2013-03

ANALYZING NAVAL STRATEGY FOR COUNTER-PIRACY OPERATIONS, USING THE MASSIVE MULTIPLAYER ONLINE WAR GAME LEVERAGING THE INTERNET (MMOWGLI) AND DISCRETE EVENT SIMULATION (DES)

Hutchins, Chad R.

Monterey, California. Naval Postgraduate School

NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

ANALYZING NAVAL STRATEGY FOR COUNTER-PIRACY OPERATIONS, USING THE MASSIVE MULTIPLAYER ONLINE WAR GAME LEVERAGING THE INTERNET (MMOWGLI) AND DISCRETE EVENT SIMULATION (DES)

by

Chad R. Hutchins
March 2013

Thesis Advisor:	Donald Brutzman
Thesis Co–Advisor:	Arnold Buss
Second Reader	Terry Norbraten

This thesis was performed at the MOVES Institute

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE			Form Approved OMB No. 0704–0188
Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704–0188) Washington DC 20503.			
1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2013	**3. REPORT TYPE AND DATES COVERED** Master's Thesis	
4. TITLE AND SUBTITLE ANALYZING NAVAL STRATEGY FOR COUNTER-PIRACY OPERATIONS, USING THE MASSIVE MULTIPLAYER ONLINE WAR GAME LEVERAGING THE INTERNET (MMOWGLI) AND DISCRETE EVENT SIMULATION (DES)			**5. FUNDING NUMBERS**
6. AUTHOR Chad R. Hutchins			
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943–5000			**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING /MONITORING AGENCY NAME(S) AND ADDRESS(ES) N/A			**10. SPONSORING/MONITORING AGENCY REPORT NUMBER**
11. SUPPLEMENTARY NOTES The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol number ____N/A____.			
12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release; distribution is unlimited			**12b. DISTRIBUTION CODE** A

13. ABSTRACT

Combating piracy is an age-old mission for international navies, as piracy has troubled ocean-going vessels for centuries. Somali piracy, like all piracy uprisings in the past, is a land-based problem stemming from a dysfunctional government that cannot enforce the laws of the land. This lack of law enforcement is what provides pirates a safe harbor to operate, which allows the problem to trickle into international waters and become a maritime problem. However, in the case of Somali piracy, leaders from the U.S. State Department and the U.S. Navy have said there is too much water in the Indian Ocean for the coalition navies to effectively patrol.

This thesis first demonstrates how the MMOWGLI platform can be used for crowd-sourced brainstorming of strategic options for counter-piracy, yielding valuable action plans that can be modeled, simulated, and analyzed to make strategic decisions. Three highly rated Action Plans from the 2012 Piracy MMOWGLI game were then modeled and simulated using Discrete Event Simulation (DES). Simulation analysis suggests that the amount of ocean is not a factor if coalition navies aggressively patrol the Somali coast, either directly off shore from active pirate camps or by the use of a naval quarantine.

Strategy development for counter-piracy, like any other wicked strategic problem, is usually conducted by senior naval leaders in the upper echelons of specific commands. The MMOWGLI game-play from Piracy MMOWGLI and other MMOWGLI games suggests the U.S. Navy needs to consider utilizing a broader range of officers, enlisted personnel and civilians for brainstorming strategic options. There are an unprecedented number of enlisted sailors with degrees and junior officers educated in joint professional military education. It is time the military taps into this knowledge base for help in planning and implementing strategy.

14. SUBJECT TERMS Crowd-sourcing, Discrete Event Simulation (DES), MMOWGLI, Somali Piracy, Simkit, Viskit, Java, KML, X3D, X3D-Edit, OpenMap, Wicked Problems			**15. NUMBER OF PAGES** 225
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UU

NSN 7540–01–280–5500 Standard Form 298 (Rev. 2–89)
Prescribed by ANSI Std. 239–18

THIS PAGE INTENTIONALLY LEFT BLANK

Approved for public release; distribution is unlimited

ANALYZING NAVAL STRATEGY FOR COUNTER-PIRACY OPERATIONS, USING THE MASSIVE MULTIPLAYER ONLINE WAR GAME LEVERAGING THE INTERNET (MMOWGLI) AND DISCRETE EVENT SIMULATION (DES)

Chad R. Hutchins
Lieutenant, United States Navy
B.S. The Citadel, 2007

Submitted in partial fulfillment of the
requirements for the degree of

**MASTER OF SCIENCE IN
MODELING, VIRTUAL ENVIRONMENTS, AND SIMULATION (MOVES)**

from the

**NAVAL POSTGRADUATE SCHOOL
March 2013**

Author: Chad R. Hutchins

Approved by: Donald Brutzman
Thesis Advisor

Arnold Buss
Thesis Co–Advisor

Terry Norbraten
Second Reader

Christian Darken
Chair, MOVES Academic Committee

Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

Combating piracy is an age-old mission for international navies, as piracy has troubled ocean-going vessels for centuries. Somali piracy, like all piracy uprisings in the past, is a land-based problem stemming from a dysfunctional government that cannot enforce the laws of the land. This lack of law enforcement is what provides pirates a safe harbor to operate, which allows the problem to trickle into international waters and become a maritime problem. However, in the case of Somali piracy, leaders from the U.S. State Department and the U.S. Navy have said there is too much water in the Indian Ocean for the coalition navies to effectively patrol.

This thesis first demonstrates how the MMOWGLI platform can be used for crowd-sourced brainstorming of strategic options for counter-piracy, yielding valuable action plans that can be modeled, simulated, and analyzed to make strategic decisions. Three highly rated Action Plans from the 2012 Piracy MMOWGLI game were then modeled and simulated using Discrete Event Simulation (DES). Simulation analysis suggests that the amount of ocean is not a factor if coalition navies aggressively patrol the Somali coast, either directly off shore from active pirate camps or by the use of a naval quarantine.

Strategy development for counter-piracy, like any other wicked strategic problem, is usually conducted by senior naval leaders in the upper echelons of specific commands. The MMOWGLI game-play from Piracy MMOWGLI and other MMOWGLI games suggests the U.S. Navy needs to consider utilizing a broader range of officers, enlisted personnel and civilians for brainstorming strategic options. There are an unprecedented number of enlisted sailors with degrees and junior officers educated in joint professional military education. It is time the military taps into this knowledge base for help in planning and implementing strategy.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF FIGURES

LIST OF TABLES

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF ACRONYMS AND ABBREVIATIONS

3D	Three Dimensional
AMISOM	African Union Mission in Somalia
API	Application Programming Interface
APS	African Partnership Station
ATC	Agent Technology Center
BII	Business Innovation Initiative
CDS	Commander Destroyer Squadron
CNMOC	Commander Naval Meteorology and Oceanography Command
CTF	Commander Task Force
DES	Discrete Event Simulation
DHS	Department of Homeland Security
DOE	Design of Experiments
DTS	Discrete Time Simulation
EEZ	Economic Exclusion Zone
EU	European Union
EVT	EDGE Virtual Training
FEL	Future Event List
GPS	Global Positioning System
GUI	Graphical User Interface
GWT	Google Web Toolkit
HOA	Horn of Africa
HTTP	Hypertext Transfer Protocol
HSDL	Homeland Security Digital Library
IA–CGF	Intelligent Agent Simulation Computer Generated Force
ICC	International Chamber of Commerce
IFTF	Institute for the Future
IMB	International Maritime Bureau
IRTC	Internationally Recommended Traffic Corridor

JCA	Joint Campaign Analysis
MMOWGLI	Massive Multiplayer Online War–game Leveraging the Internet
MOE	Measure of Effectiveness
NAVO	Naval Oceanographic Command
NCIS	Naval Criminal Investigative Service
NEC C2 M2	Network Centric Command and Control Maturity Models
NMCI	Navy/Marine Corps Intranet
NPS	Naval Postgraduate School
OBP	Oceans Beyond Piracy
ONR	Office of Naval Research
OSA	Open System Architecture
PANOPEA	Piracy Asymmetric Naval Operations Patterns modeling for Education and Analysis
PARS	Pirate Attack Risk Surface
PPS	Piracy Performance Surface
PPSN	Piracy Performance Surface Model
RF	Royalty Free
SWDG	Surface Warfare Development Group
TCP	Transmission Control Protocol
VV&A	Verification, Validation, and Accreditation
X3D	Extensible 3D Graphics Language
XML	Extensible Markup Language

ACKNOWLEDGMENTS

First and foremost I want to thank my family for their continued support while I was on this time-consuming endeavor. To my beautiful wife, Elizabeth, although it was a tough 27 months, we made it and without a doubt have a stronger relationship because of what we endured during the process. Thank you for believing in me and always pushing me to be my best. To my three amazing children, thank you for all the love and hugs that got me through each day. I'm so proud of each of you in all that you have accomplished the past 27 months.

This thesis work would not have been possible without my advisors, Dr. Brutzman, Dr. Buss, and Terry Norbraten. Dr. Brutzman, you are an innovator of great thoughts and higher learning. You always ensured I went above and beyond my potential and showed me the value of thinking outside the bounds of all problems. The benefit of your advising went well beyond simply modeling and simulation, I will forever benefit professionally and personally from you efforts during this thesis process. Dr. Buss, your methodology for conquering large problems is the reason this thesis is complete, but also a methodology I will take with me to tackle all problems in the future. Thank you for your patience and time. Terry, without your technical assistance I would not have been able to complete this thesis. I came here without a day of programming in my life, but your help in class and on this thesis allowed me to conquer Java and produce work I never thought I was capable of performing. Thanks for all your time with my questions and listening while I vented my frustrations.

Finally, I would like to thank my fellow King Cobras, at the Monterey Tennis Center. It was always a great stress relief to get out and dominate the tennis courts with each of you. I will never forget the memories and friendship developed during the time of our undefeated season. Slither on to further greatness!

THIS PAGE INTENTIONALLY LEFT BLANK

I. INTRODUCTION

"A genuine leader is not a searcher for consensus, but a molder of consensus." [1]

—Martin Luther King Jr.

A. PROBLEM STATEMENT

Piracy around the HOA has plagued the international community for the last six years. Since 2008, Somali pirates have continuously adapted to naval tactics and merchant ship best management practices; they have increased the distance in which they operate from shore, become more aggressive, and begun using more sophisticated technology, such as GPS and satellite telephones. This has resulted in increased number of piracy incidents, increased number of mariners who have been taken hostage and killed, and billions of dollars in economic cost for the international community (Bowden & Basnet, 2011). However, as of early 2012, there has been a drastic decrease in piracy incidents and successful hijackings. This decrease can be mainly attributed to the use of armed guards on merchant vessels, as well as continued presence and operations of naval forces (Major, Kline, & Fricker, 2012). With this dramatic decrease in pirate success corresponding to merchants being able to protect themselves, many analysts are asking if the international navies are still worth the cost of operations around the Horn of Africa.

This thesis analyzes and evaluates naval patrol strategies for counter–piracy operations in the Gulf of Aden and Indian Ocean. Since pirates have continually changed their tactics based on military and merchant tactics this thesis demonstrates numerous options for naval leaders to consider for future planning. These options include, a means to war game and easily model, simulate, and analyze naval strategy should new pirate tactics arise. This thesis provides analysis on how international naval strategy can continue to support policy for piracy around the Horn of Africa. The system design and methodology is also applicable for the west coast of Africa piracy, future areas that piracy may arise, and other strategic problems.

[1] From http://www.aavw.org/special_features/speeches_speech_king03.html.

B. OVERVIEW

Maritime piracy is not a new mission for the navies around the world; in fact maritime piracy has been around since at least the 14th century BC (Konstam, 2008, p. 10). However, piracy is still a real struggle for policy makers and naval strategists. Modern-day piracy around the Horn of Africa poses a serious threat to international shipping and merchant mariners in some of the busiest shipping waters in the world. It is estimated that between 20 and 30 naval vessels patrol daily around the Horn of Africa and over 42,000 merchant ships travel through the region annually (Bowden & Basnet, 2011).

Somali piracy has had a few ebbs and flows of incident frequency. Toward the end of 2008 the Internationally Recommended Traffic Corridor (IRTC) was implemented in the Gulf of Aden, which had great success in disrupting the pirate business model. Subsequently once the pirate success rate fell in the Gulf of Aden they quickly adapted and began more operations with large motherships in the Indian Ocean at distances over 1,000 nautical miles from the coasts of Somalia. In 2012, the international community has seen a substantial drop in piracy, only 75 incidents and 14 successful hijackings compared to 237 incidents and 49 successful in 2011 (ICC International Maritime Bureau, 2013). The major contributor to this success was armed security teams embarked on merchant ships to thwart pirates from successfully boarding vessels, as described in a published Proceedings article by (Major et al., 2012). This fact raises the obvious question, “Does the international community need to continue investing money in navies to patrol the Horn of Africa for piracy?” Naval leaders, government officials, and merchant companies all agree that the Navy plays a vital role in countering piracy. Therefore, it is important to ensure that navies effectively recognize, prepare, and employ the appropriate strategy that continues to contain the always evolving piracy threat and to ensure the naval strategy matches the policy objectives for counter–piracy efforts.

In these times of budget cuts and need for efficiency in the military it is imperative that simulation and war gaming play a vital role in policy and strategic planning. Simulation can assist in determining if missions are feasible, forces are being employed smartly, and all strategic options have been compared and analyzed.

Meanwhile war gaming, especially through crowd-sourcing, can ensure that all ideas are on the table and given adequate attention and consideration. The current force structure of the Navy is at a time where it is smarter and more capable than ever. However, the ideas of junior officers and enlisted personnel are often suppressed by hierarchical command structures. This thesis provides a methodology to take advantage of this high level of intellect in the Navy and a methodology to rapidly simulate and analyze the results.

C. MINDSET AND APPROACH OF CURRENT COUNTER–PIRACY EFFORTS

When Somali piracy began to peak in 2008, the international community turned to the military to defeat piracy. However, dating backing to the origins of piracy it is well known the root causes of piracy are on land. However, no one wanted to suggest any civilian or military action on the ground, due to complicated international diplomacy considerations and past military difficulties, e.g. Blackhawk Down (http://www.history.com/videos/the-true-story-of-blackhawk-down). The IRTC was implemented and the military began heavy patrols of it and piracy diminished, until the innovative use of "mother ships" allowed pirates to extend their range to over 1,000 nautical miles off the coasts of Somalia. At that time, policy makers at the U.S. State Department began making statements that suggested, the area of water off Somalia is too large to adequately patrol (Shapiro, 2009). Broad qualitative statements like those are what drives the motivation for a good portion of this thesis. It is easy to agree that there is a lot of water in the Indian Ocean, however it is most definitely not necessary to patrol every square mile of ocean in order to protect mariners on the high sea and disrupt pirate activities. Modeling and simulation can help quantify the analysis of alternatives (AoA).

The current U.S. naval strategy is to "deter, disrupt, and suppress piracy," as stated on the Commander Task Force 151 (CTF–151) website (http://www.cusnc.navy.mil/cmf/151/index.html). In the broadest sense this is a bold and probably unachievable strategy for naval forces given the current policy. To "suppress" is defined as "to put down by authority or force" (http://www.merriam–webster.com/dictionary/suppress). Without a policy of fixing the problems of Somalia or a policy that requires direct military action on the ground (which is not popular or

necessary), piracy will continue and the Navy will not be able to effectively suppress piracy. The Navy needs to redefine its strategy to match the current policy. For example, Clausewitz notes the importance of policy driving strategy, not the other way around (Clausewitz, 1984/ 1780–1831, pp. 69, 81, 605). A better strategic plan for counter–piracy forces is:

1.) Disrupt pirate activities, by naval and law enforcement means,

2.) Protect merchant shipping, and

3.) Train Africans, including Somalis on counter-piracy approaches.

This new strategy suggestion is achievable, measurable, and matches current policy objectives.

D. MOTIVATION

1. Personal Experience

In 2010, the author was deployed on USS NICHOLAS (FFG–47) as Force Protection Officer, Visit Board Search and Seizure Officer, and Legal Officer. NICHOLAS was assigned to Africa Partnership Station (APS) – East for three months of training East African military and police forces. During the APS mission he was able to gain a better understanding of the African culture, the attitudes toward piracy in Africa, and how piracy affects the countries on the east coast of Africa. Upon completion of APS NICHOLAS was assigned to CTF–67 and conducted counter–piracy operations in the sixth fleet AOR of the Indian Ocean. During this time a group of Somali pirates mistakenly identified NICHOLAS as a merchant vessel and attacked her with the intent to board her. The pirates came alongside shooting AK–47 machine guns; with the help of .50 caliber machine guns on NICHOLAS the pirates realized that, in fact, NICHOLAS was a warship. NICHOLAS was able to arrest and apprehend five pirates, where they stayed on board for 21 days at sea. The attack on NICHOLAS prompted a major investigation and federal court trial for the five pirates. The author worked closely with Naval Criminal Investigative Service (NCIS) and the Department of Justice until NICHOLAS returned to homeport upon completion of her deployment. After deployment

he went to work with Surface Warfare Development Group (SWDG), now the tactical development staff of Commander Destroyer Squadron Twenty–Six (CDS–26), and assisted in updating the Counter–Piracy Tactical Bulletin for the fleet. Simultaneously he worked extensively for the United States Attorneys (USA) who were prosecuting the case. He handled various matters for the USA including witness preparations, aiding with naval matters that arose in preparation for the trial, and worked on presentations for the trial. The author was then named the government's "Case Agent" for the trial and sat with the attorneys for its duration. The verdict of the trial was the first guilty prosecution of piracy in the U.S. since the Civil War. The trial had major effects on the definition of piracy from a law standpoint; mainly that it is possible to be guilty of piracy without having successfully plundered the vessel (U.S. Library of Congress, 2010). Since the trial he has authored the newest Counter–Piracy Tactical Bulletin for CDS–26 (Commander Destroyer Squadron Twenty-Six, 2012) and continue assisting the U.S. Attorney's Office in prosecuting pirates from the USS ASHLAND Case and the *Yacht Quest* case. He had the opportunity to assist NCIS and the FBI in interviewing pirates, which has allowed the Navy to gain a better understanding on pirate tactics and strategies. During this time he also was able to tour the Yacht Quest and shown how the four Americans on board were brutally murdered by Somali pirates.

Through these experiences the author has learned a lot about Somali piracy and considered numerous ways that the Navy can improve its counter–piracy efforts. There are many people that believe the U.S. should not be patrolling the waters off Somalia and that the easiest solution is to kill them, similar to how pirates were in the old days of piracy. However, after spending time in Africa training Africans, talking with over 30 pirates, and visiting a yacht in which four Americans were brutally murdered by ruthless pirates, the author believes navy vessels do need to be actively patrolling the waters off Somalia, but in a more efficient manner that better aligns with current policies. The author also believes that the international community must dedicate more efforts in Somalia with relief, security, training, and aide to government of Somalia and the African Union. The problem of piracy will not stop without a stable environment in Somali; an

environment that can fulfill the basic needs of the majority of its citizens and maintain peace independent of the international community.

E. RESEARCH QUESTIONS AND OBJECTIVES

This thesis addresses the following questions:

- What are the best patrol strategies for disrupting pirates and protecting merchant shipping in the Gulf of Aden and Indian Ocean?
- Is patrolling only the transit lanes a more effective strategy for detecting and disrupting pirate attacks?
- Is the Somali coastline truly too large to implement an effective quarantine, as most "experts" suggest? Does the whole coast necessarily need to be quarantined to be effective?
- Is operating closer to the Somali shore more effective at disrupting pirate activities?
- Can the online MMOWGLI game be used for crowd–sourcing innovative new ideas for long–standing difficult problems?
- Can the Massive Multiplayer Online War–Game Leveraging the Internet (MMOWGLI) action plans be simulated and analyzed?
- Can Discrete Event Simulation (DES) be used to effectively model and simulate Somali piracy?
- Does Agent Based Modeling utilizing DES provide a feasible technique for modeling multiple "moving and sensing" agents in a maritime environment?

F. SCOPE OF THESIS

This thesis leverages discrete event simulation (DES), open–source modeling and simulation software created by faculty and staff of the Naval Postgraduate School, Simkit and Viskit, the MMOWGLI innovation-game platform, and open-source X3D and GIS software for visualization. The MMOWGLI platform allows for policy and strategy ideas to be brainstormed and the leading ideas to form into action plans that give the specific details of what the policy or strategy entails. These actions plans provide the framework for the simulations for this thesis. This thesis does not aim to provide all the answers to solve piracy around the Horn of Africa. It does however demonstrate a powerful methodology and tools for policy and strategy planners to consider as the international

community moves forward in creating a policy–strategy match for counter–piracy operations and other strategic objectives.

G. THESIS ORGANIZATION

Chapter I discusses the problem statement, the motivation for the research, and the research questions for the thesis. Chapter II provides an overview of the technologies used for this thesis and past work using these technologies, as well as published work in modeling efforts for Somali Piracy. Chapter III discusses crowd-sourcing with MMOWLGI. It provides the basic overview of what the MMOWGLI game platform can enable, how it is relevant to strategy planning, and how it has been used to assist other innovators and planners. Chapter IV gives the detailed problem description and examines both the data and the MMOWGLI authored action plans that assist in modeling Somali Piracy. Chapter V provides details on the modeling and simulation of key scenarios of interest. It shows the simulation event graphs for all the major entities and discusses the major scenarios analyzed. Chapter VI gives the detailed simulation analysis for this challenging problem. Chapter VII provides thesis conclusions and recommendations for future work, emphasizing how strategy for counter–piracy operations around the Horn of Africa can be improved.

THIS PAGE INTENTIONALLY LEFT BLANK

II. BACKGROUND AND RELATED WORK

"Conformity is the jailer of freedom and the enemy of growth"[2]

—John F. Kennedy

A. INTRODUCTION

This chapter provides an overview of the technologies used for this thesis and past work using these technologies. It also acknowledges other modeling and simulation research performed on maritime piracy. The descriptions are not meant to be all–inclusive, rather give the reader a general understanding and provide references for further research. All technologies used in this thesis are open–source, royalty free (RF), and repeatable. The majority of the tools used were developed by NPS faculty, staff, and students.

B. DISCRETE EVENT SIMULATION (DES)

1. Methodology

Discrete event simulation (DES) in its simplest terms can be described with states, events, and scheduling relationships between events (Buss, 2011, p. 1–1). DES modeling represents a system as it evolves by state variables changing at distinct points in time; these points in time are where events occur. An example of a state variable from this thesis is the number of successful pirate attacks; this value increases by one e time a pirate attack is successful. An event is an instantaneous occurrence that may change the state of the system, the word may is used here because the event could simply schedule another event and not change a state variable. Along with this possible state change within an event there also needs to be a scheduling relationship between events. This is what allows the system to progress from one state to another and advance time within the system (Law, 2007, pp. 6 – 8).

[2]From http://millercenter.org/president/speeches/detail/5741.

Time advance in a DES model is called Next Event, similar names in related DES systems are called Event Queue Management and Simulation Time Clock. For each event state transition an event is scheduled with a given time delay. The basic next-event algorithm for a DES event queue is depicted in Figure 1.

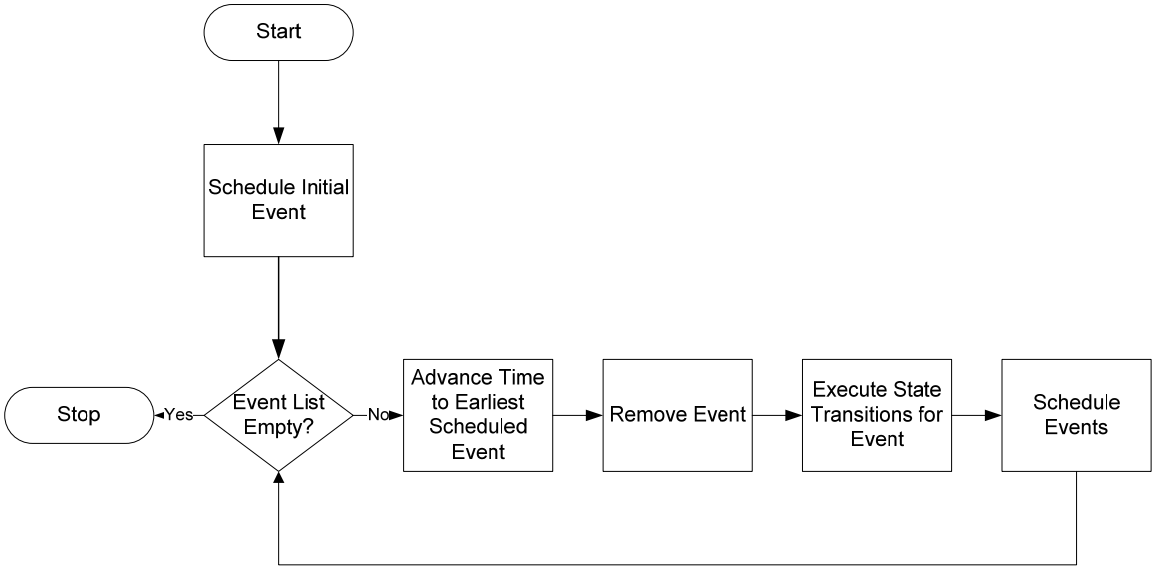


Figure 1. The logic for the Next Event Algorithm for Discrete Event Simulation (DES) (From Buss, 2011).

Two other fundamental parts of a DES model is the Future Event List (FEL) and parameters. The FEL is a structure in which pending events are stored. Each event is stored in the FEL based on time, with the nearest time on top. The structure used for the FEL must be able to add events, store them in time order, and remove an event that is due up to be processed. Parameters, also called Simulation Parameters in a DES model, are variables that do not change during the course of the simulation run (Buss, 2011, pp. 1– 4 to 1–5). An example of two simulation parameters from this thesis is the number of Navy ships and the maximum speed of a Navy ship. These values are locked and do not change during the course of a simulation run.

Event graphs are commonly used to represent a DES model (Schruben, 1983). An event graph contains nodes and edges. Each node represents a specific event, or state transition, and an edge represents the scheduling of other events. The event graph in

Figure 2 depicts a simple (yet common) event process for a DES system, an Arrival Process (Buss, 2011, pp. 3–1 to 3–3). An arrival process is a process that models how entities appear in a simulation. The Run event simply initializes the state variable for number of replications, N, to zero and schedules an arrival with a time delay of t_A. The Arrival event adds one to the state variable, N, and schedules another Arrival with a time delay of t_A. The arrival rates can be any statistical distribution and is determined based on the data for the particular model. Event graphs can also include additional functionality such as cancelling edges, assigning priorities, and implementation that functions as a "for" loop, to name a few (Buss, 2011, pp. 4–4 to 4–5).

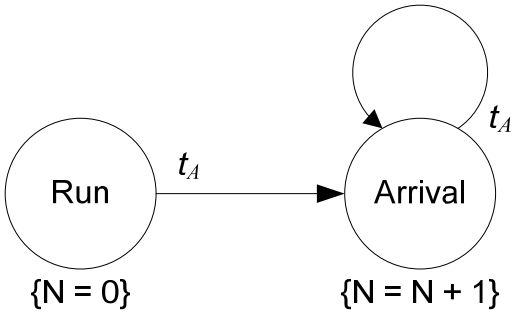


Figure 2. An Example Event Graph of an Arrival Process showing how entities arrive in a system (From Discrete Event Simulation Modeling by Dr. Arnold Buss).

An event graph model such as in Figure 2 is referred to as a component. Each component has its own set of parameters and state variables. A component allows the modeler to decompose and implement the model in pieces, rather than having one gigantic and confusing (and error prone) event graph. Therefore, the components need the ability to communicate with one another. This is done by using SimEventListeners. The SimEventListener pattern allows one, or many, components to listen for state changes in another component. Once the state change occurs in one component it triggers a state change in the listening component (Buss, 2011, pp. 5–1 to 5–2). The listening pattern is depicted in Figure 3. SimEventListeners play a huge part in the simulations of this thesis by allowing interaction between entities. More detail on on DES is provided in Chapter V.

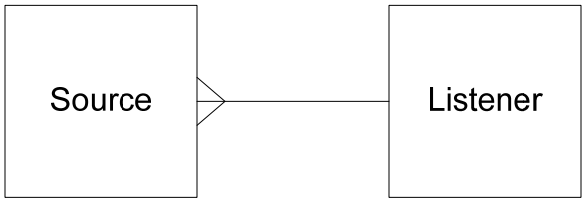


Figure 3. A Depiction of the SimEvenListener Pattern for a DES system (From Discrete Event Simulation Modeling by Dr. Arnold Buss).

2. Simkit

Simkit is an open–source application programming interface (API) that is used for creating Discrete Event Simulation models. It was developed by NPS faculty, mainly Dr. Arnold Buss, and is regularly upgraded and modified by NPS students and faculty. Simkit started out as a Java API, but has recently been implemented in the Python, Ruby, and JavaScript programming languages. The main functions of Simkit are to allow for straightforward implementation of event graphs and provide statistical analysis of simulations. Simkit allows for 2D modeling and provides a basic graphical user interface (GUI) to visualize entity level simulations, Figure 4 shows an example of this GUI. Simkit has been used in numerous theses and research projects, a few of which are discussed below (Buss, 2011, pp. 8–1 to 8–2).

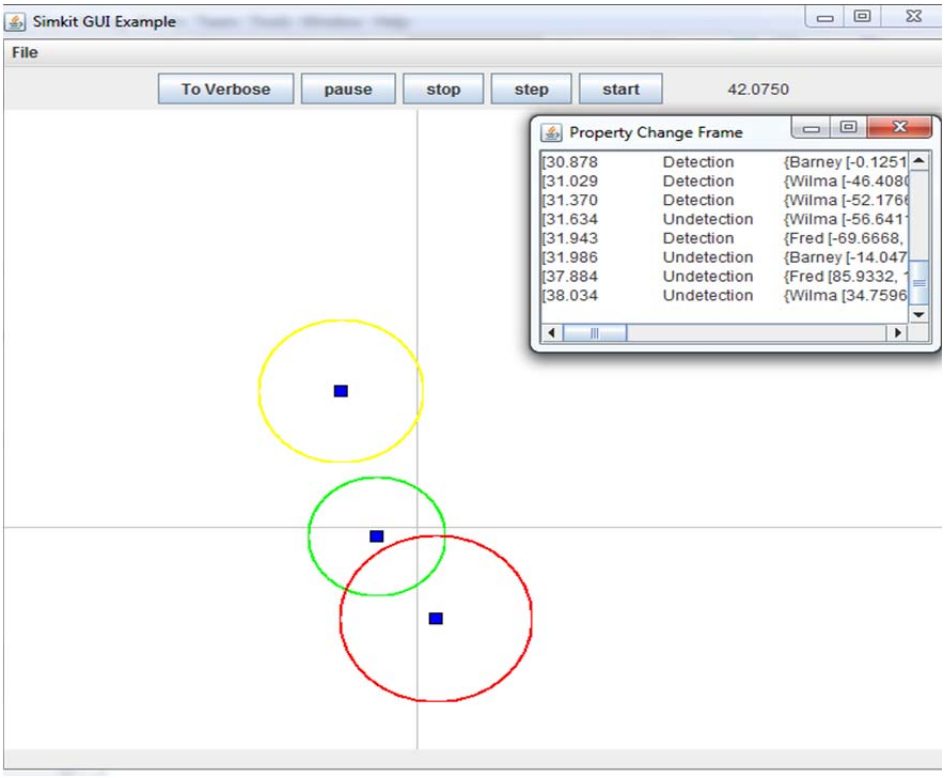


Figure 4. A simple GUI featuring a Property Change Frame displaying Detection and Undetection events.

There are two highly essential elements of DES modeling that are implemented in Simkit and used extensively for this thesis: movement and detection. It was once believed that one could not adequately model movement in a DES system, however as shown by (Buss & Sanchez, 2005) and others, modeling time-consuming movements in DES is often more desirable than utilizing more time-consuming time–step approach. The entities in this thesis model uniform linear motion by subclassing Simkit's BasicLinearMover class. For a DES model to move, it must know its initial starting location at time t_0 and a velocity v in which to move. The use of dead reckoning, or calculating the current position by utilizing past positions, can be easily computed by storing initial location, the velocity vector, and time which movement began (Buss & Sanchez, 2005). Detection is modeled in this thesis using a "cookie cutter" sensor. The sensor is given a range and if an entity comes within the range, called "enter range" of the sensor a detection event is scheduled with a time delay of zero. When the entity leaves this range, called "exit range, an undetection event is scheduled with a time delay of zero

(Buss & Sanchez, 2005). Figure 5 depicts how a cookie cutter sensor is modeled. Both movement and detection is thoroughly described in (Buss & Sanchez, 2005) if more detail is desired.

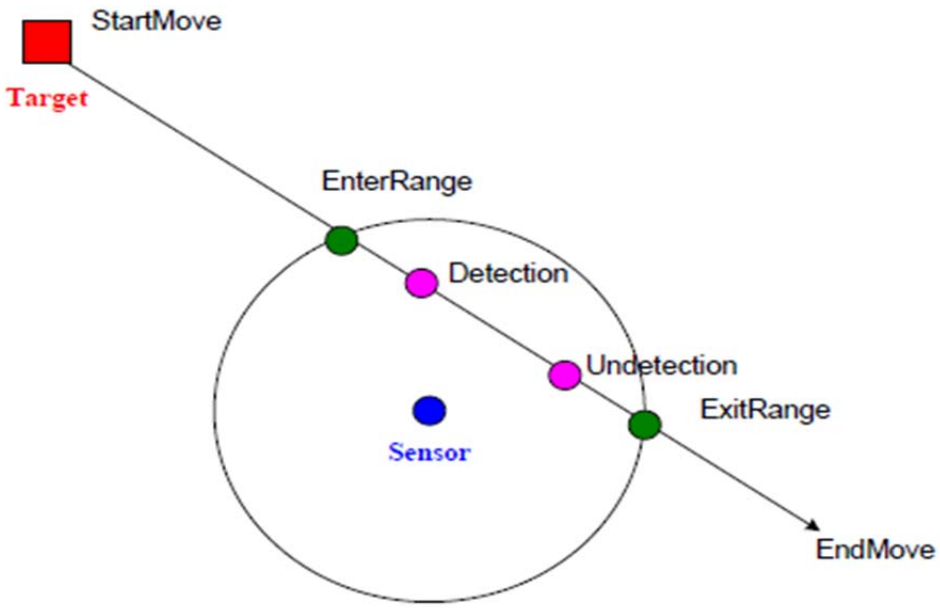


Figure 5. A graphical depiction of a Simkit Cookie cutter sensor model. From (Buss & Sanchez, 2005). Moving sensors are also possible.

Figure 5 shows many important concepts for movement and detection in DES.

- StartMove Event: The event to begin movement of an entity. It sets the velocity and destination of a mover and/or sensor. This event is also heard by listeners in order to know which sensor started moving.
- EnterRange Event: Is scheduled by the SensorMoverReferee when a mover enters the maximum range of a the sensor.
- Detection Event: The mover is detected and added to the contact list.
- Undetection Event: The mover is undetected (exits the maximum sensor range)

- ExitRange Event: Is scheduled by the SensorMoverReferee when a mover exits the maximum range of the senor. The event gives the mover that exited the ranged and the sensor that was exited.
- EndMove Event: The mover has reached its destination. The mover may immediately be ordered to startMove, if necessary.

3. Viskit

One potential hindrance of Simkit is that users are required to be proficient in computer programming. It has been noted that there is a need for students, researchers, and analysts to be able to create models and run simulations without having to be proficient at programming. An attempt to alleviate this requirement, as well as, allow for more rapid development of models and simulations, the developers of Simkit and other NPS faculty and students developed Viskit. Viskit is an open–source visual programming methodology and API. Viskit allows the user to graphically implement a normally hand-drawn event graph. Figure 6 shows the same Arrival Process as Figure 2, except the figure is drawn using Viskit.

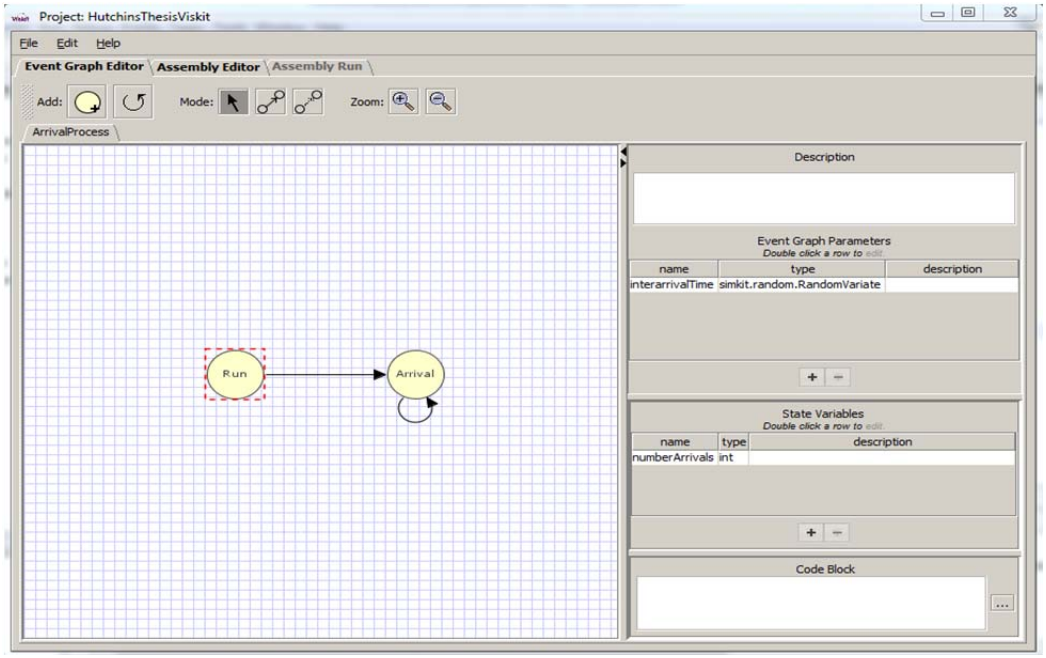


Figure 6. Arrival Process event graph using Viskit.

The event graph components are formatted into Extensible Markup Language (XML), as shown in Figure 7, and with the XML one can generate Simkit Java source code.

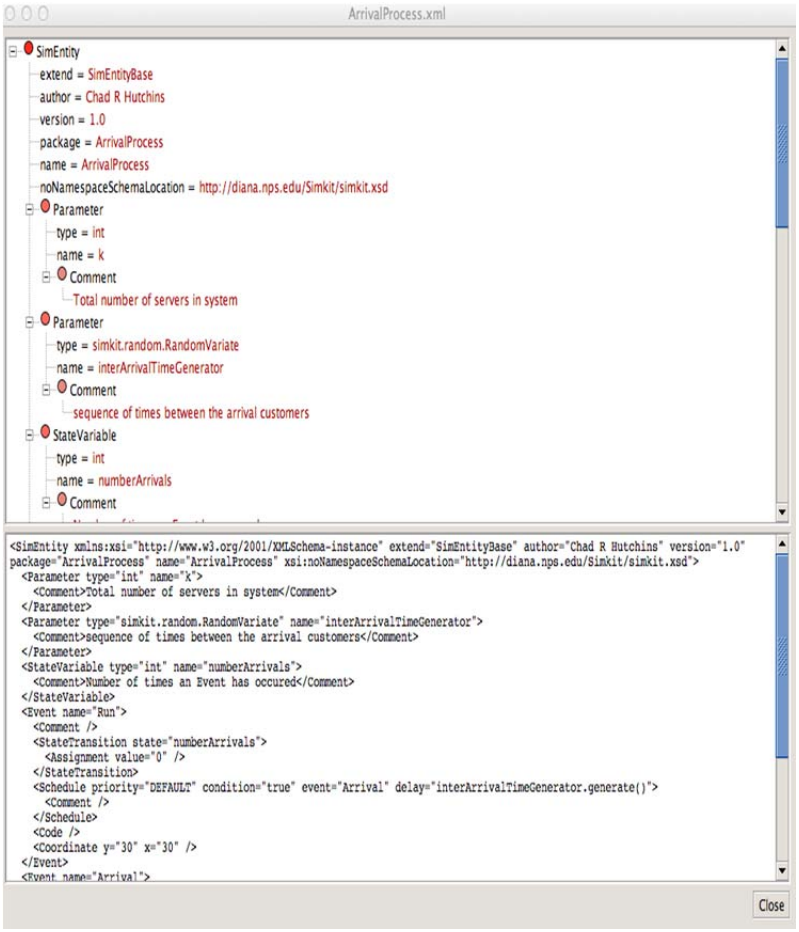


Figure 7. Viskit XML output of an ArrivalProcess. Viskit displays the XML in two views, a tree graph and standard XML format.

