; 2. Destroyer Detect Enemy; 3. Destroyer Communication with Allies;</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Make Decision</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-40 Use Case Description for Destroyer Make Decision**

### Sequence Diagram

Refer to Figure 3-4 Sequence Diagram for Use Case Make Decision.
3.3.5.3.5 Use Case: Destroyer Weapon Control

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to attack the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Weapon</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>An attacking command is received</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>1. Decide the type and quantity of Weapon to be used;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Calculate and update the Weapon quantity on board</td>
</tr>
<tr>
<td></td>
<td>3. Issue an order to Weapon launcher</td>
</tr>
<tr>
<td></td>
<td>4. A Weapon object will be created and fired by Weapon launcher</td>
</tr>
<tr>
<td></td>
<td>5. Weapon launcher will aim and fire Weapon</td>
</tr>
<tr>
<td></td>
<td>6. When Weapons are used up, recharge the Weapon on board</td>
</tr>
</tbody>
</table>

**Alternate Path**

If the Weapon is Sea-Sea Missile, it will return a massage stating whether the target is destroyed or not

| Post-Condition | Weapon is fired and exploded. |

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>Destroyer Make Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>Weapon Control</td>
</tr>
</tbody>
</table>

**Other Requirements**

| NA |

**Table 3-41** Use Case Description for Destroyer Weapon Control

**Sequence Diagram**

Refer to Figure 3-5 Sequence Diagram for Use Case Weapon Control.
3.3.5.3.6 Use Case: Destroyer Update Status

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Destroyer’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Destroyer object</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Flow of Events</th>
<th>Base Path</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Update the location of the Destroyer</td>
<td></td>
</tr>
<tr>
<td>2. Determine if the Destroyer is hit by Weapon</td>
<td></td>
</tr>
<tr>
<td>3. Get the hit points of the Destroyer</td>
<td></td>
</tr>
<tr>
<td>4. Determine if the Destroyer can recover from the hit points</td>
<td></td>
</tr>
<tr>
<td>5. Determine if the Destroyer is destroyed</td>
<td></td>
</tr>
<tr>
<td>6. Determine if the Destroyer crashes with other object</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alternate Path</th>
<th>NA</th>
</tr>
</thead>
</table>

| Post-Condition | The status of the Destroyer is updated |

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>Update Status</td>
<td></td>
</tr>
</tbody>
</table>

| Other Requirements | NA |

Table 3-42 Use Case Description for Destroyer Update Status

Sequence Diagram

Refer to Figure 3-6 Sequence Diagram for Use Case Update Status.
### 3.3.5.3.7 Use Case: Destroyer Rearm and Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to rearm and refueling the Destroyer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.</td>
</tr>
</tbody>
</table>

**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 Destroyer get rearm and refueling

**Related Use Case**

NA

**Used Use Case**

Rearm and Refueling

**Extending Use Case**

NA

**Other Requirements**

NA

<table>
<thead>
<tr>
<th>Table 3-43 Use Case Description for Destroyer Rearm and Refueling</th>
</tr>
</thead>
</table>

**Sequence Diagram**

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

![Use Case Diagram for Cruiser](image)

**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.  
*No comments.*

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 be destroyed when crash with other object.  
*No comments.*

**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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Cruiser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Exist a Cruiser object; 2. A command is received from the navigation officer</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Cruiser is moved</td>
</tr>
<tr>
<td>Related Used Use Case</td>
<td>Cruiser Make Decision</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Navigation Control</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-44 Use Case Description for Cruiser Navigation Control

Sequence Diagram

Refer to Figure 3-1 Sequence Diagram for Use Case Navigation Control.
### 3.3.6.3.2 Use Case: Cruiser Detect Enemy

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to locate the enemy using Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to detect the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Cruiser object</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>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</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Any enemy in the range are detected</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Detect Enemy</td>
</tr>
</tbody>
</table>

| Other Requirements           | NA                                                  |

Table 3-45  Use Case Description for Cruiser Navigation Control

**Sequence Diagram**

Refer to Figure 3-2  Sequence Diagram for Use Case Detect Enemy.
### 3.3.6.3.3 Use Case: Cruiser Communication with Allies

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

Table 3-46 Use Case Description for Aircraft Carrier Communication with Allies

**Sequence Diagram**

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
### 3.3.6.3.4 Use Case: Cruiser Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, and decide where to conduct the ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Exist a Cruiser object; 2. The Cruiser’s status is updated; 3. All the reports are received</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>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</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Post-Condition**

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.

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1. Cruiser Update Status; 2. Cruiser Detect Enemy; 3. Cruiser Communication with Allies</td>
</tr>
</tbody>
</table>

**Extending Use Case**

| Make Decision |

**Other Requirements**

NA

Table 3-47  Use Case Description for Cruiser Make Decision

**Sequence Diagram**

Refer to Figure 3-4  Sequence Diagram for Use Case Make Decision.
### 3.3.6.3.5 Use Case: Cruiser Weapon Control

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to attack the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Weapon</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>An attacking command is received</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>1. Decide the type and quantity of Weapon to be used;</td>
</tr>
<tr>
<td></td>
<td>2. Calculate and update the Weapon quantity on board</td>
</tr>
<tr>
<td></td>
<td>3. Issue an order to Weapon launcher</td>
</tr>
<tr>
<td></td>
<td>4. A Weapon object will be created by Weapon launcher</td>
</tr>
<tr>
<td></td>
<td>5. Weapon launcher will aim and fire Weapon</td>
</tr>
<tr>
<td></td>
<td>6. Deduct the Weapon on board</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>If the Weapon is Sea-Sea Missile, it will return a massage stating whether the target is destroyed or not</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Weapon is fired and exploded.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Cruiser Make Decision</td>
</tr>
<tr>
<td>Used Use Case</td>
<td>Weapon Control</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td></td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-48 Use Case Description for Cruiser Weapon Control**

### Sequence Diagram

Refer to figure 3-5 Sequence Diagram for Weapon Control.
### 3.3.6.3.6 Use Case: Cruiser Update Status

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Cruiser’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Cruiser object</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td><strong>Base Path</strong></td>
<td></td>
</tr>
<tr>
<td>1. Update the location of the Cruiser</td>
<td></td>
</tr>
<tr>
<td>2. Determine if the Cruiser is hit by Weapon</td>
<td></td>
</tr>
<tr>
<td>3. Get the hit points of the Cruiser</td>
<td></td>
</tr>
<tr>
<td>4. Determine if the Cruiser can recover from the hit points</td>
<td></td>
</tr>
<tr>
<td>5. Determine if the Cruiser is destroyed</td>
<td></td>
</tr>
<tr>
<td>6. Determine if the Cruiser crashes with other object</td>
<td></td>
</tr>
<tr>
<td><strong>Alternate Path</strong></td>
<td></td>
</tr>
<tr>
<td>NA</td>
<td></td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The status of the Cruiser is updated</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Used Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Update Status</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-49 Use Case Description for Cruiser Update Status**

**Sequence Diagram**

Refer to Figure 3-6 Sequence Diagram for Use Case Update Status.
3.3.6.3.7 Use Case: Cruiser Rearm and Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to rearm and refueling the Cruiser</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.</td>
</tr>
</tbody>
</table>

Flow of Events

<table>
<thead>
<tr>
<th>Base Path</th>
<th>Flow of Events</th>
</tr>
</thead>
<tbody>
<tr>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

Table 3-50  Use Case Description for Cruiser Rearm and Refueling

Sequence Diagram

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

![Use Case Diagram for Battleship](image)

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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Battleship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
</tbody>
</table>
| 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 3-51 Use Case Description for Battleship Navigation Control**

**Sequence Diagram**

Refer to Figure 3-1 Sequence Diagram for Use Case Navigation Control.
### 3.3.7.3.2 Use Case: Battleship Detect Enemy

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to locate the enemy using Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to detect the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Battleship object</td>
</tr>
<tr>
<td><strong>Flow of Events</strong></td>
<td><strong>Base Path</strong></td>
</tr>
<tr>
<td>1. Initialize a Radar object with location and radius when Battleship is created;</td>
<td>2. Update Radar location;</td>
</tr>
<tr>
<td>3. Turn on/off Radar;</td>
<td>4. Get object information around the Battleship</td>
</tr>
<tr>
<td><strong>Alternate Path</strong></td>
<td>NA</td>
</tr>
<tr>
<td><strong>Post-Condition</strong></td>
<td>Any enemy in the range are detected</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Detect Enemy</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-52 Use Case Description for Battleship Navigation Control**

**Sequence Diagram**

Refer to Figure 3-2 Sequence Diagram for Use Case Detect Enemy.
### 3.3.7.3.3 Use Case: Battleship Communication with Allies

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

**Table 3-53 Use Case Description for Battleship Communication with Allies**

**Sequence Diagram**

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
3.3.7.3.4 Use Case: Battleship Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, and decide where to conduct the ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
</tbody>
</table>

### Pre-Conditions
1. Exist a Battleship object;
2. The Battleship’s status is updated;  
3. All the reports are received

### Flow of Events

| Base Path | 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. |
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
</tbody>
</table>
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 |
| Status | Detailed description and completed scenario |
| Actor | Weapon |
| Pre-Conditions | An attacking command is received |
| Flow of Events | Base Path |
| | 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 massage stating whether the target is destroyed or not |
| Post-Condition | Weapon is fired and exploded. |

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>Extending Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Battleship Make Decision</td>
<td>Weapon Control</td>
</tr>
</tbody>
</table>

| Other Requirements | NA |

Table 3-55 Use Case for Weapon Control

**Sequence Diagram**

Refer to Figure 3-5 Sequence Diagram for Use Case Weapon Control.
### 3.3.7.3.6 Use Case: Battleship Update Status

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Battleship’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Battleship object</td>
</tr>
<tr>
<td><strong>Flow of Events</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Base Path</strong></td>
<td></td>
</tr>
<tr>
<td>1. Update the location of the Battleship</td>
<td></td>
</tr>
<tr>
<td>2. Determine if the Battleship is hit by Weapon</td>
<td></td>
</tr>
<tr>
<td>3. Get the hit points of the Battleship</td>
<td></td>
</tr>
<tr>
<td>4. Determine if the Battleship can recover from the hit points</td>
<td></td>
</tr>
<tr>
<td>5. Determine if the Battleship is destroyed</td>
<td></td>
</tr>
<tr>
<td>6. Determine if the Battleship crashes with other object</td>
<td></td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td><strong>Post-Condition</strong></td>
<td>The status of the Battleship is updated</td>
</tr>
<tr>
<td>Related Use Case</td>
<td></td>
</tr>
<tr>
<td>Used Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Update Status</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-56  Use Case Description for Battleship Update Status

**Sequence Diagram**

Refer to Figure 3-6  Sequence Diagram for Use Case Update Status.
3.3.7.3.7 Use Case: Battleship Rearm and Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to rearm and refueling the Battleship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>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.</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Battleship get rearm and refueling</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>NA</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Rearm and Refueling</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-57 Use Case Description for Battleship Rearm and Refueling

Sequence Diagram

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

![Use Case Diagram for Submarine](image)

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 on the sea in 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.  
*No comments.*

**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 when its Weapons are used up.  
*No comments.*
3.3.8.3 Use Case Description

3.3.8.3.1 Use Case: Submarine Navigation Control

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to navigate the Submarine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to move on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>NA</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Exist a Submarine object;</td>
</tr>
<tr>
<td></td>
<td>2. A command is received from the navigation officer</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>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</td>
</tr>
<tr>
<td></td>
<td>Alternate Path: NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Submarine is moved</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case: Submarine Make Decision</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Extending Use Case: Navigation Control</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-58 Use Case Description for Submarine Navigation Control

Sequence Diagram

Refer to Figure 3-1 Sequence Diagram for Use Case Navigation Control.
### 3.3.8.3.2 Use Case: Submarine Detect Enemy

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to locate the enemy using Radar</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to detect the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Submarine object</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. Initialize a Radar object with location and radius when Submarine is created;</td>
</tr>
<tr>
<td></td>
<td>2. Update Radar location;</td>
</tr>
<tr>
<td></td>
<td>3. Turn on/off Radar;</td>
</tr>
<tr>
<td></td>
<td>4. Get object information around the Submarine</td>
</tr>
<tr>
<td></td>
<td>Alternate Path</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>Any enemy in the range are detected</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>Extending Use Case</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 3-59  Use Case Description for Submarine Detect Enemy

**Sequence Diagram**

Refer to Figure 3-2  Sequence Diagram for Use Case Detect Enemy.
### 3.3.8.3.3 Use Case: Submarine Communicate with Allies

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the communication service between Submarine and its allies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Communication/Detect must have this use case in order to pass information to the Submarine's allies</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Submarine object</td>
</tr>
</tbody>
</table>

**Flow of Events**

| Base Path   | 1. Initialize a Radio object with location and radius when Submarine is created;  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. Update Radio location;</td>
</tr>
<tr>
<td></td>
<td>3. Turn on/off Radio;</td>
</tr>
<tr>
<td></td>
<td>4. Get object information around the Submarine;</td>
</tr>
<tr>
<td></td>
<td>5. Send massage to its allies</td>
</tr>
</tbody>
</table>

**Alternate Path**

NA

**Post-Condition**

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 3-60 Use Case Description for Submarine Communicate with Allies**

**Sequence Diagram**

Refer to Figure 3-3 Sequence Diagram for Use Case Communicate with Allies.
### 3.3.8.3.4 Use Case: Submarine Make Decision