Figure 8 shows the product of this powerful feature (Buss, n.d.).

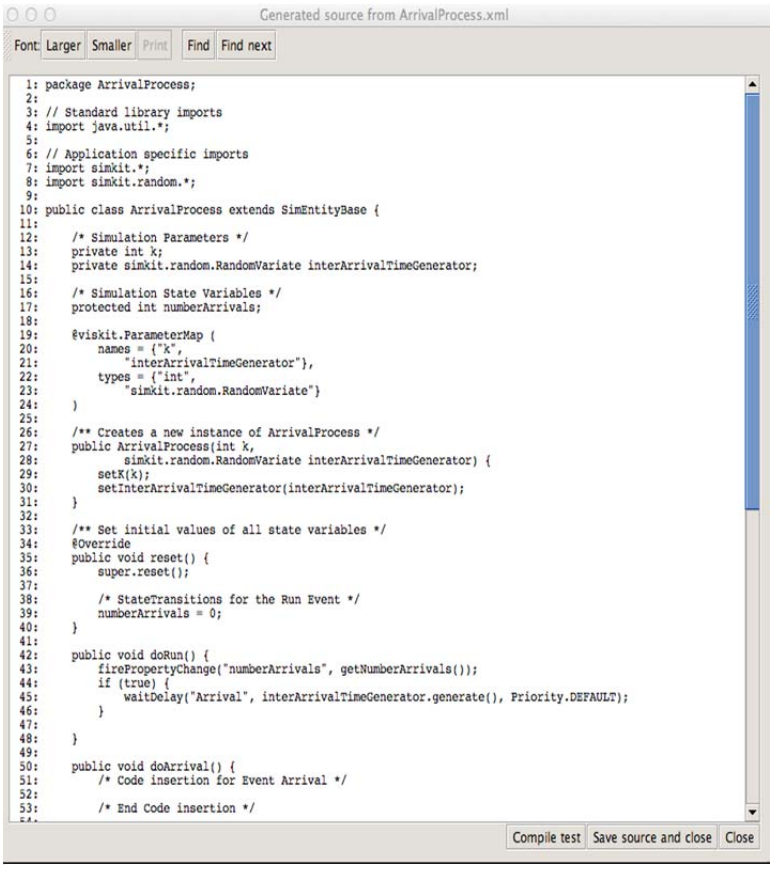


Figure 8. Viskit Java source code of an ArrivalProcess autogenerated from XML.

Viskit is still a work in progress and has the potential to be a powerful tool for military analysts and decision makers. Further programmer labor is needed to finish this effort. Sadly, an adequate sponsor has not been made aware of how powerful rapid modeling, without the use of computer programing skills can be to future military systems analysis. Fortunately, many features of Viskit are already fully functional and (as shown in several screen shots) were helpful in designing and documenting the event-graph models needed for this thesis. The corresponding auto-generated source code was also helpful for debugging and improving the human-authored source code.

C. VISUALIZATION

Visualization plays an important part in combat simulations, especially with helping leaders understand the problem and results. The phrase “a picture is worth a thousand words,” is quite true when the results of a simulation can be visualized in a simple and logical manner. Visualization can be as simple as a graph or as complex as 3D models interacting in a virtual environment. The key is to utilize the visualization tool that best expresses the simulation and supports the analysis in a manner that helps lead to confident decisions by decision makers. This thesis describes various methods for visualizing discrete event simulations, and this section presents the overview of the technologies. Chapter V shows the implementations of this thesis.

1. X3D–Edit

X3D–Edit is an authoring tool for X3D graphics. It is an open–source Java and XML program leveraging the Netbeans platform. X3D–Edit can launch X3D scenes for rendering in any X3D compliant 3D browser, including Xj3D, a Java-based 3D browser for VRML 97 and X3D authored scenes (X3D–Edit, 2013). Figure 9 shows Xj3D embedded into the X3D–Edit GUI. Recently the developers of X3D-Edit added functionality that allows users to create, edit, and validate KML. Chapter V describes how the simulations in this thesis utilize X3D–Edit to visualize KML.

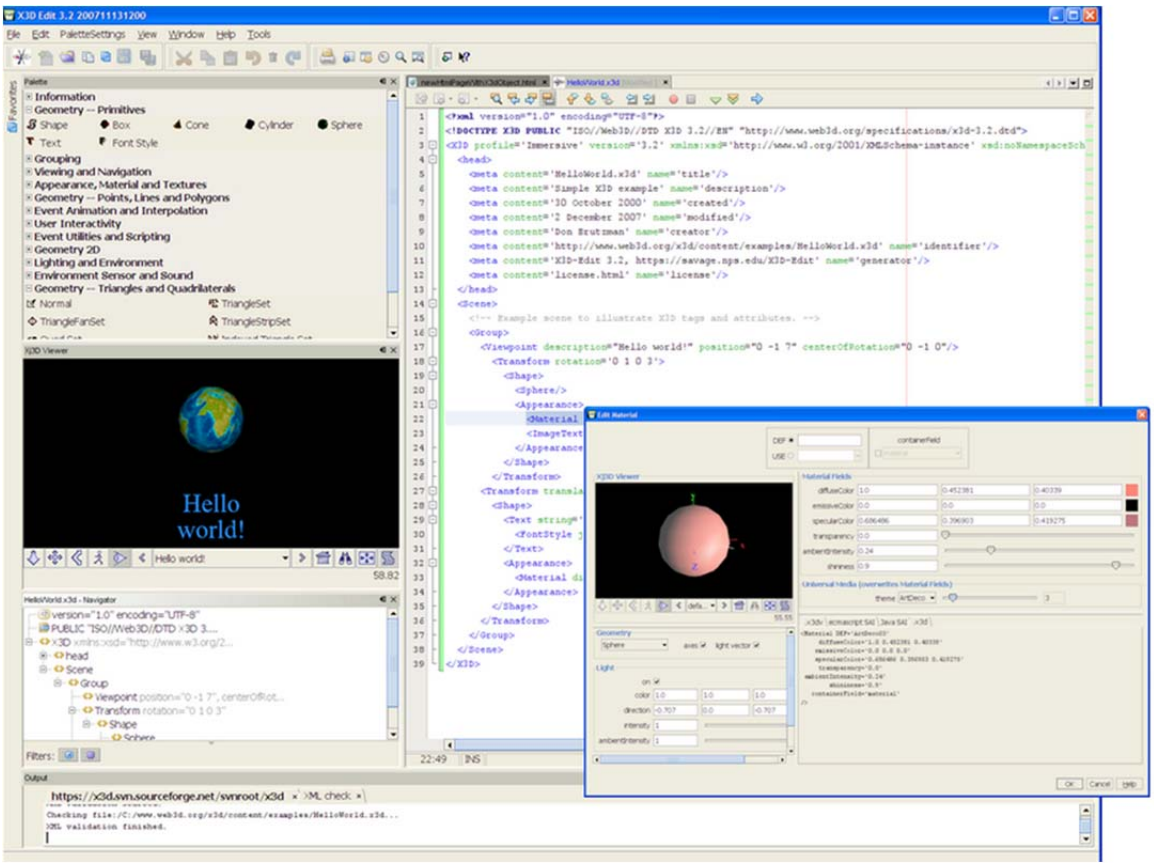


Figure 9. A screen snapshot of X3D-Edit with Xj3D browser displaying Hello World scene (From X3D-Edit Home Page).