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to analyze the report, decide attack target, and decide where to conduct the ship</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to know its next action</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Communication/Detection</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. Exist a Submarine object;</td>
</tr>
<tr>
<td></td>
<td>2. The Submarine’s status is updated;</td>
</tr>
<tr>
<td></td>
<td>3. All the reports are received</td>
</tr>
<tr>
<td>Flow of Events</td>
<td>Base Path</td>
</tr>
<tr>
<td></td>
<td>1. Upon reception of reports, the captain analyze the threats and decide attack target;</td>
</tr>
<tr>
<td></td>
<td>2. The captain gives the order to navigation officer for where to conduct the ship and at what speed;</td>
</tr>
<tr>
<td></td>
<td>3. The captain gives order to Weapon officer to prepare the attack;</td>
</tr>
<tr>
<td></td>
<td>4. The captain gives order to communication officer to send out the message ;</td>
</tr>
<tr>
<td></td>
<td>5. The Captain decide to rearm or refueling to send request to SC.</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>1. The navigation officer executes captain’s command;</td>
</tr>
<tr>
<td></td>
<td>2. The Weapon office executes captain’s command;</td>
</tr>
<tr>
<td></td>
<td>3. The communication officer execute captain’s command;</td>
</tr>
<tr>
<td></td>
<td>4. The Base Supplier perform the transaction task.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Used Use Case</td>
</tr>
<tr>
<td></td>
<td>1. Submarine Update Status;</td>
</tr>
<tr>
<td></td>
<td>2. Submarine Detect Enemy;</td>
</tr>
<tr>
<td></td>
<td>3. Submarine Communication with Allies</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>Make Decision</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Table 3-61  Use Case Description for Submarine Make Decision**

**Sequence Diagram**

Refer to Figure 3-4  Sequence Diagram for Use Case Make Decision.
3.3.8.3.5 Use Case: Submarine Weapon Control

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to select Weapon to attack, update the quantity of Weapon on board, and recharge the Weapon as needed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to attack the enemy</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Weapon</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>An attacking command is received</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>If the Weapon is Sea-Sea Missile, it will return a massage stating whether the target is destroyed or not</td>
</tr>
</tbody>
</table>

**Post-Condition**

| Weapon is fired and exploded. |

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>Submarine Make Decision</th>
</tr>
</thead>
<tbody>
<tr>
<td>Extending Use Case</td>
<td>Weapon Control</td>
</tr>
</tbody>
</table>

**Other Requirements**

| NA |

Table 3-62  Use Case description for Submarine Weapon Control

**Sequence Diagram**

Refer to Figure 3-5  Sequence Diagram for Use Case Weapon Control.
3.3.8.3.6 Use Case: Submarine Update Status

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to update Submarine’s location and other status (alive/dead)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to report status to SC</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>Simulation Controller</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Exist a Submarine object</td>
</tr>
</tbody>
</table>

**Flow of Events**

<table>
<thead>
<tr>
<th>Base Path</th>
<th>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</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
</tbody>
</table>

**Post-Condition**
The status of the Submarine is updated

**Related Use Case**

<table>
<thead>
<tr>
<th>Used Use Case</th>
<th>NA</th>
</tr>
</thead>
</table>

**Extending Use Case**

<table>
<thead>
<tr>
<th>Use Case</th>
<th>Update Status</th>
</tr>
</thead>
</table>

**Other Requirements**

| Used | NA |

Table 3-63 Use Case Description for Submarine Update Status

**Sequence Diagram**

Refer to Figure 3-6 Sequence Diagram for Use Case Update Status.
3.3.8.3.7 Use Case: Submarine Rearm and Refueling

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service to rearm and refueling the Submarine</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Would like to have this use case in order to continue moving on the sea</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>1. The base supplier has enough fuel in stock; 2. The Radio is in ON state.</td>
</tr>
</tbody>
</table>

**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 Submarine

**Alternate Path**

NA

**Post-Condition**

The Submarine get rearm and refueling

<table>
<thead>
<tr>
<th>Related Use Case</th>
<th>Used Use Case</th>
<th>Extending Use Case</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

| Other Requirements | NA |

Table 3-64  Use Case Description for Submarine Rearm and Refueling

**Sequence Diagram**

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

The Weapons subsystem has the following four sub modules:

- Weapon (Carried Weapon)
- Controller
- Ruder
- Charger

Weapons can be classified as 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 NBS need to extend its functionality.

3.3.9.1 Use Case Diagram

![Use Case Diagram for Weapon](image)
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 controlled by ballistic when it shot.

*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

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the Weapon location to SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to aim the target.</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>
| Actor                | 1. All the Weapon;  
                        | 2. Simulation Controller. |

<table>
<thead>
<tr>
<th>Pre-Conditions</th>
<th>Weapon knows its location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow of Events Base</td>
<td>Path</td>
</tr>
</tbody>
</table>
|                      | 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     | NA                        |
| Used Use Case        | NA                        |
| Extending Use Case   | NA                        |
| Other Requirements   | NA                        |

Table 3-65 Use Case Description for Provide Location

Sequence Diagram

![Sequence Diagram](image)

Figure 3-35 Sequence Diagram for Use Case Weapon Provide Location
### 3.3.9.3.2 Use Case: Aim Target

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide a service to trace the target location</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to aim the target</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
</tbody>
</table>

**Actor**
1. All the Weapon (Heavy Cannon Shell, Sea-Sub Missile and Sub-Sea Missile will be 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

**Base Path**
1. The Weapon’s owner detects the target and launches the Weapon;
2. Weapon uses its detection system to trace the location of the nearest target;

**Alternate Path**
- NA

**Post-Condition**
The location of the target has been traced by the Weapon

**Related Use Case**
- Provide Location

**Extending Use Case**
- NA

**Other Requirements**
- NA

**Table 3-66 Use Case Description for Aim Target**

**Sequence Diagram**

![Sequence Diagram for Use Case Weapon Aim Target](image)

**Figure 3-36 Sequence Diagram for Use Case Weapon Aim Target**
### 3.3.9.3.3 Use Case: Fire and Hit Target

<table>
<thead>
<tr>
<th>Description</th>
<th>Provide the service for Weapon to be fired and hit the target.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Priority</td>
<td>Must have this use case in order to hit the target</td>
</tr>
<tr>
<td>Status</td>
<td>Detailed description and completed scenario</td>
</tr>
<tr>
<td>Actor</td>
<td>1. All the Weapon; 2. Target Object.</td>
</tr>
<tr>
<td>Pre-Conditions</td>
<td>Weapon is launched and prepared to fire.</td>
</tr>
<tr>
<td>Flow of Events</td>
<td></td>
</tr>
<tr>
<td>Base Path</td>
<td>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</td>
</tr>
<tr>
<td>Alternate Path</td>
<td>NA</td>
</tr>
<tr>
<td>Post-Condition</td>
<td>The Weapon is fired and target has been hit.</td>
</tr>
<tr>
<td>Related Use Case</td>
<td>Aim Target</td>
</tr>
<tr>
<td>Extending Use Case</td>
<td>NA</td>
</tr>
<tr>
<td>Other Requirements</td>
<td>NA</td>
</tr>
</tbody>
</table>

| Table 3-67       | Use Case Description for Fire and Hit Target                  |

**Sequence Diagram**

![Sequence Diagram for Use Case Weapon Fire And Hit Target](image-url)

**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 heavy 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 corporate 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 learn ability 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.

![Interaction diagram between subsystems of the Naval Battle Simulation System](Image)
The following figure describes the architecture of the system:

![Architecture of the Naval Battle Simulation System](image)

**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 of the Naval Battle Simulation System
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 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 Communion/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 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 4-4 Simulation Controller_for_Communication/Detection](image)
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.

- **goFristDetected()** 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. 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 ship want to know the detailed detected object info.

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.

- **goFristDetected()** 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.
Figure 4-5  Radar/Sonar_for_Weapon

- CRadar or CSonar()
- emitReceive()
- goFirstDetected()
- goNextDetected()
- getDetectedInfo()
- turnOn()
- turnOff()
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 ti indicate ID of a derived object of Ship, returns no value.
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 4-7 BaseWeapon_for_Simulation Controller](image)

### 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 ships 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)

![Diagram](image)

**Figure 4-8  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 function, when one function need to use another function of other 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 SC module 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 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 4-9 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>sc</td>
<td>SC*</td>
<td>Handle to Document Class</td>
</tr>
<tr>
<td>controller</td>
<td>Controller*</td>
<td>Handle to View Class</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Float</td>
<td>Coordinate of x axis of position</td>
</tr>
<tr>
<td>Y</td>
<td>Float</td>
<td>Coordinate of y axis of position</td>
</tr>
<tr>
<td>Type</td>
<td>char</td>
<td>Ship type</td>
</tr>
<tr>
<td>Flag</td>
<td>char</td>
<td>Side flag</td>
</tr>
<tr>
<td>DrawInfo</td>
<td>Struct</td>
<td>Structure of draw information about object</td>
</tr>
<tr>
<td>m_typebutton</td>
<td>Integer</td>
<td>Ship type button flag</td>
</tr>
<tr>
<td>vInfo[15][15]</td>
<td>VehicleInfo</td>
<td>Ship info 2-D array</td>
</tr>
</tbody>
</table>

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
   X=-1; y=-1; bmp=-1;
   Loop to initialize the ship info array
   vInfo[i][j] = NULL;
   End

Name: OnLButtonDown
Input: nFlags UINT, poitn CPoint
Output: none
Description: draw the ship object on the map
Pseudo-code:
   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 undoStatck;
   delete vInfo[r][c];
   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];
   set vInfo[r][c] = NULL;
   set drawInfo array to default value
   End
4.4.1.2.3 SC class

**Traceability to SRS**

**Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PARA</td>
<td>double</td>
<td>1.3</td>
<td></td>
</tr>
<tr>
<td>RADAR_RANGE</td>
<td>double</td>
<td>150.0</td>
<td>Radar range</td>
</tr>
<tr>
<td>SONAR_RANGE</td>
<td>double</td>
<td>100.0</td>
<td>Sonar range</td>
</tr>
<tr>
<td>RADIO_RANGE</td>
<td>double</td>
<td>1000.0</td>
<td>Radio range</td>
</tr>
<tr>
<td>WEAPON_RANGE</td>
<td>double</td>
<td>140.0</td>
<td>Weapon range for all Weapon</td>
</tr>
<tr>
<td>Time</td>
<td>double</td>
<td>0.07</td>
<td>Time slice for each ship or Aircraft</td>
</tr>
</tbody>
</table>

**Private data members**

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

**Public functions**

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

```
Begin:
    Set anim 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,
Use V1 and v0 and speed to get the vector of next position randomly
End

---

**Name:** iterator_findNearest

**Input:** *vptr vector<ShipClass*>, *vptr 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]
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
{erase the element of vector}
End
**Name:** getTimeSlice  
**Input:** none  
**Output:** double  
**Description:**
**Pseudo-code:**
```plaintext
Begin:
  return time
End
```

**Name:** incrLastID  
**Input:** none  
**Output:** none  
**Description:**
**Pseudo-code:**
```plaintext
Begin:
  lastID++;
End
```

**Name:** ~SC  
**Input:** none  
**Output:** none  
**Description:** destructor  
**Pseudo-code:**
```plaintext
Begin:
  Call freeupVehicleList();
End
```

### 4.4.1.2.4 Controller Class

**Traceability to SRS**  
SC-009, SC-010, SC-013, SC-014

**Constants**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PI</td>
<td>GLfloat</td>
<td>3.1415926f</td>
<td></td>
</tr>
</tbody>
</table>

**Private data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>counterActive</td>
<td>integer</td>
<td>Counter of system</td>
</tr>
<tr>
<td>start1</td>
<td>Clock_t</td>
<td>Start system time</td>
</tr>
</tbody>
</table>

**Protected data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>percent</td>
<td>integer</td>
<td>Test variable</td>
</tr>
</tbody>
</table>
Public data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fhor</td>
<td>float</td>
<td>Time variable</td>
</tr>
<tr>
<td>fVer</td>
<td>float</td>
<td>Velocity variable</td>
</tr>
<tr>
<td>zoom</td>
<td>float</td>
<td>Image zoom variable</td>
</tr>
<tr>
<td>TextureImage</td>
<td>Struct</td>
<td>Image structure</td>
</tr>
<tr>
<td>textures[18]</td>
<td>TextureImage</td>
<td>TextureImage array</td>
</tr>
<tr>
<td>offset</td>
<td>GLfloat</td>
<td>Offset of image</td>
</tr>
</tbody>
</table>

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: handle paints of graphical ships when window size is changing
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>position</td>
<td>vector</td>
<td>vector of Vehicle position</td>
</tr>
<tr>
<td>type</td>
<td>integer</td>
<td>Type of Vehicle</td>
</tr>
<tr>
<td>flag</td>
<td>char</td>
<td>Flag of Vehicle</td>
</tr>
</tbody>
</table>

#### 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pDoc</td>
<td>Cdocument*</td>
<td>Handle for document object</td>
</tr>
</tbody>
</table>

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 false;
  End
End
```

Name: virtual ~VehicleFactory
Input: none
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>bship</td>
<td>BaseShipStructure</td>
<td>New created ship object</td>
</tr>
<tr>
<td>Fuelamount</td>
<td>double</td>
<td>Total fuel amount of base supplier</td>
</tr>
<tr>
<td>Weapon</td>
<td>struct</td>
<td>Total Weapon amount and Weapon type structure</td>
</tr>
</tbody>
</table>

**Public functions**

**Name:** BaseSupplier

**Input:** none

**Output:** none

**Description:** default constructor

**Pseudo-code:**

\[
\text{Begin:} \\
\quad \text{Fuelamount}=0; \\
\quad \text{Weapon.type}=-1; \\
\quad \text{Weapon.amount}=0; \\
\quad \text{Ship.type}=-1; \\
\quad \text{Ship.amount}=0; \\
\quad \text{End}
\]

**Name:** BaseSupplier

**Input:** double fue, Weapon wep, BaseShipStructure ship

**Output:** none

**Description:** constructor

**Pseudo-code:**

\[
\text{Begin:} \\
\quad \text{Fuelamount}=\text{fue}; \\
\quad \text{Weapon.type}=\text{wep.amount}; \\
\quad \text{Weapon.amount}=\text{wep.type}; \\
\quad \text{Bship.type}=\text{ship.type}; \\
\quad \text{Bship.amount}=\text{ship.amount} \\
\quad \text{End}
\]
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 weapon amount and type;
  Create Weapon;
  Deduct the Weapon amount of the type;
  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 for setting type and amount.
  Deduct the ship amount of the type;
  Return 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 the following figure.