2. Keyhole Markup Language (KML)

KML is XML based markup language that displays information in geographic applications, such as Google Earth. KML is a rather simple language to read, as seen in the code snippet below, and it is relatively easy to master the basics (Wernecke, 2009). The following example KML code shows a simple placemark of a known pirate camp in Somalia, Eyl.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2">
   <Placemark>
        <name>Pirate Camp Eyl</name>
        <description>Simple Placemark example of the location of the city Eyl,
        which is  known pirate camp</description>
        <Point>
               <coordinates>49.85000,7.76575</coordinates>
        </Point>
   </Placemark>
</kml>
```

The main appeal of KML for this thesis is the ability to create and view KML within the NMCI network. KML can be written in a simple text editor or a more capable editor (such as X3D-Edit). Google Earth is an approved application on NMCI networks and KML can also be run inside a web browser. The value of this approach is great and there are numerous potential applications for KML on a ship or another station within an NMCI network. There is more information on KML in the AgentC project. Chapter V demonstrates how KML was used to visualize simulation data in this thesis.

3. OpenMap™, OpenStreetMap and OpenSeaMap

OpenMap™, OpenStreetMap, and OpenSeaMap are all Java-based GIS systems that are also other alternatives for visualizing and analyzing simulations. Both are open source and provide unique capabilities for simulation and analysis. They are more complex to utilize; one has to create layer files and implement a link between the simulation code and layer file. However, they are practical and since both are open–source it makes access to the source code and development easier. OpenMap™ and OpenSeaMap are ongoing projects and both have a wealth of information on their websites: http://OpenMaptm.bbn.com, http://www.openstreetmap.org, and http://www.openseamap.org.

4. JAVA Swing Graphical User Interface (GUI)

Simkit leverages the UI windowing functionality of Java Swing in its framework. Java Swing is a simple choice for basic simulation runs or troubleshooting interactions of entities. It is relatively easily programmed and is well documented. One can easily take a simple scenario, such as Figure 4, and turn it into a more aesthetically pleasing scenario, as seen in Figure 10, with a couple lines of code that adds a background image. An unfortunate limitation of this approach, at least so far, is the need to use Cartesian X-Y coordinates rather than geospatial latitude/longitude coordinates.

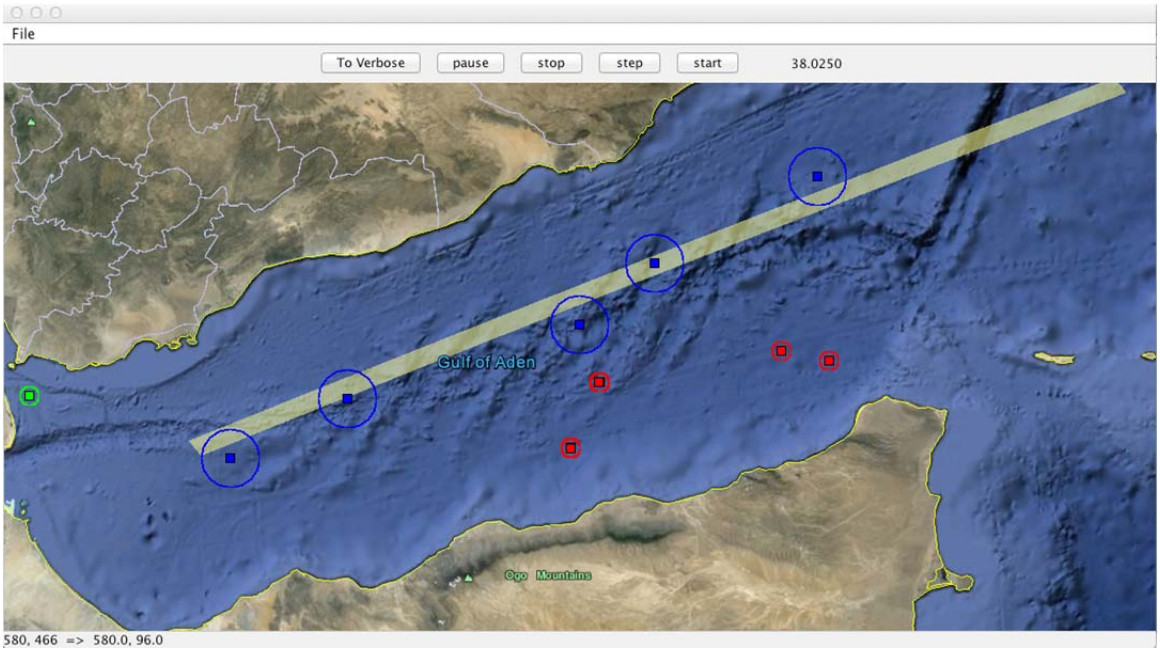


Figure 10. Java Swing functionality of Simkit depicting pirates in the Gulf of Aden and Navy ships patrolling the IRTC, using a Google Earth image as background.

D. PREVIOUS RESEARCH USING DES/SIMKIT MODELING

Many outstanding theses have emerged from NPS that utilized DES and Simkit. A simple search in the NPS library's Calhoun database or through DTIC reveals all of them. The following theses were influential to the work in this thesis.

1. Viskit Modeling of ANTI–TERRORISM/FORCE PROTECTION (AT/FP)

Harney (2003) and Sullivan (2006) laid the foundation for how AT/FP measures can be analyzed and visualized in order to provide surface vessels with a better way to train and maintain robust security. Harney (2003) produced the framework, including 3D visualization. Sullivan (2006) adds to the work of Harney and the simulation and analysis capability using DES and Viskit. Sullivan (2006) shows how large–scale scenarios can be easily managed, simulated, and analyzed in Viskit and visualized in 3D using X3D.

2. Simkit and GIS visualization

Mack (2000) uses the output of Simkit models to run in OpenMap™. It demonstrates how to use OpenMap™ layers to execute simulation code. The work of Mack (2000) was also used at the Turkish Naval Academy and published in Gurat (2010). This publication demonstrates a small–scale naval simulation using Simkit and OpenMap™. Both publications offer a great deal of information for getting a Simkit model running in OpenMap™. More detail is provided in Chapter IV.

Seguin (2007) creates a simulation that analyzes the capabilities and effectiveness of a Seadiver Unmanned Underwater Vehicle (UUV) utilizing Simkit, Viskit, and the Autonomous Unmanned Vehicle (AUV) workbench. The AUV workbench allows for physics-based models to perform mission rehearsals and real-time task level contols for robot missions with X3D (https://savage.nps.edu/AuvWorkbench) .

E. MODELING AND SIMULATING MARITIME PIRACY

The maritime community and international navies are increasingly utilizing modeling and simulation technologies. There has been some significant M&S research conducted on piracy around the Horn of Africa. As budgets get tighter and scrutiny grows by those who believe piracy is suppressed around the Horn of Africa (HOA), M&S will become more heavily relied on to assist in planning for shipping companies and military combatant commanders. The following are some of the most influential research initiatives in the area to date.

1. Agent Technology Center's AgentC Project

The Agent Technology Center (ATC) located at the Czech Technical University in Prague is a research center devoted to research in agent–based computing, multi–agent systems, and agent technologies (http://agents.felk.cvut.cz). While ATC has numerous exceptional projects and areas of research this thesis is interested in their AgentC project. The AgentC project is sponsored by the Office of Naval Research (ONR) and explores how multi–agent systems can be utilized to improve maritime security, in particularly maritime piracy. The basic principal of the research is to "develop an integrated set of algorithmic techniques for maximizing transit security given the limited number protection resources available." The project consists of a simulation engine that receives information from real–world systems and allows for visualization via Google Earth, as seen in Figure 11 (http://agents.felk.cvut.cz/projects/agentc). The research has produced stellar results in three areas of research:

(1) Data integration and analysis: a data–based piracy risk model and a probabilistic modeling of vessel trajectories have been developed.

(2) Computational modeling and simulation: a global merchant shipping model, utility based model of piracy, and an integrated model of a maritime transportation system with piracy has been produced.

(3) Computational optimization and planning: a group transit timetable optimization method, dynamic on–demand group transit scheme, traffic–coverage maximizing patrol deployment, game–theoretically optimum policies for mobile patrols and an optimum randomized transit routing have been developed (Jakob, Vanek, Hrstka, Bosansky, & Pechoucek, 2011).

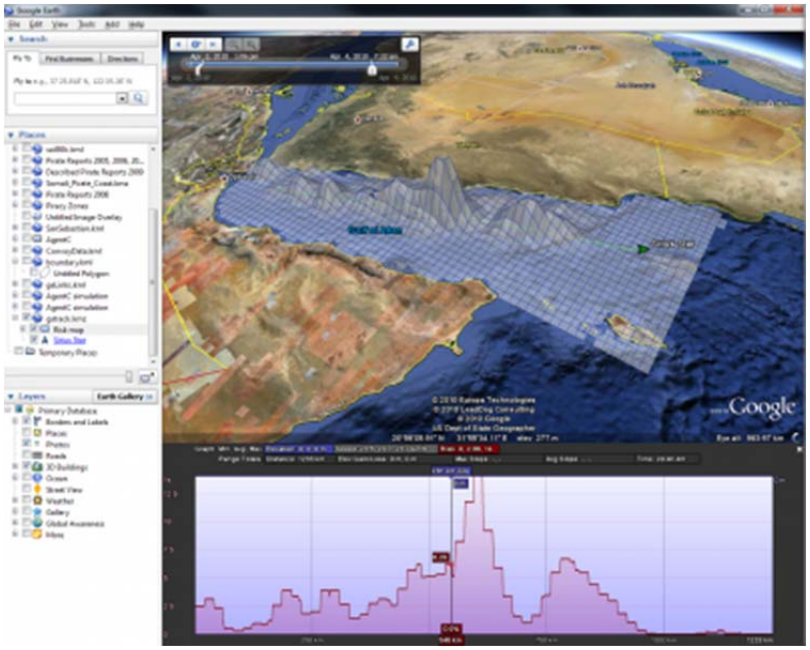


Figure 11. AgentC Google Earth visualization of risk modeling. (From Agent Technology Center's AgentC website, March 15, 2013)

The faculty and researchers at ATC have published numerous reports and publications outlining their work and success. The year–end reports are detailed and are a great resource for obtaining the latest efforts and on–going work. It is beyond the scope of this thesis to include, but all publications can be found on their website: http://agents.felk.cvut.cz/projects/agentc.

The author of this thesis considers the work being done at ATC to be the best in the field for piracy and other research. There has been quite a bit of collaboration between the author and researchers at ATC. ATC has also been collaborating with the developers of Pirate Attack Risk Surface (PARS) at the Naval Research Laboratory (NRL); this research is discussed in the next section. Currently efforts are being made to include the work from the AgentC project into the current U.S. Navy operational model, PARS.

2. Piracy Attack Risk Surface (PARS) Model

The research leading the way for PARS was called Piracy Performance Surface (PPS) model. Naval Oceanographic Command (NAVO) was directed to research piracy

by the current Oceanographer and Navigator of the Navy, Rear Admiral Titley, just days after the Maersk Alabama pirate incident occurred in 2009 (http://topics.cnn.com/topics/maersk_alabama). It was obvious at the time that weather around the HOA, in particular, two distinct monsoon seasons was a major factor in pirate success. The purpose of PPS was to produce a tool for navies and merchants to determine which areas were more susceptible to pirate attack. The model uses environmental data and historic attack data, weights each of them and displays the data on a color–coded map, as seen in Figure 12 (Slootmaker, 2011).

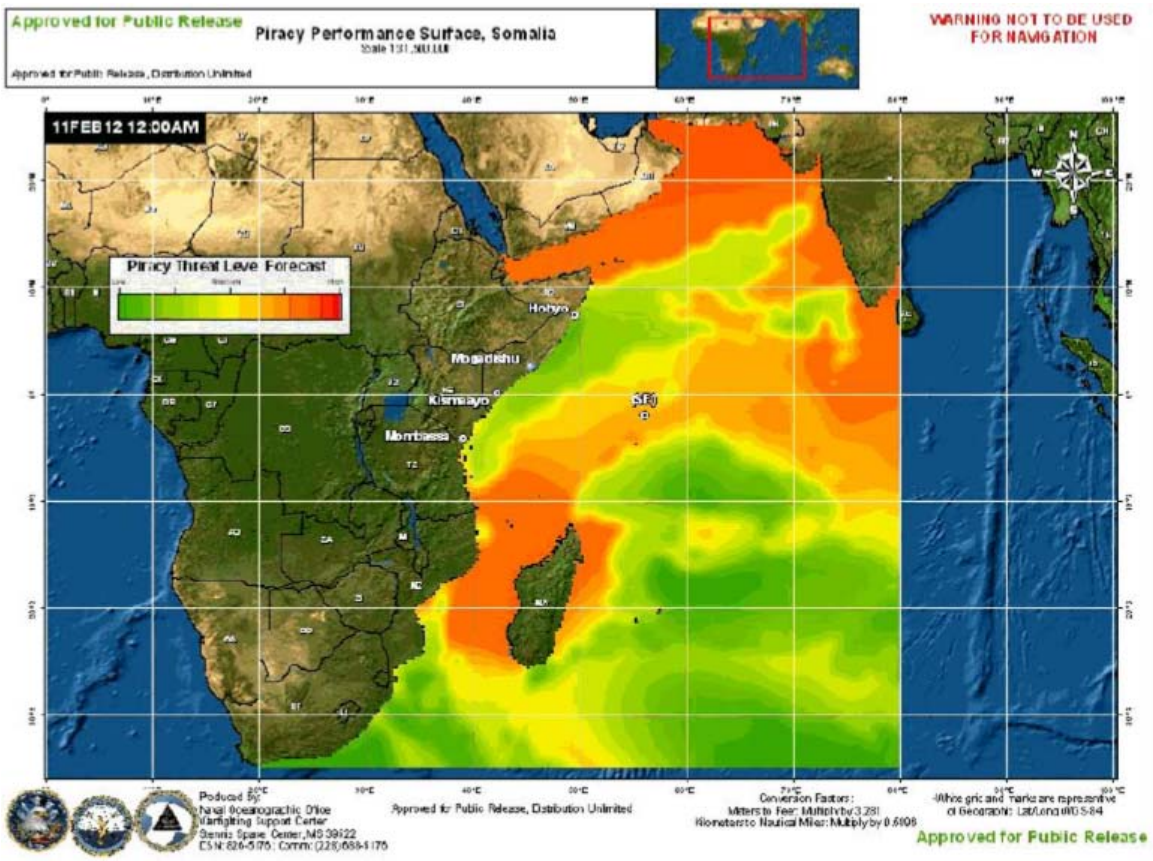


Figure 12. Visual display of the Piracy Performance Surface model on 11February2012 (From ONI Piracy Analysis and Warning Weekly (PAWW) report from 02 – 08 February 2012.)

PPS had great initial success, but needed a more advanced model in order to provide more accurate predictive power. The Naval Meteorology and Oceanography

Command (CNMOC) decided to produce a more advanced model, was called Next Generation Piracy Performance Surface Model (PPSN). To accomplish this CNMOC asked Dr. Jim Hansen at the Naval Research Laboratory in Monterey, CA to develop this new model. The PPSN is a stochastic Monte Carlo forecasting model with probabilistic weighing factors that is programmed in Python. The main functionality of PPSN included simulated pirate behavior, pirate knowledge about environmental conditions, a time-integrated environment with recurring pirate CONOPS distributions to produce relative forecast of pirate presence, and operator inputs for observed pirate locations, pirate camps, and length of time pirate can operator. The PPSN is one of the first models to combine real–time METOC and INTEL into an operational model. LT Leslie Slootmaker performed further work on the PPSN model in her 2011 Naval Postgraduate School thesis (Slootmaker, 2011). She was able to utilize design of experiments (DOE) to identify key parameters that affect the PPSN output, as well as, some optimization for memory and run–time requirements.

The PPSN model has recently changed its name to PARS and is currently an operational model that assists commanders of counter–piracy forces and units conducting counter–piracy operations in the Gulf of Aden and Indian Ocean. PARS is used by Combined Maritime Forces, European Union's (EU) Operation Atalanta, and North Atlantic Treaty Organization's (NATO) Operation Ocean Shield (Slootmaker, 2011). PARS is continually being improved and recently just passed its Verification, Validation, and Accreditation process (VV&A) (J. Hansen, personal communication, August 23, 2012). PARS is an excellent example of how valuable modeling and simulation can be to maritime security; it has been a true benefit to the fight against piracy, in both operational effectiveness and cost effectiveness.

3. Piracy Asymmetric Naval Operations Patterns Modeling for Education and Analysis (PANOPEA) Project by Simulation Team

The Simulation Team is a network of international institutions involved in M&S. They have been involved in numerous research projects and efforts scaling a broad range of interests, from business, health care, energy, telecommunications, homeland security, military, and many more (http://www.simulationteam.com). The PANOPEA project is a

discrete event simulator that is integrated with another Simulation Team project, Intelligent Agent Simulation Computer Generated Force (IA–CGF). PANOPEA models pirate activity around the Horn of Africa in an effort to evaluate various Network Centric Command and Control Maturity Models (NEC C2 M2). PANOPEA provides valuable insight on the benefit of having a robust communication network that allows for rapid information sharing during counter-piracy operations (Bruzzone, Tremori, and Merkuryev, 2011). Further research is needed to determine if such a robust network can feasibly be utilized by coalition forces. Research efforts on C2 Maturity models are ongoing by the PANOPEA researchers.

4. Naval Postgraduate School (NPS) Research on Somali Piracy

Research has also been accomplished on the subject of Somali piracy at NPS. The Joint Campaign Analysis (JCA) course, OA 4602, has produced two highly significant pieces of analysis on Somali piracy. In 2009, a team of students, two from the U.S. and one from Turkey, performed an analysis on the current state of piracy and made two foresighted recommendations: change the group transit schedule for the IRTC and for ships to defend themselves with armed guards (Bloye, Yildiz, & Scherer, 2009). The first recommendation was quickly acted upon by the EU. The second took some time to become politically popular, but in 2011 armed guards became heavily relied upon and have drastically reduced the amount of successful attacks around the Horn of Africa. More recently, analysis from the JCA class by LCDR William Major, suggested that ships with self–protection were more effective at thwarting pirates than U.S. Naval patrols (Major et al., 2012). The JCA course is a true prize for the school, the students, and sponsoring commands. Students from all services, including internationals, are given current real–world problems to analyze using the tools they have acquired thus far in their studies. Each quarter a new problem or set of problems are posed by different military commands. At the course conclusion the analysis and the recommendations are sent directly to the command where the question(s) originated for insight and consideration. Most quarters, students are able to accomplish such superb analysis that they invited to publish their work in peer reviewed journals such as PROCEEDINGS

(http://www.usni.org/magazines/proceedings), INFORMS (https://www.informs.org), or PHYLANX (http://www.mors.org).

There have been 13 graduate level theses conducted on Somali piracy at NPS since 2009, including the Slootmaker thesis that was discussed previously. There is a broad range of research areas:

- "Stopping Piracy: Refocusing on Land-based Governanc," June 2012, by Fredik Borchgrevink, http://hdl.handle.net/10945/7310.
- "Case Study of European Union Antipiracy Operation Naval Force Somalia Successes, Failures and Lessons Learned for the Hellenic Navy," September 2012, by Evangelos Soufis, http://hdl.handle.net/10945/17461.
- "Piracy in the Horn of Africa the Role of Somalia's Fishermen," December 2011, by Emmanuel Sone, http://hdl.handle.net/10945/4989.
- "Counter-piracy escort operations in the Gulf of Aden," June 2011, by Thomas Tsilis, http://hdl.handle.net/10945/5633.
- "Countering Piracy with the Next Generation Piracy Performance Surface Model," March 2011, by Leslie Slootmaker, http://hdl.handle.net/10945/5747.
- "Capacity building as an answer to piracy in the Horn of Africa," December 2010, by Loannis Nellas, http://hdl.handle.net/10945/5095.
- "Piracy and its Impact on the Economy," December 2010, by Rami Islam, http://hdl.handle.net/10945/5063.
- "Trading nets for guns the impact of illegal fishing on piracy in Somalia," September 2010, by Aaron Arky, http://hdl.handle.net/10945/5115.
- "Decreasing variance in response time to singular incidents of piracy in the horn of Africa area of operation," June 2010, by Christopher Descovich, http://hdl.handle.net/10945/5258.

- "Modern piracy and regional security cooperation in the maritime domain the Middle East and Southeast Asia," March 2010, by Michael King, http://hdl.handle.net/10945/5367.
- "Piracy in the Horn of Africa a Comparative Study with Southeast Asia," December 2009, by Stephen Riggs, http://hdl.handle.net/10945/4373.
- "Counter piracy a repeated game with asymmetric information," September 2009, by Christopher Marsh, http://hdl.handle.net/10945/4542.
- "Disrupting Somali Piracy Via Trust and Influence Operations," June 2009, by Robert Bair, http://hdl.handle.net/10945/4703.

F. SUMMARY

This chapter familiarized the reader with all the technologies utilized in this thesis in order to allow for a better understanding of the methodology utilized, especially in DES with Simkit and visualization. The chapter also highlighted some recent research conducted on Somali piracy, including theses and other institutional research projects.

THIS PAGE INTENTIONALLY LEFT BLANK

III. CROWD-SOURCING WITH MASSIVE MULTIPLAYER ONLINE WAR GAME LEVERAGING THE INTERNET (MMOWGLI)

"One thing a person cannot do, no matter how rigorous his analysis or heroic his imagination, is to draw up a list of things that would never occur to him." [3]

—Thomas Schelling

A. INTRODUCTION

Crowd-sourcing and serious games are being used by some of the most successful corporations in the world (http://www.iftf.org/iftf–you/clients–sponsors). Serious games are games that are developed for a purpose more than just entertainment, such as learning, problem solving, simulation, training, collaboration, networking, etc (http://www.seriousgamesinstitute.co.uk/about.aspx?section=18&item=41&category=16. The DoD, especially the Army, utilizes serious games frequently for training. However, Jensen and Cook (2010) suggest that these serious games can possibly play a bigger role in DoD decision–making and strategic planning. The traditional methods of decision–making and strategic planning indeed work, however, Jensen & Cook (2010) argue that there is a need to expand the participants involved and utilize a broader knowledge base.

This chapter discusses how the MMOWGLI platform uses crowd-sourcing as a means to collect ideas and information, then collaboratively produce action plans for extremely complex and wicked problems.

B. WHAT IS MMOWGLI?

MMOWGLI is message–based serious game that allows players to work together through idea generation, brainstorming, and action plan development in order to encourage innovative solutions to extremely complex and wicked problems. A wicked problem as defined by Camillus (2008) is a problem that cannot be solved by traditional

[3] From "Gaming for innovation: An open source approach to generating insight" by G. Jensen and .M. Cook, 2010, ONR Director of Innovation Newsletter, Volume 5, pp 8 – 10.

processes. He describes the problem as “tough to describe and doesn’t have a right answer.” Roberts (2000) describes a wicked problem as a problem with no consensus that is merely defined from the point-of-view of the analyst. She also describes that a wicked problem has many stakeholders from a very diverse group, all of which have to continually work together to define the continuously changing constraints of the problem (Roberts, 2000). The game seeks to solve these wicked problems by gathering ideas from all persons of an organization without regard for rank or seniority (MMOWGLI Players Portal, n.d.). The idea of MMOWGLI came from Dr. Garth Jensen, who at the time was the Director of Innovation at the Caderock Division, Naval Surface Warfare Center. His original vision was aimed at bridging the disconnect between technologists and warfighters. To turn his vision into reality Dr. Jensen led a team comprised of the ONR, NPS, and The Institute for the Future (IFTF) to form MMOWGLI (Ohab, 2011). The MMOWGLI Game design is mainly architected by IFTF and implemented by NPS MOVES (MMOWGLI Players Portal, n.d.).

C. TECHNICAL OVERVIEW

MMOWGLI is an open–source serious game platform that utilizes some of the latest web–based technologies. MMOWGLI had some significant technological hurdles to overcome in order to launch. The biggest hurdle was how to allow NMCI users to participate without installing software on a government computer. The solution to working within the NMCI is to build an interactive game that uses an approved web browser and works over Transmission Control Protocal (TCP) port 80, or Hypertext Transfer Protocol (HTTP). The development team used HTML and Javascript based content, with help from tools such as the Java Vaadin GUI, Java Google Web Toolkit (GWT), and Tomcat server technology, to name a few (Brutzman, 2011). There are plenty of references for all these tools available online or in books, but their specifics are beyond the scope of this thesis. The complete list of software, operating instructions, and software details are maintained on the MMOWGLI portal.

D. MMOWGLI GAME HISTORY

1. Piracy MMOWGLI 2011–Open to Public

The initial MMOWGLI game aimed to test the MMOWGLI idea and technology on one of the Navy's most wicked and predominately unclassified problems, Somalia Piracy. It was open to military, government employees, and civilians. The 2011 piracy game had three iterations and consisted of 2,165 players, 14,978 idea cards, and 68 action plans, additional game statistics can be viewed in Table 1. Further information, including HTML pages of all action plans and idea cards for piracy MMOWGLI 2011 can be found at:

- https://portal.mmowgli.nps.edu
- select the Piracy MMOWGLI Games link,
- in the table of contents select Piracy MMOWGLI Game 2011.1.

There is also more detail on a few of the Action Plans in Section IV of this thesis.

	Piracy 2011.1 (Move 1-2-3)	Piracy 2011.2 (Move N-Alfa)	Piracy 2011.3 (Move N-Bravo)	Total 2011
Dates	31-May-3 June, 21-23 June, 5-8 July	7-9 November	10-13 November	-
Days duration	11	3	3.5	18
Signups	16,000	31,000	31,000	31,000
Invitees	2,200	7,500	7,500	15,000
Players	832	920	413	~2,100
Signup %	30.7%	12.3%	5.5%	14%
# Idea Cards	5142	5608	4228	14,978
# Action Plans	28	18	22	68
# Game Master Accounts	29	50	46	~60

Table 1. Game statistics for the Piracy MMOWGLI 2011 game that was open to the public. Retrieved from MMOWGLI Game for Crowd –Sourcing Problem (PPT) Solutions by Don Brutzman

2. Piracy MMOWGLI 2012–Maritime Experts and Stakeholders Only

Throughout the 2011 MMOWGLI game it became apparent to those at Oceans Beyond Piracy (OBP) and those at NPS working on MMOWGLI and researching Somali piracy that MMOWGLI could be a major asset for the policy makers and strategic planners concerned with Somali piracy. The game was organized around OBP's Independent assessment and asked players to brainstorm ideas to improve each line of effort. Figure 13 shows the lines of effort in the Independent Assessment. The action plans developed by this group of experts during the "Naval Operations" week of MMOWGLI are used in this thesis to analyze and assess.

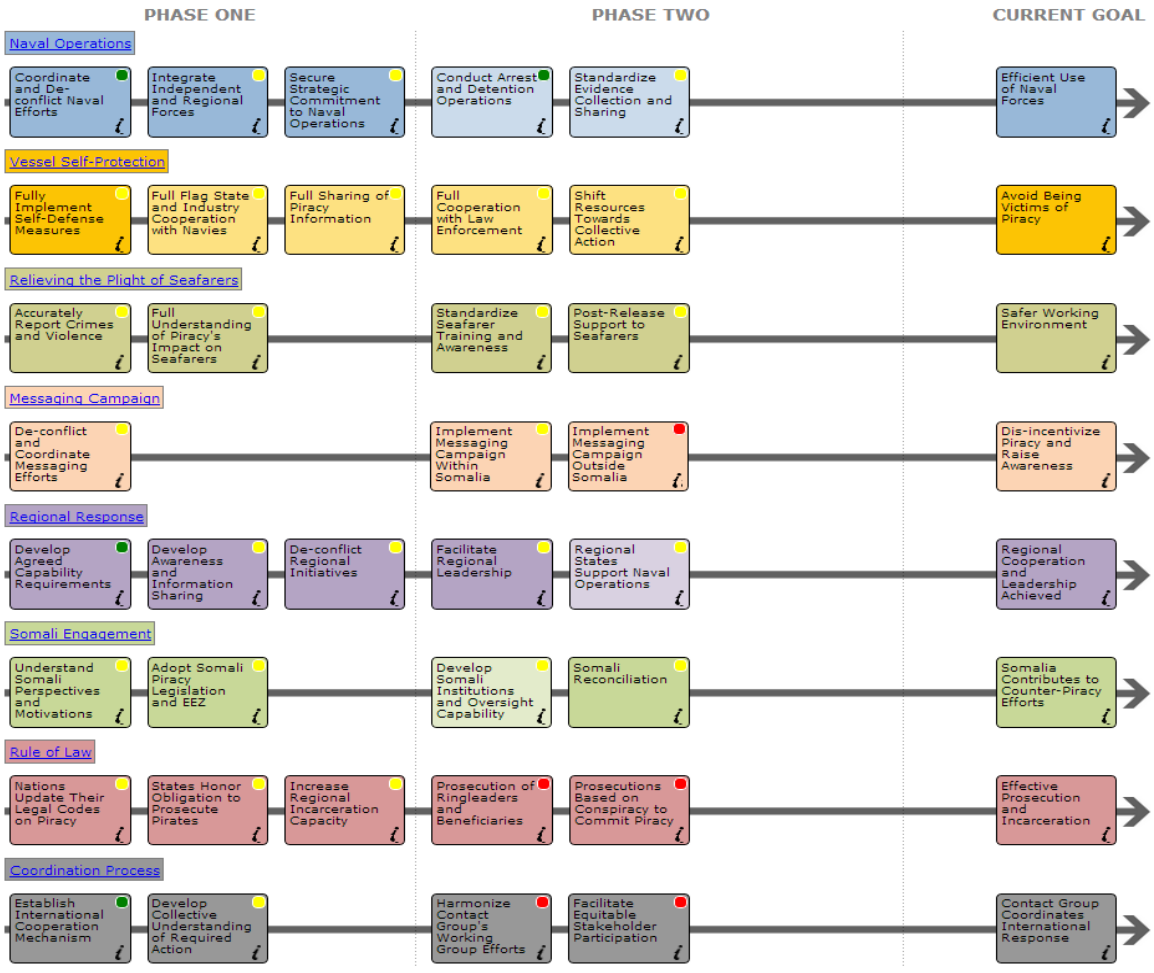


Figure 13. Oceans Beyond Piracy's Independent Assessment (From Oceans Beyond Piracy website, February 15, 2013).

Further information, including HTML pages of all action plans and idea cards for piracy MMOWGLI 2012 can be found at:

- https://portal.mmowgli.nps.edu
- select the Piracy MMOWGLI Games link,
- in the table of contents select Piracy MMOWGLI Game 2012 - 2013.

There is also more detail on the analysis of the top action plans in Section IV of this thesis.

The Piracy MMOWGLI game caught attention internationally among maritime progessional. Dr. Don Brutzman was invited to speak and hold a workshop at the 16th Hanson Wade Combating Piracy 22 – 26 October 2012 in London . Hanson Wade is a company who strives to progress organizations and businesses through conferences and workshops, which bring together top leaders and thinkers in their respected domain (http://hansonwade.com/corporate/about-us). The Combating Piracy series of conferences brings together maritime professionals, including international navies, international governments, including Somali government officials, maritime shipping companies the maritime security industry, and non-profit organizations (http://combating-piracy.com).

The initial effort between NPS and OBP never fully developed fully, as originally planned, but the individuals involved with Piracy MMOWGLI plan to continue further work on the effort. There are plans being developed to continue engaging the maritime community and developing ideas on how navies, policy makers, and industry should proceed in the fight against Somali piracy.

3. Energy MMOWGLI

Energy MMOWGLI was sponsored OPNAV N45 – Task Force Energy, the game was used MMOWGLI to gather ideas and action plans on how to secure the Navy's energy future. Energy MMOWGI produced 5,121 idea cards and 38 action plans, additional game statistics can be viewed in Table 2. Additional information on both the Energy MMOWGLI can be found at https://portal.mmowgli.nps.edu/energy and https://mmowgli.nps.edu/energy/reports.

4. EDGE Virtual Training Program (EVTP) MMOWGLI

The U.S. Department of Homeland Security (DHS) Department Science and Technology department conducted a game in order to develop a new partnership program with the U.S. Army on the EDGE Virtual Training Program (EVTP). This platform will eventually be used to train first responders. EVTP MMOWGLI produced 263 idea cards and 4 action plans, additional game statistics can be viewed in Table 2. More information can be found at: https://portal.mmowgli.nps.edu/evtp and https://mmowgli.nps.edu/evtp/reports.

	energyMMOWGLI	piracyMMOWGLI 2012	evtp: Edge Virtual Training Program
Dates	21-27 May 2012	18 June - present, ongoing	19-22 December
Days duration	5	Long-term	5
Signups	-	-	-
Invitees	797	200+	65
Players	561	115	65
Signup %	70.4%	Slow increase	100%
# Idea Cards	5121	432	263
# Action Plans	37	8	3
# Game Master Accounts	47	10	8

Table 2. Game statistics for the all the MMOWGLI games run in 2012. Retrieved from MMOWGLI Game for Crowd –Sourcing Problem Solutions (PPT) by Don Brutzman

5. Business Innovation Initiative (BII) MMOWGLI

The Navy acquisition community utilized MMOWGLI to explore how to best achieve the Navy's new Open System Architecture (OSA) strategy, called The Business Innovation Initiative (BII). This game was for navy personnel and contracting companies.

BII MMOWGLI produced 900 idea cards and 12 action plans. More information can be found at: https://portal.mmowgli.nps.edu/bii and https://mmowgli.nps.edu/bii/reports.

6. Electromagnetic Maneuver (EM2) MMOWGLI

EM2 MMOWGLI was sponsored by Naval Warfare Development Command (NWDC), ONR, and NPS to crowd-source ideas on how to innovate concept development and experimentation efforts for how the Navy should operate in the EM Environment. EM2 MMOWGLI was run for three weeks and produced 5,496 idea cards and 40 action plans. Additional information on EM2 MMOWGLI can be found at https://portal.mmowgli.nps.edu/em2 and https://mmowgli.nps.edu/em2/reports.

	bii Business Innovation Initiative	em2 Electro-magnetic Maneuver	ig NPS Inspector General	Totals 2012-2013
Dates	Round 1: 14-25 January	Rounds 1-3: 18-24 February 4-10 February 4-10 March	Round 1: 30 January – 1 February	-
Days duration	12	21	3	81
Signups	73	753	0	1900
Invitees	136	943+	1800	~4000
Players	90	578	70 + visitors	~1500
Signup %	66.2%	61.3%	4-6%	4%-70%
# Idea Cards	907	5624	521	12,868
# Action Plans	12	41	3	104
# Game Masters	29	50	9	~80

Table 3. Game statistics for the all the MMOWGLI games run in 2013, including totals for all games in 2012 and 2013. Retrieved from MMOWGLI Game for Crowd – Sourcing Problem Solutions (PPT) by Don Brutzman

E. MMOWGLI PORTAL

The developers of MMOWGLI implemented a portal in order to enable players to access information about the game, information on the current game topic, current news,

past research and publications on the current topic, and various other research tools to help make game play more valuable and informed. The portal was built using Liferay portal engine and allows for reference storage, blog pages, and other wiki pages. Figure 14 shows the main player's portal page for MMOWGLI.

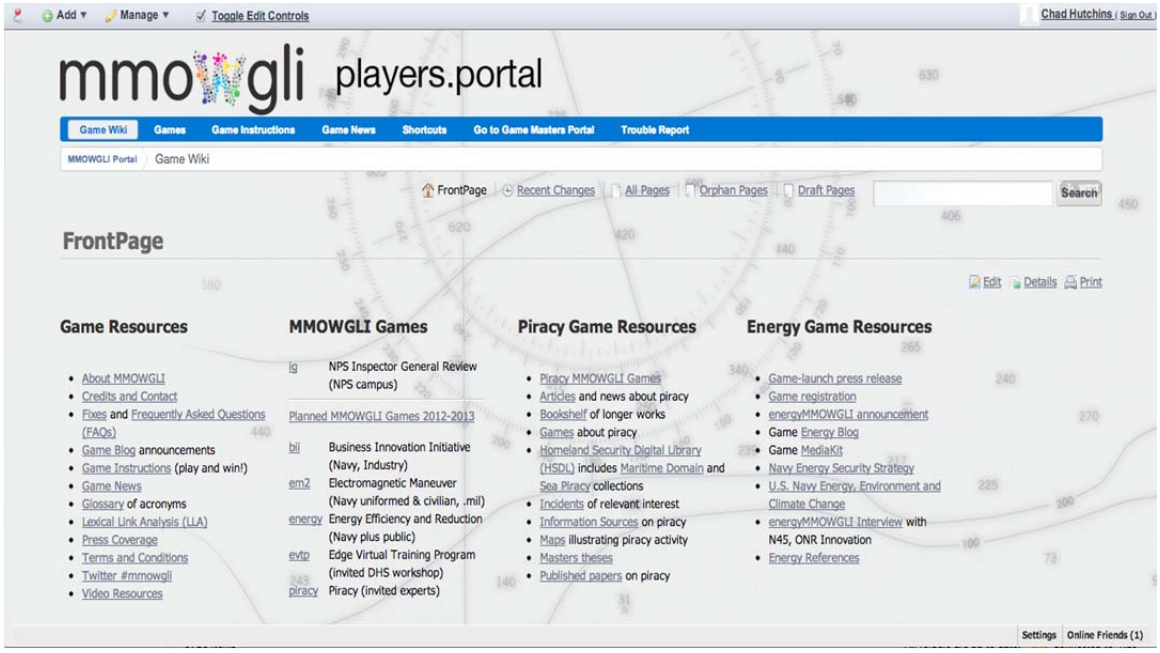


Figure 14. The MMOWGLI Portal Home Page is the home for all current and past MMOWGLI games. (From MMOWGLI Portal, February 4, 2013).

1. Piracy Portal

The piracy portal, seen in Figure 15, has greatly contributed to the success of piracy MMOWGLI. The portal enables quick access to research on piracy, relevant information sources, current news, and even the Homeland Security Digital Library (HSDL), which includes sources for maritime security and piracy. The portal also enables players to be able to access the idea cards and action plans from past piracy games (http://portal.mmowgli.nps.edu/piracy–welcome).

Figure 15. The MMOWGLI Piracy Portal Welcome Page is the start point for accessing Piracy MMOWGLI. (From MMOWGLI Portal, February 4, 2013).

F. SUMMARY

This chapter has described crowd-sourcing utilizing the MMOWGLI platform. Numerous MMOWGLI games have been run and many other possibilities exist for the military utilize MMOWGLI. Play the game, change the game!

THIS PAGE INTENTIONALLY LEFT BLANK

IV. DETAILED PROBLEM DESCRIPTION

"If I had an hour to save the world I would spend 59 minutes defining the problem and one minute finding the solutions."[4]

– Albert Einstein

A. INTRODUCTION

Piracy has been around for centuries and there are many lessons that strategists can utilize to help combat modern day piracy and future piracy. Modern day piracy is without a doubt a wicked problem, and although as of 2012 piracy has been drastically reduced around the Horn of Africa (HOA) there is still a need to analyze strategy for combating piracy. Whether it be another surge in Somali pirates, continued violence of West Africa piracy, or a rise in piracy in another part of the world, analyzing various strategy options can help rid the problem in a more cost effective and timely manner.

B. PIRACY PROBLEMS AND CHALLENGES

Throughout history there have been four requirements for maritime piracy to exist: (1) Non–existent or weak government on land, (2) Ungoverned territorial seas, (3) Access to shipping lanes, and (4) Access to boats, manpower, and arms (J. Kline, personal communication, 24 January 2011). The same is true for Somali piracy; Somalia does not have a functional government that can adequately govern and uphold the laws on land or on their territorial seas. Somalia is positioned on the busiest sea route in the world, including a major chokepoint at the Straits of Bab El Mandeb. The majority of people in Somali are poor, desperate for an opportunity, and highly susceptible to being coerced into piracy. Analyzing this historical correlation it is not difficult to see that the root causes of piracy are on land and major diplomatic and political objectives are needed to rectify the main problems. Clausewitz and Mahan would both argue the need for a military effort to engage piracy. Mahan said naval forces are what allow for sea trade (Mahan, 1918, p. 22). Clausewitz argues, military force is an instrument of policy

[4] From "Open Innovation and Crowdsourcing: Advice from Leaders Advice from Leading Experts", 2011, by Paul Sloane, p. 204.

(Clausewitz, 1984/ 1780–1831, pp. 87 & 605), and until sailors are not in danger and sea–lanes are safe, the international community needs to figure out how to use this instrument in a manner that is consistent with its policies.

In 2008, after a few high–value merchant vessels were hijacked off the coast of Somalia the international spotlight began to shine on the coasts of Somalia. NATO formed Operation OCEAN SHIELD, the EU formed Operation Atalanta, and in 2009 the Combined Maritime Force formed CTF–151 (Haywood & Spivak, pp. 50–51). Operation Ocean Shield's mission is to deter and disrupt piracy, protect merchant vessels, and provide security around the HOA (http://www.mc.nato.int/ops/Pages/OOS.aspx). Operation Atalanta's mission is to deter, prevent and repress acts of piracy. Operation Atalanta also protects the World Food Program shipping and the African Union Mission in Somalia (AMISOM) shipping (http://eunavfor.eu). CTF-151's mission, as discussed in Chapter I, is to deter, detect, and disrupt piracy (http://www.cusnc.navy.mil/cmf/151/index.html). There were also independent nations such as China, Russia, Iran, and Japan sending warships to the area to escort and patrol. This was the beginning of a military approach to suppress piracy. The "big three" have had numerous criticisms for not working together and not being under one central operational commander. They tried to circumvent some of the coordination issues with the creation of Shared Awareness and De–Confliction (SHADE), a group which attempted to bridge the gaps and share information and intelligence (Haywood & Spivak, pp. 51–52). The major issue is that all three operations have different mandates and defined missions, thus making it near impossible to organize a true central command. Clausewitz often reminded military and political leaders of the need to seek unity of command and unity of effort (Clausewitz, 1984/ 1780–1831, pp. 205 – 209).

Although the international community and its navies struggled to suppress piracy from 2008 – 2011, the year 2012 was a huge success in decreasing successful attacks and attempted attacks around the HOA. The use of armed security teams on board merchants, navies operating closer to the shores of Somalia, and other law-enforcement agencies tracking and targeting the financial flows of pirate financiers have all had a significant impact on the pirate business model. However, the shared counter-piracy mission is still

not accomplished. The non-government organization Oceans Beyond Piracy (OBP) has followed piracy more closely than any other organization and provided numerous detailed and highly utilized research efforts. Their continually updated Independent Assessment of the current state of piracy efforts show there is still quite a bit of work to be done (http://oceansbeyondpiracy.org/independent_assessment). Figure 13 in the previous chapter shows the lines of effort that OBP analyzes and their current status.

With the past struggles to suppress piracy and now the recent success in protecting the sea–lanes around the Horn of Africa, policy makers and strategist are left with the most challenging decisions: How will the international community proceed now that piracy is down? Will funding continue to be available to support a counter–piracy mission? Are international navies still needed? If so, how should we deploy navy fleet assets in order to match current policy? Does our current strategy match current policy? These questions and many others are what need to be discussed, analyzed, and agreed upon.

C. MODELING PIRACY AND COUNTER–PIRACY TACTICS

1. Data Limitations

Gathering data on Somali piracy is a difficult task. There are many variables, some of which are impossible to gather data on, so many assumptions have to be made. The data used for this thesis is all unclassified. Most of the data used for the models come from IMB data. Cyrus Moody, the Assistant Director at the IMB, graciously provided the author with all pirate incident data that IMB has record dating back to 2006. The author also relied heavily on his research from writing the U.S. Navy's unclassified TACBUL for counter–piracy, as well as, the numerous interviews he has conducted with Somali pirates. The members of the AgentC project at ATC also provided data on pirate attacks, "mother ship" movements, and merchant shipping. It is definitely difficult to gather all of the data on Somali piracy and this thesis does not claim to have it all. However, the data used for this thesis allows the author to feel confident that the processes and behaviors that occur during counter–piracy operations are captured in the models created.

2. MMOWGLI Action Plans

Although raw data can be hard to gather, it is highly beneficial to utilize a large diverse group to discuss new ideas and brainstorm methods on how to defeat piracy. After days of brainstorming ideas in the MMOWGLI platform, the major themes and highly debated topics that arose from the idea chains were formed into action plans. These action plans lay the foundation for how to solve the problem or a subset of the problem in the point of view of the authors of the action plan. As seen in Action Plan in Figure 16, the action plans give the Who, What, When, Why, and How to make the plan work. For this thesis the author selected the top three actions plans that showed the best potential for actually being implemented into naval strategy. These three action plans are measurable and they match current policy objectives. The three selected were transit lane operations, naval quarantine, and pirate camp operations. Each of these are described in depth in Chapter V.

Figure 16. Excerpt from example Action Plan #3 outlines a plan for enforcing the fishing zones around Somalia. (From Piracy MMOWGLI 2012 Action Plan #3).

D. SUMMARY

This chapter discussed the complexities of combating piracy and the difficult strategic decisions that still need to be made to ensure piracy around the Horn of Africa remains disrupted. Analyzing piracy can be difficult because data is hard to collect, but crowd-sourcing ideas and utilizing large groups of people to develop actions plans can assist in developing cohesive strategy options that can be rapidly modeled and analyzed.

THIS PAGE INTENTIONALLY LEFT BLANK

V. SIMULATION DESIGN AND MODELING

"All models are wrong, some are useful." [5]

–George Box

A. INTRODUCTION

Agent modeling has been a field of extensive research since the early 1990s, especially in the military. Most military agent systems are Discrete Time Simulation (DTS) based, also referred to as time step, rather than DES, or next-event based, (Alrowaei, 2011, p. 2). However, (Alrowaei, 2011) shows that there are many risks in using DTS if the modeler is not careful with the specified time step size, even at small time steps the analysis can be degraded (Alrowaei, 2011, pp. 244–247). This thesis utilizes a DES approach to agent based modeling, and shows that movement, sensing, and detecting is a practical and useful methodology for rapidly simulating and analyzing military applications. Alrowaei, (2011) did note that the DES approach, on average, did take more time in the coding phases of modeling (Alrowaei, 2011, pp. 244–245). However, utilizing Viskit would ensure a more rapid development of models with little to no coding. However, the Viskit code base needs further support in order to allow this methodology to be more widely used. This chapter explains the DES models used for this thesis, simulation design, and visualization implementation.

B. SIMULATION DESIGN

The simulations in this thesis are all agent-based with DES and implemented using Simkit. There are three main groups of entities modeled, pirates, navy ships, and merchant ships. Each of these groups are controlled by a Simkit Mover Manager, uniquely named, PirateMoverManager, NavyShipMoverManager, and MerchantShipMoverManager. The Mover Managers model all the logic for each entity and allow movement by scheduling "Move" events, as well as carry out entity specific

[5] From https://www.math.umass.edu/~jstauden/notes1114.pdf .

tasks, such as “Attack” or “Evade.” Each entity has a senor that is modeled by a Simkit CookieCutter Sensor. The CookieCutter Sensor has a specified range and detects any mover that enters the range. The Mover Managers and their sensors are then programmed into a Simkit assembly, as seen in Appendix L, and connected via listeners that allow interactions and detections. Using this listener pattern allows for statistics to be easily collected for the simulation analysis.

C. SIMKIT ENTITIES

1. Pirate Mover Manager

The PirateMoverManager class models the behavior of a Somali pirate. The pirate is given a pirate camp to start from and leaves the pirate camp at a specified interval by a pirate departure process. The pirate heads to a random point in either the Gulf of Aden or Indian Ocean, where it hunts for merchant vessels. If no merchant is found after all fuel and supplies are depleted the pirate returns to its pirate camp. If a merchant is located it makes a decision as to whether to attack the vessel or not. If the pirate makes the decision to attack the adjudicator will determine whether or not the pirate is successful, based on historical data and whether or not a navy vessel is within distance to disrupt the attack. If the pirate is successful it returns to the pirate camp with the merchant. If it is not successful it flees the area and continues searching for other merchants. If a pirate is detected by a navy vessel it stop and be boarded by the navy vessel. The navy either returns the pirate to the coast of Somalia or apprehends the pirate.

The above logic can be followed in the event graph depicted in Figure 17 and the java source code can be found in Appendix B.

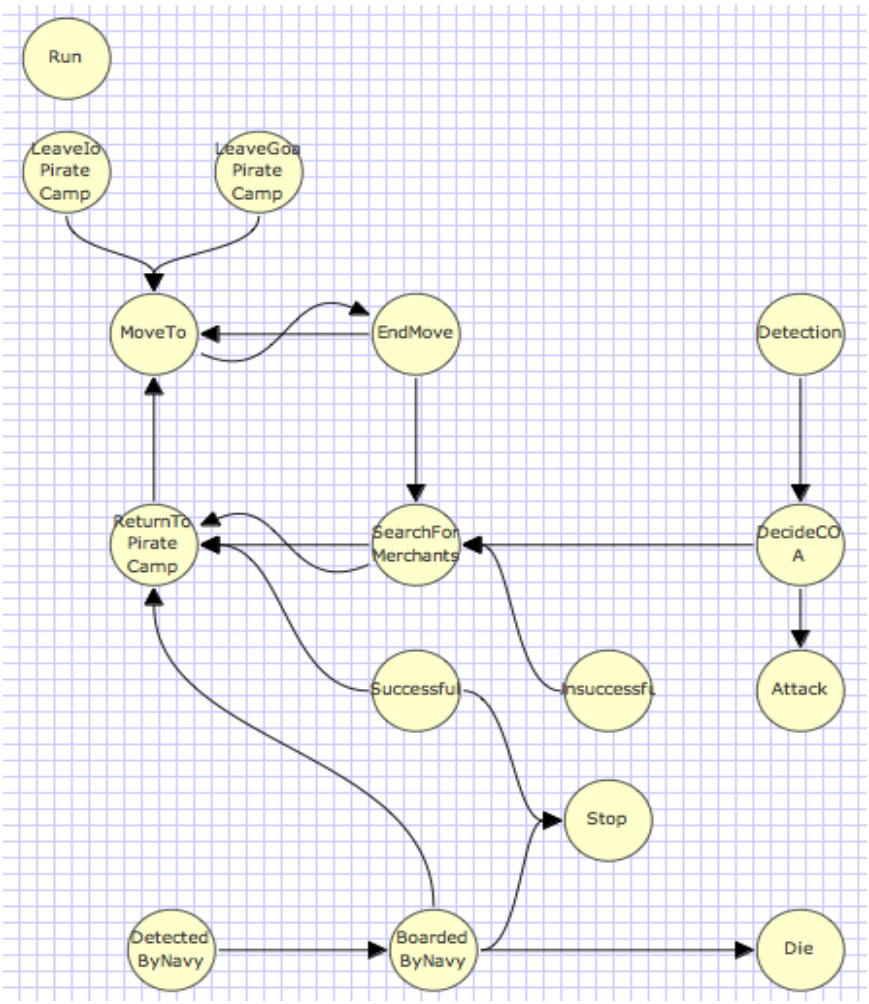


Figure 17. PirateMoverManager Viskit Event Graph shows the modeled behavior of a Somali pirate.

2. Navy Ship Mover Manager

The NavyMoverManager class models naval vessels on patrol. They are given a patrol box to patrol and patrols the box with a random search pattern. If a pirate is detected it signals the pirate and conducts a boarding. The pirate is returned to port if not in the act of attacking a merchant. But if the pirate is caught in the act of attacking the navy vessel detains the pirate. The navy vessels also receive distress calls from merchants. Once they get a distress call the closest vessel intercepts the merchant's location to search for pirates. It is assumed that navy vessels have helicopter capability,

but this is not explicitly modeled. However, it is taken into account when determining if the navy can respond to a distress call in a sufficient amount of time.

The above logic can be followed in the event graph depicted in Figure 18 and the java source code can be found in Appendix C.

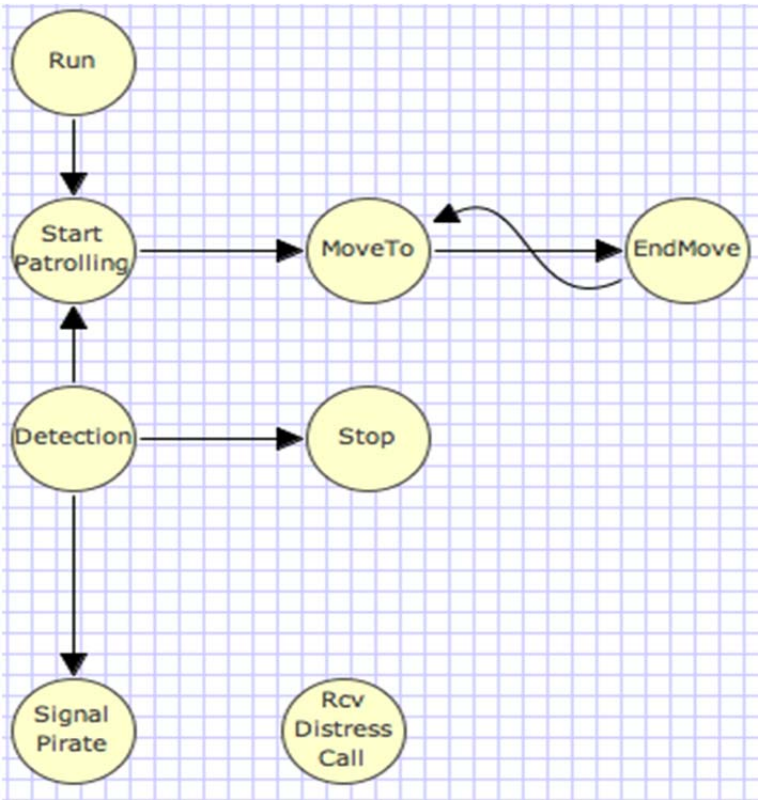


Figure 18. NavyMoverManager Viskit Event Graph shows the behavior modeled for a navy vessel conducting counter-piracy operations.

3. Merchant Ship Mover Manager

The MerchantMoverManager class is the simplest of the MoverManagers. A merchant is given a starting location and a path to its destination. The merchant proceeds at a specified speed from its starting location to the destination. It leaves its starting

location at specified intervals via a departure process. If the merchant detects a pirate vessel it radios the navy and attempt to evade the pirate attack. If hijacked it first stops, then be taken to the pirate camp.

The above logic can be followed in the event graph depicted in Figure 19 and the java source code can be found in Appendix D.

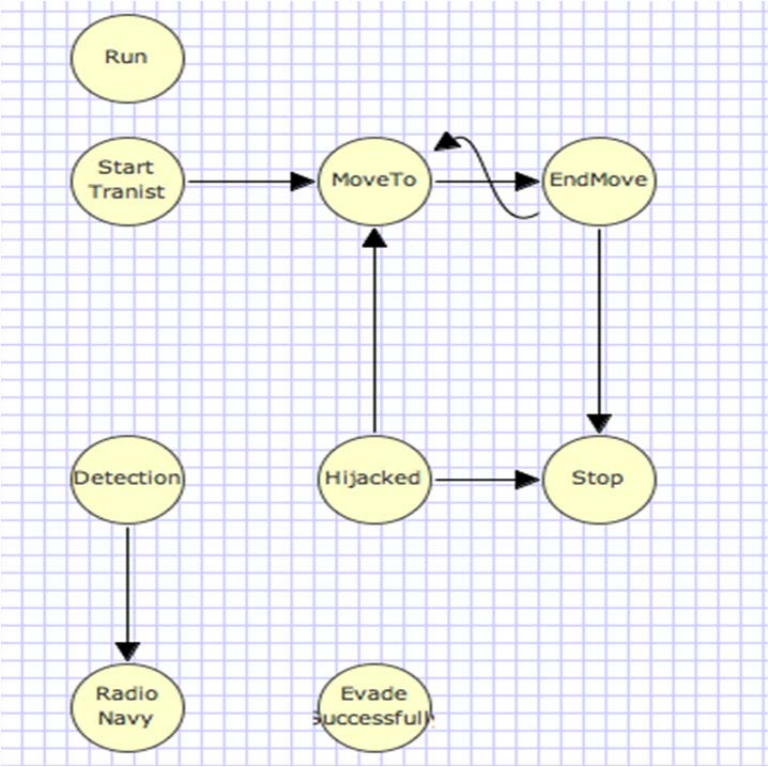


Figure 19. MerchantMoverManager Viskit Event Graph shows the modeled behavior of a merchant vessel transiting from its port of orgin to a its destination.

4. Adjudicator

The Adjudicator class acts as the referee between the entities. It processes the pirate attacks and determines whether or not the attack was successful. Once this determination is made it schedules the appropriate events for the pirate and merchant.

D. SIMKIT PROCESSES

1. Pirate Departure Processes

The pirate departure processes are just like the arrival processes described in Figure 2. Their interarrival times are Poisson distributions with a given lambda, which is defined before runtime. Since no real data exists for how many pirates depart a given port, the ability to analyze various departure rates is highly valuable.

2. Pirate Camps

Each pirate camp is modeled separately and all listen to a separate pirate departure process, as seen in Figure 20. This gives the modeler explicit control of each pirate camps rate of pirate departure. The author used information from Piracy MMOWGLI action plans and other open-source data to choose which pirate camps to model. The pirate camp component is also coded in a way that allows for pirates to leave the camp separately instead of in groups the size of the defined number of pirates. The code for one pirate camp departure process and pirate camp can be viewed in Appendices E and F, respectively.

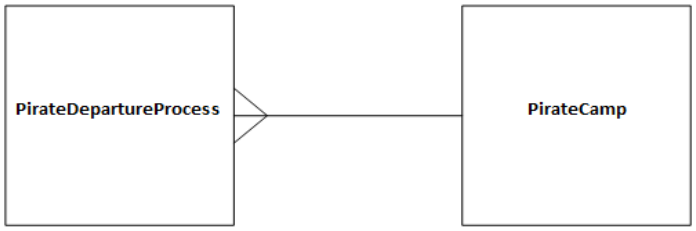


Figure 20. Visual depiction of a Pirate Departure Process and Pirate Camp SimEventListener Pattern

3. Merchant Ship Departure Processes

The merchant ship departure processes are also modeled with a typical departure process. Their inter-arrival times are Poisson distributions with a given lambda, which can be defined before runtime. This simulation utilized a lambda based on 42,000 ships per year transiting around the Horn of Africa. This thesis currently does not take into account any seasonal variation or varying intensities.

4. Merchant Ship Port of Origin

Each merchant ship leaves from one of three locations: the Red Sea, the Gulf of Oman, or just North of the Maldives. For the purpose of these models it is not important which port the ships left from, but rather the direction the ship was heading. The ports of origin components play the same role as the pirate camp components. The merchant ship acts almost identical to the pirate camp and communicates with the departure process the same way, as seen from Figure 21. The code for one merchant ship departure process and merchant port origin can be viewed in Appendices G and H, respectively.

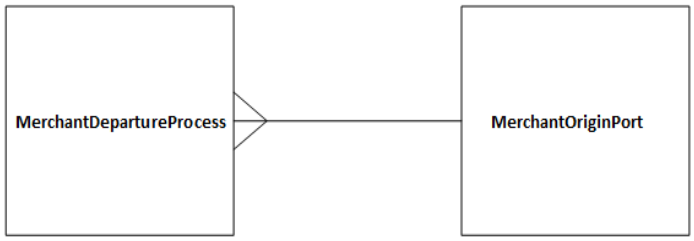


Figure 21. Merchant Departure Process and Merchant Origin Port SimEventListener Pattern

E. SIMKIT SCENARIO ASSEMBLIES

The scenarios chosen to model were based upon action plans created by players in the Piracy MMOWGLI 2012, expert only game. These scenarios give decision makers three distinct options for implementing naval strategy around the Horn of Africa. All images and concepts are taken directly from the Piracy MMOWGLI 2012 Action Plan Report. There are many ways to model and analyze these scenarios, but this thesis focuses on two measures of effectiveness (MOEs), how likely naval ships are to detect pirates and how likely pirates are to successfully hijack a merchant in each scenario. These were the most feasible MOEs given the time constraints to complete a Master's thesis. Due to these constraints the MMOWGLI action plans are not fully modeled and evaluated as the authors describe. However, enough detail is modeled in order to provide a sound analysis on which scenarios are best for the chosen MOEs, as well as give valuable insight on how to best combat pirates.

1. Defense Scenario One: Transit Lane Patrols

The transit lane operation action plan calls for naval vessels to continue patrols along the IRTC, but also implements another transit lane that extends the IRTC toward Maldives. Naval patrols are close to the merchants, but also provide a barrier of protection to merchant traffic off the coasts of Oman and India. The barrier of protection provides quarantine-like patrols without the legal framework of a traditional naval quarantine. This plan recommends that merchants travel via the specified transit lanes or provide their own security. The general concept modeled in this thesis can be viewed in Figure 22 and the full action plan can be viewed in Appendix I. The Simkit source code for the assembly is similar to what is provided in Appendix L, with the only notable difference is the location and patrol boxes of navy vessels.

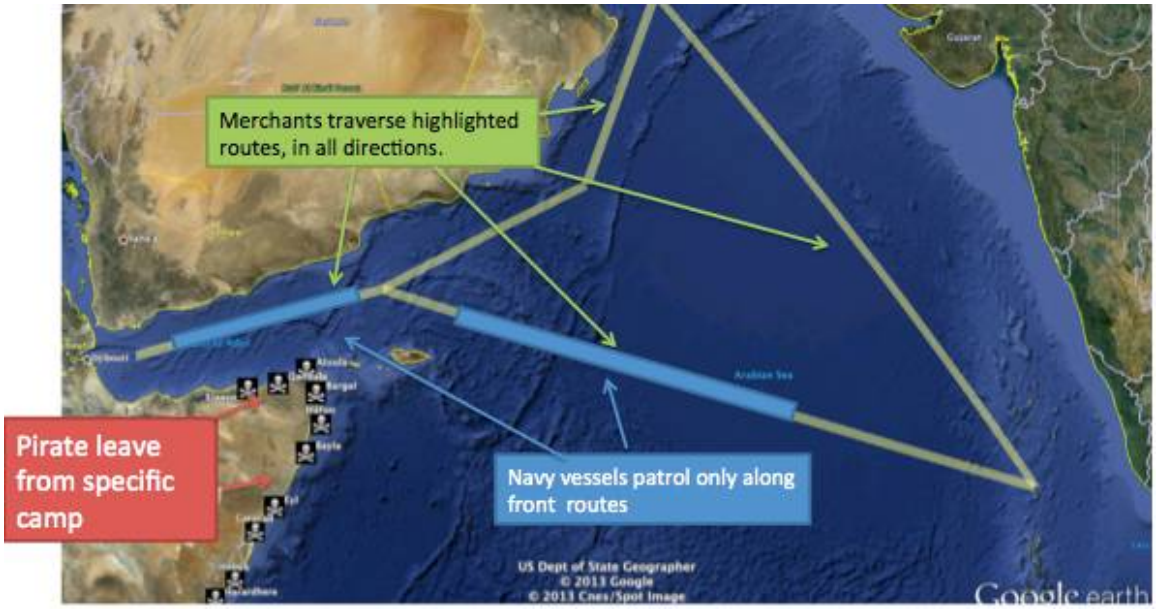


Figure 22. Illustration of Transit Lane Patrol (From Piracy MMOWGLI 2012 Action Plan Report, February 10, 2012).

2. Defense Scenario Two: Naval Quarantine

The naval quarantine action plan calls for a quarantine of the entire southeastern coast of Somalia, from Bargal to the southernmost part of Somalia. The quarantine is 200 nautical miles (NM) from the Somali coast and aims not to impede non-hijacked

merchant traffic. All vessels detected trying to enter the 200 NM quarantine zone is challenged and boarded. Vessels that have been hijacked are not allow to enter into the 200 nautical mile zone and head toward the Somali coast. If the pirates do not cooperate with naval forces then the merchant vessel is disabled in order to restrict any further movement. The aim of this plan is to ensure no merchant vessel has the opportunity to be ransomed off near the shores of Somalia. The simulated pirates do not have access to a resupply of food or additional pirate support. The concept of this plan can be viewed in Figure 23 and the full action plan can be viewed in Appendix J. The Simkit source code for the assembly is similar to what is provided in Appendix M, with the only notable difference is the location and patrol boxes of naval vessels.

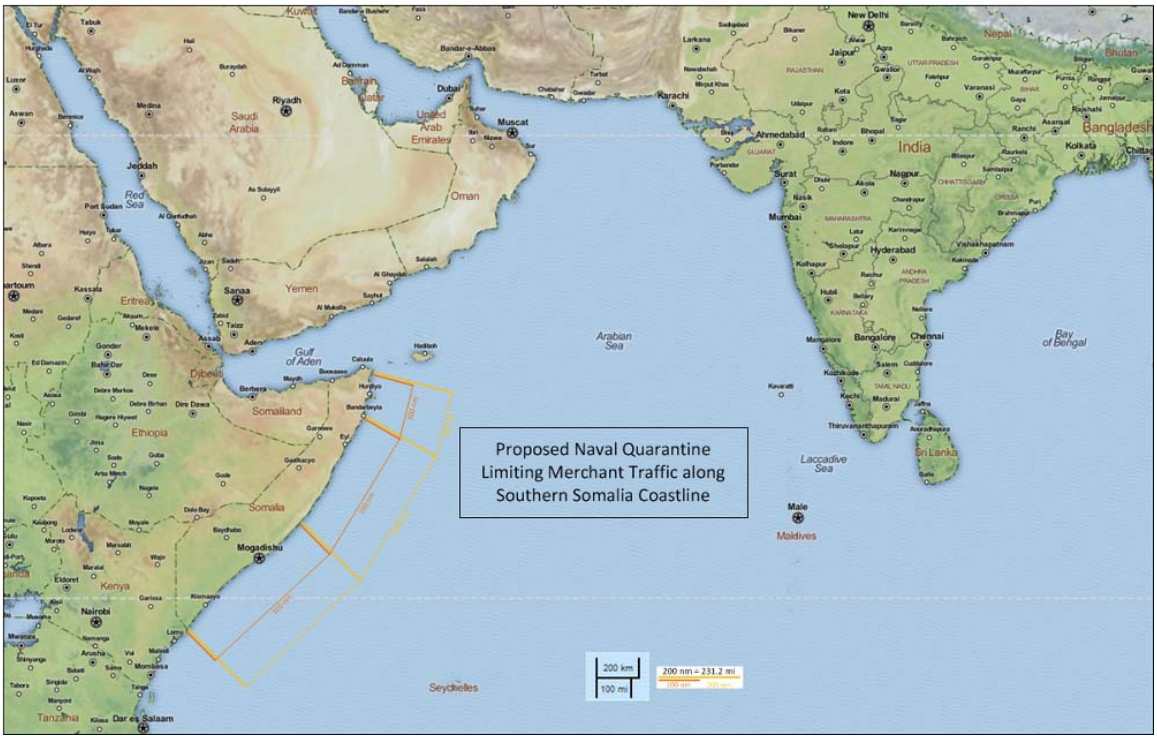


Figure 23. Illustration of a 200NM Naval Quarantine off the Southern coast of Somalia(From Piracy MMOWGLI 2012 Action Plan Report, February 10, 2012).

The MMOWGLI game is not the first time the idea of a naval quarantine has been published. Law (2011) suggests the use of a quarantine in a published Master's thesis for California State Univeristy, Monterey Bay's Panetta Institue of Health and Human Services and Public Policy. The thesis is an applied policy report that gives three alternatives for countering piracy:

1. Keep the status quo (Law, 2011, pp 23 -24).
2. Provide methods of alternative livelihood for Somalis, including a moratorium on fishing in the Somalia EEZ (Law, 2011, pp 24 -26),
3. A naval quarantine (Law, 2011, pp 27 -28).

3. Defense Scenario Three: Pirate Camp Operations

The pirate camp operation action plans are six different plans that evaluate how vulnerable specific pirate camps are to naval intervention. The assumptions used to model this are that INTEL exists on each camp and that ISR assets are continually available to identify pirate activity along the coasts of Somalia. Naval ships would operate in sight of the shoreline and actively deter pirates from launching their vessels. The concept of this plan can be viewed in Figure 24 and the full action plan can be viewed in Appendix K. The Simkit source code for the assembly can be viewed in Appendix M.

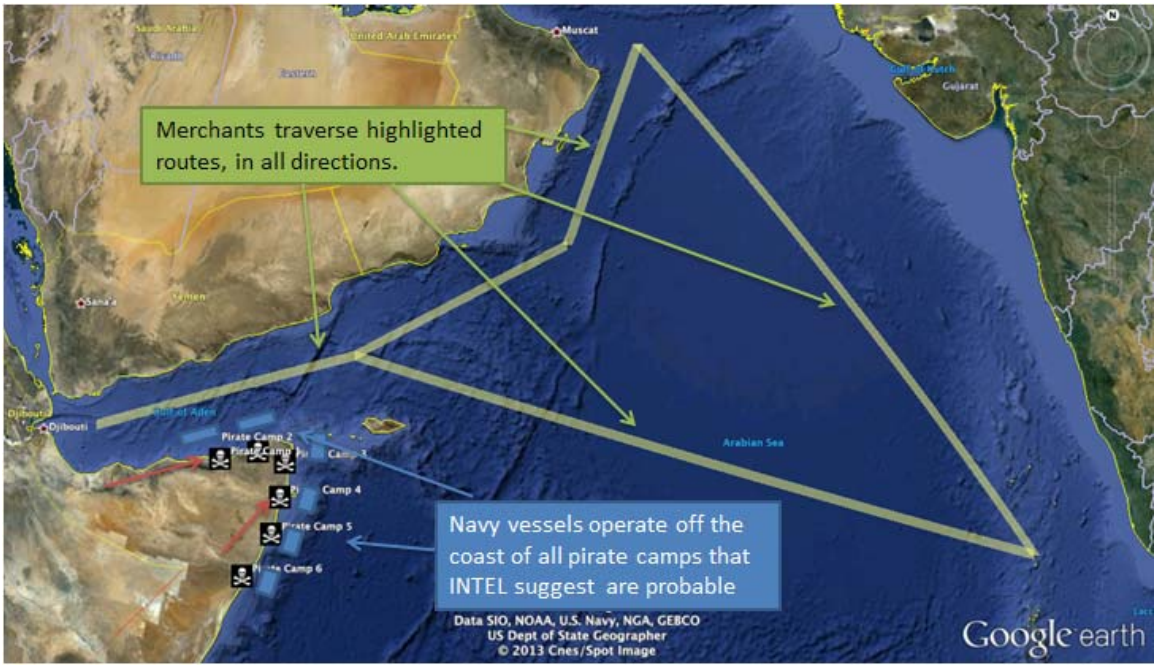


Figure 24. An illustration of Pirate Camp Operations modeled for this thesis.

The pirate camp operations described in this Action Plan can also be used for operations such as those that were conducted by EU forces in May 2012. These operations included bombing the shore basing efforts of pirates on the Somali coast (http://worldnews.nbcnews.com/_news/2012/05/15/11711225-eu-forces-attack-somali-pirates-on-land-for-first-time?lite).

F. JAVA SUPPLEMENTAL CLASSES

There are a few other classes that are highly important to the functionality of all the models and simulations in this thesis.

The Platform.java class is a subclass of Simkit's BasicLinearMover class and is used in order allow each entity to have a state implementation and to disable the functionality of the entity after it is captured or disabled, i.e., a pirate ship after it has been apprehended by the navy. Each entity mover is of class Platform, which allows it to inherit its functionality. The Java source code for Platform.java can be viewed in Appendix M.

In order to assign each Platform (or entity) their specified type, i.e., navy, merchant, or pirate, a simple enum class was created, PlatformType.java. This enum contains only enum types, NAVY, MERCHANT, and PIRATE. The assignment is made in the Simkit assembly and passed into the MoverManager's constructor. The simple enum class can be viewed in Appendix N.

Each entity also has a state class: NavyState.java, PirateState.java, and MerchantState.java. These classes also are the trigger for state transitions in the simulation. Each class accounts for all possible states the particular entity can encounter during the simulation. The java source code for all the entity state classes can be viewed in Appendix O – Appendix Q.

G. DETAILED DESCRIPTION OF VISUALIZATION IMPLEMENATION

1. X3D-Edit and KML

X3D-Edit was utilized to author and validate KML code in order to visualize simulation data. KML can be used for many purposes, in this thesis it was utilized to visualize pirate path history and attack history. To view pirate path history a KML <LineString> is used to create a path. In order to obtain a pirate's position during its mission a Java LinkedList was created in the PirateMoverManager. Then in e event that includes a change in movement for the pirate the current position is taken and put into the LinkedList. The following code snippet shows this functionality:

```
wayPoint = new WayPoint( myMover.getCurrentLocation() );
wayPointList.add(wayPoint);
```

Then at the end of Simkit scenario assembly simply iterate through the LinkedList using a java for-each loop to put the coordinates into a KML format (in KML coordinates <LineString> are expressed as longitude, latitude, elevation), as seen with the following code snippet:

```
for (Iterator it = ioPmm.getWayPointList().iterator();
it.hasNext();)
 {
      WayPoint output = (WayPoint) it.next();
      System.out.println( output.getWayPoint().getY() + ,”“ +
      output.getWayPoint().getX() + ,”“ + 0 );
  }
```

This output can then be copied and pasted into X3D-Edit as shown in Figure 25.

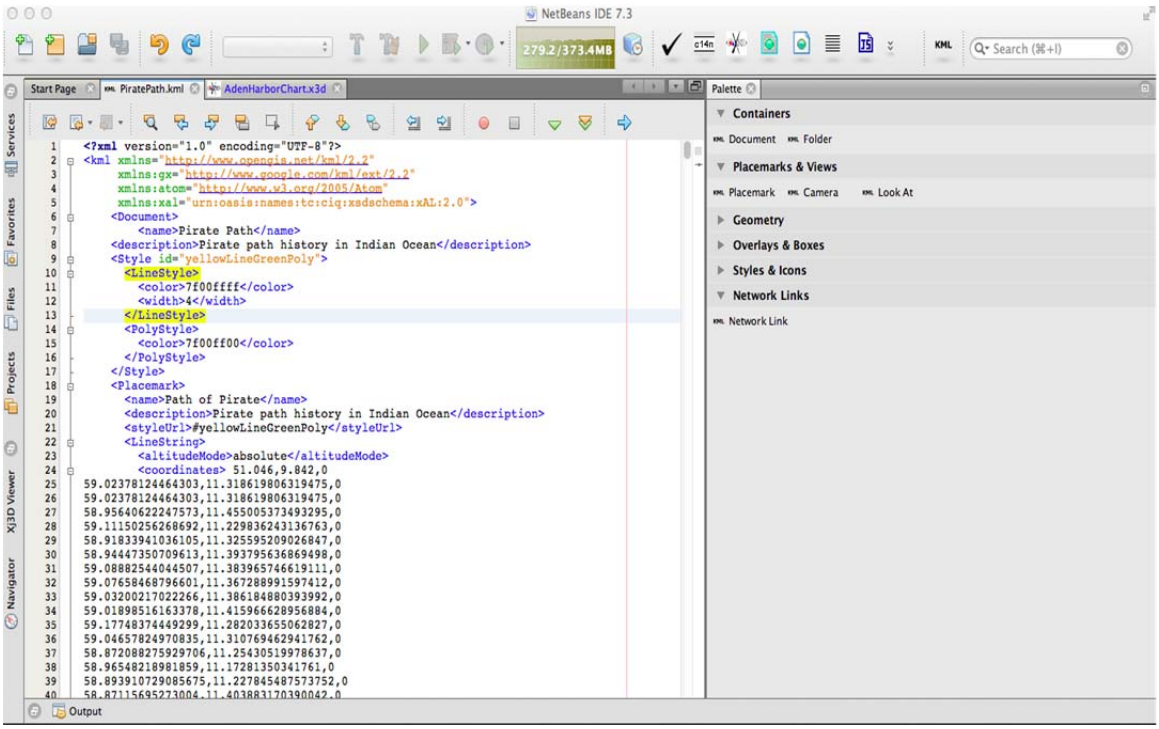


Figure 25. X3D-Edit with PiratePath.kml and the KML Palette

Once the KML file is validated in X3D-Edit it can be easily viewed in Google Earth. Figure 26 shows a simple example of a pirate that left the pirate camp of Bayla, searched a destination in the Indian Ocean and returned to camp.

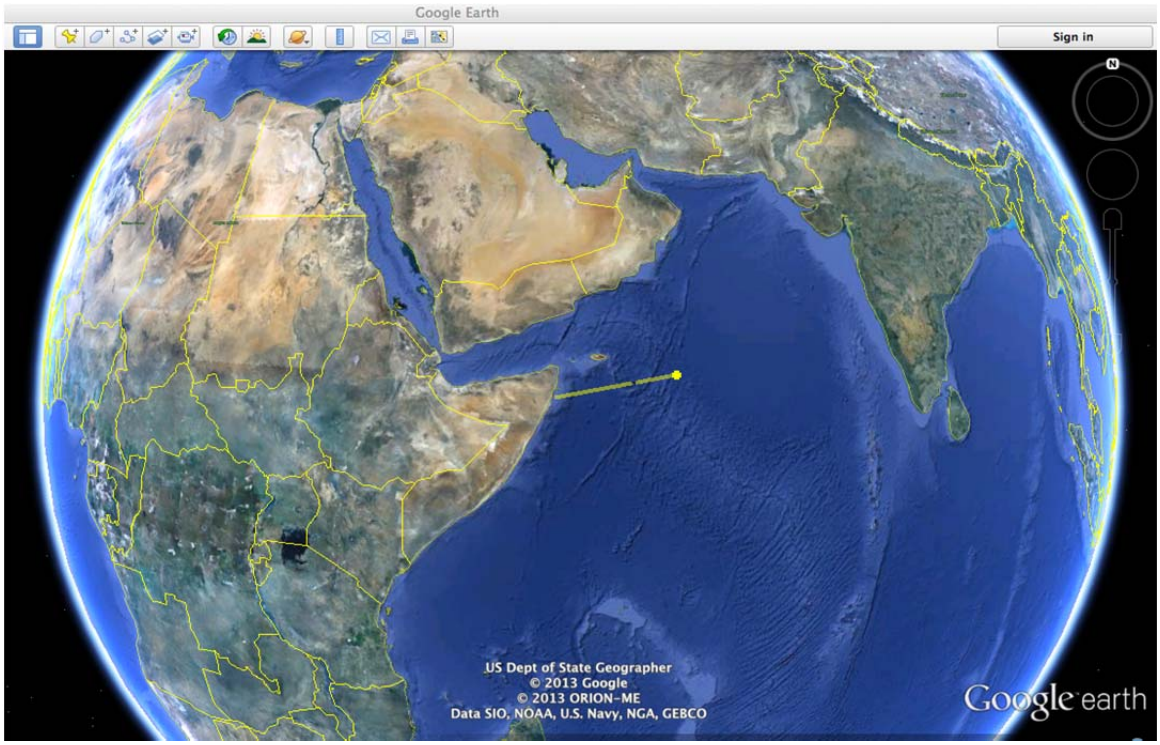


Figure 26. Pirate Path History of single pirate viewed in Google Earth

Pirate attack history can also be visualized with KML. This visualization can be helpful for decision makers in order to see if there are any specific patterns of where pirates are able to gain access to merchant vessels. This implementation is similar to the pirate path history implementation, but instead of using a <LineString>, it is a <Placemark> for each attempted attack. A Java LinkedList is created and etime there is an attack and the location of the merchant at the time of attack is stored in the LinkedList. The optimal location for this implementation was in the Adjudicator.java class. Then to output the data a Java for-each loop can be used as shown in the following code snippet:

```
for (Iterator it = adj.getWayPointList().iterator();
it.hasNext();)
   {
     WayPoint output = (WayPoint) it.next();
     System.out.println("<Placemark>");
     System.out.println("<name>Successful Pirate
     attack</name>");
     System.out.println("<descritpion>Successful Pirate
     Attack</description>");
     System.out.println("<Point>");
     System.out.println("<coordinates>" +
     output.getWayPoint().getY() + ,"" +
     output.getWayPoint().getX() +  "</coordinates>");
     System.out.println("</Point>");
     System.out.println("</Placemark>");
   }
```

Figure 27 shows the successful attack history of the first replication of the naval quarantine scenario.

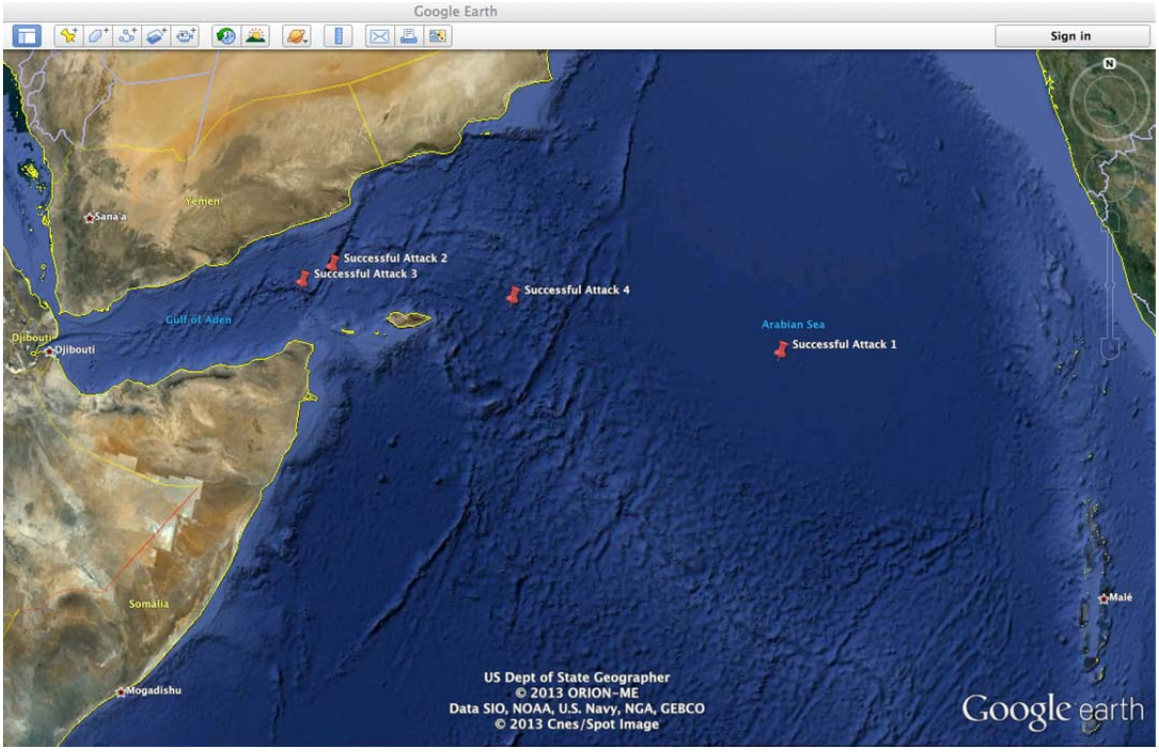


Figure 27. Pirate Successful Attack History for one simulation replication viewed in Google Earth

2. Open-source Geographical Information Systems (GIS)

Since OpenMapTM and Open Street Map are both open-source they are appealing platforms to learn and connect Simkit to. Another benefit of OpenMapTM is the ability to utilize the Mil-Std 2525 symbology. Although Mil-Std 2525 was not demonstrated as a part of this thesis, it is something that is of value and worth knowing. For a detailed description on implementing Simkit models into OpenMapTM and creating a simulation layer for GIS systems, refer to (Gunal, 2010). He provides a superb explanation, with code snippets, that is easy to follow and implement. Figure 28 shows a basic model of a quarantine implemented in OpenMapTM.

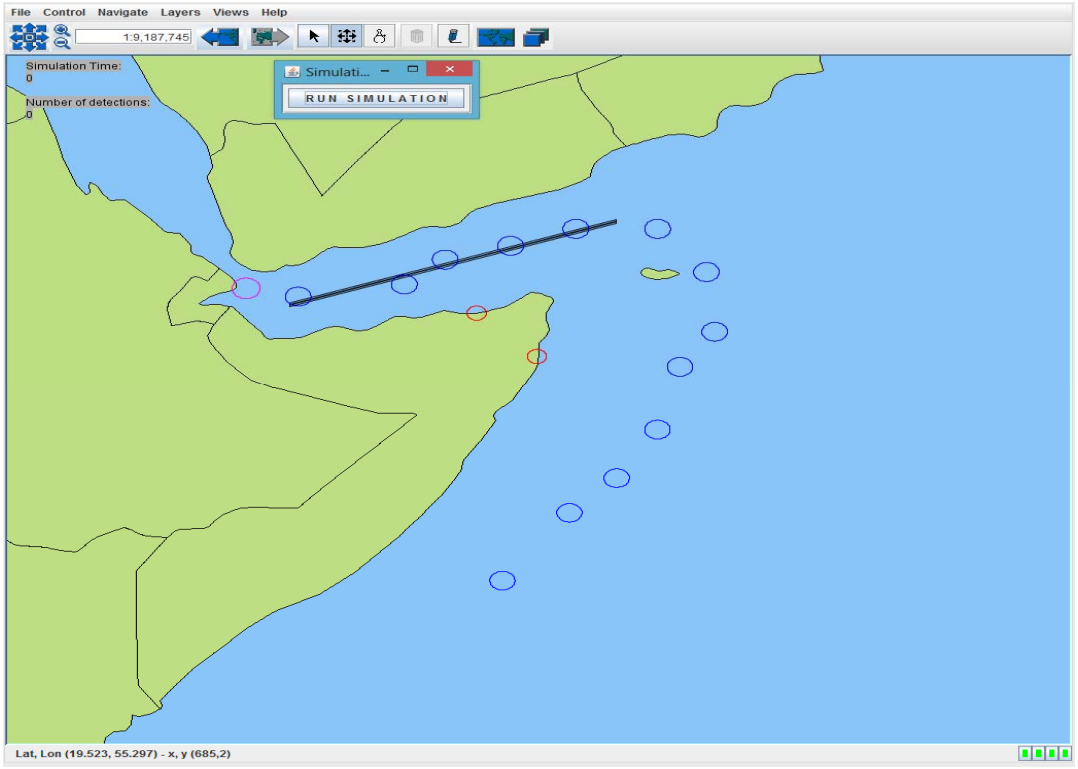


Figure 28. OpenMap^{TM} GUI with Simulation Layer Implemented

The source code for the Simulation Layer can be viewed in Appendix R. To setup and assembly to run the simulation in OpenMapTM it is similar to the Simkit assembly in Appendix L, the two major difference are all locations are in latitude and longitude and the utilization of the number to degree function, as discussed (Gunal, 2010).

```
public double nmToDeg( int latOrLon, double distance )
  {
       DistanceMouseMode xx = new DistanceMouseMode();
        if ( latOrLon == 1 )
        {
            double lonCoefficient = xx.getGreatCircleDist(
            20.0, 13.0, 20.0, 14.0, 2 );
            return distance / lonCoefficient;
         }
        else
        {
            double latCoefficient = xx.getGreatCircleDist(
            20.0, 13.0, 21.0, 13.0, 2 );
            return distance / latCoefficient;
        }
   }
```

This function uses the great circle distance equation to calculate the number of degrees in a distance based on where the entity is in the world. This is required by OpenMap™ when calculating distances.

3. Java Swing

Implementation of Java Swing visualization is made real simple with Simkit. In the Simkit library the “smd” package has an “animate” package. This package allows for basic animations to be performed using Java Swing. The first piece to implementing this is ensuring the “Actions.jar” is included by adding to the Netbeans or Eclipse library for the project. Once this is done creating a Sandbox Frame and a Sandbox is a straightforward process. The code snippet to implement this is found in Appendix S. Once the Sandbox Frame is set up the only part left is adding the movers and sensors. This is done with only a couple lines of code.

To add a single mover and sensor:

```
sandboxFrame.addMover( elaayoPirateMover], Color.RED );
sandboxFrame.addSensor( elaayoPirateSensor, Color.RED );
```

To add an array of movers and sensors:

```
for ( int i = 0 ; i < elaayoPirateMover.length ; ++i )
{
      sandboxFrame.addMover( elaayoPirateMover[i], Color.RED );
      sandboxFrame.addSensor( elaayoPirateSensor[i], Color.RED );
}
```

As seen from the code in Appendix S, a waypoint generator and mouse listener is easily implemented for added functionality. The WaypointBuilder source code can be found in Appendix T and the MouseListener in Appendix U.

H. SUMMARY

Modeling piracy around the Horn of Africa is made easier and more logical using DES and the event graph methodology. MMOWGLI action plans can indeed be modeled and are highly beneficial to decision makers. The action plans layout all the required details needed by both the decision maker and modeler. Many options exist for visualizing DES models; as such three different approaches were discussed in the chapter. This chapter also discussed how to implement simulation in each visualization technology, but the best choice as to which visualization technology to use is highly dependent on the resources available, the modeler's capabilities, and the end product detail desired.

VI. SIMULATION ANALYSIS

A. INTRODUCTION

The simulation and models in this thesis are stochastic, meaning they involve probability, therefore have random inputs that change every run. In order to make confident predictions using a stochastic simulation many replications are needed. If the model is run only a few times, then modeler sees only few random scenarios. So, for example, if a pirate has the ability to go anywhere in the Indian Ocean and the modeler only runs the model five times, then the result of the simulation is based on where the pirate was at those five times and does not take into account the other thousands of locations possible. However, if the simulation is run 10,000 times, it gives the modeler a good sense of exactly what can happen, i.e., the pirate can in reality go anywhere in the Indian Ocean. However, 10,000 runs may not be feasible due to computational cost or equipment limitations, so the analyst must decide how many runs yield enough data to ensure informed decisions can be made from the simulation data. Once these simulation runs are complete simulation analysis can be conducted. The analysis allows the modeler to analyze the data collected from the simulation runs, in order to make accurate predictions or decisions about the model. This chapter discusses the simulation analysis techniques performed for this thesis and recommendations for naval strategy around the Horn of Africa.

B. SIMULATION ANALYSIS

Each of the three scenarios, Transit Lane Operations, Naval Quarantine, and Pirate Camp Operations, were run 30 times. This thesis used 30 runs of each scenario because 30 is the minimal amount of runs needed for the data to have the needed properties for statistical significance. For each scenario the following MOEs were evaluated:

- Naval Effectiveness $= \frac{number\ pirates\ detected}{number\ pirates\ departed\ camp}$
- Pirate Effectiveness $= \frac{number\ successful\ pirate\ attacks}{number\ pirates\ departed\ camp}$

In order to evaluate which scenario offered the "best" choice a simple selection procedure was conducted. For the simple selection both naval effectiveness and pirate effectiveness values were calculated and recorded. The sample mean (or X-bar), the standard deviation, and standard error were calculated for each MOE. For the Naval Effectiveness MOE, the highest X-bar is the "best" option and for Pirate Effectiveness MOE the lowest X-bar is the "best" option. Table 4 shows the results for the Naval Effectiveness MOE and Table 5 shows the results for the Pirate Effectiveness MOE.

Scenario	**Pirate Camp Operations**	**Naval Quarantine**	**Transit Lane Operations**
Mean	0.90	0.54	0.40
Standard Deviation	0.06	0.07	0.06
Standard Error.	0.01	0.01	0.01

Table 4. Comparison of the Naval Effectiveness MOE simulation results among all three defense scenarios

Scenario	**Pirate Camp Operations**	**Naval Quarantine**	**Transit Lane Operations**
Mean	0.05	0.14	0.16
Standard Deviation	0.04	0.06	0.06
Standard Error.	0.01	0.01	0.01

Table 5. Comparison of the Pirate Effectiveness MOE simulation results among all three defense scenarios

Figure 29 shows that the Pirate Camp Operation scenario performed much better than the other two scenarios in Naval Effectiveness and pirates performed worst in Pirate Camp Operation, as seen in Figure 30.

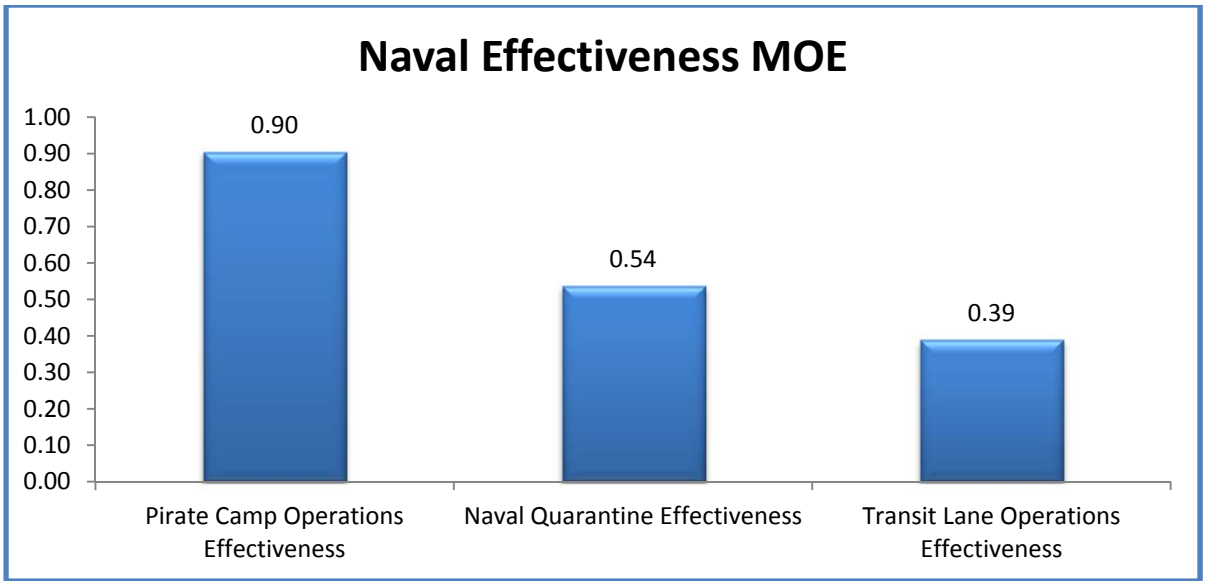


Figure 29. Histogram comparing the results of the Naval Effectiveness MOE of each defense scenario.

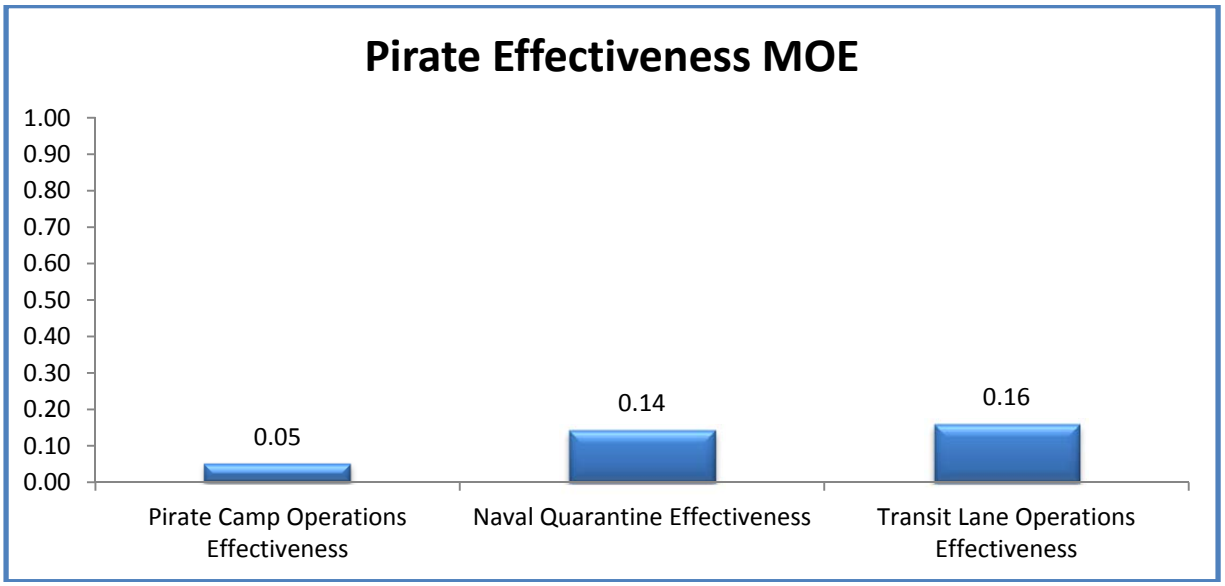


Figure 30. Histogram comparing the results of the Pirate Effectiveness MOE for each defense scenario.

An interesting observation is noted in looking at how close the Pirate Effectiveness MOE was in Naval Quarantine and Transit Lane Operations, although in Naval Quarantine performed significantly better in Naval Effectiveness. This can be attributed to the close proximity of naval vessels during Transit Lane Operations. During

these types of operations the probability of having a naval ship close enough to either interdict or launch a helicopter to interdict after receiving a merchant distress call is greater since the ships are patrolling on the transit lanes.

C. SUMMARY

Simulation analysis is the most important aspect of simulation modeling. It allows decision makers to make sense of what went on behind the scenes of the simulation and how they can use that information to make better decisions. There are many simulation techniques, ranging from simple ones, such as the simple selection process, to complex ones. The right analysis technique is dependent on what is being modeled, valid input data, and what assets are available to the analyst to achieve a desired result.

VII. CONCLUSION AND RECOMMENDATIONS

A. RECOMMENDATIONS FOR COUNTER-PIRACY STRATEGY

The scenarios modeled in this thesis gives decision makers three distinctly different approaches to combat piracy. However, as seen from Tables 1 and 2, Pirate Camp Operations performed significantly better than Naval Quarantine and Transit Lane Operations, when analyzing the Naval Effectiveness and Pirate Effectiveness MOEs. The Pirate Camp Operation was not only superior in performance, but also utilized two fewer ships than the other scenarios.

Assistant Secretary Shapiro and others who claimed there is too much ocean for naval ships to patrol (Shapiro, 2009) were correct in their assessment, however the real question is, *why are naval forces trying to patrol that much water?* Piracy has been a land problem since 14th century BC and still today in the 21st century it is being combated from the sea. Whether it be another surge in Somali piracy or a rise in maritime piracy in another region, naval forces need to cut the amount of water patrolled and attack the problem before it even reaches international waters. Not only do the simulations for this thesis show the superior effectiveness of combating piracy closer to shore, it would more than likely play a major deterrent for pirates to physically see naval vessels patrolling off their coasts. Operations like the pirate camp operation also allow for easier opportunity for capacity building engagements with Somali coast guard forces, which allows the Somali people to defeat piracy once and for all.

1. It is recommended that counter-piracy forces consider a pirate camp operation approach to prevent pirates from reaching into the merchant transit lanes. However, this approach does have some drawbacks, the major one being that navy vessels would have to operate inside the Somali Economic Exclusion Zone (EEZ). This approach might have a negative impact on current efforts to rebuild the Somali fishing industry.

2. If it is determined that the impact of navy patrols within the Somalia EEZ might negatively impact efforts to rebuild the fishing industry off the coast of Somalia, then the use of a naval quarantine provides the best strategic option. The naval quarantine does have lots of benefits as well. It not only cuts down the amount of ocean required to patrol, but it also keeps naval vessels out of the EEZ. Another key aspect to the naval quarantine is that it prevents pirated vessels from making it back to the shores of Somalia. The pirates are then forced to conduct all negotiations away from its land, financiers, and supplies.

3. Both of these solutions demonstrate that affordable naval operations are feasible for combined maritime forces to prevent the resurgence of Somali piracy on the high seas. Similar approaches are likely feasible for other regions plagued by piracy around the world.

B. RECOMMENDATIONS FOR FUTURE WORK

The following is future work that can be accomplished to add to the body of work in this thesis.

1. Implement UAVs and determine if the use of UAVs can lower the need for ships or limit the use of the ships helicopter.
2. Conduct cost/benefit analysis of each scenario.
3. Determine fuel consumption and savings for each scenario using ship's helo or UAV.
4. Conduct a comparison of pirate effectiveness when merchants traverse by routes other than dedicated transit lanes.
5. Conduct a more robust simulation analysis that includes a design of experiment
6. Create a tactical decision aid (TDA) for use by ships and shore commands that utilize simulation and visualization for better operations planning.
7. Conduct a follow-on MMOWGLI counter-piracy game to perform a renewed exploration of these operations, recent developments, and future counter-piracy strategies.

C. FINAL THOUGHTS AND CONSIDERATIONS

Maritime piracy is one of many wicked problems faced by military decision makers. However, the U.S. military is fully equipped with highly educated and trained enlisted personnel and officers to come up with the best approach to combat these problems. With this valuable asset the strategy sessions used to formulate strategic options needs to include a much broader audience, rather than simply the top echelon of the chain-of-command and its staff. War gaming via crowd sourcing affords military leaders the opportunity to tap into this precious resource. The MMOWGLI platform was designed to tackle these wicked problems and discrete event simulation allows for analysis of the action plans formed during these brainstorming sessions. This thesis has demonstrated how this methodology can be used to formulate strategically valuable options from experts in maritime piracy and the action plans can be modeled using discrete event simulation and analyzed using simulation analysis. It is highly recommended that military leaders utilize this methodology in their planning and evaluation of current efforts.

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX A. PIRATE MOVER MANAGER JAVA CODE

```
1 /*
2  * PirateMoverManager.java
3  *
4  */
5 package entities;
6
7 import java.awt.geom.Point2D;
8 import java.util.LinkedList;
9 import simkit.Priority;
10 import simkit.SimEntityBase;
11 import simkit.random.DiscreteRandomVariate;
12 import simkit.random.RandomVariate;
13 import simkit.random.RandomVariateFactory;
14 import simkit.smd.CookieCutterSensor;
15 import supplemental.PirateState;
16 import supplemental.Platform;
17 import supplemental.PlatformType;
18
19 /**
20  *
21  * Models the behavior of a Somali Pirate.
22  *
23  * @version $Id: PirateMoverManager.java 199 2013–03–03 06:10:24Z crhutchi $
24  * @author Chad R Hutchins
25  *
26  */
27 public class PirateMoverManager extends SimEntityBase {
28
29     /**
30      * Parameters. Contains Setters and Getters
31      *
32      */
33     private Platform myMover;
34     private CookieCutterSensor sensor;
35     private Point2D baseLocation;
36     private RandomVariate[] pathGenerator;
37     private RandomVariate[] patrolBoxGenerator;
```

```
38  private double timeOnPatrol;
39  private PlatformType platformType;
40  private DiscreteRandomVariate attackDecision;
41  private DiscreteRandomVariate successOrFailGenerator;
42  private RandomVariate[] unsuccessfulAttackTime;
43  private Point2D patrolBoxStartX;
44  private Point2D patrolBoxStarY;
45  private Point2D nextPathWaypoint;
46  /**
47   * State Variables. Contains only getters, no setters.
48   *
49   */
50  protected PirateState myMovementState;
51  protected double numberAttemptedAttacks; //number of attempted attacks
52  protected double numberSuccessfulAttacks;  //number of successful attacks
53  protected double numberUnsuccessfulAttacks; //number of unsuccessful attacks
54  protected double numberMerchantsDetected; //number of merchants  detected
55  protected double numberDetectedBeforeAction; //
56  protected boolean isAlive; //
57  /**
58   * String constant for firePropertyChange modification of the state
59   * variable, not visible outside this class
60   *
61   */
62  private final String MY_MOVEMENT_STATE = "myMovementState";
63  private final String NUMBER_ATTEMPTED_ATTACKS = "numberAttemptedAttacks";
64  private final String NUMBER_MERCHANTS_DETECTED = "numberMerchantsDetected";
65  private final String NUMBER_UNSUCCESSFUL_ATTACKS =
66        "numberUnsuccessfulAttacks";
67  private final String NUMBER_SUCCESSFUL_ATTACKS = "numberSuccessfulAttacks";
68  private final String IS_ALIVE = "isAlive";
69
70  /**
71   * String constant for waitDelay method scheduling, visible to other classes
72   *
73   */
74  protected final String MOVE_TO = "MoveTo";
75  protected final String SEARCH_FOR_MERCHANTS = "SearchForMerchants";
76  protected final String DECIDE_COA = "DecideCOA";
77  protected final String ATTACK = "Attack";
78  protected final String RETURN_TO_PIRATE_CAMP = "ReturnToPirateCamp";
```

```
79  protected final String STOP = “Stop”;
80  protected final String BOARDED_BY_NAVY = “BoardedByNavy”;
81  protected final String DIE = “Die”;
82  /**
83   * String constant for all other cases.
84   *
85   */
86  protected final String MERCHANT = “Merchant”;
87
88  //Local patrolbox distance coordinates
89  double scale = 0.5;
90  double localDistance = 10 * scale;  //10NM
91  double transitSpeed = 10 * scale;
92  double searchSpeed;
93  double successfulAttackTimeDelay;
94  double timeOfNavyBoarding = 2.0;
95
96  /**
97   * Default Constructor
98   *
99   */
100  public PirateMoverManager()
101  {
102     //Does not set anything
103  }
104
105  /**
106   * Main constructor: Sets mover, sensor, base location, path, patrol box,
107   * attack decision random variate, and success or Fail random variate.
108   *
109   * @param myMover
110   * @param sensor
111   * @param baseLocation
112   * @param pathGenerator
113   * @param localPatrolBoxGenerator
114   * @param attackDecision
115   * @param successOrFailGenerator
116   */
117  public PirateMoverManager( Platform myMover,
118                             CookieCutterSensor sensor,
119                             Point2D baseLocation,
```

```
120                         RandomVariate[] pathGenerator,
121                         DiscreteRandomVariate attackDecision,
122                         RandomVariate[] unsuccessfulAttackTime)
123     {
124        this.setMyMover( myMover );
125        this.setSensor( sensor );
126        this.setBaseLocation( baseLocation );
127        this.setPathGenerator( pathGenerator );
128        this.setAttackDecision( attackDecision );
129        this.setUnsuccessfulAttackTime( unsuccessfulAttackTime );
130     }
131
132     /**
133      * Reset: Resets state variables at end of each replication
134      *
135      */
136     @Override
137     public void reset()
138     {
139        super.reset();
140        myMovementState = PirateState.WAITING_AT_BASE;
141        myMover.setInitialLocation( baseLocation );
142        numberAttemptedAttacks = 0;
143        numberSuccessfulAttacks = 0;
144        numberUnsuccessfulAttacks = 0;
145        numberMerchantsDetected = 0;
146        isAlive = true;
147     }
148
149     /**
150      * Run: FirePropertyChange for all state variables in reset method
151      *
152      */
153     public void doRun()
154     {
155        firePropertyChange( MY_MOVEMENT_STATE, getMyMovementState() );
156 //       firePropertyChange( NUMBER_ATTEMPTED_ATTACKS,
157 //                           getNumberAttemptedAttacks() );
158        firePropertyChange( NUMBER_SUCCESSFUL_ATTACKS,
159                            getNumberSuccessfulAttacks() );
160        firePropertyChange( NUMBER_UNSUCCESSFUL_ATTACKS,
```

```
161                     getNumberUnsuccessfulAttacks() );
162     firePropertyChange( NUMBER_MERCHANTS_DETECTED,
163                     getNumberMerchantsDetected() );
164   }
165
166   /**
167    * LeavePirateIoPirateCamp Event: Changes myMovementState to
168    * ENROUTE_TO_PATROL, generates the next way points, and schedules MoveTo
169    * event for pirates departing from pirate camps in the Indian Ocean side of
170    * Somalia.
171    *
172    */
173   public void doLeaveIoPirateCamp()
174   {
175     PirateState oldMyMovementState = getMyMovementState();
176     myMovementState = PirateState.ENROUTE_TO_PATROL;
177     firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
178                     getMyMovementState() );
179
180     RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
181     transitSpeedGenerator[0] = RandomVariateFactory.
182           getInstance("Uniform," 8 * scale, 12 * scale);
183
184     transitSpeed = transitSpeedGenerator[0].generate();
185
186     myMover.setMaxSpeed( transitSpeed );
187
188     nextPathWaypoint = new Point2D.Double(
189           getPathGenerator()[0].generate(),
190           getPathGenerator()[1].generate() );
191
192     waitDelay( MOVE_TO, 0.0, nextPathWaypoint );
193   }
194
195   /**
196    * LeavePirateGoaPirateCamp Event: Changes myMovementState to
197    * ENROUTE_TO_PATROL, generates the next way points, and schedules MoveTo
198    * event for pirates departing from pirate camps in the Gulf of Aden side of
199    * Somalia.
200    */
201   public void doLeaveGoaPirateCamp()
```

```
202   {
203      PirateState oldMyMovementState = getMyMovementState();
204      myMovementState = PirateState.ENROUTE_TO_PATROL;
205      firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
206                        getMyMovementState() );
207
208      RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
209      transitSpeedGenerator[0] = RandomVariateFactory.
210            getInstance("Uniform," 8 * scale, 12 * scale);
211
212      transitSpeed = transitSpeedGenerator[0].generate();
213
214      myMover.setMaxSpeed( transitSpeed );
215
216      nextPathWaypoint = new Point2D.Double(
217            getPathGenerator()[0].generate(),
218            getPathGenerator()[1].generate() );
219
220      waitDelay( MOVE_TO, 0.0, nextPathWaypoint );
221   }
222
223   /**
224    * EndMove Event: Generates nextWayPoint and if myMovementState is
225    * PATROLLING it schedules MoveTo. If myMovementState is ENROUTE_TO_PATROL
226    * it schedules SEARCH_FOR_MERCHANTS.
227    *
228    * @param mover
229    */
230   public void doEndMove( Platform mover )
231   {
232      double xVal = nextPathWaypoint.getX();
233      double yVal = nextPathWaypoint.getY();
234
235      RandomVariate[] localPatrolBoxGenerator = new RandomVariate[ 2 ];
236      localPatrolBoxGenerator[0] = RandomVariateFactory.
237            getInstance( "Uniform,"
238                  ( xVal - localDistance ),
239                  ( xVal + localDistance ));
240      localPatrolBoxGenerator[1] = RandomVariateFactory.
241            getInstance( "Uniform,"
242                  ( yVal - localDistance ),
```