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.

![Class Diagram for Communication/Detection Module](image)

Figure 4-10 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
<td>integer</td>
<td>1 is Radar, 2 is Sonar</td>
</tr>
<tr>
<td>ID</td>
<td>integer</td>
<td>Radar/Sonar object ID</td>
</tr>
<tr>
<td>ddb</td>
<td>CDetectedDatabase</td>
<td>All the detected information class</td>
</tr>
<tr>
<td>state</td>
<td>integer</td>
<td>Radar/Sonar on/off state (“on” for object creation)</td>
</tr>
<tr>
<td>range</td>
<td>double</td>
<td>Radar/Sonar radius of detection</td>
</tr>
</tbody>
</table>

Public functions

<table>
<thead>
<tr>
<th>Name: CommunicationBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: integer ty</td>
</tr>
<tr>
<td>Output: none</td>
</tr>
<tr>
<td>Description: default constructor</td>
</tr>
<tr>
<td>Pseudo-code:</td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>Initialize the member data;</td>
</tr>
<tr>
<td>ID=0;</td>
</tr>
<tr>
<td>Range=0;</td>
</tr>
<tr>
<td>type=ty;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: CommunicationBase</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input: nId integer, nRange integer, ty integer</td>
</tr>
<tr>
<td>Output: none</td>
</tr>
<tr>
<td>Description: constructor</td>
</tr>
<tr>
<td>Pseudo-code:</td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>Id= nid;</td>
</tr>
<tr>
<td>//initialize id</td>
</tr>
<tr>
<td>Range=nRange;</td>
</tr>
<tr>
<td>//initialize Range</td>
</tr>
<tr>
<td>Type=ty;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>
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=o;  // 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  object;  // 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: none
Description:
Pseudo-code:
Begin:
    get detected object by calling getDetectedFromList() in DetectedDatabase
End

Name: getFirstDetected
Input: none
Output: none
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 setNextDetected() in DetectedDatabase;
   
   End

Name: **turnOff**
Input: ty integer, id integer
Output: integer
Description:
Pseudo-code:
   
   Begin:
   assgin 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:
   assgin 1 to State for object.ID=id for Radar 1, for Sonar 2.
   return State;
   
   End

Name: **~ communicationBase**
Input: none
Output: none
Description: virtual distructor
Pseudo-code:
   
   Begin:
   
   End

### 4.4.2.2.2 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>Detected object ID</td>
</tr>
<tr>
<td>flag</td>
<td>integer</td>
<td>Detected object flag</td>
</tr>
<tr>
<td>type</td>
<td>integer</td>
<td>Detected object type</td>
</tr>
<tr>
<td>powerswitch</td>
<td>integer</td>
<td>Detected object power switch</td>
</tr>
<tr>
<td>pos</td>
<td>vector</td>
<td>Detected object position</td>
</tr>
<tr>
<td>velocity</td>
<td>vector</td>
<td>Detected object velocity</td>
</tr>
</tbody>
</table>

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: destructor
Pseudo-code:
```
Begin:
End
```

### 4.4.2.2.3 DetectedDatabase Class

**Traceability to SRS**
CD-004, CD-008

**Constants**
NA.

**Private data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>DetList</td>
<td>typedef vector&lt;CDetected*&gt; DetList;</td>
</tr>
<tr>
<td>itCurrDetected</td>
<td>DetList::iterator</td>
<td>Iterator to the vector of DetList</td>
</tr>
</tbody>
</table>

**Public functions**

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

Name: `~CdetectedDatabase`
Input: none
Output: none
Description: destructor
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>myRadioId</td>
<td>integer</td>
<td>Radio object ID</td>
</tr>
<tr>
<td>range</td>
<td>float</td>
<td>Range of Radio radius</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Msg</td>
<td>Message</td>
<td>struct define Message include senderID, receiverID, senderType, command, sender Position, destination position and enemyInfo of CDetected type.</td>
</tr>
<tr>
<td>pVehicle</td>
<td>BaseShip*</td>
<td>Pointer variable of BaseShip type to indicate the ship information.</td>
</tr>
</tbody>
</table>

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  
```
<table>
<thead>
<tr>
<th>Name: GetReceiverId</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Message inMsg</td>
</tr>
<tr>
<td><strong>Output:</strong> Integer</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Pseudo-code:</strong></td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>return Msg.receiverID;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: SetCommand</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> int pCommand</td>
</tr>
<tr>
<td><strong>Output:</strong> none</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Pseudo-code:</strong></td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>Msg.command = pCommand;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: GetCommandId</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> none</td>
</tr>
<tr>
<td><strong>Output:</strong> Integer</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Pseudo-code:</strong></td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>return Msg.command;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: SetSenderPosition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Vector Pos</td>
</tr>
<tr>
<td><strong>Output:</strong> Integer</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Pseudo-code:</strong></td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>Msg.sPos[1] = Pos[1];</td>
</tr>
<tr>
<td>Msg.sPos[2] = Pos[2];</td>
</tr>
<tr>
<td>Msg.sPos[3] = Pos[3];</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name: GetSenderPosition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> none</td>
</tr>
<tr>
<td><strong>Output:</strong> Vector</td>
</tr>
<tr>
<td><strong>Description:</strong></td>
</tr>
<tr>
<td><strong>Pseudo-code:</strong></td>
</tr>
<tr>
<td>Begin:</td>
</tr>
<tr>
<td>return Msg.sPos;</td>
</tr>
<tr>
<td>End</td>
</tr>
</tbody>
</table>
**Name:** SetDestinationPosition  
**Input:** Vector Pos  
**Output:** none  
**Description:**  
**Pseudo-code:**  
Begin:  
  Msg.dPos = Pos;  
End

**Name:** GetDestinationPosition  
**Input:** none  
**Output:** Vector  
**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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td>MsgList</td>
<td>Typedef std::vector&lt;Message*&gt; MsgList</td>
</tr>
</tbody>
</table>
**Private functions**

Name: MessageDatabase
Input: none
Output: none
Description: constructor
Pseudo-code:

```
Begin:
    //Initialize the member data
    Message *msgData= new Message();
    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 (check if receiver is alive and within range;
    for broadcast message, define a list of message with receiverId equal to
    the ID of those objects alive and within range;)
```
if this Msg is valid, keep this message to the list; For broadcast message, keep that list of 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 the following figure.

4.4.3.1 Module Detailed Design

The class operation and attribute are not list 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 $\Rightarrow$. 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 5.5
Figure 4-11  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)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_RESISTANCE</td>
<td>integer</td>
<td>Depends on ship</td>
<td>resistance value when ship and Aircraft first created</td>
</tr>
<tr>
<td>RECOVERABLE_RESISTANCE</td>
<td>integer</td>
<td>Depends on ship</td>
<td>minimum resistance that the can make reparation</td>
</tr>
<tr>
<td>MAX_REPAIR_TIME</td>
<td>integer</td>
<td>Depends on ship</td>
<td>Maximum time the ship and Aircraft needs to restore the resistance</td>
</tr>
</tbody>
</table>

Protected data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>Ship and Aircraft ID</td>
</tr>
<tr>
<td>Check</td>
<td>int</td>
<td>used to indicate if the ship object is selected or not</td>
</tr>
</tbody>
</table>

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 description. Radar/Sonar represents the Radar class for all the applicable ships and Sonar class foe all the applicable ship in different class implementation.

Traceability to SRS
SC-001, SS-002

Constants(Redefined in Different Derived Ship Class if applicable)

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_RESISTANCE</td>
<td>integer</td>
<td>resistance value when ship and Aircraft first created</td>
</tr>
<tr>
<td>RECOVERABLE_RESISTANCE</td>
<td>integer</td>
<td>minimum resistance that the can make reparation</td>
</tr>
<tr>
<td>MAX_REPAIR_TIME</td>
<td>integer</td>
<td>Maximum time the ship and Aircraft needs to restore the resistance</td>
</tr>
</tbody>
</table>

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ID</td>
<td>integer</td>
<td>Ship and Aircraft ID</td>
</tr>
<tr>
<td>active</td>
<td>bool</td>
<td>used to distinguish between alive and dead</td>
</tr>
<tr>
<td>flag</td>
<td>char</td>
<td>used to distinguish between allies and enemies</td>
</tr>
<tr>
<td>type</td>
<td>integer</td>
<td>used to distinguish among different ships and Aircraft</td>
</tr>
<tr>
<td>fuelamount</td>
<td>integer</td>
<td>fuel amount at ship creation</td>
</tr>
<tr>
<td>fuellimit</td>
<td>integer</td>
<td>Fuel limit when need to send request</td>
</tr>
<tr>
<td>Weaponamount</td>
<td>integer</td>
<td>Amount of on board Weapon when ship is created</td>
</tr>
<tr>
<td>captain</td>
<td>Captain</td>
<td>an object of the class Captain</td>
</tr>
<tr>
<td>n_officer</td>
<td>NavigationOfficer</td>
<td>An instance of class NavigationOfficer</td>
</tr>
<tr>
<td>Radar_officer</td>
<td>DetectionOffice</td>
<td>An instance of class DetectionOffice</td>
</tr>
<tr>
<td>Radio_officer</td>
<td>RadioOfficer</td>
<td>An instance of class RadioOfficer</td>
</tr>
<tr>
<td>w_officer</td>
<td>WeaponOfficer</td>
<td>An instance of class WeaponOfficer</td>
</tr>
<tr>
<td>w_launcher</td>
<td>WeaponLauncher</td>
<td>An instance of class WeaponLauncher</td>
</tr>
<tr>
<td>S_Radar</td>
<td>Radar</td>
<td>An instance of class Radar</td>
</tr>
<tr>
<td>s_Radio</td>
<td>Radio</td>
<td>An instance of class Radio</td>
</tr>
<tr>
<td>time_counter</td>
<td>long</td>
<td>records the simulation time</td>
</tr>
<tr>
<td>resistance</td>
<td>integer</td>
<td>The value stands for the status of the ship and Aircraft, i.e. how serious the ship is damaged</td>
</tr>
</tbody>
</table>
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(Weapon amount of base supplier >= Weapon amount request) and Weapon type == ship’s Weapon type)
  { Basesupplier->createWeapon();
    Return true;
  }
  else return false;
End

4.4.3.3.3 Captain Class

Traceability to SRS

Constants
NA

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>friend_list</td>
<td>ObjectList</td>
<td>the node of the ObjectList will contain information of id, position, flag, speed of object. This list contains friends info.</td>
</tr>
<tr>
<td>enemy_list</td>
<td>ObjectList</td>
<td>list after update</td>
</tr>
<tr>
<td>previous_enemy_list</td>
<td>ObjectList</td>
<td>list before update</td>
</tr>
<tr>
<td>crash</td>
<td>bool</td>
<td>if true, the ship or Aircraft collides with another object</td>
</tr>
<tr>
<td>attack_target</td>
<td>Detected</td>
<td>Target object the ship and Aircraft will attack</td>
</tr>
</tbody>
</table>
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  
**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 enemy_list to previous_enemy_list;
  remove all elements in the enemy_list;

  1. Information from Radar/Sonar
     get number of objects detected by calling function
     RadarOfficer/SonarOffice->getNumOfDetected(Radar, Vector currPos ).
     check the first detected object:
     RadarOfficer/SonarOffice[] getFirstDetected(Radar, currPos);
     if Detected[] getFlag() is the same as the flag of the ship or Aircraft,
      store in friend_list by calling addToFriendList(Detected);
     if the flag is different, store in enemy_list: addToEnemyList(Detected);
     loop until all object detected have been checked
```
RadarOfficer/SonarOffice  getNextDetected(Radar)
check returned object Detected,
if Detected  getFlag is the same as the flag of the ship or Aircraft
store in friend_list, call: addToFriendList(Detected)
else store in enemy_list: addToEnemyList(Detected);
}

2. Information from Radio
while (return value of receiveMessage () in the RadioOfficer is not
    NULL, which means there is at least one message)
{
    Cmessage  getDetectedInfo() which return Detected object
    check if it is friend, if yes, store in friend-list,
    else store in enemy-list, similarly step as info from Radar
}

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

<table>
<thead>
<tr>
<th>Name</th>
<th>isCrash</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>none</td>
</tr>
<tr>
<td>Output</td>
<td>bool</td>
</tr>
<tr>
<td>Description:</td>
<td>if true, the ship and Aircraft collides with other object</td>
</tr>
<tr>
<td>Pseudo-code:</td>
<td></td>
</tr>
</tbody>
</table>
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, for example, there are only under water enemies (Submarines) or air enemies (Aircrafts), return false for Battleship
            --check all elements in the enemy_list from the first one to the last one
            --get position (Vector) of the each object
            --get z value of the position
            --check if the z value is equal to 0, that means the object is sea-borne object for Battleship eg.
            --if z values of all objects are not equal to 0, no object can be attacked for Battleship eg., return false
    case 3: there is at least one enemy for this object, for example, sea-borne enemy for Battleship
            Following the same procedure as case 2 to find the number of sea-borne enemy for Battleship eg.
            /*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.
                    }
                }
                can not attack the target,
                return false;
            } 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
}