```
243                     ( yVal + localDistance ));
244
245     Point2D nextWaypoint = new Point2D.Double(
246             localPatrolBoxGenerator[0].generate(),
247             localPatrolBoxGenerator[1].generate() );
248
249     if ( myMovementState == PirateState.ENROUTE_TO_PATROL )
250     {
251         waitDelay( SEARCH_FOR_MERCHANTS, 0.0, nextWaypoint );
252     }
253
254     if ( myMovementState == PirateState.PATROLLING )
255     {
256         waitDelay( MOVE_TO, 0.0, nextWaypoint );
257     }
258 }
259
260 /**
261  * SearchForMerchants Event: Changes myMovementState to PATROLLING.
262  * Generates patrolBox to hunt for merchant ships, and schedules MOVE_TO
263  * with nextWaypoint in patrol box.
264  *
265  */
266 public void doSearchForMerchants( Point2D nextWaypoint  )
267 {
268     PirateState oldMyMovementState = getMyMovementState();
269     myMovementState = PirateState.PATROLLING;
270     firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
271                     getMyMovementState() );
272
273     RandomVariate[] searchSpeedGenerator = new RandomVariate[2];
274     searchSpeedGenerator[0] = RandomVariateFactory.
275             getInstance("Uniform," 2 * scale, 8 * scale);
276
277     searchSpeed = searchSpeedGenerator[0].generate();
278
279     myMover.setMaxSpeed( searchSpeed );
280
281     double xVal = nextWaypoint.getX();
282     double yVal = nextWaypoint.getY();
283
```

```
284 //      System.out.println(myMover.getName() + " Next WayPoint X Value: " + xVal);
285 //      System.out.println("Next WayPoint Y Value: " + yVal);
286
287       RandomVariate[] localPatrolBoxGenerator = new RandomVariate[ 2 ];
288       localPatrolBoxGenerator[0] = RandomVariateFactory.
289             getInstance( "Uniform,"
290                         xVal - localDistance,
291                         xVal + localDistance);
292       localPatrolBoxGenerator[1] = RandomVariateFactory.
293             getInstance( "Uniform,"
294                         yVal - localDistance,
295                         yVal + localDistance);
296
297       Point2D nextPatrolWaypoint = new Point2D.Double(
298             localPatrolBoxGenerator[0].generate(),
299             localPatrolBoxGenerator[1].generate() );
300
301       waitDelay( MOVE_TO, 0.0, nextPatrolWaypoint );
302
303       //IO pirates: Fuel is a RV from 2 weeks - 2 months
304       if ( myMover.getInitialLocation().getY () <= 300.0 )
305       {
306          RandomVariate[] lowFuelIOGenerator = new RandomVariate[ 1 ];
307          lowFuelIOGenerator[0] = RandomVariateFactory.
308                getInstance ( "Uniform," 336.0, 1460.0 );
309
310          double lowFuelIO = ((lowFuelIOGenerator[0].generate ()) -
311                (getEventList().getSimTime()));
312
313          if (lowFuelIO < 0)
314          {
315             lowFuelIO = 12.0;
316          }
317
318          //If fuel is low go back to camp
319          waitDelay ( RETURN_TO_PIRATE_CAMP, lowFuelIO, Priority.HIGH );
320       }
321
322       //GOA pirates: Fuel is a RV from 3 days - 3 weeks
323       if ( myMover.getInitialLocation().getY () > 300.0 )
324       {
```

```
325         RandomVariate[] lowFuelGOAGenerator = new RandomVariate[ 1 ];
326         lowFuelGOAGenerator[0] = RandomVariateFactory.
327               getInstance ( "Uniform," 72.0, 504.0 );
328
329         double lowFuelGOA = ((lowFuelGOAGenerator[0].generate ()) -
330               (getEventList().getSimTime()));
331
332         if (lowFuelGOA < 0)
333         {
334            lowFuelGOA = 12.0;
335         }
336
337         //If fuel is low go back to camp
338         waitDelay ( RETURN_TO_PIRATE_CAMP, lowFuelGOA, Priority.HIGH );
339      }
340   }
341
342   /**
343    * ReturnToPirateCamp Event: Changes myMovementState to RETURNING_TO_BASE.
344    * Schedules MOVE_TO with baseLocation coordinate.
345    *
346    */
347   public void doReturnToPirateCamp()
348   {
349      PirateState oldMyMovementState = getMyMovementState();
350      myMovementState = PirateState.RETURNING_TO_BASE;
351      firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
352                  getMyMovementState() );
353
354
355
356      RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
357      transitSpeedGenerator[0] = RandomVariateFactory.
358            getInstance("Uniform," 8 * scale, 12 * scale);
359
360      transitSpeed = transitSpeedGenerator[0].generate();
361
362      myMover.setMaxSpeed( transitSpeed );
363
364      waitDelay( MOVE_TO, 0.0, Priority.HIGH, myMover.getInitialLocation() );
365   }
```

```
366
367     /**
368      * Detection Event: Detects any mover within the sensor range. If contact is
369      * a Merchant and the merchant hasn't been detected before increments
370      * numberMerchantsDetected. Schedules DecideCOA. Adds merchant to list of
371      * detectedMerchants.
372      *
373      * @param contact
374      */
375     public void doDetection( Platform contact )
376     {
377        LinkedList<Platform> detectedMerchants = new LinkedList();
378
379 //         System.out.println( "I " + myMover.getName () +" got a detection" );
380 //
381 //            System.out.println("Contact detected by Pirate: " + contact);
382
383        if ( ( contact.getType() == PlatformType.MERCHANT &&
384             !detectedMerchants.contains( contact  )  )
385
386           (   myMovementState == PirateState.ENROUTE_TO_PATROL ||
387             myMovementState == PirateState.PATROLLING ||
388             myMovementState == PirateState.RETURNING_TO_BASE ) )
389        {
390 //           System.out.println( "Detected a Merchant" );
391
392           detectedMerchants.add( contact );
393
394           numberMerchantsDetected = getNumberMerchantsDetected() + 1;
395
396           waitDelay( DECIDE_COA, 0.0, Priority.HIGH, contact );
397        }
398     }
399
400     /**
401      * DecideCOA Event: generates attack decision based on Bernoulli random
402      * variable. If choice does not equal 1 the decision is to attack, and
403      * cancels (interrupts) prior MOVE_TO events and schedules ATTACK event. If
404      * choice equals 1 then the decision is not to attack. This logic is based
405      * on size of merchant, weather, and various statistics.
406      *
```

```
407     * @param target
408     */
409    public void doDecideCOA( Platform contact  )
410    {
411       int choice = attackDecision.generateInt();
412 //       System.out.println( “Attack Decision: “ + choice );
413
414       if ( choice == 0 )
415       {
416 //          System.out.println( “Decided not to attack” );
417
418          double xValue = myMover.getCurrentLocation().getX();
419          double yValue = myMover.getCurrentLocation().getY();
420
421          RandomVariate[] localPatrolBoxGenerator = new RandomVariate[ 2 ];
422          localPatrolBoxGenerator[0] = RandomVariateFactory.
423                getInstance( “Uniform,”
424                            xValue - localDistance,
425                            xValue + localDistance );
426          localPatrolBoxGenerator[1] = RandomVariateFactory.
427                getInstance( “Uniform,”
428                            yValue - localDistance,
429                            yValue + localDistance );
430
431          Point2D nextWaypoint = new Point2D.Double(
432                localPatrolBoxGenerator[0].generate(),
433                localPatrolBoxGenerator[1].generate() );
434
435
436          waitDelay( SEARCH_FOR_MERCHANTS, 0.0, nextWaypoint );
437
438       }
439       if ( choice == 1 )
440       {
441 //          System.out.println( “Decided to Attack!!” );
442
443          waitDelay( ATTACK, 0.0, Priority.HIGH, myMover, contact );
444       }
445    }
446
447    /**
```

```
448     * Attack Event: Change myMovementState to ATTACKING. Generates success or
449     * fail Bournoulli random variable, based upon statistics on merchant BMP
450     * practices, armed guards on board, etc. If successOrFail does not equal 1
451     * it is a successful attack and schedules SuccessfulAttack event. If the
452     * random variable does equal 1 it is an unsuccessful attack and schedules
453     * UnsuccessfulAttack event. Increments numberAttemptedAttacks.
454     *
455     * @param target
456     */
457    public void doAttack( Platform myMover, Platform contact  )
458    {
459        PirateState oldMovementState = getMyMovementState();
460
461        myMovementState = PirateState.ATTACKING;
462
463        double oldNumberAttemptedAttacks = getNumberAttemptedAttacks();
464        numberAttemptedAttacks = getNumberAttemptedAttacks() + 1;
465
466 //       System.out.println("I am attacking yer ship!!!");
467
468        firePropertyChange( MY_MOVEMENT_STATE, oldMovementState,
469                        getMyMovementState() );
470
471        firePropertyChange( NUMBER_ATTEMPTED_ATTACKS, oldNumberAttemptedAttacks,
472                        getNumberAttemptedAttacks() );
473    }
474
475    /**
476     * UnsuccessfulAttack Event: Increments numberUnsuccessfulAttacks. Schedules
477     * SEARCH_FOR_MERCHANTS with a delay determined by random variate.
478     *
479     */
480    public void doUnsuccesfulAttack()
481    {
482        double oldNumberUnSuccessfulAttacks = getNumberUnsuccessfulAttacks();
483        numberUnsuccessfulAttacks = getNumberUnsuccessfulAttacks() + 1;
484
485 //       System.out.println("My attack has been foiled!!");
486
487        double timeOfAttack = unsuccessfulAttackTime[0].generate();
488
```

```
489 //       System.out.println("Duration of Pirate Attack: " + timeOfAttack);
490
491     waitDelay( SEARCH_FOR_MERCHANTS, timeOfAttack, Priority.HIGH );
492
493     firePropertyChange( NUMBER_UNSUCCESSFUL_ATTACKS,
494                         oldNumberUnSuccessfulAttacks,
495                         getNumberUnsuccessfulAttacks() );
496   }
497
498   /**
499    * A successful attack equals a successful hijacking. Increments
500    * numberSuccessfulAttacks. Schedules returnToPirateCamp.
501    *
502    */
503   public void doSuccessfulAttack()
504   {
505     double oldNumberSuccessfulAttacks = getNumberSuccessfulAttacks();
506     numberSuccessfulAttacks = getNumberSuccessfulAttacks() + 1;
507
508     PirateState oldMovementState = getMyMovementState();
509
510     myMovementState = PirateState.RETURNING_WITH_MERCHANT;
511
512     firePropertyChange( MY_MOVEMENT_STATE, oldMovementState,
513                         getMyMovementState() );
514
515     System.out.println("I got me a ship... aaarrrgghhh!!");
516
517     RandomVariate[] successfulAttackTimeGenerator = new RandomVariate[ 2 ];
518       successfulAttackTimeGenerator[0] = RandomVariateFactory.
519             getInstance ( "Uniform," 1.0, 3.0 );
520
521       successfulAttackTimeDelay =
522             successfulAttackTimeGenerator[0].generate ();
523
524     firePropertyChange( NUMBER_SUCCESSFUL_ATTACKS,
525                         oldNumberSuccessfulAttacks,
526                         getNumberSuccessfulAttacks() );
527
528     waitDelay( STOP, 0.0, Priority.HIGH );
529
```

```
530         waitDelay( RETURN_TO_PIRATE_CAMP, successfulAttackTimeDelay,
531                                 Priority.HIGH );
532
533     }
534
535
536     /**
537      * DetectedByNavy Event: Is triggered when a Navy vessel detects it... this
538      * is setup in main class via adapter. Schedules STOP event and
539      * BOARDED_BY_NAVY event.
540      *
541      * @param contact
542      */
543     public void doDetectedByNavy( Platform contact, double boardingTime )
544     {
545 //      System.out.println( "Contact:" + contact );
546 //
547 //      System.out.println( "Pirate Speed after detection: " + myMover.
548 //              getCurrentSpeed() );
549
550         waitDelay( STOP, 0.0, Priority.HIGH );
551
552         contact.waitDelay( BOARDED_BY_NAVY, 0.0, Priority.HIGH, boardingTime );
553     }
554
555     /**
556      * BoardedByNavy Event: Changes myMovementState to NAVY_BOARDED. If pirate
557      * is attacking when detected schedule DIE event. In all other conditions
558      * schedule pirate to RETURN_TO_CAMP.
559      *
560      */
561     public void doBoardedByNavy( double boardingTime )
562     {
563         PirateState oldMyMovementState = getMyMovementState();
564
565         myMovementState = PirateState.NAVY_BOARDED;
566         firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
567                         getMyMovementState() );
568
569 //      System.out.println( "I'm being boarded" );
570
```

```
571     if ( oldMyMovementState == PirateState.ENROUTE_TO_PATROL
572         || oldMyMovementState == PirateState.RETURNING_TO_BASE )
573     {
574
575         waitDelay( RETURN_TO_PIRATE_CAMP, boardingTime,
576                 Priority.HIGH );
577
578         System.out.println( "DETECTED AND RELEASED TO CAMP" );
579     }
580
581     if ( oldMyMovementState == PirateState.ATTACKING ||
582          oldMyMovementState == PirateState.PATROLLING )
583     {
584          sensor.interruptAll();
585          myMover.interruptAll();
586          myMover.removeMover(myMover);
587
588          waitDelay( DIE, 0.0, Priority.HIGHEST, sensor );
589          waitDelay( DIE, 0.0, Priority.HIGHEST, myMover);
590          myMover.removeMover(myMover);
591
592          System.out.println( "DETECTED AND APPREHENDED" );
593     }
594 }
595
596 /**
597  * Returns a String containing the type of Player.
598  */
599 @Override
600 public String toString()
601 {
601     return "I am a (" + myMover.getType() + ")";
603 }
604
605 ///*************************REMOVED ALL SETTERS AND GETTERS*****************//
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX B. NAVY MOVER MANAGER JAVA CODE

```
1 /*
2  * NavyShipMoverManager.java
3  */
4 package entities;
5
6 import java.awt.geom.Point2D;
7 import java.util.LinkedList;
8 import simkit.Priority;
9 import simkit.SimEntityBase;
10 import simkit.random.RandomVariate;
11 import simkit.random.RandomVariateFactory;
12 import simkit.smd.CookieCutterSensor;
13 import supplemental.NavyState;
14 import supplemental.Platform;
15 import supplemental.PlatformType;
16
17 /**
18  * Models the behavior of a Navy ship on patrol in the Indian Ocean and Gulf of
19  * Aden
20  *
21  * @version $Id: NavyShipMoverManager.java 199 2013–03–03 06:10:24Z crhutchi $
22  * @author Chad R Hutchins
23  *
24  */
25 public class NavyShipMoverManager extends SimEntityBase {
26
27     /**
28      * Parameters: Contains getters and setters
29      *
30      *
31      */
32     private Platform myMover;
33     private Point2D startLocation;
34     private RandomVariate[] patrolBoxGenerator;
35     private double maxSpeed;
36     private PlatformType platformType;
37     private CookieCutterSensor sensor;
```

```
38  public static final double EPSILON = 1.0E-5;
39  //Scales all distances and speeds for Java Swing. This works for this
40  //particular set of simulations. You need to ensure proper scale of any
41  //area other than the exact same location as this sim.
42  double scale = 0.5;
43  double patrolSpeed = 8 * scale;
44  /**
45   * State Variables: Contains only getters, no setters.
46   */
47  protected NavyState myMovementState;
48  protected double timeOnPatrol;
49  protected double numberPiratesDetected;
50  protected double numberDistressCallRcv;
51  protected Platform target;
52  protected Point2D interceptPoint;
53  protected double timeOfBoarding;
54  /**
55   * String constant for firePropertyChange modification of the state
56   * variable, not visible outside this class
57   *
58   */
59  private final String MY_MOVEMENT_STATE = "myMovementState";
60  private final String TARGET = "target";
61  private final String INTERCEPT_POINT = "interceptPoint";
62  private final String NUMBER_PIRATES_DETECTED = "numberPiratesDetected";
63  private final String NUMBER_DISTRESS_CALL_RCV = "numberDistressCallRcv";
64  /**
65   * String constant for waitDelay method scheduling, visible to other classes
66   *
67   */
68  protected final String MOVE_TO = "MoveTo";
69  protected final String START_PATROLLING = "StartPatrolling";
70  protected final String STOP = "Stop";
71  protected final String SIGNAL_PIRATE = "SignalPirate";
72  protected final String START_INTERCEPT = "StartIntercept";
73
74  /**
75   * String constant for all other cases.
76   *
77   */
78  protected final String PIRATE = "Pirate";
```

```
79
80  /**
81   * Main constructor: Sets mover, sensor, starting location, and patrol box,
82   * id, and max speed of ship
83   *
84   * @param myMover
85   * @param sensor
86   * @param startLocation
87   * @param patrolBoxGenerator
88   * @param maxSpeed
89   */
90  public NavyShipMoverManager( Platform myMover,
91                               CookieCutterSensor sensor,
92                               Point2D startLocation,
93                               RandomVariate[] patrolBoxGenerator,
94                               double maxSpeed )
95  {
96     this.setMyMover( myMover );
97     this.setSensor( sensor );
98     this.setStartLocation( startLocation );
99     this.setPatrolBoxGenerator( patrolBoxGenerator );
100     this.setMaxSpeed(maxSpeed);
101  }
102
103  /**
104   * Default Constructor
105   *
106   */
107  public NavyShipMoverManager()
108  {
109     //Does not set anything
110  }
111
112  /**
113   * Reset: Resets state variables at end of each replication
114   *
115   */
116  @Override
117  public void reset()
118  {
119     super.reset();
```

```
120     myMovementState = NavyState.DEAD_IN_WATER;
121     numberPiratesDetected = 0;
122     numberDistressCallRcv = 0;
123     myMover.setInitialLocation( startLocation );
124     this.target = null;
125     this.interceptPoint = Platform.NaP;
126  }
127
128  /**
129   * Run: FirePropertyChange for all state variables in reset method.
130   * Schedules StarPatrolling.
131   *
132   */
133  public void doRun()
134  {
135     firePropertyChange( MY_MOVEMENT_STATE, getMyMovementState() );
136     firePropertyChange( TARGET, getTarget() );
137     firePropertyChange( INTERCEPT_POINT, getInterceptPoint() );
138     waitDelay( START_PATROLLING, 0.0, Priority.HIGH );
139  }
140
141  /**
142   * StartPatrolling: Changes state to PATROLLING, generates next way point in
143   * patrol box, and schedules MoveTo.
144   *
145   */
146  public void doStartPatrolling()
147  {
148     NavyState oldMyMovementState = getMyMovementState();
149     myMovementState = NavyState.PATROLLING;
150     firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
151                         getMyMovementState() );
152
153     myMover.setMaxSpeed(patrolSpeed);
154
155     Point2D nextWaypoint = new Point2D.Double(
156           patrolBoxGenerator[0].generate(),
157           patrolBoxGenerator[1].generate() );
158
159     waitDelay( MOVE_TO, 0.0, nextWaypoint );
160  }
```

```
161
162     /**
163      * EndMove: Generates nextWayPoint and if myMovementState is PATROLLING it
164      * schedules MoveTo. If myMovementState is INTERCEPTING it schedules MoveTo
165      * with intercept point
166      *
167      * @param mover
168      */
169     public void doEndMove( Platform mover )
170     {
171        Point2D nextWaypoint = new Point2D.Double(
172              patrolBoxGenerator[0].generate(),
173              patrolBoxGenerator[1].generate() );
174
175        if ( myMovementState == NavyState.PATROLLING )
176        {
177           waitDelay( MOVE_TO, 0.0, nextWaypoint );
178        }
179     }
180
181     /**
182      * Detection Event: Detects any mover within the sensor range. If it is a
183      * pirate while PATROLLING it increments numberPiratesDetected, adds pirate
184      * to list of detected pirates, stops the ship, and signals the pirate
185      * (which stops the pirate vessel) by an adapter in main class. Schedules
186      * StartPatrolling after a determined amount of time via a random variate.
187      * Schedules: Stop, SignalPirate, and Start Patrolling.
188      *
189      * @param contact
190      */
191     public void doDetection( Platform contact )
192     {
193        double oldNumberPiratesDetected = getNumberPiratesDetected();
194
195        LinkedList detectedPirates = new LinkedList();
196
197        if ( contact.getType() == PlatformType.PIRATE  &&
198             myMovementState == NavyState.PATROLLING  &&
199             !detectedPirates.contains( contact ) )
200        {
201           detectedPirates.add( contact );
```

```
202
203         numberPiratesDetected = getNumberPiratesDetected() + 1;
204         firePropertyChange( NUMBER_PIRATES_DETECTED,
205                         oldNumberPiratesDetected,
206                         getNumberPiratesDetected() );
207
208         RandomVariate[] timeOfBoardingGenerator = new RandomVariate[ 2 ];
209         timeOfBoardingGenerator[0] = RandomVariateFactory.
210               getInstance ( "Uniform," 1.0, 3.0 );
211
212         timeOfBoarding = timeOfBoardingGenerator[0].generate ();
213
214         contact.waitDelay( "OrderStop," 0.0, Priority.HIGHEST, contact );
215
216         NavyState oldMyMovementState = getMyMovementState ();
217         myMovementState = NavyState.BOARDING;
218         firePropertyChange ( MY_MOVEMENT_STATE, oldMyMovementState,
219               getMyMovementState () );
220
221         waitDelay( STOP, 0.0, Priority.HIGHER, myMover );
222         waitDelay( SIGNAL_PIRATE, 0.0, Priority.HIGHER, contact,
223                   timeOfBoarding );
224
225 //        System.out.println( "Detected you dirty Pirate " + contact.getName()
226 //                  + " by " + myMover.getName() );
227
228         waitDelay( START_PATROLLING, getTimeOfBoarding (), Priority.HIGH );
229
230       }
231    }
232
233    /**
234     * SignalPirate: Signals pirate that it has been detected.
235     *
236     * @param contact
237     *
238     */
239    public void doSignalPirate( Platform contact, double boardingTime )
240    {
241 //       System.out.println( "I see you!!!" );
242       //Does nothing but signals to pirate
```

```
243    }
244
245    /**
246     * RcvDistressCall: Receives call from Merchant using adapter in main class.
247     * If Navy within 40NM increments numberDistressCallRcv. Assumes helo on
248     * board and can respond to distress in less than 30 min.
249     *
250     * @param caller
251     */
252    public void doRcvDistressCall( Platform caller )
253    {
254 //       System.out.println( "Caller: " + caller );
255 //       System.out.println( "Here I come to save the day!!" );
256
257
258
259       double upperBoundCallerX = caller.getCurrentLocation().getX() + 20;
260       double lowerBoundCallerX = caller.getCurrentLocation().getX() - 20;
261       double upperBoundCallerY = caller.getCurrentLocation().getY() + 20;
262       double lowerBoundCallerY = caller.getCurrentLocation().getY() - 20;
263
264
265
266       if (( myMover.getCurrentLocation().getX() <= upperBoundCallerX   &&
267             myMover.getCurrentLocation().getX() >= lowerBoundCallerX  ) &&
268           ( myMover.getCurrentLocation().getY() <=  upperBoundCallerY &&
269             myMover.getCurrentLocation().getY() >= lowerBoundCallerY  )   )
270       {
271          double oldNumberDistressCallRcv = getNumberDistressCallRcv();
272          numberDistressCallRcv = getNumberDistressCallRcv() + 1;
273
274 //           System.out.println("Navy Received Distress Call: " + myMover +
275 //                              " From: " + caller);
276
277          firePropertyChange(NUMBER_DISTRESS_CALL_RCV,
278                oldNumberDistressCallRcv,
279                getNumberDistressCallRcv());
280       }
281    }
282
283    /**
```

```
284     * Returns a String containing the type of Player.
285     */
286    @Override
287    public String toString()
288    {
289        return "(" + myMover.getType() + ")";
290    }
291 ///*************************REMOVED ALL SETTERS AND GETTERS******************//
```

APPENDIX C. MERCHANT MOVER MANAGER JAVA CODE

```
1 /*
2  * MerchantShipMoverManager.java
3  */
4 package entities;
5
6 import java.awt.geom.Point2D;
7 import java.util.LinkedList;
8 import java.util.ListIterator;
9 import simkit.Priority;
10 import simkit.SimEntityBase;
11 import simkit.random.RandomVariate;
12 import simkit.smd.CookieCutterSensor;
13 import supplemental.MerchantState;
14 import supplemental.Platform;
15 import supplemental.PlatformType;
16
17 /**
18  * Models the behavior of merchant traffic in the GOA and Indian Ocean.
19  *
20  * @version $Id: MerchantShipMoverManager.java 70 2012–07–11 15:48:44Z crhutchi
21  * $
22  * @author Chad R Hutchins
23  **/
24 public class MerchantShipMoverManager extends SimEntityBase {
25
26     /**
27      * Parameters. Contains Setters and Getters
28      **/
29     private Platform myMover;
30     private CookieCutterSensor sensor;
31     private Point2D startLocation;
32     private RandomVariate[] pathGenerator;
33     private PlatformType platformType;
34     private LinkedList<Point2D> wayPoint;
35
36     /**
37      * State Variables. Contains only getters, no setters.
```

```
38   **/
39  protected MerchantState myMovementState;
40  protected ListIterator<Point2D> nextWayPointIter;
41  protected double numberPiratesEncountered;
42  protected double numberPiratesEvaded;
43  protected double numberHijacked;
44  protected double numberSuccessfulTransits;
45  protected Point2D wayPointOne;
46  protected Point2D wayPointTwo;
47  protected Point2D wayPointThree;
48  protected Point2D wayPointFour;
49  protected boolean isAlive;
50
51  private double scale = 0.5;
52  private double transitSpeed = 15 * scale;
53
54  /**
55   * String constant for firePropertyChange modification of the state
56   * variable, not visible outside this class
57   **/
58  private final String MY_MOVEMENT_STATE = "myMovementState";
59  private final String NUMBER_PIRATES_ENCOUNTERED =
60                  "numberPiratesEncountered";
61  private final String NUMBER_PIRATES_EVADED = "numberPiratesEvaded";
62  private final String NUMBER_HIJACKED = "numberHijacked";
63  private final String NUMBER_SUCCESSFUL_TRANSITS =
64                "numberSuccessfulTransits";
65  private final String NEXT_WAY_POINT = "nextWaypoint";
66  private final String IS_ALIVE = "isAlive";
67
68  /**
69   * String constant for waitDelay method scheduling, visible to other classes
70   **/
71  protected final String MOVE_TO = "MoveTo";
72  protected final String STOP = "Stop";
73  protected final String ORDER_STOP = "OrderStop";
74  protected final String RADIO_NAVY = "RadioNavy";
75  protected final String DIE = "Die";
76
77  /**
78   * String constant for all other cases.
```

```
79    **/
80   protected final String PIRATE = “Pirate”;
81
82   /**
83    * Main constructor. Sets mover, sensor, starting location, and path
84    *
85    * @param myMover
86    * @param sensor
87    * @param startLocation
88    * @param pathGenerator
89    */
90   public MerchantShipMoverManager( Platform myMover,
91                                    CookieCutterSensor sensor,
92                                    Point2D startLocation,
93                                    RandomVariate[] pathGenerator )
94   {
95      this.setMyMover( myMover );
96      this.setSensor( sensor );
97      this.setStartLocation( startLocation );
98      this.setPathGenerator( pathGenerator );
99   }
100
101   /**
102    * Default constructor
103    **/
104   public MerchantShipMoverManager()
105   {
106   }
107
108   /**
109    * Reset:  Resets state variables at end of each replication
110    **/
111   @Override
112   public void reset()
113   {
114      super.reset();
115      myMovementState = MerchantState.DEAD_IN_WATER;
116      numberPiratesEncountered = 0;
117      numberPiratesEvaded = 0;
118      numberHijacked = 0;
119      wayPoint = wayPoint = new LinkedList<>();;
```

```
120     myMover.setInitialLocation( startLocation );
121     wayPointOne = new Point2D.Double(
122             getPathGenerator()[0].generate(),
123             getPathGenerator()[1].generate() );
124     wayPointTwo = new Point2D.Double(
125             getPathGenerator()[2].generate(),
126             getPathGenerator()[3].generate() );
127     wayPointThree = new Point2D.Double(
128             getPathGenerator()[4].generate(),
129             getPathGenerator()[5].generate() );
130     wayPointFour = new Point2D.Double(
131             getPathGenerator()[6].generate(),
132             getPathGenerator()[7].generate() );
133     isAlive = true;
134
135   }
136
137   /**
138    * Run Event: FirePropertyChange for all state variables in reset method
139    **/
140   public void doRun()
141   {
142     firePropertyChange( MY_MOVEMENT_STATE, getMyMovementState() );
143     firePropertyChange( NUMBER_PIRATES_ENCOUNTERED,
144                         getNumberPiratesEncountered() );
145     firePropertyChange( NUMBER_PIRATES_EVADED, getNumberPiratesEvaded() );
146     firePropertyChange( NUMBER_HIJACKED, getNumberHijacked() );
147     firePropertyChange( IS_ALIVE, getIsAlive());
148
149   }
150
151   /**
152    * Start Transit:  Changes myMovementState to TRANSITTING, set the path and
153    * schedule MoveTo to move merchant to next point on path.
154    **/
155   public void doStartTransit()
156   {
157     MerchantState oldMyMovementState = getMyMovementState();
158     myMovementState = MerchantState.TRANSITTING;
159
160     myMover.setMaxSpeed ( transitSpeed );
```

```
161
162     wayPoint.add( 0, wayPointOne );
163     wayPoint.add( 1, wayPointTwo );
164     wayPoint.add( 2, wayPointThree );
165     wayPoint.add( 3, wayPointFour );
166
167     nextWayPointIter = getWayPoint().
168           listIterator();
169
170     Point2D nextWaypoint = nextWayPointIter.hasNext() ? nextWayPointIter.
171           next() : null;
172
173     firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
174                                 getMyMovementState());
175     firePropertyChange( NEXT_WAY_POINT, nextWaypoint );
176
177     if ( nextWaypoint != null )
178     {
179        waitDelay( MOVE_TO, 0.0, nextWaypoint );
180     }
181  }
182
183  /**
184   * End Move: Checks if at the end of the path. If not it schedules MoveTo,
185   * if it is at the end of that path it stops the merchant.
186   *
187   * @param mover
188   */
189  public void doEndMove( Platform mover )
190  {
191     Point2D next = nextWayPointIter.hasNext() ?
192                 nextWayPointIter.next() : null;
193     firePropertyChange( NEXT_WAY_POINT, next );
194
195     if (myMovementState == MerchantState.TRANSITTING)
196     {
197
198        if (next != null)
199        {
200           waitDelay(MOVE_TO, 0.0, next);
201        }
```

```
202
203         if (next == null)
204         {
205            waitDelay( STOP, 0.0, myMover );
206
207            double oldNumberSuccessfulTransits = getNumberSuccessfulTransits();
208            numberSuccessfulTransits = numberSuccessfulTransits + 1;
209
210            firePropertyChange(NUMBER_SUCCESSFUL_TRANSITS,
211                  oldNumberSuccessfulTransits,
212                  numberSuccessfulTransits);
213         }
214      }
215
216      if (myMovementState == MerchantState.HIJACKED)
217      {
218         //System.out.println("Merchant Location: " +
219         //myMover.getCurrentLocation());
220      }
221   }
222
223
224   /**
225    * Detection: Detects any mover within the sensor range. If it is a pirate
226    * the merchant will radio the Navy, increment numberPiratesEncountered, and
227    * add the pirate to detectedPirates list.
228    *
229    * @param contact
230    */
231   public void doDetection( Platform contact )
232   {
233      LinkedList detectedPirates = new LinkedList();
234
235      if ( contact.getType() == PlatformType.PIRATE &&
236           !detectedPirates.contains(contact) &&
237           (myMovementState == MerchantState.TRANSITTING ||
238            myMovementState == MerchantState.EVADING))
239      {
240         detectedPirates.add(contact);
241
242         double oldNumberPiratesEncountered = getNumberPiratesEncountered();
```

```
243
244         numberPiratesEncountered = getNumberPiratesEncountered() + 1;
245
246         //System.out.println( “I see you Pirate! “ + contact );
247
248         waitDelay( RADIO_NAVY, 0.0, Priority.HIGHER, this.myMover );
249
250         firePropertyChange( NUMBER_PIRATES_ENCOUNTERED,
251                         oldNumberPiratesEncountered,
252                         getNumberPiratesEncountered() );
253     }
254   }
255
256   /**
257    * RadioNavy: Signals nearest Navy vessel for help. This is done via an
258    * adapter in the “main” file.
259    *
260    * @param merchant
261    */
262   public void doRadioNavy( Platform merchant )
263   {
264     MerchantState oldMyMovementState = getMyMovementState();
265     myMovementState = MerchantState.EVADING;
266
267     //System.out.println( “Help me!!!” );
268     //Send message to nearest Navy vessel
269
270     firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
271                     getMyMovementState() );
272   }
273
274   /**
275    * EvadeSuccessfully: Increments numberPiratesEvaded. Merchant continues on
276    * voyage.
277    **/
278   public void doEvadeSuccessfully()
279   {
280     double oldNumberPiratesEvaded = getNumberPiratesEvaded();
281     numberPiratesEvaded = getNumberPiratesEvaded() + 1;
282
283     firePropertyChange( NUMBER_PIRATES_EVADED,
```

```
284                     oldNumberPiratesEvaded,
285                     getNumberPiratesEvaded() );
286
287   }
288
289   /**
290    * Hijacked: Increments numberHijacked. Takes merchant back to pirate base
291    * camp for ransom negotiations.
292    **/
293   public void doHijacked( Platform pirate )
294   {
295      double oldNumberHijacked = getNumberHijacked();
296      numberHijacked = getNumberHijacked() + 1;
297
298      MerchantState oldState = getMyMovementState();
299      myMovementState = MerchantState.HIJACKED;
300
301      isAlive = false;
302
303      myMover.setIsAlive( isAlive );
304
305      waitDelay(STOP, 0.0, Priority.HIGH );
306
307      double pirateCampX = pirate.getInitialLocation ().getX ();
308      double pirateCampY = pirate.getInitialLocation ().getY();
309
310      //If pirate Camp is on GOA
311      if(pirateCampY > 285)
312      {
313         double hijackedIOMerchantX;
314         double hijackedIOMerchantY;
315         hijackedIOMerchantX = pirateCampY + 5;
316         hijackedIOMerchantY = pirateCampX;
317
318         Point2D merchantIOHijackLocation = new Point2D.Double (
319               hijackedIOMerchantX, hijackedIOMerchantY );
320
321         waitDelay ( MOVE_TO, 2.0, merchantIOHijackLocation );
322      }
323      //IF pirate camp is on Indian Ocean
324      else
```

```
325     {
326        double hijackedGOAMerchantX;
327        double hijackedGOAMerchantY;
328
329        hijackedGOAMerchantX = pirateCampX +5;
330        hijackedGOAMerchantY = pirateCampY;
331
332        Point2D merchantGOAHijackLocation = new Point2D.Double (
333              hijackedGOAMerchantX, hijackedGOAMerchantY );
334
335        waitDelay ( MOVE_TO, 2.0, merchantGOAHijackLocation );
336     }
337
338     firePropertyChange( NUMBER_HIJACKED, oldNumberHijacked,
339                    getNumberHijacked() );
340     firePropertyChange( MY_MOVEMENT_STATE, oldState, getMyMovementState() );
341  }
342
343  /**
344   * Returns a String containing the type of Player.
345   *
346   **/
347  @Override
348  public String toString()
349  {
350     return "I am a (" + myMover.getType() + ")";
351  }
352 ///*************************REMOVED ALL SETTERS AND GETTERS******************//
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX D. BAYLA PIRATE DEPARTURE PROCESS JAVA CODE.

```
1 /*
2  * PirateGoaDepartureProcess.java
3  */
4 package process;
5
6 import simkit.SimEntityBase;
7 import simkit.random.RandomVariate;
8
9 /**
10  * Generates departure times for pirates leaving the Gulf of Aden(GOA).
11  *
12  * @version $Id: BaylaPirateDepartureProcess.java 168 2013–02–14 06:59:16Z crhutchi $
13  * @author Chad R Hutchins
14  */
15 public class BaylaPirateDepartureProcess extends SimEntityBase {
16
17     /**
18      * Parameters. Contains Setters and Getters
19      **/
20     private RandomVariate IoDepartureTimeGenerator; //Generates depature times
21
22     /**
23      * State Variables. Contains only getters, no setters.
24      **/
25     protected int numberDepartedIO;
26
27     /**
28      * String constant for firePropertyChange modification of the state
29      * variable, not visible outside this class
30      **/
31     private final String NUMBER_DEPARTED_IO = “numberDepartedIO”;
32
33     /**
34      * String constant for waitDelay method scheduling, visible to other classes
35      **/
36     protected final String DEPART = “Depart”;
37
```

```
38  /**
39   * Main constructor. Sets IoDepartureTimeGenerator.
40   *
41   * @param rv The RandomVariate instance for DepartureTimeGeneratorSB times
42   */
43  public BaylaPirateDepartureProcess( RandomVariate rv )
44  {
45     this.setIoDepartureTimeGenerator( rv );
46  }
47
48  /**
49   * Reset Event:  resets all state variables after each replication.
50   */
51  @Override
52  public void reset()
53  {
54     super.reset();
55     numberDepartedIO = 0;
56  }
57
58  /**
59   * Run Event: Initial event - put on event list at the start of e run.
60   * State Transition: in reset() Schedule: First LeaveCampIo event with
61   * departureTime delay
62   */
63  public void doRun()
64  {
65     firePropertyChange( NUMBER_DEPARTED_IO, getNumberDepartedIO() );
66
67     waitDelay( DEPART, IoDepartureTimeGenerator.generate() );
68  }
69
70  /**
71   * LeaveGoaPirateCamp Event: increments numberDepartedSB and schedules
72   * it's self with delay of departureTime.
73   */
74  public void doDepart()
75  {
76     int oldState = getNumberDepartedIO();
77     numberDepartedIO = getNumberDepartedIO() + 1;
78     firePropertyChange( NUMBER_DEPARTED_IO, oldState,
```

```
79                      getNumberDepartedIO() );
80
81      //**Comment for visual testing**//
82      waitDelay( DEPART, IoDepartureTimeGenerator.generate() );
83    }
84
85    /**
86     * @return the IoDepartureTimeGenerator
87     */
88    public RandomVariate getIoDepartureTimeGenerator()
89    {
90      return IoDepartureTimeGenerator;
91    }
92
93    /**
94     * @param IoDepartureTimeGenerator the IoDepartureTimeGenerator to set
95     */
96    public void setIoDepartureTimeGenerator(
97          RandomVariate goaDepartureTimeGenerator )
98    {
99      this.IoDepartureTimeGenerator = goaDepartureTimeGenerator;
100   }
101
102   /**
103    * @return the numberDepartedIO
104    */
105   public int getNumberDepartedIO()
106   {
107     return numberDepartedIO;
108   }
109 }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX E. BAYLA PIRATE CAMP JAVA CODE.

```
1 /*
2  * BaylaPirateCamp.java
3  */
4 package process;
5
6 import entities.PirateMoverManager;
7 import java.util.Arrays;
8 import java.util.LinkedList;
9 import simkit.Priority;
10 import simkit.SimEntityBase;
11
12 /**
13  *
14  * @author Chad R Hutchins
15  *
16  */
17 public class BaylaPirateCamp extends SimEntityBase
18 {
19
20    private PirateMoverManager[] pirateMM;
21
22    protected LinkedList<PirateMoverManager> myPirates;
23    protected int numberDepartedIO;
24
25    /**
26     * String constant for firePropertyChange modification of the state
27     * variable, not visible outside this class
28     **/
29    private final String NUMBER_DEPARTED_IO = "numberDepartedIO";
30
31    /**
32     * String constant for waitDelay method scheduling, visible to other classes
33     **/
34    protected final String LEAVE = "Leave";
35    protected final String LEAVE_IO_PIRATE_CAMP = "LeaveIoPirateCamp";
36
37    public BaylaPirateCamp( PirateMoverManager[] pirateMM )
```

```
38  {
39     this.setPirateMM(pirateMM);
40     this.myPirates = new LinkedList<PirateMoverManager>();
41  }
42
43  /**
44   * Reset Event:  resets all state variables after each replication.
45   */
46  @Override
47  public void reset()
48  {
49     super.reset();
50     numberDepartedIO = 0;
51     myPirates.clear();
52     myPirates.addAll(Arrays.asList( pirateMM));
53  }
54
55  public void doRun()
56  {
57     //firePropertyChange( NUMBER_DEPARTED_IO, getNumberDepartedIO() );
58  }
59
60
61  public void doDepart()
62  {
63     if( !myPirates.isEmpty() )
64     {
65        //System.out.println("myPirate size: " + myPirates.size());
66        waitDelay(LEAVE, 0.0);
67     }
68  }
69
70  public void doLeave()
71  {
72     PirateMoverManager p = myPirates.removeFirst();
73     p.waitDelay( LEAVE_IO_PIRATE_CAMP, 0.0, Priority.HIGH );
74
75     int oldState = getNumberDepartedIO();
76     numberDepartedIO = getNumberDepartedIO() + 1;
77     firePropertyChange( NUMBER_DEPARTED_IO, oldState,
78                    getNumberDepartedIO() );
```

```
79
80 //       System.out.println(
81 //               "Number Pirate Departures from Bayla " +
82 //                       getNumberDepartedIO() );
83
84    }
85
86    /**
87     * @return the myPirates
88     */
89    public LinkedList<PirateMoverManager> getMyPirates() {
90       return myPirates;
91    }
92
93    /**
94     * @return the numberDepartedIO
95     */
96    public int getNumberDepartedIO() {
97       return numberDepartedIO;
98    }
99
100    /**
101     * @return the pirateMM
102     */
103    public PirateMoverManager[] getPirateMM() {
104       return pirateMM.clone();
105    }
106
107    /**
108     * @param pirateMM the pirateMM to set
109     */
110    public void setPirateMM(PirateMoverManager[] pirateMM) {
111       this.pirateMM = pirateMM.clone();
112    }
113
114 }
115
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX F. SUEZ TO OMAN MERCHANT DEPARTURE JAVA CODE

```
1 /*
2  * SuezToOmanDepartureProcess.java
3  */
4 package process;
5
6 import simkit.SimEntityBase;
7 import simkit.random.RandomVariate;
8
9 /**
10  * Generates departure times for merchants sailing out of the Suez to Oman.
11  *
12  * @version $Id: SuezToOmanMerchantDepartureProcess.java 169 2013–02–14
13  * 20:56:17Z crhutchi $
14  * @author Chad R Hutchins
15  */
16 public class SuezToOmanMerchantDepartureProcess extends SimEntityBase {
17
18     /**
19      * Parameters. Contains Setters and Getters
20      **/
21     //Generates depature times
22     private RandomVariate merchantDepartureTimeGenerator;
23
24     /**
25      * State Variables. Contains only getters, no setters.
26      **/
27     protected int numberDeparted;
28
29     /**
30      * String constant for firePropertyChange modification of the state
31      * variable, not visible outside this class
32      **/
33     private final String NUMBER_DEPARTED = “numberDeparted”;
34
35     /**
36      * String constant for waitDelay method scheduling, visible to other classes
37      **/
```

```
38  protected final String DEPART = “Depart”;
39
40  /**
41   * Main constructor. Sets merchantDepartureTimeGenerator.
42   *
43   * @param rv The RandomVariate instance for DepartureTimeGeneratorSB times
44   */
45  public SuezToOmanMerchantDepartureProcess( RandomVariate rv )
46  {
47     this.setMerchantDepartureTimeGenerator( rv );
48  }
49
50  /**
51   * Reset Event:  resets all state variables after each replication.
52   */
53  @Override
54  public void reset()
55  {
56     super.reset();
57     numberDeparted = 0;
58  }
59
60  /**
61   * Run Event: Initial event - put on event list at the start of e run.
62   * State Transition: in reset() Schedule: First LeaveCampIo event with
63   * departureTime delay
64   */
65  public void doRun()
66  {
67     firePropertyChange( NUMBER_DEPARTED, getNumberDeparted() );
68
69     waitDelay( DEPART, merchantDepartureTimeGenerator.generate() );
70  }
71
72  /**
73   * Depart Event: increments numberDeparted and schedules
74   * it’s self with delay of departureTime.
75   */
76  public void doDepart()
77  {
78     int oldState = getNumberDeparted();
```

```
79      numberDeparted = getNumberDeparted() + 1;
80      firePropertyChange( NUMBER_DEPARTED, oldState,
81                      getNumberDeparted() );
82
83      //**Comment for visual testing**//
84      waitDelay( DEPART, merchantDepartureTimeGenerator.generate() );
85    }
86
87    /**
88     * @return the merchantDepartureTimeGenerator
89     */
90    public RandomVariate getMerchantDepartureTimeGenerator()
91    {
92      return merchantDepartureTimeGenerator;
93    }
94
95    /**
96     * @param merchantDepartureTimeGenerator the merchantDepartureTimeGenerator
97     * to set
98     */
99    public void setMerchantDepartureTimeGenerator(
100           RandomVariate merchantDepartureTimeGenerator )
101     {
102       this.merchantDepartureTimeGenerator = merchantDepartureTimeGenerator;
103     }
104
105     /**
106      * @return the numberDeparted
107      */
108     public int getNumberDeparted()
109     {
110       return numberDeparted;
111     }
112 }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX G. SUEZ TO OMAN ORIGIN PORT JAVA CODE

```
1 /*
2  * SuezToOmanOrginPort.java
3  */
4 package process;
5
6 import entities.MerchantShipMoverManager;
7 import java.util.Arrays;
8 import java.util.LinkedList;
9 import simkit.Priority;
10 import simkit.SimEntityBase;
11
12 /**
13  * Port of Origin for merchants sailing from Suez to Oman.
14  *
15  * @author Chad R Hutchins
16  *
17  */
18 public class SuezToOmanOriginPort extends SimEntityBase
19 {
20    private MerchantShipMoverManager[] merchantMM;
21
22    protected LinkedList<MerchantShipMoverManager> myMerchants;
23    protected int numberDepartedPort;
24
25    /**
26     * String constant for firePropertyChange modification of the state
27     * variable, not visible outside this class
28     **/
29    private final String NUMBER_DEPARTED_PORT = "numberDepartedPort";
30
31    /**
32     * String constant for waitDelay method scheduling, visible to other classes
33     **/
34    protected final String LEAVE = "Leave";
35    protected final String START_TRANSIT = "StartTransit";
36
37    public SuezToOmanOriginPort( MerchantShipMoverManager[] merchantMM )
```

```
38  {
39     this.setMerchantMM(merchantMM);
40     this.myMerchants = new LinkedList<MerchantShipMoverManager>();
41
42  }
43
44  /**
45   * Reset Event:  resets all state variables after each replication.
46   */
47  @Override
48  public void reset()
49  {
50     super.reset();
51     numberDepartedPort = 0;
52     myMerchants.clear();
53     myMerchants.addAll(Arrays.asList(getMerchantMM()));
54  }
55
56  public void doRun()
57  {
58     //firePropertyChange( NUMBER_DEPARTED_PORT, getNumberDepartedPort() );
59  }
60
61
62  public void doDepart()
63  {
64     if( !myMerchants.isEmpty() )
65     {
66        waitDelay(LEAVE, 0.0);
67     }
68  }
69
70  public void doLeave()
71  {
72     MerchantShipMoverManager m = myMerchants.removeFirst();
73     m.waitDelay( START_TRANSIT, 0.0, Priority.HIGH );
74
75     int oldState = getNumberDepartedPort();
76     numberDepartedPort = getNumberDepartedPort() + 1;
77     firePropertyChange( NUMBER_DEPARTED_PORT, oldState,
78                    getNumberDepartedPort() );
```

```
79
80 //       System.out.println(
81 //               “Number Merchant Ship Departures from SuezToMaldives Port “ +
82 //                       getNumberDepartedPort() );
83
84    }
85
86    /**
87     * @return the myMerchants
88     */
89    public LinkedList<MerchantShipMoverManager> getMyMerchants() {
90       return myMerchants;
91    }
92
93    /**
94     * @return the numberDepartedPort
95     */
96    public int getNumberDepartedPort() {
97       return numberDepartedPort;
98    }
99
100    /**
101     * @return the merchantMM
102     */
103    public MerchantShipMoverManager[] getMerchantMM() {
104       return merchantMM.clone();
105    }
106
107    /**
108     * @param merchantMM the merchantMM to set
109     */
110    public void setMerchantMM(MerchantShipMoverManager[] merchantMM) {
111       this.merchantMM = merchantMM.clone();
112    }
113
114 }
115
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX H. MMOWGLI ACTION PLAN 16: TRANSIT LANE PATROLS BY INTERNATIONAL NAVIES

URL: https://mmowgli.nps.edu/piracy/reports/ActionPlanList_Piracy2012.html#ActionPlan16

Action Plan 16

ID

Action Plan 16 for **piracyMMOWGLI 2012**

Description

Transit Lane Patrols by International Navies

Rating

3.0 "thumbs up" average score from 0 to 3

Idea Card Chain

Idea Card Chain 504 started by player *Banaadirre:* It seems logical for the Navy to operate solely on that transit lane and the IRTC.

Who Is Involved

International navies, merchant mariners, and IMB. EU, NATO, CTF (AKA the "big 3") are going to need to coordinate as "the big 1."

What Is It

It is in the best interest of merchant mariners to get from port to port using the shortest possible distance. If IMB would approve an "IRTC" like transit lane that extends to Oman and Maldives the navies could set up patrols on those lanes as they do the IRTC. (See Image of proposed transit lanes).

What Will It Take

Merchants transitting only via preferred transit lanes and navies organizing patrol boxes to operate solely in the these transit lanes. If merchants have to travel outside these lanes they will need to either coordinate for a convoy or use onboard armed security.

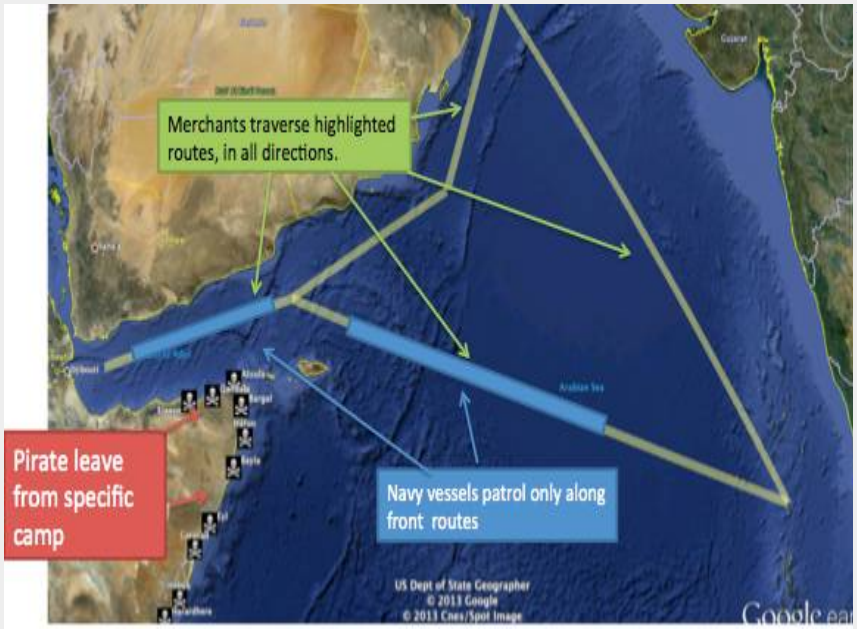
How Will It Change Things

Cuts down on the amount of ocean required to patrol. Keeps mariners safe by focusing all naval attention to designated transit lanes. It is a more passive option for those who do not want to get the navies involved on land.

Authors

LawDawg, gm chad, Banaadirre

Image

1		https://mmowgli.nps.edu/piracy/images/16/Optimized-TransitLane.jpg

(From Piracy MMOWGLI 2012 Action Plan Report)

APPENDIX I. MMOWGLI ACTION PLAN 6: NAVAL QUARANTINE OF SOUTHEASTERN SOMALIA COAST CAN PREVENT SUCCESSFUL PIRATE CAPTURE AND RANSOM OF HOSTAGE VICTIMS AND MERCHANT SHIPS.

URL: https://mmowgli.nps.edu/piracy/reports/ActionPlanList_Piracy2012.html#ActionPlan6

Action Plan 6

ID

Action Plan 6 for **piracyMMOWGLI 2012**

Description

Naval Quarantine of southeastern Somalia coast can prevent successful pirate capture and ransom of hostage victims and merchant ships.

Rating

2.7 "thumbs up" average score from 0 to 3

Idea Card Chain

Idea Card Chain 209 started by player *EdwardPreble:* A naval quarantine along the southern Somali coast can prevent captured ships from returning to pirate havens for ransom

Who Is Involved

Combined maritime forces and the merchant marine industry can cooperate directly. Large commercial ships above an agreed-upon tonnage (which are easily detected using AIS, radar or remote sensing) are considered to be commandeered against their will unless they have registered their intent to visit Somalia prior to approaching the 200nm limit.

What Is It

Naval forces can significantly reduce patrol and response requirements by establishing a naval quarantine on large merchant vessels along the southern Somalia coastline. Unless it has filed prior notification of intent, merchant ships approaching within 200 nautical miles of shore are considered pirate captives and in need of rescue. Naval intervention on the high seas can prevent captured ships from reaching pirate camps, where hostage ransom negotiations can take years to resolve.

What Will It Take

Merchant ships within 200 nm of the Somali coastline are considered captured, and naval forces can intervene to prevent

hostages being held ransom ashore. Needed: reporting mechanism for commercial ships to combined maritime forces. Other aspects of this simple plan fit well with current naval operations, simplifying detection of piracy capture. Pirates have no way to reinforce and are contained within the vessel until they surrender. International law then takes over.

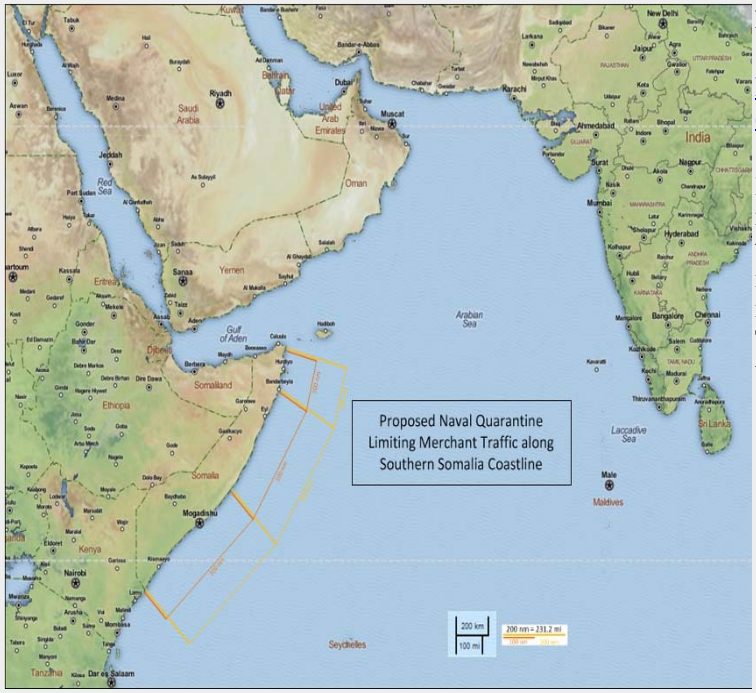
How Will It Change Things

Reduced cost and greater effectiveness for naval forces. Reduced risk and greater protection for merchant ships. Greatly reduced protection and income for pirates, undercutting their profits and business model. Criminal threats against the crew are possible at sea or ashore - international forces are able to act against pirates with much greater impact while at sea.

Authors

EdwardPreble, gm_becca, LawDawg, briefer, WillyRobert, Banaadirre

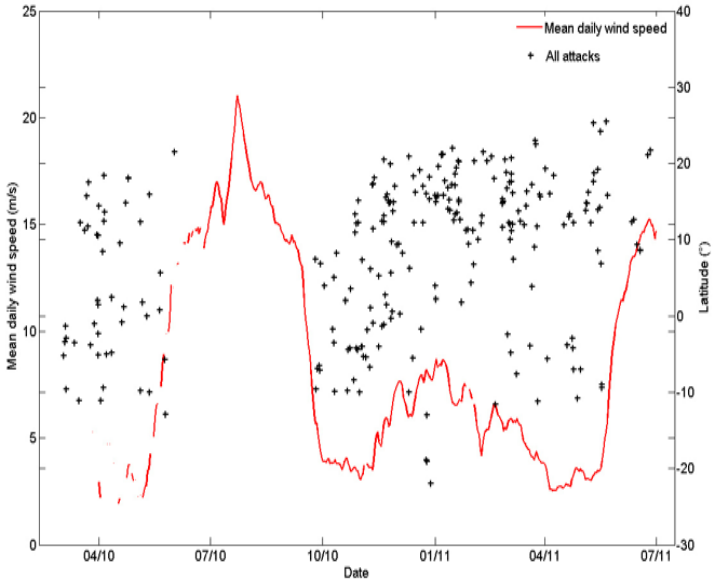
Images



Proposed Naval Quarantine of Southern Somali Coastline

https://mmowgli.nps.edu/piracy/images/6/NavalQuarantineSouthernSomalia.reduced.png

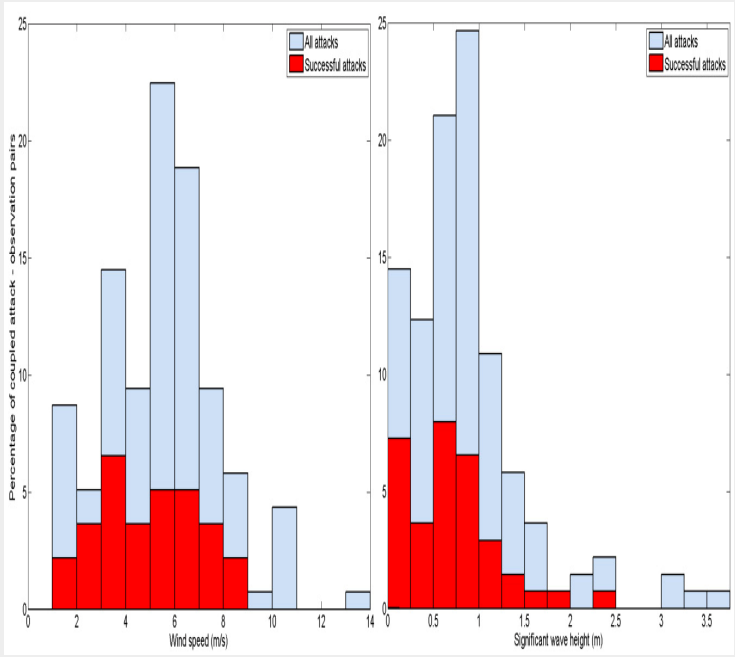
Naval forces can significantly reduce patrol and response requirements by establishing a naval quarantine on large merchant vessels along the southern Somalia coastline. Unless it has filed prior notification of intent, merchant ships approaching within 200 nautical miles of shore are considered pirate captives and in need of rescue. This prevents ships from reaching port where hostage ransoms can take years to resolve.



Satellites and Piracy on the High Seas: Wind Speed and Pirate Attacks

http://www.esa.int/images/Wind_speed_and_attacks_H.jpg

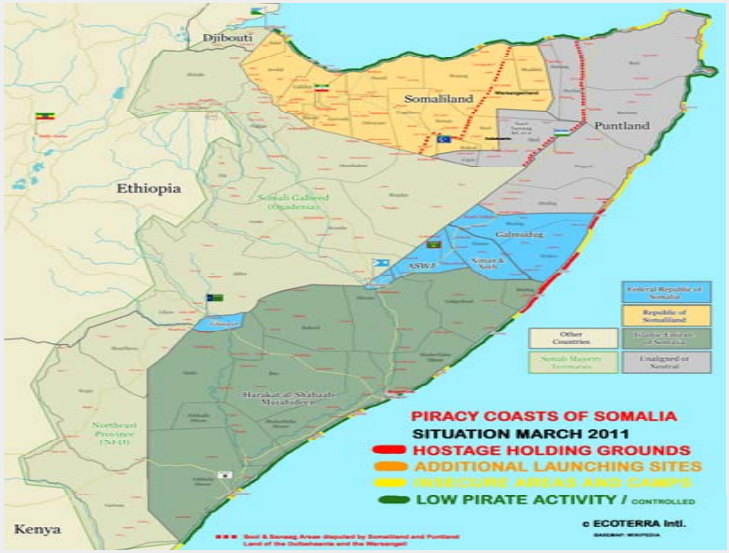
Mean daily wind speed at Socotra (Yemen) and pirate attacks by latitude for April 2010 to July 2011. When the wind speed dropped, pirate attacks increased. Credits: D. Cook, S. Garrett and M. Rutherford, 2011.



Satellites and Piracy on the High Seas: Wave Height and Pirate Attacks

http://www.esa.int/images/Wave_height_and_attackes_H.jpg

Satellite observations of wind speed (left) and significant wave height (right) for 2010–2011 attempted and successful pirate attacks off Somalia. The GlobWave databases provided observations of significant wave height and surface wind speed for 54% of all pirate attacks. Wind speeds during pirate attacks were mainly low but once wind speeds exceeded 9 m/s, no successful attacks occurred. Nearly all piracy was in seas with wave height less than 2.5 m, and most attacks were conducted in calm oceans with waves less than 1 m in height. No successful attacks occurred on days where wave height exceeded 2.5 m. Credits: D. Cook, S. Garrett and M. Rutherford, 2011.



Piracy Coasts of Somalia: Situation March 2011

https://mmowgli.nps.edu/piracy/images/6/Somalia_Piracy_Camps.png

Political map, hostage holding grounds, launching sites, and other information.

Video

Dangerous Waters

https://www.youtube.com/watch?v=tb0R1JVvzic

STORY: The waters off Somalia are the most dangerous in the world. Piracy has flourished in lawless Somalia since the collapse of central government 17 years ago. In an effort to combat the problem, the U.N. Security Council earlier this year passed a resolution allowing foreign warships to enter Somalia's territorial waters to fight piracy. But it hasn't made Somali waters any safer. Attacks at sea have soared this year. This is the pirate's base - Eyl is a lawless former fishing outpost, part of the self-declared autonomous Puntland region within Somalia. The Puntland authorities are critical of foreign efforts to stamp out piracy. [Abdul-Kadir Yusuf Muse, Puntland Region Fishing and Ports Assistant Minister]: "We know they have been given full mandate by the security council to intervene when the pirates strikes on Somali waters." The Puntland authorities want the United Nations to set up an international force to police Somali territorial waters. Dozens of ships have been hijacked for ransom this year. It's a lucrative business. Most captured vessels fetch thousands sometime millions of dollars in ransoms. Hostages are usually treated well. Shipping companies are urged not to pay...but most do. On Thursday, a German ship and Japanese tanker were freed along with their crew, but pirates are currently holding about 10 ships for ransom and more than 130 crew members.

Author-to-Author Chat Messages

1	Monday, 25 June 2012 11:36:38-PDT	*LawDawg:* Would you include the UN in this?
2	Saturday, 30 June 2012 09:20:19-PDT	*EdwardPreble:* not sure. thanks for initial setup - finally had a chance to elaborate this plan. maybe we should explore UN and diplomatic issues during Rule of Law discussions.
3	Friday, 6 July 2012 10:39:43-PDT	*gm_donb:* Needed: openly available maps of where ships are being held for ransom, and tracks taken when captured ships are brought back to Somaiia by the pirates.
4	Tuesday, 17 July 2012 10:54:38-PDT	*LawDawg:* Would the use of weather ballons be beneficial in this scenario? They would be less expensive then maintaining a multi-force naval presence to quarantine the area, is perceived as 'less threatening' by pirates (and thus helps "protect" the hostages), and would probably require less political will to put into action.
5	Monday, 23 July 2012 16:16:30-PDT	*LawDawg:* http://www.esa.int/esaEO/SEMATD8X73H_index_0.html This article explores how environmental conditions limit pirate activity. Conclusions show that wave hight and pirate attacks were correlated as well as wind speed and pirate activty. (Once wind speeds exceeded 9 m/s, no successful attacks occurred. Nearly all piracy was in seas with wave height less than 2.5 m, and most attacks were conducted in calm oceans with waves less than 1 m in height. No successful attacks occurred on days where wave height exceeded 2.5 m.) Weather patterns (and proper weather balloon placement) could help determine the correct boundaries for the naval quarantine.
6	Tuesday, 24 July 2012 13:35:56-PDT	*WillyRobert:* interesting and something that we can without a doubt simulate! Thanks.
7	Sunday, 29 July 2012 11:46:21-PDT	*WillyRobert:* As I'm working on a model for this, we need to consider how we handle patrols around Socotra Islands. It is within the 200NM zone, but thinking we need to add units between it and Somalia which stops easy access to this key location. I am thinking at least 2 units need to be placed on the inside of the 200NM zone and between the island and Somali mainland. Thoughts??
8	Sunday, 29 July 2012 11:47:44-PDT	*WillyRobert:* I'll hopefully have a pic up within the next day or so to give example of what I'm thinking.
9	Thursday, 2 August 2012 09:32:52-PDT	*LawDawg:* I think the Socotra Islands would make an potential "check point" in the quarantine. Obviously shippers don't want to navigate around it (greater fuel costs, etc.) but with proper tracking and reporting it would be known when ships travel through this particular area. This could result in increased vigulance on the part of naval ships enforcing the quarantine. As for those who don't report or check in, enter at your own risk.
10	Thursday, 2 August 2012 09:33:30-PDT	*LawDawg:* This could all be enforced through insurance rates as well...something to consider.
11	Thursday, 2 August 2012 11:56:04-PDT	*LawDawg:* I'm working on information sharing and coordination efforts which could tie nicely into this. I would also look into the where Lloyds of London specifically defines their War Risk Zone for that area. Could hold some impliacations for placement of naval vessels.

12	Friday, 23 November 2012 13:33:45-PST	*gm_donb:* The maps of pirate camps don't really pertain to this plan. They should be in separate plans for each pirate camp.
13	Friday, 23 November 2012 16:56:59-PST	*gm_donb:* Multiple separate idea cards and action plans have been spun off for each pirate camp.
14	Thursday, 31 January 2013 21:40:05-PST	*WillyRobert:* Will the patrols in the IRTC remain the same? Or will they more of a quarantine role as well?

Player Comments

1	Saturday, 30 June 2012 08:32:02-PDT	*EdwardPreble:* Smaller ships might also seek protection by registering prior intent to NEVER cross the quarantine barrier. This allows naval forces to have a clear indication of a smaller ship's intent if it appears to be heading towards a pirate sanctuary.
2	Sunday, 1 July 2012 19:51:31-PDT	*Finius Stormfroth:* Boarding ships full of hostages at sea is a risky business. Does the quarantine continue if the pirates execute hostages or rig ships to sink to deter rescue attempts?
3	Monday, 2 July 2012 13:48:56-PDT	*EdwardPreble:* Executing hostages and sinking ransomed ships can also occur while the ship is held at a pirate camp ashore. So it is always a pirate option. The difference in the situation is that pirate captors have no shore infrastructure at sea, no help from other pirates, no communications with the crime bosses, and no other exit (for themselves personally) besides capture by naval forces.
4	Thursday, 31 January 2013 21:41:51-PST	*WillyRobert:* Should we consider this along Northern Somalia too? Not just Southern? It wouldn't be much different than the normal IRTC patrols.

(From Piracy MMOWGLI 2012 Action Plan Report)

APPENDIX J. MMOWGLI ACTION PLAN 9: PIRATE CAMP OPERATIONS

URL: https://mmowgli.nps.edu/piracy/reports/ActionPlanList_Piracy2012.html#ActionPlan9

ACTION PLAN 9

ID

Action Plan 9 for piracyMMOWGLI 2012

Title

How vulnerable are pirate camps at Eyl Somalia to naval quarantine or hostage rescue?

Rating

1.5 ”thumbs up” average score from 1 to 3

Idea Card Chain

Idea Card Chain 480 started by player *EdwardPreble:* It will be interesting to look at each publicly reported pirate camp to see how vulnerable they are to recapture of hostages.

Who Is Involved

Combined maritime forces, EU, NATO, DoS, DoJ, African Union. These are most of the “players” involved, however, the exact mix and other agency involvement is dependent on other policy mandates.

What Is It

Eyl Somalia has been publicly identified as a place where pirates keep hostages and hold ships ransom. For more details see card 482 . Naval assets and other law enforcement agencies actively patrolling and disrupting pirate activities on shore or before pirates reach international waters. It could be as aggressive as the EU bombings of pirate camps (http://www.bloomberg.com/news/2012–05–15/eu-navy-destroys-somali-pirates-supplies-in-shore-attack-1-.html) or like the French hostage rescue from Somali pirates (http://articles.washingtonpost.com/2008–04–12/world/36840240_1_somali-pirates-semiautonomous-puntland-region-french-luxury-yacht). Or it could simply be more passive as a deterrent for pirates by having naval ships patrolling within view of the shorelines.

What Will It Take

It will need persistent ISR assets patrolling the Somali coasts, identifying actual pirates from fishermen. INTEL is continuously needed to track pirate activity on shore and notifying task force commanders of probable pirate activity.

How Will It Change Things

It stops pirates from leaving the shores and getting into international waters. It also allows for a deterrent effect and a means to train Somali coast guard.

Authors

LawDawg, gm_chad, EdwardPreble, gm_becca, WillyRobert

Images

1

Pirate camps identified in public press

"GIS & Satellite: Applications for Piracy-Monitoring" by Josh Lyons, Freedom From Fear magazine, 17 July 2012. https://mmowgli.nps.edu/piracy/images/9/SomaliaPirateCamps.png

2

Horn of Africa, Socotra Island, Garaad, Eyl Somalia
Horn of Africa closeup showing one northern camp at Garaad Somalia, Socotra Island (Yemen) and eastern camp at Eyl Somalia.
https://mmowgli.nps.edu/piracy/images/9/HornOfAfricaSocotraIslandGaraadEylSomalia.png

---

3

Shoreline Eyl Somalia: Dinghies, Merchant Ship
More information on Eyl Somalia can be found on Wikipedia at http://en.wikipedia.org/wiki/Eyl
https://mmowgli.nps.edu/piracy/images/9/ShorelineEylSomaliaDinghiesMerchantShip.png

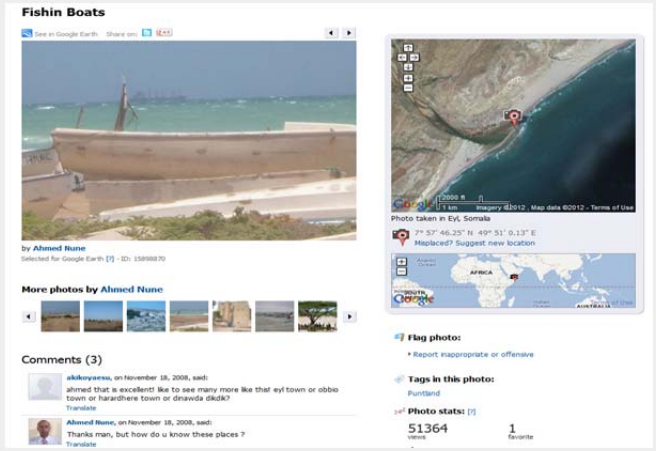
---

4

Shoreline Eyl Somalia: Dinghies On Sand
Port facilities could not be much simpler, skiffs are dragged up on the sand. Not a single pier is present. Captive freighters are kept offshore at anchor.
https://mmowgli.nps.edu/piracy/images/9/ShorelineEylSomaliaDinghiesOnSand.reduced.png

---

5



Fishin Boats (near Eyl) with Freighter in Background
Publicly posted photograph gives beach perspective of fishing boats, also shows freighter just offshore
http://www.panoramio.com/photo/15898870
https://mmowgli.nps.edu/piracy/images/9/FishinBoatsFreighterInBackground0.png

(From Piracy MMOWGLI 2012 Action Plan Report)

APPENDIX K. PIRATE CAMP OPERATIONS SIMKIT ASSEMBLY

```
1 /*
2  * PirateCampOperations.java
3  */
//****************Imports Removed*****************************************//
58
59 /**
60  *
61  * @author Chad R Hutchins
62  * @version $Id:
63  */
64 public class PirateCampOperations {
65
66     /**
67      * @param args the command line arguments
68      */
69     public static void main(String[] args) {
70       //******Constants for Scenario*************************************//
71
72        //**Simulation specific contants**//
73        double simTime = 730.0;//1 Month //2208.0;// 3 months//8765.81 = 1 year;
74        double scaleDistance = 0.5;  //scales the distances in the simulation
75
76        //**Pirate Constants**//
77        int numElaayoPirates = 6;
78        int numQandalaPirates = 8;
79        int numAluulaPirates = 6;
80        int numBargalPirates = 6;
81        int numHafunPirates = 8;
82        int numBaylaPirates = 6;
83        int numEylPirates = 6;
84        int numGaracadPirates = 8;
85        int numHobyoPirates = 6;
86        int numHarardherePirates = 8;
87        double pirateMaxSpeed = 15 * scaleDistance;
88        double pirateVisualSensorRange = 15 * scaleDistance;
89        //**Navy Constants**//
90        int numIoNavyShips = 7;
91        int numGoaNavyShips = 3;
92        double navyMaxSpeed = 30.0 * scaleDistance;
93        double navySurfaceRadarRange = 25 * scaleDistance;
94        //**Merchant Constants**//
95        int numSuezToOmanMerchants = 370;
96        int numSuezToMaldivesMerchants = 370;
97        int numOmanToSuezMerchants = 370;
98        int numOmanToMaldivesMerchants = 370;
99        int numMaldivesToSuezMerchants = 370;
100        int numMaldivesToOmanMerchants = 370;
101        double merchantSurfaceRadarRange = 20 * scaleDistance;
102        double merchantMaxSpeed = 20 * scaleDistance;
103
104        //**Probability Distribution Constants**//
105        double elaayoInterarrivalTimeLambda = 150.0;
106        double qandalaInterarrivalTimeLambda = 100.0;
107        double aluulaInterarrivalTimeLambda = 150.0;
108        double bargalInterarrivalTimeLambda = 150.0;
109        double hafunInterarrivalTimeLambda = 100.0;
```

```
110         double baylaInterarrivalTimeLambda = 150.0;
111         double eylInterarrivalTimeLambda = 150.0;
112         double garacadInterarrivalTimeLambda = 100.0;
113         double hobyoInterarrivalTimeLambda = 150.0;
114         double harardhereInterarrivalTimeLambda = 100.0;
115         double stoInterarrivalTimeLambda = 2.2;
116         double stmInterarrivalTimeLambda = 2.21;
117         double otsInterarrivalTimeLambda = 2.22;
118         double otmInterarrivalTimeLambda = 2.23;
119         double mtsInterarrivalTimeLambda = 2.24;
120         double mtoInterarrivalTimeLambda = 2.25;
121         double probOfAttackingDecision = 0.75;
122         double minUnsuccessfulAttackTime = 0.1;
123         double maxUnsuccessfulAttackTime = 0.75;
124
125
126 //******Constants FOR INTIIAL LOCATIONS OF ENTITIES*****************************//
127         //**Pirate Camps**//
128         Point2D pirateCampElaayo = new Point2D.Double( 306.0, 301.0 );
129         Point2D pirateCampQandala = new Point2D.Double( 339.0, 310.0 );
130         Point2D pirateCampAluula = new Point2D.Double( 367.0, 323.0 );
131         Point2D pirateCampBargal = new Point2D.Double( 379.0, 300.0 );
132         Point2D pirateCampHafun = new Point2D.Double( 384.0, 273.0 );
133         Point2D pirateCampBayla = new Point2D.Double( 370.0, 240.0 );
134         Point2D pirateCampEyl = new Point2D.Double( 345.0, 183.0 );
135         Point2D pirateCampGaracad = new Point2D.Double( 322.0, 155.0 );
136         Point2D pirateCampHobyo = new Point2D.Double( 305.0, 103.0 );
137         Point2D pirateCampHarardhere = new Point2D.Double( 283.0, 79.0 );
138
139         //**Navy Ships**//
140         Point2D initialLocationNavyPB1 = new Point2D.Double( 294.0, 325.0 );
141         Point2D initialLocationNavyPB2 = new Point2D.Double( 331.0, 337.0 );
142         Point2D initialLocationNavyPB3 = new Point2D.Double( 365.0, 346.0 );
143         Point2D initialLocationNavyPB4 = new Point2D.Double( 408.0, 313.0 );
144         Point2D initialLocationNavyPB5 = new Point2D.Double( 410.0, 276.0 );
145         Point2D initialLocationNavyPB6 = new Point2D.Double( 396.0, 243.0 );
146         Point2D initialLocationNavyPB7 = new Point2D.Double( 370.0, 185.0 );
147         Point2D initialLocationNavyPB8 = new Point2D.Double( 344.0, 154.0 );
148         Point2D initialLocationNavyPB9 = new Point2D.Double( 330.0, 101.0 );
149         Point2D initialLocationNavyPB10 = new Point2D.Double( 310.0, 76.0 );
150
151         //**Merchant Ships starting**//
152         Point2D initialLocationMerchantSuezToMaldives =
153                 new Point2D.Double( 145.0, 345.0 );
154         Point2D initialLocationMerchantSuezToOman =
155                 new Point2D.Double( 145.0, 345.0 );
156         Point2D initialLocationMerchantMaldivesToSuez =
157                 new Point2D.Double( 1135.0, 250.0 );
158         Point2D initialLocationMerchantMaldivesToOman =
159                 new Point2D.Double( 1135.0, 250.0 );
160         Point2D initialLocationMerchantOmanToMaldives =
161                 new Point2D.Double( 655.0, 725.0 );
162         Point2D initialLocationMerchantOmanToSuez =
163                 new Point2D.Double( 655.0, 725.0 );
164
165         //**Pirate Paths**//
166         double minLatGoaPiratePath = 145.00;
167         double maxLatGoaPiratePath = 465.00;
168         double minLonGoaPiratePath = 340.0;
169         double maxLonGoaPiratePath = 460.0;
170         double minLatIoPiratePath = 400.0;
```