---

**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 $q$ with current direction to get away from enemy.
    NavigationOfficer steer($q$);
    double acc$= 525$; // 525 km/hr$^2$ for Battleship
    NavigationOfficer adjustSpeed(acc$= MAX\_SPEED)$;
}
if (no friend on the heading direction and |speed|<Max)
{
    find a direction which has angle $q$ with current direction where there is no friends and object on the way;
    NavigationOfficer steer($q$);
}
if (friends or object on the way)
{
    find a direction which has angle $q$ with current direction where there is no friends and object on the way;
    NavigationOfficer steer($q$);
    double deceleration$= -700$; // -700 km/hr$^2$ 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$= -700$; // -700 km/hr$^2$ 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>det</td>
<td>Detected</td>
<td>a Detected object, store object information</td>
</tr>
<tr>
<td>Radar_on/Sonar_on</td>
<td>bool</td>
<td>Radar/Sonar is on if true</td>
</tr>
</tbody>
</table>
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/Sonar: Radar&/Sonar&
Output: void
Description: turn off Radar/Sonar
Pseudo-code:
   Begin:
       Radar/Sonar [] turnoff();
   End

Name: turnOnRadar/turnOnSonar
Input: Radar/Sonar: 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
### 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
```

### getNextDetected
**Input:** Radar/Sonar: Radar&/Sonar&,
**Output:** Detected
**Description:** get the next detected object information
**Pseudo-code:**
```
Begin:
    Radar/Sonar & goNextDetected();
    return Radar/Sonar & 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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>message</td>
<td>CMessage</td>
<td>an instance of CMessage, store message info</td>
</tr>
</tbody>
</table>

#### 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: CRadio&
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:
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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAX_SPEED</td>
<td>integer</td>
<td>Maximum speed of the Battleship</td>
</tr>
</tbody>
</table>

**Private data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Curr_position</td>
<td>Vector</td>
<td>Current position</td>
</tr>
<tr>
<td>Temp_position</td>
<td>Vector</td>
<td>Temporarily position before update</td>
</tr>
<tr>
<td>velocity</td>
<td>Vector</td>
<td>including direction</td>
</tr>
</tbody>
</table>

**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:** dt: integer, decPos: Vector

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

**Pseudo-code:**

```plaintext
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 Dt.
    Vector velocity_ori = velocity-aDt;
    //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 Dt and update position of ship or aircraft.
    curr_position = curr_position + VelocityDt;
End
```

---

**Name:** `getPosition`
**Input:** none
**Output:** Vector
**Description:** get current position

**Pseudo-code:**

```plaintext
Begin
    return curr_position;
End
```

---

**Name:** `getVelocity`
**Input:** none
**Output:** Vector
**Description:** get current velocity

**Pseudo-code:**

```plaintext
Begin
    return velocity;
End
```

---

**Name:** `setPosition`
**Input:** pos: Vector
**Output:** void
**Description:**

**Pseudo-code:**

```plaintext
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 + accl[t];
    //decelerate to a velocity smaller than original
    else if (accl<0 & (targetVelocity<Velocity)&(targetVelocity>=0))
        {temp_Velocity = velocity + accl[t];
         if (temp_Velocity<0) velocity =0;
         else velocity = velocity + accl[t];
    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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CANNON_QTY</td>
<td>integer</td>
<td>The quantity of cannon (Battleship eg.)</td>
</tr>
<tr>
<td>MISSILE_QTY</td>
<td>integer</td>
<td>The quantity of sea-sea Missile (Battleship eg.)</td>
</tr>
</tbody>
</table>

Private data members

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

Public member functions

Name: WeaponOfficer
Input: none
Output: none
Description: Constructor initializes attributes
Pseudo-code:
```plaintext
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:
```plaintext
Begin:
End
```

Name: prepareAttack
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 cannon 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:
```
taget_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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GRAVITY_ACCELERATION</td>
<td>double</td>
<td>Physic constant (Battleship)</td>
</tr>
</tbody>
</table>

**Private data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cannon_attribute</td>
<td>WAttribute</td>
<td>contain the attributes of cannon (Battleship)</td>
</tr>
<tr>
<td>Missile_attribute</td>
<td>WAttribute</td>
<td>contain the attributes of Missile (Battleship)</td>
</tr>
<tr>
<td>cannon_list</td>
<td>List</td>
<td>keep the created cannon shell until it is detonated (Battleship)</td>
</tr>
<tr>
<td>Missile_list</td>
<td>List</td>
<td>keep the created Missile until it is detonated (Battleship)</td>
</tr>
</tbody>
</table>

**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) \( V\cos a \cdot t = \frac{(g\cdot t^2)}{2} \)
  \( V\cos b \cdot t = (Y_m - Y_e) - V_x \cdot t \)
  \( V\cos g \cdot t = (X_m - X_e) - V_y \cdot t \)
  \( (\cos a)^2 + (\cos b)^2 + (\cos g)^2 = 1 \)
  Note: \( V \) is the magnitude of the cannon shell speed.
  \( a \), \( b \), \( g \) 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:
  \( V^2 \cdot t^2 = \left(\frac{(g\cdot t^2)}{2}\right)^2 + ((Y_m - Y_e) - V_x \cdot t)^2 + ((X_m - X_e) - V_y \cdot t)^2 \)
  In order to make the above equation has a definite solution, we have to make some assumption to simplify it. We observe that the sum of the last two items in the equation is the distance from the position of my ship to the final position where the cannon shell falls. Therefore we make the following assumptions so that we can get a solution from the equation:
  One, we suppose \( V \) is constant with its value to be the maximum speed.
  Two, we divide the attack range of the cannon into different areas. For each area we make the sum of the last two items is outer boundary value of the area. So it is a constant value.
  Through this way, we can get a fixed time the cannon shells fly in each of the areas. Then we can get the fire angle, as well as the fire speed of the cannon shells in \( x \), \( y \), \( z \) directions for any intended fire destination within cannon fire range, using different fixed times for different fire areas. These fire speeds in \( x \), \( y \), \( z \) directions are what we should provide to the Weapon subsystem. However, the Weapon subsystem asks for the intended destination of cannon shell. We can also provide this
destination, but we think it is more reasonable to provide initial velocity of cannon shells.
Return the destination Vector of cannon shells;

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 the following figure.

4.4.4.1 Module Detailed Design

The class operation and attribute are not list 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.

![Class Diagram for Weapon Module](image-url)
4.4.4.2 Class Definition

4.4.4.2.1 CWeapon

Traceability to SRS

WP-001

Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHARGE_RANGE</td>
<td>float</td>
<td>Depends on Weapon</td>
<td>Take from structure wAttr.wMaxSpeed/15</td>
</tr>
</tbody>
</table>

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wFlag</td>
<td>integer</td>
<td>friend and enemy</td>
</tr>
<tr>
<td>time_len</td>
<td>double</td>
<td>record time length for each loop</td>
</tr>
<tr>
<td>wPosContr</td>
<td>CWPositionController</td>
<td></td>
</tr>
<tr>
<td>wAimContr</td>
<td>CWAutoAimController</td>
<td></td>
</tr>
<tr>
<td>wChgContr</td>
<td>CWChargeController</td>
<td></td>
</tr>
<tr>
<td>wStaContr</td>
<td>CWActiveStateController</td>
<td></td>
</tr>
</tbody>
</table>

Protected data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wAttr</td>
<td>struct WAttribute</td>
<td>Weapon Attribute Structure</td>
</tr>
</tbody>
</table>

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 AutoAimming 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  
    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: ~CWeapon
Input: none
Output: none
Description: destructor
Pseudo-code:
Begin:
End

4.4.4.2.2 WCommon Class

Traceability to SRS
WP-002, WP-003

Constants

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>DOUBLE_MAX</td>
<td>double</td>
<td>(1.0e+60)</td>
<td>Maximum double</td>
</tr>
<tr>
<td>INVALID_VEC</td>
<td>Vector</td>
<td>(Vector(-DOUBLE_MAX, -DOUBLE_MAX, -DOUBLE_MAX))</td>
<td>Invalid Vector for speed</td>
</tr>
<tr>
<td>W_RADAR_RANG</td>
<td>integer</td>
<td>50</td>
<td>50000 meters</td>
</tr>
<tr>
<td>MAX_TARGET_DIST</td>
<td>double</td>
<td>DOUBLE_MAX</td>
<td>Maximum target distance</td>
</tr>
<tr>
<td>WeaponTypeStart</td>
<td>integer</td>
<td>7</td>
<td>the begin type of Weapon</td>
</tr>
<tr>
<td>WRadar_Type</td>
<td>integer</td>
<td>0</td>
<td>//aiming device no.</td>
</tr>
<tr>
<td>Ballistic</td>
<td>integer</td>
<td>2</td>
<td>//aiming device no.</td>
</tr>
<tr>
<td>DOUBLE_PREC</td>
<td>double</td>
<td>0.00001</td>
<td>Precise of double</td>
</tr>
<tr>
<td>AircraftCarrier_Type</td>
<td>integer</td>
<td>1</td>
<td>Ship type</td>
</tr>
<tr>
<td>Aircraft_Type</td>
<td>integer</td>
<td>2</td>
<td>Ship type</td>
</tr>
<tr>
<td>Destroyer_Type</td>
<td>integer</td>
<td>3</td>
<td>Ship type</td>
</tr>
<tr>
<td>Cruiser_Type</td>
<td>integer</td>
<td>4</td>
<td>Ship type</td>
</tr>
<tr>
<td>Name</td>
<td>Type</td>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>---------------------------</td>
<td>---------------</td>
<td>----------------------------------</td>
<td></td>
</tr>
<tr>
<td>Battleship Type</td>
<td>integer 5</td>
<td>Ship type</td>
<td></td>
</tr>
<tr>
<td>Submarine Type</td>
<td>integer 6</td>
<td>Ship type</td>
<td></td>
</tr>
<tr>
<td>HeavyCannonShell</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>AirAir Missile</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>AirSea Missile</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>SeaSea Missile</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>SeaAir Missile</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>SeaSub Missile</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>Torpedo</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
<tr>
<td>SubSea Torpedo</td>
<td>integer</td>
<td>Weapon Type</td>
<td></td>
</tr>
</tbody>
</table>

**Private data members**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>struct WAttribute</td>
<td>struct</td>
<td>Weapon Attribute</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doublePI</td>
<td>const double</td>
<td>3.1415926;</td>
<td>radius of Circle</td>
</tr>
</tbody>
</table>

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wTargetPosition</td>
<td>Position</td>
<td>Target position</td>
</tr>
<tr>
<td>wType</td>
<td>TYPE_WEAPON</td>
<td>Weapon type</td>
</tr>
<tr>
<td>MyFlag</td>
<td>int</td>
<td>Friend and enemy</td>
</tr>
<tr>
<td>StaContrPt;</td>
<td>CWActiveStateController*</td>
<td>Weapon state controller</td>
</tr>
</tbody>
</table>
**Public functions**

**Name:** CWAutoAimController  
**Input:** none  
**Output:** none  
**Description:** default constructor  
**Pseudo-code:**
```plaintext
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:**
```plaintext
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 Aimming Weapons: Rudder, Radar/Sonar system  
**Pseudo-code:**
```plaintext
Begin:
  wType(id),
  myFlag(flag),
  staContrPt(state),
  rudderPt(rud),
  RSDetect(RSpt),
  wTargetPosition(INVALID_VEC)
End
```

**Name:** *init*  
**Input:** TYPE_WEAPON id, int flag,  
  CWActiveStateController *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 Aimming 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 postion
  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:
• Call Radar/Sonar EmitReceive() to check how many objects is in the Radar/Sonar range. If it returns zero, then it is finished and return 0.
• For each object, it gets target using Radar/Sonar getFirstDetect() for first time. It gets target using Radar/Sonar getNextDetect() if it isn't the first time.

And it calls isTargetType() to check object type If type is invalid, then go to second step for next object.
• And then it call Radar/Sonar getPosition() to get position of object.

And then it counts the distance between object position and the target position.
• And 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.

From above steps, it gets the nearest object and sets wTargetPosition to the position of the nearest object, and return 1. If there are not valid object in the valid range because of object type, then it doesn't change wTargetPosition,
End
```
**Name: setInitTargetPosition**

**Input:** Position targetPos

**Output:** none

**Description:** called in fire() function

**Pseudo-code:**
```
Begin:
    if( ( wType == SeaSea Missile ) || ( 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firepower</td>
<td>integer</td>
<td>Fire power of Weapon</td>
</tr>
<tr>
<td>precision</td>
<td>double</td>
<td>Weapon precision</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hitDet</td>
<td>ChitDetect</td>
<td>Hit detected object</td>
</tr>
<tr>
<td>Ch</td>
<td>CWCharge*</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Asc</td>
<td>CWActiveStateController *</td>
<td>Weapon state controller object</td>
</tr>
<tr>
<td>MyFlag</td>
<td>integer</td>
<td>Enemy or friend</td>
</tr>
<tr>
<td>FirePower</td>
<td>integer</td>
<td>Fire power of Weapon</td>
</tr>
<tr>
<td>HitRange</td>
<td>double</td>
<td>Hit range of Weapon</td>
</tr>
<tr>
<td>WeaponType</td>
<td>integer</td>
<td>Type of Weapon</td>
</tr>
<tr>
<td>pObject</td>
<td>baseClass*</td>
<td>Target object</td>
</tr>
</tbody>
</table>

Private functions

Name: detonate
Input: BaseClass *pO
Output: none
Description:
Pseudo-code:

```
Begin:
    Detonate the Weapon;
End
```

Public functions

Name: CWChargeController
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;
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
    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
CWAActiveStateController.setState(INACTIVE)

return 1;
End
if state is INACTIVE Return 1;
else return 0;
End

Name: ~CWChargeController
Input: none
Output: none
Description: destructor
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WCurrentPositon</td>
<td>Position</td>
<td>current position</td>
</tr>
<tr>
<td>wDestinationPosition</td>
<td>Position</td>
<td>Destination position</td>
</tr>
<tr>
<td>wNextPosition</td>
<td>Position</td>
<td>next time slice position</td>
</tr>
<tr>
<td>wRoute</td>
<td>double</td>
<td>Route for this Weapon</td>
</tr>
<tr>
<td>WVelocity</td>
<td>Velocity</td>
<td>Cannon Shell and carrier Weapons don't have Radar and Sonar system.</td>
</tr>
<tr>
<td>wType</td>
<td>TYPE_WEAPON</td>
<td>Type of Weapon</td>
</tr>
<tr>
<td>CWrudder *rudderPt</td>
<td>CWAActiveStateController*</td>
<td>None of Cannon shell</td>
</tr>
<tr>
<td>StaContrPt</td>
<td>CWAActiveStateController*</td>
<td>Weapon active state control object</td>
</tr>
</tbody>
</table>

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 Aiming 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 Aiming Weapons
   //Parameters TYPE_WEAPON id, CWActiveStateController *state,
   CWRudder *rud
```plaintext
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  
**Input:** 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**

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wActive</td>
<td>bool</td>
<td>Weapon state (Alive or dead)</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>doublePI</td>
<td>const double</td>
<td>3.1415926;</td>
<td>radius of Circle</td>
</tr>
</tbody>
</table>

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>wpSpeed</td>
<td>wSpeed</td>
<td>Weapon speed (velocity) wSpeed is structure of speed</td>
</tr>
<tr>
<td>currentPos</td>
<td>Vector</td>
<td>Weapon current position</td>
</tr>
<tr>
<td>TargetPos</td>
<td>Vector</td>
<td>target position</td>
</tr>
<tr>
<td>currentRad</td>
<td>double</td>
<td>current steering angle</td>
</tr>
<tr>
<td>NewRad</td>
<td>double</td>
<td>new steering angle</td>
</tr>
<tr>
<td>Steering</td>
<td>bool</td>
<td>steering on/off</td>
</tr>
<tr>
<td>maxSpeed</td>
<td>integer</td>
<td>maxium Weapon speed (speed)</td>
</tr>
</tbody>
</table>

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 launched 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

Constants
N/A

Private data members

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudder</td>
<td>CWRudder</td>
<td>Weapon Rudder object</td>
</tr>
<tr>
<td>Charge</td>
<td>CWCharge</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Radar</td>
<td>CRadar</td>
<td>Radar object</td>
</tr>
</tbody>
</table>

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, Preceision;
  initialize Radar;
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";
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudder</td>
<td>CWRudder</td>
<td>Weapon Rudder object</td>
</tr>
<tr>
<td>Charge</td>
<td>CWCharge</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Radar</td>
<td>CRadar</td>
<td>Radar object</td>
</tr>
</tbody>
</table>

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

Name: operator delete
Input: void * mem
Output: none
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";
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudder</td>
<td>CWRudder</td>
<td>Weapon Rudder object</td>
</tr>
<tr>
<td>Charge</td>
<td>CWCharge</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Radar</td>
<td>CRadar</td>
<td>Radar object</td>
</tr>
</tbody>
</table>
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: 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();
  return SC::vpVehicles[sz-1];
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";
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudder</td>
<td>CWRudder</td>
<td>Weapon Rudder object</td>
</tr>
<tr>
<td>Charge</td>
<td>CWCharge</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Radar</td>
<td>CRadar</td>
<td>Radar object</td>
</tr>
</tbody>
</table>

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: 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();
    return SC::vpVehicles[sz-1];
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";
    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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CarriedTorpedo</td>
<td>Wtorpedo*</td>
<td>Carriage Missile object</td>
</tr>
</tbody>
</table>
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: 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();
    return SC::vpVehicles[sz-1];
  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";
  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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>carriedMissile</td>
<td>WMissileSeaSea*</td>
<td>Carriage Missile object</td>
</tr>
</tbody>
</table>

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: operator new  
Input: size_t  
Output: none  
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();
return SC::vpVehicles[sz-1];

End

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";
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>charge</td>
<td>CWCharge*</td>
<td>Weapon Charge Object</td>
</tr>
</tbody>
</table>

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:** 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();
  return SC::vpVehicles[sz-1];
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";
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

<table>
<thead>
<tr>
<th>Name</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Rudder</td>
<td>CNRudder</td>
<td>Weapon Rudder object</td>
</tr>
<tr>
<td>Charge</td>
<td>CWCharge</td>
<td>Weapon charge object</td>
</tr>
<tr>
<td>Sonar</td>
<td>CSonar</td>
<td>Sonar object</td>
</tr>
</tbody>
</table>