```
171     double maxLatIoPiratePath = 1060.0;
172     double minLonIoPiratePath = 0.0;
173     double maxLonIoPiratePath = 720.0;
174     double minLatGoaAndIoPiratePath = 145.0;
175     double maxLatGoaAndIoPiratePath = 1060.0;
176     double minLonGoaAndIoPiratePath = 0.0;
177     double maxLonGoaAndIoPiratePath = 720.0;
178
179     //**Merchant Paths**//
180     double minLatSuezToMaldivesMerchantWaypoint1 = 170.00;
181     double maxLatSuezToMaldivesMerchantWaypoint1 = 194.0;
182     double minLonSuezToMaldivesMerchantWaypoint1 = 320.0;
183     double maxLonSuezToMaldivesMerchantWaypoint1 = 328.0;
184     double minLatSuezToMaldivesMerchantWaypoint2 = 425.0;
185     double maxLatSuezToMaldivesMerchantWaypoint2 = 450.0;
186     double minLonSuezToMaldivesMerchantWaypoint2 = 390.0;
187     double maxLonSuezToMaldivesMerchantWaypoint2 = 415.0;
188     double minLatSuezToMaldivesMerchantWaypoint3 = 1055.0;
189     double maxLatSuezToMaldivesMerchantWaypoint3 = 1090.0;
190     double minLonSuezToMaldivesMerchantWaypoint3 = 250.0;
191     double maxLonSuezToMaldivesMerchantWaypoint3 = 265.0;
192     double minLatSuezToMaldivesMerchantWaypoint4 = 1115.0;
193     double maxLatSuezToMaldivesMerchantWaypoint4 = 1140.0;
194     double minLonSuezToMaldivesMerchantWaypoint4 = 220.0;
195     double maxLonSuezToMaldivesMerchantWaypoint4 = 260.0;
196     double minLatSuezToOmanMerchantWaypoint1 = 170.00;
197     double maxLatSuezToOmanMerchantWaypoint1 = 194.00;
198     double minLonSuezToOmanMerchantWaypoint1 = 320.0;
199     double maxLonSuezToOmanMerchantWaypoint1 = 328.0;
200     double minLatSuezToOmanMerchantWaypoint2 = 425.0;
201     double maxLatSuezToOmanMerchantWaypoint2 = 450.0;
202     double minLonSuezToOmanMerchantWaypoint2 = 390.0;
203     double maxLonSuezToOmanMerchantWaypoint2 = 415.0;
204     double minLatSuezToOmanMerchantWaypoint3 = 625.0;
205     double maxLatSuezToOmanMerchantWaypoint3 = 645.0;
206     double minLonSuezToOmanMerchantWaypoint3 = 515.0;
207     double maxLonSuezToOmanMerchantWaypoint3 = 530.0;
208     double minLatSuezToOmanMerchantWaypoint4 = 685.0;
209     double maxLatSuezToOmanMerchantWaypoint4 = 700.0;
210     double minLonSuezToOmanMerchantWaypoint4 = 720.0;
211     double maxLonSuezToOmanMerchantWaypoint4 = 725.0;
212     double minLatMaldivesToSuezMerchantWaypoint1 = 1065.0;
213     double maxLatMaldivesToSuezMerchantWaypoint1 = 1090.00;
214     double minLonMaldivesToSuezMerchantWaypoint1 = 265.0;
215     double maxLonMaldivesToSuezMerchantWaypoint1 = 280.0;
216     double minLatMaldivesToSuezMerchantWaypoint2 = 425.0;
217     double maxLatMaldivesToSuezMerchantWaypoint2 = 460.0;
218     double minLonMaldivesToSuezMerchantWaypoint2 = 410.0;
219     double maxLonMaldivesToSuezMerchantWaypoint2 = 420.0;
220     double minLatMaldivesToSuezMerchantWaypoint3 = 170.0;
221     double maxLatMaldivesToSuezMerchantWaypoint3 = 200.0;
222     double minLonMaldivesToSuezMerchantWaypoint3 = 325.0;
223     double maxLonMaldivesToSuezMerchantWaypoint3 = 340.0;
224     double minLatMaldivesToSuezMerchantWaypoint4 = 140.0;
225     double maxLatMaldivesToSuezMerchantWaypoint4 = 155.0;
226     double minLonMaldivesToSuezMerchantWaypoint4 = 330.0;
227     double maxLonMaldivesToSuezMerchantWaypoint4 = 350.0;
228     double minLatMaldivesToOmanMerchantWaypoint1 = 1065.0;
229     double maxLatMaldivesToOmanMerchantWaypoint1 = 1090.0;
230     double minLonMaldivesToOmanMerchantWaypoint1 = 265.0;
231     double maxLonMaldivesToOmanMerchantWaypoint1 = 280.0;
```

```
232     double minLatMaldivesToOmanMerchantWaypoint2 = 890.0;
233     double maxLatMaldivesToOmanMerchantWaypoint2 = 900.0;
234     double minLonMaldivesToOmanMerchantWaypoint2 = 500.0;
235     double maxLonMaldivesToOmanMerchantWaypoint2 = 515.0;
236     double minLatMaldivesToOmanMerchantWaypoint3 = 695.0;
237     double maxLatMaldivesToOmanMerchantWaypoint3 = 715.0;
238     double minLonMaldivesToOmanMerchantWaypoint3 = 700.0;
239     double maxLonMaldivesToOmanMerchantWaypoint3 = 720.0;
240     double minLatMaldivesToOmanMerchantWaypoint4 = 685.0;
241     double maxLatMaldivesToOmanMerchantWaypoint4 = 695.0;
242     double minLonMaldivesToOmanMerchantWaypoint4 = 725.0;
243     double maxLonMaldivesToOmanMerchantWaypoint4 = 730.0;
244     double minLatOmanToMaldivesMerchantWaypoint1 = 700.0;
245     double maxLatOmanToMaldivesMerchantWaypoint1 = 720.0;
246     double minLonOmanToMaldivesMerchantWaypoint1 = 685.0;
247     double maxLonOmanToMaldivesMerchantWaypoint1 = 695.0;
248     double minLatOmanToMaldivesMerchantWaypoint2 = 890.0;
249     double maxLatOmanToMaldivesMerchantWaypoint2 = 900.0;
250     double minLonOmanToMaldivesMerchantWaypoint2 = 470.0;
251     double maxLonOmanToMaldivesMerchantWaypoint2 = 490.0;
252     double minLatOmanToMaldivesMerchantWaypoint3 = 1060.0;
253     double maxLatOmanToMaldivesMerchantWaypoint3 = 1085.0;
254     double minLonOmanToMaldivesMerchantWaypoint3 = 245.0;
255     double maxLonOmanToMaldivesMerchantWaypoint3 = 265.0;
256     double minLatOmanToMaldivesMerchantWaypoint4 = 1110.0;
257     double maxLatOmanToMaldivesMerchantWaypoint4 = 1125.0;
258     double minLonOmanToMaldivesMerchantWaypoint4 = 230.0;
259     double maxLonOmanToMaldivesMerchantWaypoint4 = 250.0;
260     double minLatOmanToSuezMerchantWaypoint1 = 700.0;
261     double maxLatOmanToSuezMerchantWaypoint1 = 720.0;
262     double minLonOmanToSuezMerchantWaypoint1 = 685.0;
263     double maxLonOmanToSuezMerchantWaypoint1 = 695.0;
264     double minLatOmanToSuezMerchantWaypoint2 = 620.0;
265     double maxLatOmanToSuezMerchantWaypoint2 = 635.0;
266     double minLonOmanToSuezMerchantWaypoint2 = 530.0;
267     double maxLonOmanToSuezMerchantWaypoint2 = 545.0;
268     double minLatOmanToSuezMerchantWaypoint3 = 170.0;
269     double maxLatOmanToSuezMerchantWaypoint3 = 200.00;
270     double minLonOmanToSuezMerchantWaypoint3 = 325.0;
271     double maxLonOmanToSuezMerchantWaypoint3 = 340.0;
272     double minLatOmanToSuezMerchantWaypoint4 = 140.0;
273     double maxLatOmanToSuezMerchantWaypoint4 = 155.0;
274     double minLonOmanToSuezMerchantWaypoint4 = 335.0;
275     double maxLonOmanToSuezMerchantWaypoint4 = 350.0;
276
277 //**********PROBABILITY DISTRIBUTIONS FOR ENTIRE SIMULATION******************//
278     //Arrival and Departure Processes
279     /*
280      * TODO: Discuss this distribution
281      */
282     RandomVariate elaayoInterarrivalTime = RandomVariateFactory.
283         getInstance("Poisson," elaayoInterarrivalTimeLambda);
284     RandomVariate qandalaInterarrivalTime = RandomVariateFactory.
285         getInstance("Poisson," qandalaInterarrivalTimeLambda);
286     RandomVariate aluulaInterarrivalTime = RandomVariateFactory.
287         getInstance("Poisson," aluulaInterarrivalTimeLambda);
288     RandomVariate bargalInterarrivalTime = RandomVariateFactory.
289         getInstance("Poisson," bargalInterarrivalTimeLambda);
290     RandomVariate hafunInterarrivalTime = RandomVariateFactory.
291         getInstance("Poisson," hafunInterarrivalTimeLambda);
292     RandomVariate baylaInterarrivalTime = RandomVariateFactory.
```

```
293         getInstance("Poisson," baylaInterarrivalTimeLambda);
294     RandomVariate eylInterarrivalTime = RandomVariateFactory.
295         getInstance("Poisson," eylInterarrivalTimeLambda);
296     RandomVariate garacadInterarrivalTime = RandomVariateFactory.
297         getInstance("Poisson," garacadInterarrivalTimeLambda);
298     RandomVariate hobyoInterarrivalTime = RandomVariateFactory.
299         getInstance("Poisson," hobyoInterarrivalTimeLambda);
300     RandomVariate harardhereInterarrivalTime = RandomVariateFactory.
301         getInstance("Poisson," harardhereInterarrivalTimeLambda);
302
303
304     RandomVariate stoMerchantInterarrivalTime = RandomVariateFactory.
305         getInstance("Poisson," stoInterarrivalTimeLambda);
306     RandomVariate stmMerchantInterarrivalTime = RandomVariateFactory.
307         getInstance("Poisson," stmInterarrivalTimeLambda);
308     RandomVariate otsMerchantInterarrivalTime = RandomVariateFactory.
309         getInstance("Poisson," otsInterarrivalTimeLambda);
310     RandomVariate otmMerchantInterarrivalTime = RandomVariateFactory.
311         getInstance("Poisson," otmInterarrivalTimeLambda);
312     RandomVariate mtsMerchantInterarrivalTime = RandomVariateFactory.
313         getInstance("Poisson," mtsInterarrivalTimeLambda);
314     RandomVariate mtoMerchantInterarrivalTime = RandomVariateFactory.
315         getInstance("Poisson," mtoInterarrivalTimeLambda);
316
317     //**Applies to both IO and GOA pirates**//
318
319     /*
320      * This distribution attempts to capture whether or not the pirate will
321      * attack a detected merchant vessel. We attempt to capture the types
322      * of vessels the proportion of the types of vessels that traverse
323      * around the Horn of Africa and weather factors. However, without any
324      * real data, which no one will ever have, this is just an educated
325      * guess, and the best COA is to either say 50/50 or that the pirates
326      * are more likely to attack than not.
327      */
328     DiscreteRandomVariate attackDecision =
329         RandomVariateFactory.
330         getDiscreteRandomVariateInstance("Bernoulli,"
331                                 probOfAttackingDecision);
332
333     //Random variable for how long an attack on a merchant takes
334     RandomVariate[] unsuccessfulAttackTime = new RandomVariate[1];
335     unsuccessfulAttackTime[0] =
336         RandomVariateFactory.getInstance("Uniform,"
337                                     minUnsuccessfulAttackTime,
338                                     maxUnsuccessfulAttackTime);
339
340     //**GOA Pirates probability distributions**//
341     RandomVariate[] goaPiratePathGenerator = new RandomVariate[2];
342     goaPiratePathGenerator[0] =
343         RandomVariateFactory.getInstance("Uniform,"
344         minLatGoaPiratePath,
345         maxLatGoaPiratePath);
346     goaPiratePathGenerator[1] =
347         RandomVariateFactory.getInstance("Uniform,"
348         minLonGoaPiratePath,
349         maxLonGoaPiratePath);
350
351     //**IO Pirates probability distributions**//
352     /*
353      * TODO: Discuss this distribution
```

```
354         */
355         RandomVariate[] ioPiratePathGenerator = new RandomVariate[2];
356         ioPiratePathGenerator[0] =
357                 RandomVariateFactory.getInstance("Uniform,"
358                 minLatIoPiratePath,
359                 maxLatIoPiratePath);
360         ioPiratePathGenerator[1] =
361                 RandomVariateFactory.getInstance("Uniform,"
362                 minLonIoPiratePath,
363                 maxLonIoPiratePath);
364 /*
365  * Bargal pirates are known to attack in GOA and in IO so their paths are
366  * distributed over both areas.
367  */
368         /*
369         * TODO: Discuss this distribution
370         */
371         RandomVariate[] bargalPiratePathGenerator = new RandomVariate[2];
372         bargalPiratePathGenerator[0] =
373                 RandomVariateFactory.getInstance("Uniform,"
374                 minLatGoaAndIoPiratePath,
375                 maxLatGoaAndIoPiratePath);
376         bargalPiratePathGenerator[1] =
377                 RandomVariateFactory.getInstance("Uniform,"
378                 minLonGoaAndIoPiratePath,
379                 maxLonGoaAndIoPiratePath);
380
381         /*
382         * The successOrFailGenerator distributions captures the probability
383         * that an attack is successful or not. We utilize IMB data from 2008 -
384         * 2011 in order to obtain the probability.
385         */
386         DiscreteRandomVariate successOrFailGenerator = RandomVariateFactory.
387                 getDiscreteRandomVariateInstance( "Bernoulli," 0.26 );
388
389 //******************END OF PROBABILITY  DISTRIBUTIONS***************************//
390
391 //******************START OF STAT KEEPING VARIABLES*****************************//
392         double totalNumDepartedGOA = 0;
393         double totalNumDepartedIO = 0;
394         double totalNumberPiratesDeparted = 0;
395         double numberOfGoaPiratesDetected = 0;
396         double numberOfIoPiratesDetected = 0;
397         double totalNumberPiratesDetected = 0;
398         double numberAttemptedAttacksEllayoPirate = 0;
399         double numberAttemptedAttacksQandalaPirate = 0;
400         double numberAttemptedAttacksAluulaPirate = 0;
401         double numberAttemptedAttacksBargalPirate = 0;
402         double numberAttemptedAttacksHafunPirate = 0;
403         double numberAttemptedAttacksBaylaPirate = 0;
404         double numberAttemptedAttacksEylPirate = 0;
405         double numberAttemptedAttacksGaracadPirate = 0;
406         double numberAttemptedAttacksHobyoPirate = 0;
407         double numberAttemptedAttacksHarardherePirate = 0;
408         double totalAttemptedAttacks = 0;
409         double totalNumberSuccessfulHijacksStM = 0;
410         double totalNumberSuccessfulHijacksStO = 0;
411         double totalNumberSuccessfulHijacksOtM = 0;
412         double totalNumberSuccessfulHijacksOtS = 0;
413         double totalNumberSuccessfulHijacksMtS = 0;
414         double totalNumberSuccessfulHijacksMtO = 0;
```

```
415     double totalNumberSuccessfulHijacks = 0;
416     double numberStOMerchantTransits = 0;
417     double numberStMMerchantTransits = 0;
418     double numberOtSMerchantTransits = 0;
419     double numberOtMMerchantTransits = 0;
420     double numberMtSMerchantTransits = 0;
421     double numberMtOMerchantTransits = 0;
422     double totalNumberMerchantTransits = 0;
423     double numberStOSuccessfulTransits = 0;
424     double numberStMSuccessfulTransits = 0;
425     double numberOtSSuccessfulTransits = 0;
426     double numberOtMSuccessfulTransits = 0;
427     double numberMtSSuccessfulTransits = 0;
428     double numberMtOSuccessfulTransits = 0;
429     double totalNumberSuccessfulMerchantTransits = 0;
430     double navalEffectiveness = 0;
431     double pirateEffectiveness1 = 0;
432     double pirateEffectiveness2 = 0;
433     double merchantSuccessRate = 0;
434
435 //**********START OF PIRATE ENTITIES**********//
436     PlatformType typePirate = PlatformType.PIRATE;
437 //**********START OF GOA Pirates**********//
438
439 //**********START OF ELAAYO PIRATE IMPLEMENTATION**********//
440     ElaayoPirateDepartureProcess elaayoDepartureTimeProcess=
441         new ElaayoPirateDepartureProcess(elaayoInterarrivalTime);
442
443     Platform[] elaayoPirateMover = new Platform[ numElaayoPirates ];
444     for (int i = 0; i < numElaayoPirates; ++i) {
445       elaayoPirateMover[i] = new Platform("Pirate-Ellayo" + i,
446             pirateCampElaayo,
447             pirateMaxSpeed, typePirate);
448     }
449
450     System.out.println("Pirate: " + elaayoPirateMover[0].paramString());
451
452     CookieCutterSensor[] elaayoPirateSensor =
453         new CookieCutterSensor[elaayoPirateMover.length];
454     for (int i = 0; i < elaayoPirateMover.length; ++i) {
455       elaayoPirateSensor[i] =
456             new CookieCutterSensor(elaayoPirateMover[i],
457             pirateVisualSensorRange);
458     }
459
460     PirateMoverManager[] elaayoPirateManager =
461         new PirateMoverManager[elaayoPirateMover.length];
462     for (int i = 0; i < elaayoPirateMover.length; ++i) {
463       elaayoPirateManager[i] =
464             new PirateMoverManager(elaayoPirateMover[i],
465             elaayoPirateSensor[i],
466             pirateCampElaayo,
467             goaPiratePathGenerator,
468             attackDecision,
469             unsuccessfulAttackTime);
470     }
471
472     ElaayoPirateCamp epc = new ElaayoPirateCamp( elaayoPirateManager );
473       elaayoDepartureTimeProcess.addSimEventListener(epc);
474
475
```

```
476 //******************END OF ELAAYO PIRATE IMPLEMENTATION************************//
477 //******************START OF QANDALA PIRATE IMPLEMENTATION*********************//
478       QandalaPirateDepartureProcess qandalaDepartureTimeProcess=
479             new QandalaPirateDepartureProcess(qandalaInterarrivalTime);
480
481       Platform[] qandalaPirateMover = new Platform[ numQandalaPirates ];
482       for (int i = 0; i < numQandalaPirates; ++i) {
483          qandalaPirateMover[i] = new Platform("Pirate-Qandala" + i,
484                pirateCampQandala,
485                pirateMaxSpeed, typePirate);
486       }
487
488       System.out.println("Pirate: " + qandalaPirateMover[0].paramString());
489
490       CookieCutterSensor[] qandalaPirateSensor =
491             new CookieCutterSensor[qandalaPirateMover.length];
492       for (int i = 0; i < qandalaPirateMover.length; ++i) {
493          qandalaPirateSensor[i] =
494                new CookieCutterSensor(qandalaPirateMover[i],
495                pirateVisualSensorRange);
496       }
497
498       PirateMoverManager[] qandalaPirateManager =
499             new PirateMoverManager[qandalaPirateMover.length];
500       for (int i = 0; i < qandalaPirateMover.length; ++i) {
501          qandalaPirateManager[i] =
502                new PirateMoverManager(qandalaPirateMover[i],
503                qandalaPirateSensor[i],
504                pirateCampQandala,
505                goaPiratePathGenerator,
506                attackDecision,
507                unsuccessfulAttackTime);
508       }
509
510       QandalaPirateCamp qpc = new QandalaPirateCamp(qandalaPirateManager);
511          qandalaDepartureTimeProcess.addSimEventListener(qpc);
512
513 //******************END OF QANDALA PIRATE IMPLEMENTATION************************//
514 //******************START OF ALUULA PIRATE IMPLEMENTATION***********************//
515
516       AluulaPirateDepartureProcess aluulaDepartureTimeProcess=
517             new AluulaPirateDepartureProcess(aluulaInterarrivalTime);
518
519       Platform[] aluulaPirateMover = new Platform[ numAluulaPirates ];
520       for (int i = 0; i < numAluulaPirates; ++i) {
521          aluulaPirateMover[i] = new Platform("Pirate-Aluula" + i,
522                pirateCampAluula,
523                pirateMaxSpeed, typePirate);
524       }
525
526       System.out.println("Pirate: " + aluulaPirateMover[0].paramString());
527
528       CookieCutterSensor[] aluulaPirateSensor =
529             new CookieCutterSensor[aluulaPirateMover.length];
530       for (int i = 0; i < aluulaPirateMover.length; ++i) {
531          aluulaPirateSensor[i] =
532                new CookieCutterSensor(aluulaPirateMover[i],
533                pirateVisualSensorRange);
534       }
535
536       PirateMoverManager[] aluulaPirateManager =
```

```
537             new PirateMoverManager[aluulaPirateMover.length];
538         for (int i = 0; i < aluulaPirateMover.length; ++i) {
539           aluulaPirateManager[i] =
540                 new PirateMoverManager(aluulaPirateMover[i],
541                 aluulaPirateSensor[i],
542                 pirateCampAluula,
543                 goaPiratePathGenerator,
544                 attackDecision,
545                 unsuccessfulAttackTime);
546         }
547
548         AluulaPirateCamp apc = new AluulaPirateCamp(aluulaPirateManager);
549           aluulaDepartureTimeProcess.addSimEventListener(apc);
550
551 //*****************END OF ALUULA PIRATE IMPLEMENTATION**********************//
552 //************************END OF GOA Pirates*********************************//
553 //********************START OF IO Pirates*************************************//
554
555 //*****************START OF BARGAL Pirate Implemenation**********************//
556         BargalPirateDepartureProcess bargalDepartureTimeProcess=
557             new BargalPirateDepartureProcess(bargalInterarrivalTime);
558
559         Platform[] bargalPirateMover = new Platform[ numBargalPirates ];
560         for (int i = 0; i < numBargalPirates; ++i) {
561           bargalPirateMover[i] = new Platform("Pirate-Bargal" + i,
562                 pirateCampBargal,
563                 pirateMaxSpeed, typePirate);
564         }
565
566         System.out.println("Pirate: " + bargalPirateMover[0].paramString());
567
568         CookieCutterSensor[] bargalPirateSensor =
569             new CookieCutterSensor[bargalPirateMover.length];
570         for (int i = 0; i < bargalPirateMover.length; ++i) {
571           bargalPirateSensor[i] =
572                 new CookieCutterSensor(bargalPirateMover[i],
573                 pirateVisualSensorRange);
574         }
575
576         PirateMoverManager[] bargalPirateManager =
577             new PirateMoverManager[bargalPirateMover.length];
578         for (int i = 0; i < bargalPirateMover.length; ++i) {
579           bargalPirateManager[i] =
580                 new PirateMoverManager(bargalPirateMover[i],
581                 bargalPirateSensor[i],
582                 pirateCampBargal,
583                 bargalPiratePathGenerator,
584                 attackDecision,
585                 unsuccessfulAttackTime);
586         }
587
588         BargalPirateCamp bpc = new BargalPirateCamp(bargalPirateManager);
589           bargalDepartureTimeProcess.addSimEventListener(bpc);
590
591 //*****************END OF BARGAL PIRATE IMPLEMENTATION**********************//
592 //*****************START OF HAFUN PIRATE IMPLEMENTATION**********************//
593         HafunPirateDepartureProcess hafunDepartureTimeProcess=
594             new HafunPirateDepartureProcess(hafunInterarrivalTime);
595
596         Platform[] hafunPirateMover = new Platform[ numHafunPirates ];
597         for (int i = 0; i < numHafunPirates; ++i) {
```

```
598         hafunPirateMover[i] = new Platform("Pirate-Hafun" + i,
599                 pirateCampHafun,
600                 pirateMaxSpeed, typePirate);
601     }
602
603     System.out.println("Pirate: " + hafunPirateMover[0].paramString());
604
605     CookieCutterSensor[] hafunPirateSensor =
606             new CookieCutterSensor[hafunPirateMover.length];
607     for (int i = 0; i < hafunPirateMover.length; ++i) {
608         hafunPirateSensor[i] =
609                 new CookieCutterSensor(hafunPirateMover[i],
610                 pirateVisualSensorRange);
611     }
612
613     PirateMoverManager[] hafunPirateManager =
614             new PirateMoverManager[hafunPirateMover.length];
615     for (int i = 0; i < hafunPirateMover.length; ++i) {
616         hafunPirateManager[i] =
617                 new PirateMoverManager(hafunPirateMover[i],
618                 hafunPirateSensor[i],
619                 pirateCampHafun,
620                 ioPiratePathGenerator,
621                 attackDecision,
622                 unsuccessfulAttackTime);
623     }
624
625     HafunPirateCamp hpc = new HafunPirateCamp(hafunPirateManager);
626         hafunDepartureTimeProcess.addSimEventListener(hpc);
627
628 //******************END OF HAFUN PIRATE IMPLEMENTATION*************************//
629 //******************START OF BAYLA PIRATE IMPLEMENTATION*************************//
630     BaylaPirateDepartureProcess baylaDepartureTimeProcess=
631             new BaylaPirateDepartureProcess(baylaInterarrivalTime);
632
633     Platform[] baylaPirateMover = new Platform[ numBaylaPirates ];
634     for (int i = 0; i < numBaylaPirates; ++i) {
635         baylaPirateMover[i] = new Platform("Pirate-Bayla" + i,
636                 pirateCampBayla,
637                 pirateMaxSpeed, typePirate);
638     }
639
640     System.out.println("Pirate: " + baylaPirateMover[0].paramString());
641
642     CookieCutterSensor[] baylaPirateSensor =
643             new CookieCutterSensor[baylaPirateMover.length];
644     for (int i = 0; i < baylaPirateMover.length; ++i) {
645         baylaPirateSensor[i] =
646                 new CookieCutterSensor(baylaPirateMover[i],
647                 pirateVisualSensorRange);
648     }
649
650     PirateMoverManager[] baylaPirateManager =
651             new PirateMoverManager[baylaPirateMover.length];
652     for (int i = 0; i < baylaPirateMover.length; ++i) {
653         baylaPirateManager[i] =
654                 new PirateMoverManager(baylaPirateMover[i],
655                 baylaPirateSensor[i],
656                 pirateCampBayla,
657                 ioPiratePathGenerator,
658                 attackDecision,
```

```
659                 unsuccessfulAttackTime);
660         }
661
662         BaylaPirateCamp baypc = new BaylaPirateCamp(baylaPirateManager);
663           baylaDepartureTimeProcess.addSimEventListener(baypc);
664
665 //******************END OF BAYLA PIRATE IMPLEMENTATION************************//
666 //******************START OF EYL PIRATE IMPLEMENTATION************************//
667         EylPirateDepartureProcess eylDepartureTimeProcess=
668             new EylPirateDepartureProcess(eylInterarrivalTime);
669
670         Platform[] eylPirateMover = new Platform[ numEylPirates ];
671         for (int i = 0; i < numEylPirates; ++i) {
672           eylPirateMover[i] = new Platform("Pirate-Eyl" + i,
673                 pirateCampEyl,
674                 pirateMaxSpeed, typePirate);
675         }
676
677         System.out.println("Pirate: " + eylPirateMover[0].paramString());
678
679         CookieCutterSensor[] eylPirateSensor =
680             new CookieCutterSensor[eylPirateMover.length];
681         for (int i = 0; i < eylPirateMover.length; ++i) {
682           eylPirateSensor[i] =
683                 new CookieCutterSensor(eylPirateMover[i],
684                 pirateVisualSensorRange);
685         }
686
687         PirateMoverManager[] eylPirateManager =
688             new PirateMoverManager[eylPirateMover.length];
689         for (int i = 0; i < eylPirateMover.length; ++i) {
690           eylPirateManager[i] =
691                 new PirateMoverManager(eylPirateMover[i],
692                 eylPirateSensor[i],
693                 pirateCampEyl,
694                 ioPiratePathGenerator,
695                 attackDecision,
696                 unsuccessfulAttackTime);
697         }
698
699         EylPirateCamp eylpc = new EylPirateCamp(eylPirateManager);
700           eylDepartureTimeProcess.addSimEventListener(eylpc);
701
702 //******************END OF EYL PIRATE IMPLEMENTATION**************************//
703 //***************START OF GARACAD PIRATE IMPLEMENTATION************************//
704         GaracadPirateDepartureProcess garacadDepartureTimeProcess=
705             new GaracadPirateDepartureProcess(garacadInterarrivalTime);
706
707         Platform[] garacadPirateMover = new Platform[ numGaracadPirates ];
708         for (int i = 0; i < numGaracadPirates; ++i) {
709           garacadPirateMover[i] = new Platform("Pirate-Garacad" + i,
710                 pirateCampGaracad,
711                 pirateMaxSpeed, typePirate);
712         }
713
714         System.out.println("Pirate: " + garacadPirateMover[0].paramString());
715
716         CookieCutterSensor[] garacadPirateSensor =
717             new CookieCutterSensor[garacadPirateMover.length];
718         for (int i = 0; i < garacadPirateMover.length; ++i) {
719           garacadPirateSensor[i] =
```

```
720                 new CookieCutterSensor(garacadPirateMover[i],
721                 pirateVisualSensorRange);
722         }
723
724         PirateMoverManager[] garacadPirateManager =
725               new PirateMoverManager[garacadPirateMover.length];
726         for (int i = 0; i < garacadPirateMover.length; ++i) {
727            garacadPirateManager[i] =
728                 new PirateMoverManager(garacadPirateMover[i],
729                 garacadPirateSensor[i],
730                 pirateCampGaracad,
731                 ioPiratePathGenerator,
732                 attackDecision,
733                 unsuccessfulAttackTime);
734         }
735
736         GaracadPirateCamp gpc = new GaracadPirateCamp(garacadPirateManager);
737            garacadDepartureTimeProcess.addSimEventListener(gpc);
738
739 //*****************END OF GARACAD PIRATE IMPLEMENTATION**********************//
740 //***************START OF HOBYO PIRATE IMPLEMENTATION**************************//
741         HobyoPirateDepartureProcess hobyoDepartureTimeProcess=
742               new HobyoPirateDepartureProcess(hobyoInterarrivalTime);
743
744         Platform[] hobyoPirateMover = new Platform[ numHobyoPirates ];
745         for (int i = 0; i < numHobyoPirates; ++i) {
746            hobyoPirateMover[i] = new Platform("Pirate-Hobyo" + i,
747                 pirateCampHobyo,
748                 pirateMaxSpeed, typePirate);
749         }
750
751         System.out.println("Pirate: " + hobyoPirateMover[0].paramString());
752
753         CookieCutterSensor[] hobyoPirateSensor =
754               new CookieCutterSensor[hobyoPirateMover.length];
755         for (int i = 0; i < hobyoPirateMover.length; ++i) {
756            hobyoPirateSensor[i] =
757                 new CookieCutterSensor(hobyoPirateMover[i],
758                 pirateVisualSensorRange);
759         }
760
761         PirateMoverManager[] hobyoPirateManager =
762               new PirateMoverManager[hobyoPirateMover.length];
763         for (int i = 0; i < hobyoPirateMover.length; ++i)
764         {
765            hobyoPirateManager[i] =
766                 new PirateMoverManager(hobyoPirateMover[i],
767                 hobyoPirateSensor[i],
768                 pirateCampHobyo,
769                 ioPiratePathGenerator,
770                 attackDecision,
771                 unsuccessfulAttackTime);
772         }
773
774         HobyoPirateCamp hobpc = new HobyoPirateCamp(hobyoPirateManager);
775            hobyoDepartureTimeProcess.addSimEventListener(hobpc);
776 //*****************END OF HOBYO PIRATE IMPLEMENTATION************************//
777 //***************START OF HARARDHERE PIRATE IMPLEMENTATION*********************//
778         HarardherePirateDepartureProcess harardhereDepartureTimeProcess=
779               new HarardherePirateDepartureProcess(harardhereInterarrivalTime);
780
```

```
781     Platform[] harardherePirateMover = new Platform[ numHarardherePirates ];
782     for (int i = 0; i < numHarardherePirates; ++i) {
783       harardherePirateMover[i] = new Platform("Pirate-Harardhere" + i,
784             pirateCampHarardhere,
785             pirateMaxSpeed, typePirate);
786     }
787
788     System.out.println("Pirate: " + harardherePirateMover[0].paramString());
789
790     CookieCutterSensor[] harardherePirateSensor =
791           new CookieCutterSensor[harardherePirateMover.length];
792     for (int i = 0; i < harardherePirateMover.length; ++i) {
793       harardherePirateSensor[i] =
794             new CookieCutterSensor(harardherePirateMover[i],
795             pirateVisualSensorRange);
796     }
797
798     PirateMoverManager[] harardherePirateManager =
799           new PirateMoverManager[harardherePirateMover.length];
800     for (int i = 0; i < harardherePirateMover.length; ++i) {
801       harardherePirateManager[i] =
802             new PirateMoverManager( harardherePirateMover[i],
803                         harardherePirateSensor[i],
804                         pirateCampHarardhere,
805                         ioPiratePathGenerator,
806                         attackDecision,
807                         unsuccessfulAttackTime);
808     }
809
810     HarardherePirateCamp harpc = new HarardherePirateCamp(
811                          harardherePirateManager);
812       harardhereDepartureTimeProcess.addSimEventListener(harpc);
813
814 //******************END OF HARARDHERE PIRATE IMPLEMENTATION*********************//
815 //************************END OF IO PIRATES*************************************//
816 //********************END OF PIRATE IMPLEMENTATION*****************************//
817 //********************START OF NAVY IMPLEMENTATION*****************************//
818     PlatformType typeNavy = PlatformType.NAVY;
819     //**Navy Patroling in Indian Ocean**//
820     //Navy patrol points Box 1
821     RandomVariate[] navyPatrolBox1Generator = new RandomVariate[ 2 ];
822     navyPatrolBox1Generator[0] = RandomVariateFactory.getInstance(
823           "Uniform,"
824           290.00,
825           300.00 );
826     navyPatrolBox1Generator[1] = RandomVariateFactory.getInstance(
827           "Uniform,"
828           325.00,
829           328.00 );
830
831     //Navy patrol points Box 2
832     RandomVariate[] navyPatrolBox2Generator = new RandomVariate[ 2 ];
833     navyPatrolBox2Generator[0] = RandomVariateFactory.getInstance(
834           "Uniform,"
835           326.00,
836           336.00 );
837     navyPatrolBox2Generator[1] = RandomVariateFactory.getInstance(
838           "Uniform,"
839           335.00,
840           338.00 );
841
```

```
842     //Navy patrol points Box 3
843     RandomVariate[] navyPatrolBox3Generator = new RandomVariate[ 2 ];
844     navyPatrolBox3Generator[0] = RandomVariateFactory.getInstance(
845         "Uniform,"
846         360.00,
847         370.00 );
848     navyPatrolBox3Generator[1] = RandomVariateFactory.getInstance(
849         "Uniform,"
850         344.00,
851         347.00 );
852
853     //Navy patrol points Box 4
854     RandomVariate[] navyPatrolBox4Generator = new RandomVariate[ 2 ];
855     navyPatrolBox4Generator[0] = RandomVariateFactory.getInstance(
856         "Uniform,"
857         407.00,
858         410.00 );
859     navyPatrolBox4Generator[1] = RandomVariateFactory.getInstance(
860         "Uniform,"
861         310.00,
862         320.00 );
863
864     //Navy patrol points Box 5
865     RandomVariate[] navyPatrolBox5Generator = new RandomVariate[ 2 ];
866     navyPatrolBox5Generator[0] = RandomVariateFactory.getInstance(
867         "Uniform,"
868         408.00,
869         410.00 );
870     navyPatrolBox5Generator[1] = RandomVariateFactory.getInstance(
871         "Uniform,"
872         270.00,
873         280.00 );
874
875     //Navy patrol points Box 6
876     RandomVariate[] navyPatrolBox6Generator = new RandomVariate[ 2 ];
877     navyPatrolBox6Generator[0] = RandomVariateFactory.getInstance(
878         "Uniform,"
879         395.00,
880         398.00 );
881     navyPatrolBox6Generator[1] = RandomVariateFactory.getInstance(
882         "Uniform,"
883         238.00,
884         248.00 );
885
886     //Navy patrol points Box 7
887     RandomVariate[] navyPatrolBox7Generator = new RandomVariate[ 2 ];
888     navyPatrolBox7Generator[0] = RandomVariateFactory.getInstance(
889         "Uniform,"
890         363.00,
891         366.00 );
892     navyPatrolBox7Generator[1] = RandomVariateFactory.getInstance(
893         "Uniform,"
894         180.00,
895         190.00 );
896
897     //Navy patrol points Box 8
898     RandomVariate[] navyPatrolBox8Generator = new RandomVariate[ 2 ];
899     navyPatrolBox8Generator[0] = RandomVariateFactory.getInstance(
900         "Uniform,"
901         342.00,
902         345.00 );
```

```
903     navyPatrolBox8Generator[1] = RandomVariateFactory.getInstance(
904         "Uniform,"
905         150.00,
906         160.00 );
907
908     //Navy patrol points in IO Box 9
909     RandomVariate[] navyPatrolBox9Generator = new RandomVariate[ 2 ];
910     navyPatrolBox9Generator[0] = RandomVariateFactory.getInstance(
911         "Uniform,"
912         322.00,
913         325.00 );
914     navyPatrolBox9Generator[1] = RandomVariateFactory.getInstance(
915         "Uniform,"
916         96.00,
917         106.00 );
918
919     //Navy patrol points Box 10
920     RandomVariate[] navyPatrolBox10Generator = new RandomVariate[ 2 ];
921     navyPatrolBox10Generator[0] = RandomVariateFactory.getInstance(
922         "Uniform,"
923         301.00,
924         304.00 );
925     navyPatrolBox10Generator[1] = RandomVariateFactory.getInstance(
926         "Uniform,"
927         71.00,
928         81.00 );
929
930     Platform[] ioNavyMover = new Platform[numIoNavyShips];
931       ioNavyMover[0] = new Platform( "IO Navy-6," initialLocationNavyPB4,
932                               navyMaxSpeed, typeNavy );
933       ioNavyMover[1] = new Platform( "Navy-7," initialLocationNavyPB5,
934                       navyMaxSpeed, typeNavy );
935       ioNavyMover[2] = new Platform( "Navy-8," initialLocationNavyPB6,
936                       navyMaxSpeed, typeNavy );
937       ioNavyMover[3] = new Platform( "Navy-9," initialLocationNavyPB7,
938                       navyMaxSpeed, typeNavy );
939       ioNavyMover[4] = new Platform( "Navy-10," initialLocationNavyPB8,
940                       navyMaxSpeed, typeNavy );
941       ioNavyMover[5] = new Platform( "Navy-11," initialLocationNavyPB9,
942                       navyMaxSpeed, typeNavy );
943       ioNavyMover[6] = new Platform( "Navy-12," initialLocationNavyPB10,
944                       navyMaxSpeed, typeNavy );
945
946       CookieCutterSensor[] ioNavySensor =
947           new CookieCutterSensor[numIoNavyShips];
948       ioNavySensor[0] = new CookieCutterSensor(ioNavyMover[0],
949           navySurfaceRadarRange);
950       ioNavySensor[1] = new CookieCutterSensor(ioNavyMover[1],
951           navySurfaceRadarRange);
952       ioNavySensor[2] = new CookieCutterSensor(ioNavyMover[2],
953           navySurfaceRadarRange);
954       ioNavySensor[3] = new CookieCutterSensor(ioNavyMover[3],
955           navySurfaceRadarRange);
956       ioNavySensor[4] = new CookieCutterSensor(ioNavyMover[4],
957           navySurfaceRadarRange);
958       ioNavySensor[5] = new CookieCutterSensor(ioNavyMover[5],
959           navySurfaceRadarRange);
960       ioNavySensor[6] = new CookieCutterSensor(ioNavyMover[6],
961           navySurfaceRadarRange);
962
963       NavyShipMoverManager[] ioNavyManager =
```