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: operator new
Input: size_t
Output: none
Pseudo-code:

```cpp
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();
return SC::vpVehicles[sz-1];
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";
End

Name: ~Wtorpedo
Input: none
Output: none
Description: destructor
Pseudo-code:

Begin:
End
5. System Testing

We use white-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**

<table>
<thead>
<tr>
<th>Unit Static testing</th>
<th>Identifying coding errors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Objective</td>
<td>Code inspection</td>
</tr>
<tr>
<td></td>
<td>All inspection questions must be checked</td>
</tr>
<tr>
<td>Technique</td>
<td>Every line of code has been inspected</td>
</tr>
<tr>
<td></td>
<td>And each kind of error in the check list has been checked and corrected</td>
</tr>
<tr>
<td>Completion criteria</td>
<td>None</td>
</tr>
</tbody>
</table>

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 |

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
Bitmap:1 X:300 Y:300 | No overlap | SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02 |
| TC_SC-002 | Bitmap:1 X:500 Y:500
Bitmap:2 X:500 Y:500 | No overlap | SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02 |
| TC_SC-003 | Bitmap:2 X:400 Y:400
Bitmap:2 X:400 Y:400 | No overlap | SC-001, SC-002, SC-007, SC-008, SC-008-01, SC-008-02 |
| 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 X:730 Y:600 | Out of region | SC-001, SC-002, SC-007, SC-008, SC-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 |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Table Case #</strong></td>
</tr>
<tr>
<td>TC_SC-013</td>
</tr>
<tr>
<td>TC_SC-014</td>
</tr>
<tr>
<td>TC_SC-015</td>
</tr>
<tr>
<td>TC_SC-016</td>
</tr>
<tr>
<td>TC_SC-017</td>
</tr>
<tr>
<td>TC_SC-018</td>
</tr>
</tbody>
</table>

Table 5-4  Unit Test Case for SetUpDlg Undo function

5.1.1.1.2 Error Reports

a) Window is flashing when undo. We changed the called OnPaint() function by draw() function.
b) The image is drawn overlap for test case 2. We construct a 15*15 matrix and trace each image sizing 40 by 40 pixels,
c) 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 |
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case #</strong></td>
</tr>
<tr>
<td>TC_SC-019</td>
</tr>
<tr>
<td>TC_SC-020</td>
</tr>
<tr>
<td>TC_SC-021</td>
</tr>
<tr>
<td>TC_SC-022</td>
</tr>
</tbody>
</table>

Table 5-5  Unit Test Case for Controller LoadTGA function
### Function Name: calDir

**Objective:** To test the calculation of direction vector

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-023</td>
<td>Vector(10.0, 10.0, 0), Vector(10.0, 10.0, 0)</td>
<td>9.0f (as flag)</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-024</td>
<td>Vector(10.0, 10.0, 0), Vector(10.0, 20.0, 0)</td>
<td>PI/2.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-025</td>
<td>Vector(10.0, 10.0, 0), Vector(10.0, 0.0, 0)</td>
<td>3.0*PI/2.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-026</td>
<td>Vector(10.0, 10.0, 0), Vector(0.0, 10.0, 0)</td>
<td>PI</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-027</td>
<td>Vector(10.0, 10.0, 0), Vector(20.0, 10.0, 0)</td>
<td>0.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-028</td>
<td>Vector(0.0, 0.0, 0), Vector(10.0, 10.0, 0)</td>
<td>PI/4.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-029</td>
<td>Vector(0.0, 0.0, 0), Vector(-10.0, 10.0, 0)</td>
<td>3*PI/4.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-030</td>
<td>Vector(0.0, 0.0, 0), Vector(-10.0, -10.0, 0)</td>
<td>5*PI/4.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
<tr>
<td>TC_SC-031</td>
<td>Vector(0.0, 0.0, 0), Vector(10.0, -10.0, 0)</td>
<td>7*PI/4.0f</td>
<td>SC-013, SC-013-01, SC-013-02, SC-013-03, SC-014</td>
</tr>
</tbody>
</table>

Table 5-6  Unit Test Case for Controller calDir function

### Function Name: OnKeyDown

**Objective:** To test the mouse key down function

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-032</td>
<td>Press key &quot;F12&quot;</td>
<td>Images get bigger</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-033</td>
<td>Press key &quot;F11&quot;</td>
<td>Images get smaller</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-034</td>
<td>Press key &quot;←&quot;</td>
<td>Images move left</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-035</td>
<td>Press key &quot;→&quot;</td>
<td>Images move right</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-036</td>
<td>Press key &quot;↑&quot;</td>
<td>Images move up</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-037</td>
<td>Press key &quot;↓&quot;</td>
<td>Images move down</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-038</td>
<td>Press key &quot;space&quot; and &quot;a&quot; and &quot;1&quot;</td>
<td>Image position no change</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-039</td>
<td>Vector(0.0, 0.0, 0), Vector(-10.0, -10.0, 0)</td>
<td>5*PI/4.0f</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
<tr>
<td>TC_SC-040</td>
<td>Vector(0.0, 0.0, 0), Vector(10.0, -10.0, 0)</td>
<td>7*PI/4.0f</td>
<td>SC-001, SC-006, SC-007, SC-008, SC-008-01, SC-008-02</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Class Controller</th>
<th>Class SetUpDlg</th>
<th>Class SC</th>
</tr>
</thead>
<tbody>
<tr>
<td>drawVehicles();</td>
<td>OnLButtonDown();</td>
<td>OnClickDestroyerB();</td>
</tr>
<tr>
<td>SetupPixelFormat();</td>
<td>OnClickAircaftcarrierB();</td>
<td>OnClickDestroyerR();</td>
</tr>
<tr>
<td>OnRButtonUp();</td>
<td>OnClickAircaftcarrierR();</td>
<td>OnClickSubmarineB();</td>
</tr>
<tr>
<td>pauseSimulation();</td>
<td>OnClickBattleshipB();</td>
<td>OnClickSubmarineR();</td>
</tr>
<tr>
<td>resumeSimulation();</td>
<td>OnClickBattleshipR();</td>
<td>OnPaint();</td>
</tr>
<tr>
<td>startSimulation();</td>
<td>OnClickCruiserB();</td>
<td>OnClearall();</td>
</tr>
<tr>
<td>endSimulation();</td>
<td>OnClickCruiserR();</td>
<td>OnStartSetup();</td>
</tr>
</tbody>
</table>

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 CDetected Class Functions

5.1.2.1.1 Unit Test Cases and Results

<table>
<thead>
<tr>
<th>Test Case</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-001</td>
<td>Pointer to AC state=0 or state = 1</td>
<td>AircraftCarrier (ID, type, flag, Powerswitch=0, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-002</td>
<td>Pointer to Aircraft, state=0</td>
<td>Aircraft (ID, type, flag, powerswitch=0, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-003</td>
<td>Pointer to Aircraft, state=1</td>
<td>Aircraft (ID, type, flag, Powerswitch=1, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-004</td>
<td>Pointer to Destroyer, state=0</td>
<td>Destroyer (ID, type, flag, Powerswitch=0, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-005</td>
<td>Pointer to Destroyer, state=1</td>
<td>Destroyer (ID, type, flag, Powerswitch=1, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-006</td>
<td>Pointer to Cruiser, state=0</td>
<td>Cruiser (ID, type, flag, powerswitch=0, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-007</td>
<td>Pointer to Cruiser, state=1</td>
<td>Cruiser (ID, type, flag, Powerswitch=1, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-008</td>
<td>Pointer to Battleship, state=0</td>
<td>Battleship (ID, type, flag, powerswitch=0, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
<tr>
<td>TC_CD-009</td>
<td>Pointer to Battleship, state=1</td>
<td>Battleship (ID, type, flag, Powerswitch=1, pos, velocity )</td>
<td>CD-004, CD-004-01, CD-004-02</td>
</tr>
</tbody>
</table>
Table 5-9 Unit Test Case for CDetected setDetData function

| TC_CD-010 | Pointer to Submarine, state=0 | Cruiser (ID, type, flag, powerswitch=0, pos, velocity) | CD-008, CD-008-01, CD-008-02 |
| TC_CD-011 | Pointer to Submarine, state=1 | Submarine (ID, type, flag, powerswitch=1, pos, velocity) | CD-008, CD-008-01, CD-008-02 |
| TC_CD-012 | Pointer 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 | Pointer 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 5-10 Unit Test Case for CDetected operator << overloading function

Function Name: ostream& operator << overloading
Objective: To overloading operator << to output CDetected object

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-014</td>
<td>CDetected Object</td>
<td>Output object's ID, flag, type, pos (x,y,z) and velocity (x,y,z)</td>
<td>CD-004, CD-008</td>
</tr>
</tbody>
</table>

Table 5-11 Unit Test Case for CDetectedDatabase DeleteAll function

Function Name: DeleteAll
Objective: To delete old detected object inside the Radar's (Sonar's) database

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-015</td>
<td>Delare Radar (Sonar) object and call emitReceive two times</td>
<td>The number of detected object calling the second time is same as that calling in the first time</td>
<td>CD-004, CD-008</td>
</tr>
</tbody>
</table>

Table 5-12 Unit Test Case for CDetectedDatabase addDeleted function

Function Name: addDetected
Objective: To add the object that is within the Radar (Sonar) range to Radar's (Sonar's) database list.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-016</td>
<td>CDetected object</td>
<td>There is an error message to indicate the input should be constant type</td>
<td>CD-004, CD-008</td>
</tr>
<tr>
<td>TC_CD-017</td>
<td>The pointer of CDetected object</td>
<td>Insert the pointer of CDetected object to database</td>
<td>CD-004, CD-008</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-018</td>
<td>State = 0</td>
<td>Output message to user: &quot;Radar is turned off, no object can be detected&quot;</td>
<td>CD-004</td>
</tr>
<tr>
<td>TC_CD-019</td>
<td>State = 1 and Range = 0</td>
<td>Output error message to user: &quot;Radar’s range can’t be less or equal to zero&quot;</td>
<td>CD-004</td>
</tr>
<tr>
<td>TC_CD-020</td>
<td>State = 1 and Range = -1</td>
<td>Output error message to user: &quot;Radar’s range can’t be less or equal to zero&quot;</td>
<td>CD-004</td>
</tr>
<tr>
<td>TC_CD-021</td>
<td>State = 1 and Range = 10</td>
<td>Output the number of detected object and a list of pointer to detected objects.</td>
<td>CD-004</td>
</tr>
<tr>
<td>TC_CD-022</td>
<td>State = 1 and Range = 10</td>
<td>the number of detected object is zero. the list of pointer to detected objects.</td>
<td>CD-004</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-023</td>
<td>State = 0</td>
<td>Output message to user: &quot;Sonar is turned off, no object can be detected&quot;</td>
<td>CD-008</td>
</tr>
<tr>
<td>TC_CD-024</td>
<td>State = 1 and Range = 0</td>
<td>Output error message to user: &quot;Sonar’s range can’t be less or equal to zero&quot;</td>
<td>CD-008</td>
</tr>
<tr>
<td>TC_CD-025</td>
<td>State = 1 and Range = -1</td>
<td>Output error message to user: &quot;Sonar’s range can’t be less or equal to zero&quot;</td>
<td>CD-008</td>
</tr>
</tbody>
</table>
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 or not.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-028</td>
<td>Pointer to the vehicle’s base class set as parameter to the Cmessage object</td>
<td>True is returned</td>
<td>CD-011</td>
</tr>
<tr>
<td>TC_CD-029</td>
<td>Pointer to the vehicle’s base class NOT set as parameter to the Cmessage object</td>
<td>False is returned</td>
<td>CD-011</td>
</tr>
</tbody>
</table>

Table 5-15 Unit Test Case for CMessage validToSend function

Function Name: updateSenderInfo
Objective: To update the sender’s Id, type and position

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-030</td>
<td>Pointer to the vehicle’s base class set as parameter to the Cmessage object</td>
<td>Sender’s Id is updated. Sender’s Type is updated. Sender’s Position is updated.</td>
<td>CD-011</td>
</tr>
<tr>
<td>TC_CD-031</td>
<td>Pointer to the vehicle’s base class NOT set as parameter to the Cmessage object</td>
<td>Function is not called</td>
<td>CD-011</td>
</tr>
</tbody>
</table>

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: **DeleteMyMessages**  
**Objective:** To delete all the messages from the database of messages corresponding to the Radio calling this function.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-032</td>
<td>Receiver’s Id is the Radio’s Id and is passed as parameter</td>
<td>All messages belonging to the Radio’s id are deleted from the message database.</td>
<td>CD-012</td>
</tr>
<tr>
<td>TC_CD-033</td>
<td>Receiver’s Id IS NOT the Radio’s Id and is passed as parameter</td>
<td>No messages are deleted.</td>
<td>CD-012</td>
</tr>
</tbody>
</table>

Table 5-17 Unit Test Case for CMessage validToSend function

Function Name: **DeleteAllMsg**  
**Objective:** To delete all the messages from the database of messages

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-034</td>
<td>Function Call</td>
<td>All messages from the message database are deleted.</td>
<td>CD-012</td>
</tr>
</tbody>
</table>

Table 5-18 Unit Test Case for CMessage DeleteAllMsg function

Function Name: **AddOneMsgIntheList**  
**Objective:** To add one message in the message database

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-035</td>
<td>CMessage object sent for broadcast.</td>
<td>The message is added in the database for all the receivers.</td>
<td>CD-012</td>
</tr>
<tr>
<td>TC_CD-036</td>
<td>CMessage object sent for a specific receiver.</td>
<td>The message is added in the database for the specific receiver.</td>
<td>CD-012</td>
</tr>
</tbody>
</table>

Table 5-19 Unit Test Case for CMessage AddOneMsgIntheList function

Function Name: **GetMyMsg**  
**Objective:** To retrieve one message from the message database.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-037</td>
<td>Radio id = receiver id and is different from sender id.</td>
<td>A message object is returned.</td>
<td>CD-012</td>
</tr>
<tr>
<td>TC_CD-038</td>
<td>Radio’s id != receiver’s id but is still different from sender’s id.</td>
<td>A default message object with data set to default values (0) is returned</td>
<td>CD-012</td>
</tr>
<tr>
<td>TC_CD-039</td>
<td>Radio’s id != receiver’s id and is not different from sender’s id.</td>
<td>A default message object with data set to default values (0) is returned</td>
<td>CD-012</td>
</tr>
<tr>
<td>TC_CD-040</td>
<td>Radio’s id = id and is not different from the sender’s id.</td>
<td>A default message object with data set to default values (0) is returned</td>
<td>CD-012</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function Name: DeleteMessages</th>
<th>Objective: To delete all the messages from the database of messages corresponding to the Radio calling this function.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case #</td>
<td>Test Data</td>
</tr>
<tr>
<td>TC_CD-041</td>
<td>Function call</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Name: SendMessage</th>
<th>Objective: To add one message in the message database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case #</td>
<td>Test Data</td>
</tr>
<tr>
<td>TC_CD-042</td>
<td>CMessage object is passed as parameter.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Function Name: ReceiveMessage</th>
<th>Objective: To retrieve one message from the message database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Case #</td>
<td>Test Data</td>
</tr>
<tr>
<td>TC_CD-043</td>
<td>Function call</td>
</tr>
</tbody>
</table>

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, Navigation Officer, Radio Officer, Weapon Officer, Weapon Launcher 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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-001</td>
<td>Battleship() ;</td>
<td>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) B_RadarOfficer created. 5) B_RadioOfficer created: type=5. 6) Radio created: id=myID. 7) BWeaponOfficer created. 8) BWeaponLauncher created.</td>
<td>SC-001, SC-002, BS-001</td>
</tr>
<tr>
<td>TC_BS-002</td>
<td>Battleship('R', Vector(2,2,0), Vector(5,5,0))</td>
<td>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_counter=0. 3) Radar created: id=myID, radius=75. 4) B_RadarOfficer created. 5) B_RadioOfficer created: type=5. 6) Radio created: id=myID. 7) BWeaponOfficer created. 8) BWeaponLauncher created.</td>
<td>SC-001, SC-002, BS-001</td>
</tr>
</tbody>
</table>

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)

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-003</td>
<td>Resistance= 190. T = 5600.</td>
<td>Resistance=190. IsActive = ture.</td>
<td>BS-024 to BS-031 BS-032 to BS-034</td>
</tr>
<tr>
<td>TC_BS-004</td>
<td>Resistance =201 T = 5399</td>
<td>Resistance=201. IsActive = ture.</td>
<td>BS-024 to BS-031 BS-032 to BS-034</td>
</tr>
<tr>
<td>TC_BS-005</td>
<td>Resistance = 201 T = 5401</td>
<td>Resistance=300 IsActive = ture.</td>
<td>BS-024 to BS-031 BS-032 to BS-034</td>
</tr>
</tbody>
</table>

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:** ifAttack

**Objective:** analyzing the situation, and making decision whether and which enemy should be attacked.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-006</td>
<td>enemy_list = NULL or dist = 120000</td>
<td>return false</td>
<td>BS-015</td>
</tr>
<tr>
<td>TC_BS-007</td>
<td>sea_enemy_count = 0</td>
<td>return false</td>
<td>BS-015</td>
</tr>
<tr>
<td>TC_BS-008</td>
<td>sea_enemy_count = 2, dist = 90000, wtype = 0, cQty = 50 OR sea_enemy_count = 2, dist = 90000, wtype = 1, mQty = 10</td>
<td>return attack = true</td>
<td>BS-015</td>
</tr>
<tr>
<td>TC_BS-009</td>
<td>sea_enemy_count = 2, dist = 90000, mQty = 0, cQty = 0</td>
<td>Return attack = false</td>
<td>BS-015</td>
</tr>
</tbody>
</table>

Table 5-26 Unit Test Case for Derived Captain ifAttack function

**Function Name:** isOntheway, adjustNavigation

**Objective:** calculating the distance between this Battleship and object detected.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-010</td>
<td>pos = vector (50, 30, 0) my_pos = vector(40, 20, 0)</td>
<td>return dist = 80.62</td>
<td>BS-001 to BS-003</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-011</td>
<td>nofficer3. adjustSpeed(40,80,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-139,0) velocity=Vector(0,-70,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-012</td>
<td>nofficer3. adjustSpeed(30,80,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-125,0) velocity=Vector(0,-63,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-013</td>
<td>nofficer3. adjustSpeed(40,60,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-119,0) velocity=Vector(0,-60,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-014</td>
<td>nofficer3. adjustSpeed(40,80,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-85,0) velocity=Vector(0,-43,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-015</td>
<td>nofficer3. adjustSpeed(-2,-1,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,1,0) velocity=Vector(0,0,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-016</td>
<td>nofficer3. adjustSpeed(-2,-1,1)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,0,0) velocity=Vector(0,-1,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-017</td>
<td>nofficer3. adjustSpeed(-2,-1,1)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-1,0) velocity=Vector(0,-2,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-018</td>
<td>nofficer3. adjustSpeed(-1.2,1,2)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-1,0) velocity=Vector(0,-1,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-019</td>
<td>nofficer3. adjustSpeed(-1.2,9,2)</td>
<td>error message temp_position=curr_position=Vector(3,1,0) velocity=Vector(0,-3,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-020</td>
<td>nofficer3. adjustSpeed(2,1,2)</td>
<td>error message temp_position=curr_position=Vector(3,1,0) velocity=Vector(0,-3,0)</td>
<td>BS-001 to BS-003</td>
</tr>
</tbody>
</table>

Table 5-28 Unit Test Case for NavigationOfficer adjustSpeed function
**Function Name:** other functions  
**Objective:** Test the others function (in bold font)

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-021</td>
<td>BNavigation nofficer1(); BNavigationOfficer() function</td>
<td>curr_position=Vector(0,0,0) temp_position=Vector(0,0,0) velocity=Vector(0,0,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-022</td>
<td>BNavigationOfficer nofficer2( Vector(2,2,0), Vector(5,5,0), Vector(1,3,0)) BNavigationOfficer( Vector curPos, Vector desPos, Vector spd)</td>
<td>curr_position=Vector(2,2,0) temp_position=Vector(5,5,0) velocity=Vector(1,3,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-023</td>
<td>BNavigationOfficer nofficer2( Vector(2,2,0), Vector(5,5,0), Vector(70,70,0)) BNavigationOfficer( Vector curPos, Vector desPos, Vector spd)</td>
<td>curr_position=Vector(2,2,0) temp_position=Vector(5,5,0) velocity=Vector(49.4975,49.4975,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-024</td>
<td>BNavigationOfficer nofficer3(Vector(2,2,0), Vector(5,5,0)) BNavigationOfficer( Vector curPos, Vector desPos);</td>
<td>curr_position=Vector(2,2,0) temp_position=Vector(5,5,0) velocity=Vector(49.4975,49.4975,0)</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-025</td>
<td>None ~BNavigationOfficer();</td>
<td>main() runs without error.</td>
<td>BS-001 to BS-003</td>
</tr>
<tr>
<td>TC_BS-026</td>
<td>nofficer3.getPosition(); getPosition()</td>
<td>Vector(2,2,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-027</td>
<td>nofficer3.getVelocity(); getVelocity()</td>
<td>Vector(7.07107,7.07107,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-028</td>
<td>nofficer3.setPosition(Vector(3,1,0)) setPosition(Vector pos)</td>
<td>curr_position=Vector(3,1,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-029</td>
<td>nofficer3.setVelocity(Vector(60,80,0)) setVelocity(Vector spd)</td>
<td>Velocity=Vector(42.56,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-030</td>
<td>nofficer3.setVelocity(Vector(4,3,0)) setVelocity(Vector spd)</td>
<td>Velocity=Vector(4,3,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-031</td>
<td>nofficer3.cruise(Vector(3,1,0),1 cruise(Vector targetPos, double t) cruise(Vector targetPos, double t)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,1,0) velocity=Vector(0,0,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-032</td>
<td>nofficer3.cruise(Vector(-3,0),1 cruise(Vector targetPos, double t)</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-2,0) velocity=Vector(0,-3,0)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-033</td>
<td>nofficer3.steer(0.1) steer(a)</td>
<td>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)</td>
<td>BS-013 to BS-018</td>
</tr>
<tr>
<td>TC_BS-034</td>
<td>nofficer3.setVelocity (Vector (0, -3,0)) nofficer3.adjustSpeed(30,80,2) nofficer3.updatePosition() ; updatePosition()</td>
<td>curr_position=Vector(3,1,0) temp_position=Vector(3,-125,0) velocity=vector(-0.63)</td>
<td>BS-013 to BS-018</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-035</td>
<td>In every time i, cp=Vector(i,0), tp=Vector(26,26,0), ts=Vector(20,20,0), tid=1, ct=10+5*i</td>
<td>After 5 times call, the object of WeaponLauncher be called and the cannon shell sent, 8 times, sent again</td>
<td>BS-019 to BS-023</td>
</tr>
<tr>
<td>TC_BS-036</td>
<td>In every time i, cp=Vector(i,0), tp=Vector(27,27,0), ts=Vector(20,20,0), tid=1, ct=10+10*i</td>
<td>After 4 times call, the object of WeaponLauncher be called and a Missile sent, 7 time calls, sent again.</td>
<td>BS-019 to BS-023</td>
</tr>
<tr>
<td>TC_BS-037</td>
<td>In every time i, cp=Vector(80,80,0), tp=Vector(85,85,0), ts=Vector(20,20,0), tid=1, ct=10+10*i</td>
<td>After 4 times call, the object of WeaponLauncher be called and a Missile sent, 7 time calls, sent again.</td>
<td>BS-019 to BS-023</td>
</tr>
<tr>
<td>TC_BS-038</td>
<td>In every time i, cp=Vector(85,85,0), tp=Vector(85,85,0), ts=Vector(20,20,0), tid=1, ct=10+10*i</td>
<td>The object of WeaponLauncher is not called, so neither cannon shells nor missiles is launched.</td>
<td>BS-019 to BS-023</td>
</tr>
</tbody>
</table>
In every time i, 
\[ cp=\text{Vector}(i, i, 0), \]
\[ tp=\text{Vector}(26+i, 26+i, 0) \]
\[ ts=\text{Vector}(20, 20, 0), \]
\[ tid=1, ct=10+10^i \]
The object of WeaponLauncher is not called, so neither cannon shells nor missiles is launched.

Table 5-30  Unit Test Case for WeaponOfficer prepareAttack function

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-041</td>
<td>cp=Vector(0, 0, 0), tp=Vector(26, 26, 0) cannon_qty=4.</td>
<td>return value is 1.</td>
<td>BS-019</td>
</tr>
<tr>
<td>TC_BS-042</td>
<td>cp=Vector(0, 0, 0), tp=Vector(26, 26, 0) cannon_qty=2.</td>
<td>return value is -1.</td>
<td>BS-019</td>
</tr>
<tr>
<td>TC_BS-043</td>
<td>cp=Vector(0, 0, 0), tp=Vector(27, 27, 0) Missile_qty=1.</td>
<td>return value is 0.</td>
<td>BS-019</td>
</tr>
<tr>
<td>TC_BS-044</td>
<td>cp=Vector(0, 0, 0), tp=Vector(27, 27, 0) Missile_qty=0.</td>
<td>return value is -1.</td>
<td>BS-019</td>
</tr>
<tr>
<td>TC_BS-045</td>
<td>cp=Vector(0, 0, 0), tp=Vector(80, 80, 0) Missile_qty=1.</td>
<td>return value is 0.</td>
<td>BS-019</td>
</tr>
<tr>
<td>TC_BS-046</td>
<td>cp=Vector(0, 0, 0), tp=Vector(85, 85, 0) Missile_qty=1.</td>
<td>return value is 0</td>
<td>BS-019</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Function Name: aimByBallistic</th>
<th>Objective: For each case, some significant or critical data are inputted and the output results are to be compared with the expected results</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case #</strong></td>
<td><strong>Test Data</strong></td>
</tr>
<tr>
<td>TC_BS-047</td>
<td>cp=Vector(0, 0, 0), tp=Vector(12, 13, 0), ts=Vector(10, 20,0),</td>
</tr>
<tr>
<td>TC_BS-048</td>
<td>cp=Vector(0, 0, 0), tp=Vector(18, 20, 0), ts=Vector(20,15,0),</td>
</tr>
<tr>
<td>TC_BS-049</td>
<td>cp=Vector(0, 0, 0), tp=Vector(25, 28, 0), ts=Vector(15, 20,0),</td>
</tr>
</tbody>
</table>

Table 5-32 Unit Test Case for WeaponLauncher aimByBallistic function

<table>
<thead>
<tr>
<th>Function Name: fireCannonShell</th>
<th>Objective: The function is used to create cannon shells, insert them in the cannon shell list and fire them when they are required</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case #</strong></td>
<td><strong>Test Data</strong></td>
</tr>
<tr>
<td>TC_BS-050</td>
<td>cp=Vector(0, 0, 0), tp=Vector(20, 20, 0), b-flag=R.</td>
</tr>
</tbody>
</table>

Table 5-33 Unit Test Case for WeaponLauncher fireCannonShell function

<table>
<thead>
<tr>
<th>Function Name: fireMissile</th>
<th>Objective: This function is used to create Missiles, insert them in the Missile list and fire them when they are required</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case #</strong></td>
<td><strong>Test Data</strong></td>
</tr>
<tr>
<td>TC_BS-051</td>
<td>cp=Vector(0, 0, 0), tp=Vector(50, 50, 0), b-flag=R.</td>
</tr>
</tbody>
</table>

Table 5-34 Unit Test Case for WeaponLauncher fireMissile function

<table>
<thead>
<tr>
<th>Function Name: deleteWeapon</th>
<th>Objective: This function is used to delete cannon shells or Missiles when they have been detonated.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Test Case #</strong></td>
<td><strong>Test Data</strong></td>
</tr>
<tr>
<td>TC_BS-052</td>
<td>Create a Missile list with Missiles some marked active, some inactive</td>
</tr>
</tbody>
</table>
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).

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-001</td>
<td>Create an active instance</td>
<td><code>getState</code> return true</td>
<td>WP-005</td>
</tr>
<tr>
<td>TC_WP-002</td>
<td>SetState to inactive</td>
<td><code>.setState()</code> returns false.</td>
<td>WP-005</td>
</tr>
<tr>
<td>TC_WP-003</td>
<td>Create an inactive instance</td>
<td><code>getState</code> return false</td>
<td>WP-005</td>
</tr>
<tr>
<td>TC_WP-004</td>
<td>SetState to active</td>
<td><code>getState</code> return true</td>
<td>WP-005</td>
</tr>
</tbody>
</table>

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:** `initialposition`

**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 rudder will be created to update the velocity.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-005</td>
<td>(100,0,0) Active</td>
<td>(50,0,0) (0.014,0,0)</td>
<td>WP-001</td>
</tr>
<tr>
<td>TC_WP-006</td>
<td>(100,0,100) Active</td>
<td>(35.36,0,35.36) (0.01,0,0.01)</td>
<td>WP-001</td>
</tr>
<tr>
<td>TC_WP-007</td>
<td>(100,0,-100) Inactive</td>
<td>N/A N/A</td>
<td>WP-001</td>
</tr>
</tbody>
</table>

Table 5-37 Unit Test Case for CWActiveStateController initialposition function
Submarine Weapons

**Function Name:** initialposition

**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 rudder will be created to update the velocity.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>destination</td>
<td>state</td>
<td>velocity</td>
</tr>
<tr>
<td>TC_WP-008</td>
<td>(100,0,0)</td>
<td>Active</td>
<td>(50,0,0)</td>
</tr>
<tr>
<td>TC_WP-009</td>
<td>(100,0,100)</td>
<td>Inactive</td>
<td>N/A</td>
</tr>
<tr>
<td>TC_WP-010</td>
<td>(100,0,-100)</td>
<td>Active</td>
<td>(35.36,0,35.36)</td>
</tr>
</tbody>
</table>

Table 5-38  Unit Test Case for CWActiveStateController initialposition 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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Initial Position</td>
<td>Detected Object 1</td>
<td>Detected Object 2</td>
</tr>
<tr>
<td>TC_WP-011</td>
<td>(100,0,0)</td>
<td>Ship, (100,0,0)</td>
<td>Ship, (100,10,0)</td>
</tr>
<tr>
<td>TC_WP-012</td>
<td>(100,0,0)</td>
<td>Ship, (300,0,0)</td>
<td>Aircraft, (100,0,50)</td>
</tr>
<tr>
<td>TC_WP-013</td>
<td>(50,0,0)</td>
<td>Aircraft, (50,0,100)</td>
<td>Submarine (50,0,-50)</td>
</tr>
</tbody>
</table>

Table 5-39  Unit Test Case for CWAutoAimController tracetarget function
### Target: Aircraft

**Function Name:**

**Objective:** self-guided Weapons to trace target touse Radar/Sonar to detect all objects in Radar/Sonar’s range, select the valid nearest target to the previous target position

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Initial Position</th>
<th>Detected Object 1</th>
<th>Detected Object 2</th>
<th>Detected Object 3</th>
<th>Detected Object 4</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-014</td>
<td>(100,0,100)</td>
<td>Ship, (100,0,0)</td>
<td>Aircraft, (100,0,100)</td>
<td>Submarine, (100,0,-50)</td>
<td>Aircraft, (100,10,80)</td>
<td>Aircraft, (100,0,100) Return success</td>
<td>WP-002, WP-004</td>
</tr>
<tr>
<td>TC_WP-015</td>
<td>(100,0,50)</td>
<td>Ship, (100,0,0)</td>
<td>Aircraft, (200,0,100)</td>
<td>Submarine, (100,0,-50)</td>
<td>N/A</td>
<td>Aircraft, (200,0,100) Return success</td>
<td>WP-002, WP-004</td>
</tr>
<tr>
<td>TC_WP-016</td>
<td>(50,0,50)</td>
<td>Ship, (100,0,0)</td>
<td>Submarine (100,0,-50)</td>
<td>N/A</td>
<td>N/A</td>
<td>Return failure</td>
<td>WP-002, WP-004</td>
</tr>
</tbody>
</table>

Table 5-40 Unit Test Case for CWAutoAimController tracetarget (Aircraft) function

### Target: Submarine

**Function Name:**

**Objective:** self-guided Weapons to trace target touse Radar/Sonar to detect all objects in Radar/Sonar’s range, select the valid nearest target to the previous target position

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Initial Position</th>
<th>Detected Object 1</th>
<th>Detected Object 2</th>
<th>Detected Object 3</th>
<th>Detected Object 4</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-017</td>
<td>(100,0,-50)</td>
<td>Ship, (100,0,0)</td>
<td>Submarine, (80,0,-50)</td>
<td>Submarine, (100,0,-50)</td>
<td>Aircraft, (100,10,80)</td>
<td>Submarine, (100,0,-50) Return success</td>
<td>WP-002, WP-003 Wp-004</td>
</tr>
<tr>
<td>TC_WP-018</td>
<td>(100,0,-50)</td>
<td>Ship, (100,0,0)</td>
<td>Aircraft, (100,0,50)</td>
<td>Submarine (150,0,-50)</td>
<td>N/A</td>
<td>Submarine (150,0,-50) Return success</td>
<td>WP-002, WP-003 Wp-004</td>
</tr>
<tr>
<td>TC_WP-019</td>
<td>(50,0,-50)</td>
<td>Ship, (50,0,0)</td>
<td>Aircraft, (50,0,50)</td>
<td>N/A</td>
<td>N/A</td>
<td>Return failure</td>
<td>WP-002, WP-003 Wp-004</td>
</tr>
</tbody>
</table>

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**

**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.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Weapon Position</td>
<td>Weapon Velocity</td>
<td>Detected Object 1</td>
</tr>
<tr>
<td>TC_WP-020</td>
<td>(0,0,0)</td>
<td>(1000,0,0)</td>
<td>Ship1, (0,0,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Ship2, (0,2,0,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detonation Check</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return success (detonate 2 objects)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-021</td>
<td>(0,0,0)</td>
<td>(1000,0,0)</td>
<td>Ship1, (0.1,0,0)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return success (detonate 1 objects)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-022</td>
<td>(0,0,0)</td>
<td>(1000,0,0)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return failure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
</tbody>
</table>

**Table 5-42  Unit Test Case for CWChargeController HitDetect function**

**Aircraft**

**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.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Weapon Position</td>
<td>Weapon Velocity</td>
<td>Detected Object 1</td>
</tr>
<tr>
<td>TC_WP-023</td>
<td>(0,0,100)</td>
<td>(1000,0,0)</td>
<td>Aircraft, (0,0,100)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Aircraft, (0,100.2)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Detonation Check</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return success (detonate 2 objects)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-024</td>
<td>(0,0,100)</td>
<td>(1000,0,0)</td>
<td>Aircraft, (0,0,100.1)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return success (detonate 1 objects)</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-025</td>
<td>(0,0,100)</td>
<td>(1000,0,0)</td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>N/A</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Return failure</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>WP-005, Wp-006</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-026</td>
<td>(0,0,-50) (1000,0,0) Submarine, (0,0,0) Submarine, (0.2,0,0) N/A</td>
<td>Return success (detonate 2 objects)</td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-027</td>
<td>(0,0,-50) (1000,0,0) Submarine, (0.1,0,0) N/A N/A</td>
<td>Return success (detonate 1 objects)</td>
<td>WP-005, Wp-006</td>
</tr>
<tr>
<td>TC_WP-028</td>
<td>(0,0,-50) (1000,0,0) N/A N/A N/A</td>
<td>Return failure</td>
<td>WP-005, Wp-006</td>
</tr>
</tbody>
</table>

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:

```cpp
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:

```cpp
int vehicleFlag = infoDet.getFlag();
if( ( vehicleFlag != myFlag )&& ( IsTargetType(WeaponType,vehicleType) == TRUE ) )
```

Old version:

```cpp
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 not according the Weapon’s precision. If hit, it calls target’s hit() function with the parameter fire power.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-029</td>
<td>baseClass * ship1, resistance 50, precision=80%, fire power=1.</td>
<td>20 iterations are run and 18 times hit, 2 times miss, resistance=42.</td>
<td>WP-006, WP-0007, WP-008</td>
</tr>
</tbody>
</table>

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 baseClass, 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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data Current Velocity</th>
<th>Current Position</th>
<th>Target Position</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-030</td>
<td>(1000,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>(0,0,0)</td>
<td>WP-001</td>
</tr>
<tr>
<td>TC_WP-031</td>
<td>(1000,0,0)</td>
<td>(0,0,0)</td>
<td>(100,0,0)</td>
<td>(1000,0,0)</td>
<td>WP-001</td>
</tr>
<tr>
<td>TC_WP-032</td>
<td>(1000,0,0)</td>
<td>(0,0,0)</td>
<td>(0,100,0)</td>
<td>(866.03, 500.0)</td>
<td>WP-001</td>
</tr>
<tr>
<td>TC_WP-033</td>
<td>(1000,0,0)</td>
<td>(0,0,0)</td>
<td>(0,100,100)</td>
<td>(612.37,353.55,707.11)</td>
<td>WP-001</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-041</td>
<td>Click icon, click map</td>
<td>Bitmap, position, type, flag</td>
<td>SC-001 to SC-006</td>
</tr>
<tr>
<td>TC_SC-042</td>
<td>Click icon, click map</td>
<td>Full of vehicles within the map</td>
<td>SC-001 to SC-006</td>
</tr>
<tr>
<td>TC_SC-043</td>
<td>Click clear all button</td>
<td>All vehicles disappear from the map</td>
<td>SC-001 to SC-006</td>
</tr>
<tr>
<td>TC_SC-044</td>
<td>Click Undo button</td>
<td>The most recent object is removed from the map</td>
<td>SC-001 to SC-006</td>
</tr>
<tr>
<td>TC_SC-045</td>
<td>Click Ok button</td>
<td>Setup dialogue window closed and main window display</td>
<td>SC-001 to SC-006</td>
</tr>
<tr>
<td>TC_SC-046</td>
<td>Click Cancel button</td>
<td>Setup dialogue will be closed.</td>
<td>SC-001 to SC-006</td>
</tr>
</tbody>
</table>

Table 5-47 Test Case for Simulation Controller(SetUpDlg) Subsystem

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-047</td>
<td>V1(0.0, 0.0, 0.0) + V2(1.0, 1.0, 1.0)</td>
<td>V(1.0, 1.0, 1.0)</td>
<td>SC-013-01</td>
</tr>
<tr>
<td>TC_SC-048</td>
<td>V2(1.0, 1.0, 1.0) * 2.0</td>
<td>V(2.0, 2.0, 2.0)</td>
<td>SC-013-01</td>
</tr>
<tr>
<td>TC_SC-049</td>
<td>V2(1.0, 1.0, 1.0) / 2.0</td>
<td>V(0.5, 0.5, 0.5)</td>
<td>SC-013-01</td>
</tr>
<tr>
<td>TC_SC-050</td>
<td>V2(1.0, 1.0, 1.0) / 0.0</td>
<td>Error</td>
<td>SC-013-01</td>
</tr>
<tr>
<td>TC_SC-051</td>
<td>V1(1.0, 1.0, 1.0) - V1(0.0, 0.0, 0.0)</td>
<td>Setup dialogue will be closed.</td>
<td>SC-013-01</td>
</tr>
<tr>
<td>TC_SC-052</td>
<td>Click Cancel button</td>
<td>Setup dialogue will be closed.</td>
<td>SC-013-01</td>
</tr>
</tbody>
</table>

Table 5-48 Test Case for Simulation Controller(Vector) Subsystem
**Objective:** For class SC, test the user action to interface is correct.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Trace-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-053</td>
<td>Click on SETUP button on the Toolbar or Set up item from Start menu of the main window</td>
<td>Set up dialog window is to be displayed, iconic buttons and a cyan rectangle shown</td>
<td>SC-012</td>
</tr>
<tr>
<td>TC_SC-054</td>
<td>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</td>
<td>Output the text info of the 2-D array to a text file out.txt via cout. The same number of VehicleInfo expected</td>
<td>SC-001</td>
</tr>
<tr>
<td>TC_SC-055</td>
<td>Click on OK button of the SetUpDlg dialog window after picking up a number of vehicles</td>
<td>Created objects will display in the simulated naval battle fields in the main window</td>
<td>SC-012</td>
</tr>
</tbody>
</table>

Table 5-49  Test Case for Simulation Controller (SC) Subsystem

**Objective:** For class VehicleFactory, test creation of ship object is correct.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Trace-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-056</td>
<td>Aircraft Carrier 1</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-057</td>
<td>Aircraft Carrier 10</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-058</td>
<td>Aircraft Carrier 1</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-059</td>
<td>Aircraft Carrier 10 Aircraft 10</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-060</td>
<td>Aircraft Carrier 10 Aircraft 10 2</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-061</td>
<td>Aircraft Carrier 10 Aircraft 10 2</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-062</td>
<td>Aircraft Carrier 10 Aircraft 10 2</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
<tr>
<td>TC_SC-063</td>
<td>Aircraft Carrier 10 Aircraft 10 2</td>
<td>Created objects will display in the simulated naval battle fields</td>
<td>SC-001, SC-012</td>
</tr>
</tbody>
</table>

Table 5-50  Test Case for Simulation Controller (VehicleFactory) Subsystem
**Objective:** For class Controller, test the start, pause, stop, and resume functions.

<table>
<thead>
<tr>
<th>Test Case#</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_SC-064</td>
<td>Click “start” with vehicles created</td>
<td>Animation starts. (Fig. 4)</td>
<td>SC-012</td>
</tr>
<tr>
<td>TC_SC-065</td>
<td>Click “start” without vehicles created</td>
<td>No action.</td>
<td>SC-012</td>
</tr>
<tr>
<td>TC_SC-066</td>
<td>Click “start” when animation is running</td>
<td>No effect.</td>
<td>SC-012</td>
</tr>
<tr>
<td>TC_SC-067</td>
<td>Click “Pause” when animation is running</td>
<td>Animation is paused.</td>
<td>SC-016</td>
</tr>
<tr>
<td>TC_SC-068</td>
<td>Click “Pause” when animation isn’t running</td>
<td>No action.</td>
<td>SC-016</td>
</tr>
<tr>
<td>TC_SC-069</td>
<td>Click “Resume” when animation is paused</td>
<td>Animation is resumed.</td>
<td>SC-017</td>
</tr>
<tr>
<td>TC_SC-070</td>
<td>Click “Pause” when animation running</td>
<td>No action.</td>
<td>SC-016</td>
</tr>
<tr>
<td>TC_SC-071</td>
<td>Click “Stop” when animation is running</td>
<td>Animation is terminated and is reset for next simulation.</td>
<td>SC-018</td>
</tr>
<tr>
<td>TC_SC-072</td>
<td>Click “Stop” when animation isn’t running</td>
<td>No action. System is set for new simulation if necessary.</td>
<td>SC-018</td>
</tr>
</tbody>
</table>

Table 5-51  Test Case for Simulation 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/Sonar the detailed information about detected object within its range. For all test cases, fifteen vehicles, three of each Aircraft, Submarine, Cruiser, Battleship, Destroyer, are created. Each vehicles can use their Radar/Sonar 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.

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_CD-044</td>
<td>Vehicle declare Radar object r(ID, 200), state=1, and its position(4,5,6)</td>
<td>Get the number of detected object within Radar’s range and a list of pointer to detected objects</td>
<td>CD-004 CD-008</td>
</tr>
<tr>
<td>TC_CD-045</td>
<td>Vehicle declare default Radar object r(), range=1000, state=1, position(4,5,6)</td>
<td>Get the number of detected object within Radar’s range and a list of pointer to detected objects</td>
<td>CD-004 CD-008</td>
</tr>
<tr>
<td>TC_CD-046</td>
<td>the number of detected object within Radar’s range.</td>
<td>Go through all pointer inside the detected list</td>
<td>CD-004 CD-008</td>
</tr>
<tr>
<td>TC_CD-047</td>
<td>vehicle detects the number of objects within Radar’s range, and access detected objects by declaring a detected object.</td>
<td>Get each detected object pointed by pointer inside the detected list</td>
<td>CD-004 CD-008</td>
</tr>
<tr>
<td>TC_CD-048</td>
<td>Call turnoff</td>
<td>Nothing is detected.</td>
<td>CD-004 CD-008</td>
</tr>
</tbody>
</table>

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

**Objective:** All the ship and Aircraft can moved on the map, receive the message, detect the enemy, attack enemy, and can be attacked by the enemy and recover or dead.

<table>
<thead>
<tr>
<th>Test Case#</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Traceability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_BS-054</td>
<td>Ship/Aircraft current position Vector(10,10,0) destination position Vector(100,10,0)</td>
<td>Ship/Aircraft move at fix speed towards to the destination</td>
<td>BS-001 to BS-002</td>
</tr>
<tr>
<td>TC_BS-055</td>
<td>1)An underwater object: Vector(10,10,-10) is aimed at ship/Aircraft. 2)Message send by allies.</td>
<td>Ship/Aircraft changes its direction 180° at max speed.</td>
<td>BS-003</td>
</tr>
<tr>
<td>TC_BS-056</td>
<td>1)Ship/Aircraft position : Vector(0,0,0) 2)Object Vector(26,26,0).</td>
<td>Ship/Aircraft reduces its speed and fire cannon.</td>
<td>BS-001 to BS-002, BS-021</td>
</tr>
<tr>
<td>TC_BS-057</td>
<td>1)Ship/Aircraft position: Vector(0,0,0) 2)Object Vector(80,80,0)</td>
<td>Ship/Aircraft reduces its speed and fire Missile.</td>
<td>BS-002, BS-021</td>
</tr>
<tr>
<td>TC_BS-058</td>
<td>1)Ship/Aircraft position : Vector(0,0,0) 2)Object Vector(53,53,0)</td>
<td>Ship/Aircraft reduces its speed and fire Missile.</td>
<td>BS-002, BS-021</td>
</tr>
<tr>
<td>TC_BS-059</td>
<td>Fire Weapon to ship/Aircraft</td>
<td>The resistance points of ship/Aircraft reduces continuously without recovery and finally reduces to zero.</td>
<td>BS-025, BS-026, BS-027</td>
</tr>
<tr>
<td>TC_BS-060</td>
<td>Fire Weapon to ship/Aircraft</td>
<td>The resistance points reduced first and recovered to maximum 300 later on.</td>
<td>BS-025, BS-026, BS-027</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Trace-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-034</td>
<td>Destroyer(0,0,0) Flag1</td>
<td>Both Destroyer are hit</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Destroyer(30,0,0) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-035</td>
<td>Destroyer (0,0,0) Flag1</td>
<td>None of Destroyer is hit because of range</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Destroyer (50,0,0) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-036</td>
<td>Destroyer (0,0,0) Flag1</td>
<td>Both vehicle are hit</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Submarine (30,0,-10) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-037</td>
<td>Destroyer (0,0,0) Flag1</td>
<td>Invalid position</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Aircraft(30,0,100) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-038</td>
<td>Destroyer (0,0,0) Flag1</td>
<td>Trace Target and hit</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Destroyer(10,0,0) to(10,100,0) Flag0</td>
<td>Trace Target and hit</td>
<td>WP-007,WP-008</td>
</tr>
</tbody>
</table>

Table 5-54 Test Case for Weapon(Wtorpedo) Subsystem

<table>
<thead>
<tr>
<th>Test Case #</th>
<th>Test Data</th>
<th>Expected Result</th>
<th>Trace-ability</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC_WP-039</td>
<td>Battle Ship(0,0,0) Flag1</td>
<td>Both Battle ships are hit.</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Battle Ship(30,0,0) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-040</td>
<td>Battle Ship(0,0,0) Flag1</td>
<td>None of battle ship is hit because of range.</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Battle Ship(50,0,0) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-041</td>
<td>Battle Ship(0,0,0) Flag1</td>
<td>Invalid Weapon position</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Battle Ship(30,0,-10) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
<tr>
<td>TC_WP-042</td>
<td>Battle Ship(0,0,0) Flag1</td>
<td>Submarine cannot be detonate</td>
<td>WP-005,WP-006</td>
</tr>
<tr>
<td></td>
<td>Submarine(30,0,0) Flag0</td>
<td></td>
<td>WP-007,WP-008</td>
</tr>
</tbody>
</table>

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 operate coordinately 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.

<table>
<thead>
<tr>
<th>Test case #</th>
<th>Descriptive</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TC1</td>
<td>Test Description</td>
<td>This test case is to check the navigation aspect of ship/Aircraft subsystem when other ships and Aircrafts are present.</td>
</tr>
<tr>
<td></td>
<td>Input states</td>
<td>Some other ships, Aircrafts on both sides are created and put in places relatively near, then relatively far away to the ship/Aircrafts.</td>
</tr>
<tr>
<td></td>
<td>Expected results</td>
<td>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</td>
</tr>
</tbody>
</table>

| 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. 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 some 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. 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.