```
964             new NavyShipMoverManager[numIoNavyShips];
965         ioNavyManager[0] = new NavyShipMoverManager( ioNavyMover[0],
966                         ioNavySensor[0], initialLocationNavyPB4,
967                         navyPatrolBox4Generator, navyMaxSpeed );
968         ioNavyManager[1] = new NavyShipMoverManager( ioNavyMover[1],
969                         ioNavySensor[1], initialLocationNavyPB5,
970                         navyPatrolBox5Generator, navyMaxSpeed );
971         ioNavyManager[2] = new NavyShipMoverManager( ioNavyMover[2],
972                         ioNavySensor[2], initialLocationNavyPB6,
973                         navyPatrolBox6Generator, navyMaxSpeed );
974         ioNavyManager[3] = new NavyShipMoverManager( ioNavyMover[3],
975                         ioNavySensor[3], initialLocationNavyPB7,
976                         navyPatrolBox7Generator, navyMaxSpeed );
977         ioNavyManager[4] = new NavyShipMoverManager( ioNavyMover[4],
978                         ioNavySensor[4], initialLocationNavyPB8,
979                         navyPatrolBox8Generator, navyMaxSpeed );
980         ioNavyManager[5] = new NavyShipMoverManager( ioNavyMover[5],
981                         ioNavySensor[5], initialLocationNavyPB9,
982                         navyPatrolBox9Generator, navyMaxSpeed );
983         ioNavyManager[6] = new NavyShipMoverManager( ioNavyMover[6],
984                         ioNavySensor[6], initialLocationNavyPB10,
985                         navyPatrolBox10Generator, navyMaxSpeed );
986
987         System.out.println ( "ioNavyManager Length: " +
988                         ioNavyManager.length );
989
990         //**Navy Patrols in the Gulf of Aden**//
991         Platform[] goaNavyMover = new Platform[numGoaNavyShips];
992         goaNavyMover[0] = new Platform( "IO Navy-1," initialLocationNavyPB1,
993                             navyMaxSpeed, typeNavy );
994         goaNavyMover[1] = new Platform( "Navy-2," initialLocationNavyPB2,
995                         navyMaxSpeed, typeNavy );
996         goaNavyMover[2] = new Platform( "Navy-3," initialLocationNavyPB3,
997                         navyMaxSpeed, typeNavy );
998
999         CookieCutterSensor[] goaNavySensor =
1000              new CookieCutterSensor[numGoaNavyShips];
1001         goaNavySensor[0] = new CookieCutterSensor(goaNavyMover[0],
1002              navySurfaceRadarRange);
1003         goaNavySensor[1] = new CookieCutterSensor(goaNavyMover[1],
1004              navySurfaceRadarRange);
1005         goaNavySensor[2] = new CookieCutterSensor(goaNavyMover[2],
1006              navySurfaceRadarRange);
1007
1008         NavyShipMoverManager[] goaNavyManager =
1009              new NavyShipMoverManager[numGoaNavyShips];
1010         goaNavyManager[0] = new NavyShipMoverManager( goaNavyMover[0],
1011                             goaNavySensor[0], initialLocationNavyPB1,
1012                             navyPatrolBox1Generator, navyMaxSpeed );
1013         goaNavyManager[1] = new NavyShipMoverManager( goaNavyMover[1],
1014                             goaNavySensor[1], initialLocationNavyPB2,
1015                             navyPatrolBox2Generator, navyMaxSpeed );
1016         goaNavyManager[2] = new NavyShipMoverManager( goaNavyMover[2],
1017                             goaNavySensor[2], initialLocationNavyPB3,
1018                             navyPatrolBox3Generator, navyMaxSpeed );
1019
1020         System.out.println("goaNavyManager length: " +
1021                         goaNavyManager.length);
1022
1023
1024 //********************END OF NAVY IMPLEMENTATION*********************************//
```

```
1025 //*****************START OF MERCHANT SHIP IMPLEMENTATION***********************//
1026       PlatformType typeMerchant = PlatformType.MERCHANT;
1027       //Creates Instance of ArrivalProcess w/ interarrival time passed in
1028       SuezToMaldivesMerchantDepartureProcess stmDepartureTimeProcess = new
1029             SuezToMaldivesMerchantDepartureProcess(
1030                   stmMerchantInterarrivalTime );
1031
1032 //**********START OF SUEZ TO MALDIVES MERCHANT SHIP IMPLEMENTATION*************//
1033       RandomVariate[] suezToMaldivesMerchantPathGenerator =
1034             new RandomVariate[ 8 ];
1035    suezToMaldivesMerchantPathGenerator[0] = RandomVariateFactory.getInstance(
1036             "Uniform,"
1037             minLatSuezToMaldivesMerchantWaypoint1,
1038             maxLatSuezToMaldivesMerchantWaypoint1 );
1039    suezToMaldivesMerchantPathGenerator[1] = RandomVariateFactory.getInstance(
1040             "Uniform,"
1041             minLonSuezToMaldivesMerchantWaypoint1,
1042             maxLonSuezToMaldivesMerchantWaypoint1 );
1043    suezToMaldivesMerchantPathGenerator[2] = RandomVariateFactory.getInstance(
1044             "Uniform,"
1045             minLatSuezToMaldivesMerchantWaypoint2,
1046             maxLatSuezToMaldivesMerchantWaypoint2 );
1047    suezToMaldivesMerchantPathGenerator[3] = RandomVariateFactory.getInstance(
1048             "Uniform,"
1049             minLonSuezToMaldivesMerchantWaypoint2,
1050             maxLonSuezToMaldivesMerchantWaypoint2 );
1051
1052    suezToMaldivesMerchantPathGenerator[4] = RandomVariateFactory.getInstance(
1053             "Uniform,"
1054             minLatSuezToMaldivesMerchantWaypoint3,
1055             maxLatSuezToMaldivesMerchantWaypoint3 );
1056    suezToMaldivesMerchantPathGenerator[5] = RandomVariateFactory.getInstance(
1057             "Uniform,"
1058             minLonSuezToMaldivesMerchantWaypoint3,
1059             maxLonSuezToMaldivesMerchantWaypoint3 );
1060
1061    suezToMaldivesMerchantPathGenerator[6] = RandomVariateFactory.getInstance(
1062             "Uniform,"
1063             minLatSuezToMaldivesMerchantWaypoint4,
1064             maxLatSuezToMaldivesMerchantWaypoint4 );
1065    suezToMaldivesMerchantPathGenerator[7] = RandomVariateFactory.getInstance(
1066             "Uniform,"
1067             minLonSuezToMaldivesMerchantWaypoint4,
1068             maxLonSuezToMaldivesMerchantWaypoint4 );
1069
1070       Platform [] suezToMaldivesMerchantMover =
1071             new Platform[numSuezToMaldivesMerchants];
1072       for (int i = 0; i < suezToMaldivesMerchantMover.length; ++i)
1073       {
1074          suezToMaldivesMerchantMover[i] =
1075             new Platform( "Merchant: SuezToMaldives " + i,
1076                         initialLocationMerchantSuezToMaldives,
1077                         merchantMaxSpeed, typeMerchant );
1078       }
1079
1080       CookieCutterSensor[] suezToMaldivesMerchantSensor =
1081             new CookieCutterSensor[suezToMaldivesMerchantMover.length];
1082       for (int i = 0; i < suezToMaldivesMerchantMover.length; ++i)
1083       {
1084          suezToMaldivesMerchantSensor [i] =
1085                new CookieCutterSensor(suezToMaldivesMerchantMover[i],
```

```
                                merchantSurfaceRadarRange);
    }

    MerchantShipMoverManager [] suezToMaldivesMerchantManager =
        new MerchantShipMoverManager[suezToMaldivesMerchantMover.length];
    for (int i = 0; i < suezToMaldivesMerchantMover.length; ++i)
    {
       suezToMaldivesMerchantManager[i] =
           new MerchantShipMoverManager (
                        suezToMaldivesMerchantMover[i],
                        suezToMaldivesMerchantSensor[i],
                        initialLocationMerchantSuezToMaldives,
                        suezToMaldivesMerchantPathGenerator );
    }

    SuezToMaldivesOriginPort stm = new
        SuezToMaldivesOriginPort(suezToMaldivesMerchantManager);
       stmDepartureTimeProcess.addSimEventListener( stm );


//************END OF SUEZ TO MALDIVES MERCHANT IMPLEMENTATION*****************//
//***********START OF SUEZ TO OMAN MERCHANT SHIP IMPLEMENTATION***************//
    //Creates Instance of ArrivalProcess w/ interarrival time passed in
    SuezToOmanMerchantDepartureProcess stoDepartureTimeProcess = new
        SuezToOmanMerchantDepartureProcess(stoMerchantInterarrivalTime);

    RandomVariate[] suezToOmanMerchantPathGenerator =
        new RandomVariate[ 8 ];
    suezToOmanMerchantPathGenerator[0] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLatSuezToOmanMerchantWaypoint1,
        maxLatSuezToOmanMerchantWaypoint1 );
    suezToOmanMerchantPathGenerator[1] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLonSuezToOmanMerchantWaypoint1,
        maxLonSuezToOmanMerchantWaypoint1 );
    suezToOmanMerchantPathGenerator[2] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLatSuezToOmanMerchantWaypoint2,
        maxLatSuezToOmanMerchantWaypoint2 );
    suezToOmanMerchantPathGenerator[3] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLonSuezToOmanMerchantWaypoint2,
        maxLonSuezToOmanMerchantWaypoint2 );

    suezToOmanMerchantPathGenerator[4] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLatSuezToOmanMerchantWaypoint3,
        maxLatSuezToOmanMerchantWaypoint3 );
    suezToOmanMerchantPathGenerator[5] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLonSuezToOmanMerchantWaypoint3,
        maxLonSuezToOmanMerchantWaypoint3 );

    suezToOmanMerchantPathGenerator[6] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLatSuezToOmanMerchantWaypoint4,
        maxLatSuezToOmanMerchantWaypoint4 );
    suezToOmanMerchantPathGenerator[7] = RandomVariateFactory.getInstance(
        "Uniform,"
        minLonSuezToOmanMerchantWaypoint4,
```

```
1147            maxLonSuezToOmanMerchantWaypoint4 );
1148
1149        Platform [] suezToOmanMerchantMover =
1150            new Platform[numSuezToOmanMerchants];
1151        for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
1152        {
1153           suezToOmanMerchantMover[i] =
1154              new Platform( "Merchant: SuezToOman " + i,
1155                      initialLocationMerchantSuezToOman,
1156                      merchantMaxSpeed, typeMerchant );
1157        }
1158
1159        CookieCutterSensor[] suezToOmanMerchantSensor =
1160            new CookieCutterSensor[suezToOmanMerchantMover.length];
1161        for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
1162        {
1163           suezToOmanMerchantSensor [i] =
1164               new CookieCutterSensor(suezToOmanMerchantMover[i],
1165                           merchantSurfaceRadarRange);
1166        }
1167
1168        MerchantShipMoverManager [] suezToOmanMerchantManager =
1169            new MerchantShipMoverManager[suezToOmanMerchantMover.length];
1170        for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
1171        {
1172           suezToOmanMerchantManager[i] =
1173               new MerchantShipMoverManager (
1174                           suezToOmanMerchantMover[i],
1175                           suezToOmanMerchantSensor[i],
1176                           initialLocationMerchantSuezToOman,
1177                           suezToOmanMerchantPathGenerator );
1178        }
1179
1180        SuezToOmanOriginPort sto = new
1181            SuezToOmanOriginPort( suezToOmanMerchantManager );
1182           stoDepartureTimeProcess.addSimEventListener( sto );
1183
1184 //*************END OF SUEZ TO OMAN MERCHANT IMPLEMENTATION*****************//
1185 //************START OF MALDIVES TO SUEZ MERCHANT SHIP IMPLEMENTATION***********//
1186        //Creates Instance of ArrivalProcess w/ interarrival time passed in
1187        MaldivesToSuezMerchantDepartureProcess mtsDepartureTimeProcess = new
1188            MaldivesToSuezMerchantDepartureProcess(
1189                        mtsMerchantInterarrivalTime );
1190
1191        RandomVariate[] maldivesToSuezMerchantPathGenerator =
1192            new RandomVariate[ 8 ];
1193     maldivesToSuezMerchantPathGenerator[0] = RandomVariateFactory.getInstance(
1194            "Uniform,"
1195            minLatMaldivesToSuezMerchantWaypoint1,
1196            maxLatMaldivesToSuezMerchantWaypoint1 );
1197     maldivesToSuezMerchantPathGenerator[1] = RandomVariateFactory.getInstance(
1198            "Uniform,"
1199            minLonMaldivesToSuezMerchantWaypoint1,
1200            maxLonMaldivesToSuezMerchantWaypoint1 );
1201     maldivesToSuezMerchantPathGenerator[2] = RandomVariateFactory.getInstance(
1202            "Uniform,"
1203            minLatMaldivesToSuezMerchantWaypoint2,
1204            maxLatMaldivesToSuezMerchantWaypoint2 );
1205     maldivesToSuezMerchantPathGenerator[3] = RandomVariateFactory.getInstance(
1206            "Uniform,"
1207            minLonMaldivesToSuezMerchantWaypoint2,
```

```
1208            maxLonMaldivesToSuezMerchantWaypoint2 );
1209
1210    maldivesToSuezMerchantPathGenerator[4] = RandomVariateFactory.getInstance(
1211            “Uniform,”
1212            minLatMaldivesToSuezMerchantWaypoint3,
1213            maxLatMaldivesToSuezMerchantWaypoint3 );
1214    maldivesToSuezMerchantPathGenerator[5] = RandomVariateFactory.getInstance(
1215            “Uniform,”
1216            minLonMaldivesToSuezMerchantWaypoint3,
1217            maxLonMaldivesToSuezMerchantWaypoint3 );
1218
1219    maldivesToSuezMerchantPathGenerator[6] = RandomVariateFactory.getInstance(
1220            “Uniform,”
1221            minLatMaldivesToSuezMerchantWaypoint4,
1222            maxLatMaldivesToSuezMerchantWaypoint4 );
1223    maldivesToSuezMerchantPathGenerator[7] = RandomVariateFactory.getInstance(
1224            “Uniform,”
1225            minLonMaldivesToSuezMerchantWaypoint4,
1226            maxLonMaldivesToSuezMerchantWaypoint4 );
1227
1228      Platform [] maldivesToSuezMerchantMover =
1229            new Platform[numMaldivesToSuezMerchants];
1230      for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
1231      {
1232         maldivesToSuezMerchantMover[i] =
1233            new Platform( “Merchant: MaldivesToSuez “ + i,
1234                     initialLocationMerchantMaldivesToSuez,
1235                     merchantMaxSpeed, typeMerchant );
1236      }
1237
1238      CookieCutterSensor[] maldivesToSuezMerchantSensor =
1239            new CookieCutterSensor[maldivesToSuezMerchantMover.length];
1240      for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
1241      {
1242         maldivesToSuezMerchantSensor [i] =
1243              new CookieCutterSensor(maldivesToSuezMerchantMover[i],
1244                           merchantSurfaceRadarRange);
1245      }
1246
1247      MerchantShipMoverManager [] maldivesToSuezMerchantManager =
1248           new MerchantShipMoverManager[maldivesToSuezMerchantMover.length];
1249      for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
1250      {
1251         maldivesToSuezMerchantManager[i] =
1252              new MerchantShipMoverManager (
1253                           maldivesToSuezMerchantMover[i],
1254                           maldivesToSuezMerchantSensor[i],
1255                           initialLocationMerchantMaldivesToSuez,
1256                           maldivesToSuezMerchantPathGenerator );
1257      }
1258
1259      MaldivesToSuezOriginPort mts = new
1260            MaldivesToSuezOriginPort( maldivesToSuezMerchantManager );
1261         mtsDepartureTimeProcess.addSimEventListener( mts );
1262
1263
1264 //************END OF MALDIVES TO SUEZ MERCHANT IMPLEMENTATION*****************//
1265 //***********START OF MALDIVES TO OMAN MERCHANT SHIP IMPLEMENTATION***********//
1266      //Creates Instance of ArrivalProcess w/ interarrival time passed in
1267      MaldivesToOmanMerchantDepartureProcess mtoDepartureTimeProcess = new
1268            MaldivesToOmanMerchantDepartureProcess(
```

```
1269                    mtoMerchantInterarrivalTime );
1270
1271    RandomVariate[] maldivesToOmanMerchantPathGenerator =
1272          new RandomVariate[ 8 ];
1273    maldivesToOmanMerchantPathGenerator[0] = RandomVariateFactory.getInstance(
1274          "Uniform,"
1275          minLatMaldivesToOmanMerchantWaypoint1,
1276          maxLatMaldivesToOmanMerchantWaypoint1 );
1277    maldivesToOmanMerchantPathGenerator[1] = RandomVariateFactory.getInstance(
1278          "Uniform,"
1279          minLonMaldivesToOmanMerchantWaypoint1,
1280          maxLonMaldivesToOmanMerchantWaypoint1 );
1281    maldivesToOmanMerchantPathGenerator[2] = RandomVariateFactory.getInstance(
1282          "Uniform,"
1283          minLatMaldivesToOmanMerchantWaypoint2,
1284          maxLatMaldivesToOmanMerchantWaypoint2 );
1285    maldivesToOmanMerchantPathGenerator[3] = RandomVariateFactory.getInstance(
1286          "Uniform,"
1287          minLonMaldivesToOmanMerchantWaypoint2,
1288          maxLonMaldivesToOmanMerchantWaypoint2 );
1289
1290    maldivesToOmanMerchantPathGenerator[4] = RandomVariateFactory.getInstance(
1291          "Uniform,"
1292          minLatMaldivesToOmanMerchantWaypoint3,
1293          maxLatMaldivesToOmanMerchantWaypoint3 );
1294    maldivesToOmanMerchantPathGenerator[5] = RandomVariateFactory.getInstance(
1295          "Uniform,"
1296          minLonMaldivesToOmanMerchantWaypoint3,
1297          maxLonMaldivesToOmanMerchantWaypoint3 );
1298
1299    maldivesToOmanMerchantPathGenerator[6] = RandomVariateFactory.
1300          getInstance( "Uniform," minLatMaldivesToOmanMerchantWaypoint4,
1301          maxLatMaldivesToOmanMerchantWaypoint4 );
1302    maldivesToOmanMerchantPathGenerator[7] = RandomVariateFactory.
1303          getInstance( "Uniform," minLonMaldivesToOmanMerchantWaypoint4,
1304          maxLonMaldivesToOmanMerchantWaypoint4 );
1305
1306    Platform [] maldivesToOmanMerchantMover =
1307          new Platform[numMaldivesToOmanMerchants];
1308    for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
1309    {
1310       maldivesToOmanMerchantMover[i] =
1311          new Platform( "Merchant: MaldivesToOman " + i,
1312                initialLocationMerchantMaldivesToOman,
1313                merchantMaxSpeed, typeMerchant );
1314    }
1315
1316    CookieCutterSensor[] maldivesToOmanMerchantSensor =
1317          new CookieCutterSensor[maldivesToOmanMerchantMover.length];
1318    for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
1319    {
1320       maldivesToOmanMerchantSensor [i] =
1321            new CookieCutterSensor(maldivesToOmanMerchantMover[i],
1322                  merchantSurfaceRadarRange);
1323    }
1324
1325    MerchantShipMoverManager [] maldivesToOmanMerchantManager =
1326         new MerchantShipMoverManager[maldivesToOmanMerchantMover.length];
1327    for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
1328    {
1329       maldivesToOmanMerchantManager[i] =
```

```
1330                new MerchantShipMoverManager (
1331                            maldivesToOmanMerchantMover[i],
1332                            maldivesToOmanMerchantSensor[i],
1333                            initialLocationMerchantMaldivesToOman,
1334                            maldivesToOmanMerchantPathGenerator );
1335        }
1336
1337        MaldivesToOmanOriginPort mto = new
1338             MaldivesToOmanOriginPort( maldivesToOmanMerchantManager );
1339           mtoDepartureTimeProcess.addSimEventListener( mto );
1340
1341 //************END OF MALDIVES TO OMAN MERCHANT IMPLEMENTATION*****************//
1342 //***********START OF OMAN TO MALDIVES MERCHANT SHIP IMPLEMENTATION***********//
1343        //Creates Instance of ArrivalProcess w/ interarrival time passed in
1344        OmanToMaldivesMerchantDepartureProcess otmDepartureTimeProcess = new
1345             OmanToMaldivesMerchantDepartureProcess(
1346                  otmMerchantInterarrivalTime );
1347
1348        RandomVariate[] omanToMaldivesMerchantPathGenerator =
1349             new RandomVariate[ 8 ];
1350        omanToMaldivesMerchantPathGenerator[0] = RandomVariateFactory.
1351             getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint1,
1352             maxLatOmanToMaldivesMerchantWaypoint1 );
1353        omanToMaldivesMerchantPathGenerator[1] = RandomVariateFactory.
1354             getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint1,
1355             maxLonOmanToMaldivesMerchantWaypoint1 );
1356        omanToMaldivesMerchantPathGenerator[2] = RandomVariateFactory.
1357             getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint2,
1358             maxLatOmanToMaldivesMerchantWaypoint2 );
1359        omanToMaldivesMerchantPathGenerator[3] = RandomVariateFactory.
1360             getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint2,
1361             maxLonOmanToMaldivesMerchantWaypoint2 );
1362
1363        omanToMaldivesMerchantPathGenerator[4] = RandomVariateFactory.
1364             getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint3,
1365             maxLatOmanToMaldivesMerchantWaypoint3 );
1366        omanToMaldivesMerchantPathGenerator[5] = RandomVariateFactory.
1367             getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint3,
1368             maxLonOmanToMaldivesMerchantWaypoint3 );
1369
1370        omanToMaldivesMerchantPathGenerator[6] = RandomVariateFactory.
1371             getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint4,
1372             maxLatOmanToMaldivesMerchantWaypoint4 );
1373        omanToMaldivesMerchantPathGenerator[7] = RandomVariateFactory.
1374             getInstance( "Uniform,"minLonOmanToMaldivesMerchantWaypoint4,
1375             maxLonOmanToMaldivesMerchantWaypoint4 );
1376
1377        Platform [] omanToMaldivesMerchantMover =
1378             new Platform[numOmanToMaldivesMerchants];
1379        for (int i = 0; i < omanToMaldivesMerchantMover.length; ++i)
1380        {
1381           omanToMaldivesMerchantMover[i] =
1382             new Platform( "Merchant: OmanToMaldives " + i,
1383                       initialLocationMerchantOmanToMaldives,
1384                       merchantMaxSpeed, typeMerchant );
1385        }
1386
1387        CookieCutterSensor[] omanToMaldivesMerchantSensor =
1388             new CookieCutterSensor[omanToMaldivesMerchantMover.length];
1389        for (int i = 0; i < omanToMaldivesMerchantMover.length; ++i)
1390        {
```

```
1391          omanToMaldivesMerchantSensor [i] =
1392               new CookieCutterSensor(omanToMaldivesMerchantMover[i],
1393                                      merchantSurfaceRadarRange);
1394       }
1395
1396       MerchantShipMoverManager [] omanToMaldivesMerchantManager =
1397            new MerchantShipMoverManager[omanToMaldivesMerchantMover.length];
1398       for (int i = 0; i < omanToMaldivesMerchantMover.length; ++i)
1399       {
1400          omanToMaldivesMerchantManager[i] =
1401               new MerchantShipMoverManager (
1402                                      omanToMaldivesMerchantMover[i],
1403                                      omanToMaldivesMerchantSensor[i],
1404                                      initialLocationMerchantOmanToMaldives,
1405                                      omanToMaldivesMerchantPathGenerator );
1406       }
1407
1408       OmanToMaldivesOriginPort otm = new
1409            OmanToMaldivesOriginPort( omanToMaldivesMerchantManager );
1410          otmDepartureTimeProcess.addSimEventListener( otm );
1411
1412 //************END OF OMAN TO MALDIVES MERCHANT IMPLEMENTATION*****************//
1413 //***********START OF OMAN TO SUEZ MERCHANT SHIP IMPLEMENTATION***************//
1414       //Creates Instance of ArrivalProcess w/ interarrival time passed in
1415       OmanToSuezMerchantDepartureProcess otsDepartureTimeProcess = new
1416            OmanToSuezMerchantDepartureProcess(
1417                 otsMerchantInterarrivalTime );
1418
1419       RandomVariate[] omanToSuezMerchantPathGenerator =
1420            new RandomVariate[ 8 ];
1421       omanToSuezMerchantPathGenerator[0] = RandomVariateFactory.getInstance(
1422            “Uniform,”
1423            minLatOmanToSuezMerchantWaypoint1,
1424            maxLatOmanToSuezMerchantWaypoint1 );
1425       omanToSuezMerchantPathGenerator[1] = RandomVariateFactory.getInstance(
1426            “Uniform,”
1427            minLonOmanToSuezMerchantWaypoint1,
1428            maxLonOmanToSuezMerchantWaypoint1 );
1429       omanToSuezMerchantPathGenerator[2] = RandomVariateFactory.getInstance(
1430            “Uniform,”
1431            minLatOmanToSuezMerchantWaypoint2,
1432            maxLatOmanToSuezMerchantWaypoint2 );
1433       omanToSuezMerchantPathGenerator[3] = RandomVariateFactory.getInstance(
1434            “Uniform,”
1435            minLonOmanToSuezMerchantWaypoint2,
1436            maxLonOmanToSuezMerchantWaypoint2 );
1437
1438       omanToSuezMerchantPathGenerator[4] = RandomVariateFactory.getInstance(
1439            “Uniform,”
1440            minLatOmanToSuezMerchantWaypoint3,
1441            maxLatOmanToSuezMerchantWaypoint3 );
1442       omanToSuezMerchantPathGenerator[5] = RandomVariateFactory.getInstance(
1443            “Uniform,”
1444            minLonOmanToSuezMerchantWaypoint3,
1445            maxLonOmanToSuezMerchantWaypoint3 );
1446
1447       omanToSuezMerchantPathGenerator[6] = RandomVariateFactory.getInstance(
1448            “Uniform,”
1449            minLatOmanToSuezMerchantWaypoint4,
1450            maxLatOmanToSuezMerchantWaypoint4 );
1451       omanToSuezMerchantPathGenerator[7] = RandomVariateFactory.getInstance(
```

```
1452            "Uniform,"
1453            minLonOmanToSuezMerchantWaypoint4,
1454            maxLonOmanToSuezMerchantWaypoint4 );
1455
1456        Platform [] omanToSuezMerchantMover =
1457            new Platform[numOmanToSuezMerchants];
1458        for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
1459        {
1460           omanToSuezMerchantMover[i] =
1461            new Platform( "Merchant: OmanToSuez " + i,
1462                    initialLocationMerchantOmanToSuez,
1463                    merchantMaxSpeed, typeMerchant );
1464        }
1465
1466        CookieCutterSensor[] omanToSuezMerchantSensor =
1467            new CookieCutterSensor[omanToSuezMerchantMover.length];
1468        for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
1469        {
1470           omanToSuezMerchantSensor [i] =
1471              new CookieCutterSensor(omanToSuezMerchantMover[i],
1472                          merchantSurfaceRadarRange);
1473        }
1474
1475        MerchantShipMoverManager [] omanToSuezMerchantManager =
1476            new MerchantShipMoverManager[omanToSuezMerchantMover.length];
1477        for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
1478        {
1479           omanToSuezMerchantManager[i] =
1480              new MerchantShipMoverManager (
1481                          omanToSuezMerchantMover[i],
1482                          omanToSuezMerchantSensor[i],
1483                          initialLocationMerchantOmanToSuez,
1484                          omanToSuezMerchantPathGenerator );
1485        }
1486
1487        OmanToSuezOriginPort ots = new
1488            OmanToSuezOriginPort( omanToSuezMerchantManager );
1489           otsDepartureTimeProcess.addSimEventListener( ots );
1490
1491 //************END OF OMAN TO SUEZ MERCHANT IMPLEMENTATION*********************//
1492 //*****************END OF MERCHANT SHIP IMPLEMENTATION**************************//
1493 //******************START OF ADUDICATOR IMPLEMENTATION*************************//
1494        Adjudicator adj = new Adjudicator(successOrFailGenerator);
1495
1496 //*****************END OF ADUDICATOR IMPLEMENTATION**************************//
1497 //****************Referees, Mediators, and EventListeners*********************//
1498        //Create a SensorMoverReferee
1499        SensorMoverReferee smr = new SensorMoverReferee();
1500
1501        //Add a mediator for each sesnor and mediator
1502        smr.addMediator( CookieCutterSensor.class, Platform.class,
1503                    new CookieCutterMediator() );
1504
1505        adj.addSimEventListener( smr );
1506
1507        for ( int i = 0 ; i < elaayoPirateMover.length ; ++i )
1508        {
1509           elaayoPirateMover[i].addSimEventListener( smr );
1510           elaayoPirateManager[i].addSimEventListener( smr );
1511           elaayoPirateSensor[i].addSimEventListener( smr );
1512           elaayoPirateSensor[i].addSimEventListener( elaayoPirateManager[i] );
```

```
1513        }
1514
1515        for ( int i = 0 ; i < qandalaPirateMover.length ; ++i )
1516        {
1517           qandalaPirateMover[i].addSimEventListener( smr );
1518           qandalaPirateManager[i].addSimEventListener( smr );
1519           qandalaPirateSensor[i].addSimEventListener( smr );
1520           qandalaPirateSensor[i].addSimEventListener(qandalaPirateManager[i]);
1521        }
1522
1523        for ( int i = 0 ; i < aluulaPirateMover.length ; ++i )
1524        {
1525           aluulaPirateMover[i].addSimEventListener( smr );
1526           aluulaPirateManager[i].addSimEventListener( smr );
1527           aluulaPirateSensor[i].addSimEventListener( smr );
1528           aluulaPirateSensor[i].addSimEventListener( aluulaPirateManager[i] );
1529        }
1530
1531        for ( int i = 0 ; i < bargalPirateMover.length ; ++i )
1532        {
1533           bargalPirateMover[i].addSimEventListener( smr );
1534           bargalPirateManager[i].addSimEventListener( smr );
1535           bargalPirateSensor[i].addSimEventListener( smr );
1536           bargalPirateSensor[i].addSimEventListener( bargalPirateManager[i] );
1537        }
1538
1539        for ( int i = 0 ; i < hafunPirateMover.length ; ++i )
1540        {
1541           hafunPirateMover[i].addSimEventListener( smr );
1542           hafunPirateManager[i].addSimEventListener( smr );
1543           hafunPirateSensor[i].addSimEventListener( smr );
1544           hafunPirateSensor[i].addSimEventListener( hafunPirateManager[i] );
1545        }
1546
1547        for ( int i = 0 ; i < baylaPirateMover.length ; ++i )
1548        {
1549           baylaPirateMover[i].addSimEventListener( smr );
1550           baylaPirateManager[i].addSimEventListener( smr );
1551           baylaPirateSensor[i].addSimEventListener( smr );
1552           baylaPirateSensor[i].addSimEventListener( baylaPirateManager[i] );
1553        }
1554
1555        for ( int i = 0 ; i < eylPirateMover.length ; ++i )
1556        {
1557           eylPirateMover[i].addSimEventListener( smr );
1558           eylPirateManager[i].addSimEventListener( smr );
1559           eylPirateSensor[i].addSimEventListener( smr );
1560           eylPirateSensor[i].addSimEventListener( eylPirateManager[i] );
1561        }
1562
1563        for ( int i = 0 ; i < garacadPirateMover.length ; ++i )
1564        {
1565           garacadPirateMover[i].addSimEventListener( smr );
1566           garacadPirateManager[i].addSimEventListener( smr );
1567           garacadPirateSensor[i].addSimEventListener( smr );
1568           garacadPirateSensor[i].addSimEventListener(
1569                          garacadPirateManager[i] );
1570        }
1571
1572        for ( int i = 0 ; i < hobyoPirateMover.length ; ++i )
1573        {
```

```
1574        hobyoPirateMover[i].addSimEventListener( smr );
1575        hobyoPirateManager[i].addSimEventListener( smr );
1576        hobyoPirateSensor[i].addSimEventListener( smr );
1577        hobyoPirateSensor[i].addSimEventListener( hobyoPirateManager[i] );
1578     }
1579
1580     for ( int i = 0 ; i < harardherePirateMover.length ; ++i )
1581     {
1582        harardherePirateMover[i].addSimEventListener( smr );
1583        harardherePirateManager[i].addSimEventListener( smr );
1584        harardherePirateSensor[i].addSimEventListener( smr );
1585        harardherePirateSensor[i].addSimEventListener(
1586              harardherePirateManager[i] );
1587     }
1588
1589     for ( int i = 0 ; i < goaNavyMover.length ; ++i )
1590     {
1591        goaNavyMover[i].addSimEventListener( smr );
1592        goaNavyManager[i].addSimEventListener( smr );
1593        goaNavySensor[i].addSimEventListener( smr );
1594        goaNavySensor[i].addSimEventListener( goaNavyManager[i] );
1595     }
1596
1597     for ( int i = 0 ; i < ioNavyMover.length ; ++i )
1598     {
1599        ioNavyMover[i].addSimEventListener( smr );
1600        ioNavyManager[i].addSimEventListener( smr );
1601        ioNavySensor[i].addSimEventListener( smr );
1602        ioNavySensor[i].addSimEventListener( ioNavyManager[i] );
1603     }
1604
1605     for ( int i = 0 ; i < suezToOmanMerchantMover.length ; ++i )
1606     {
1607        suezToOmanMerchantMover[i].addSimEventListener( smr );
1608        suezToOmanMerchantManager[i].addSimEventListener( smr );
1609        suezToOmanMerchantSensor[i].addSimEventListener( smr );
1610        suezToOmanMerchantSensor[i].addSimEventListener(
1611              suezToOmanMerchantManager[i] );
1612     }
1613
1614     for ( int i = 0 ; i < suezToMaldivesMerchantMover.length ; ++i )
1615     {
1616        suezToMaldivesMerchantMover[i].addSimEventListener( smr );
1617        suezToMaldivesMerchantManager[i].addSimEventListener( smr );
1618        suezToMaldivesMerchantSensor[i].addSimEventListener( smr );
1619        suezToMaldivesMerchantSensor[i].addSimEventListener(
1620              suezToMaldivesMerchantManager[i] );
1621     }
1622
1623     for ( int i = 0 ; i < omanToSuezMerchantMover.length ; ++i )
1624     {
1625        omanToSuezMerchantMover[i].addSimEventListener( smr );
1626        omanToSuezMerchantManager[i].addSimEventListener( smr );
1627        omanToSuezMerchantSensor[i].addSimEventListener( smr );
1628        omanToSuezMerchantSensor[i].addSimEventListener(
1629              omanToSuezMerchantManager[i] );
1630     }
1631
1632     for ( int i = 0 ; i < omanToMaldivesMerchantMover.length ; ++i )
1633     {
1634        omanToMaldivesMerchantMover[i].addSimEventListener( smr );
```

```
1635          omanToMaldivesMerchantManager[i].addSimEventListener( smr );
1636          omanToMaldivesMerchantSensor[i].addSimEventListener( smr );
1637          omanToMaldivesMerchantSensor[i].addSimEventListener(
1638                omanToMaldivesMerchantManager[i] );
1639       }
1640
1641       for ( int i = 0 ; i < maldivesToSuezMerchantMover.length ; ++i )
1642       {
1643          maldivesToSuezMerchantMover[i].addSimEventListener( smr );
1644          maldivesToSuezMerchantManager[i].addSimEventListener( smr );
1645          maldivesToSuezMerchantSensor[i].addSimEventListener( smr );
1646          maldivesToSuezMerchantSensor[i].addSimEventListener(
1647                maldivesToSuezMerchantManager[i] );
1648       }
1649
1650       for ( int i = 0 ; i < maldivesToOmanMerchantMover.length ; ++i )
1651       {
1652          maldivesToOmanMerchantMover[i].addSimEventListener( smr );
1653          maldivesToOmanMerchantManager[i].addSimEventListener( smr );
1654          maldivesToOmanMerchantSensor[i].addSimEventListener( smr );
1655          maldivesToOmanMerchantSensor[i].addSimEventListener(
1656                maldivesToOmanMerchantManager[i] );
1657       }
1658
1659 //**************END OF Referees, Mediators, and EventListeners******************//
1660
1661 //*****************Start of Adapters for Simulation*****************************//
1662     Adapter decision = new Adapter("Attack," "DecideSuccessOrFail");
1663
1664       for(int i = 0; i < eylPirateManager.length; ++i)
1665       {
1666          decision.connect( eylPirateManager[i], adj);
1667       }
1668
1669       for(int i = 0; i < qandalaPirateManager.length; ++i)
1670       {
1671          decision.connect( qandalaPirateManager[i], adj);
1672       }
1673
1674       for(int i = 0; i < aluulaPirateManager.length; ++i)
1675       {
1676          decision.connect( aluulaPirateManager[i], adj);
1677       }
1678
1679       for(int i = 0; i < bargalPirateManager.length; ++i)
1680       {
1681          decision.connect( bargalPirateManager[i], adj);
1682       }
1683
1684       for(int i = 0; i < hafunPirateManager.length; ++i)
1685       {
1686          decision.connect( hafunPirateManager[i], adj);
1687       }
1688
1689       for(int i = 0; i < baylaPirateManager.length; ++i)
1690       {
1691          decision.connect( baylaPirateManager[i], adj);
1692       }
1693
1694       for(int i = 0; i < eylPirateManager.length; ++i)
1695       {
```

```
1696        decision.connect( eylPirateManager[i], adj);
1697     }
1698
1699     for(int i = 0; i < garacadPirateManager.length; ++i)
1700     {
1701        decision.connect( garacadPirateManager[i], adj);
1702     }
1703
1704     for(int i = 0; i < hobyoPirateManager.length; ++i)
1705     {
1706        decision.connect( hobyoPirateManager[i], adj);
1707     }
1708
1709     for(int i = 0; i < harardherePirateManager.length; ++i)
1710     {
1711        decision.connect( harardherePirateManager[i], adj);
1712     }
1713
1714     //**Allows Navy vessels to signal pirates when detections occur**//
1715     Adapter signalPiarteAdapter = new Adapter( "SignalPirate,"
1716                                        "DetectedByNavy" );
1717     for ( int i = 0 ; i < elaayoPirateManager.length ; ++i )
1718     {
1719        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1720        {
1721           signalPiarteAdapter.connect( goaNavyManager[j],
1722                                        elaayoPirateManager[i] );
1723        }
1724     }
1725     for ( int i = 0 ; i < elaayoPirateManager.length ; ++i )
1726     {
1727        for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1728        {
1729           signalPiarteAdapter.connect( ioNavyManager[j],
1730                                        elaayoPirateManager[i] );
1731        }
1732     }
1733
1734     for ( int i = 0 ; i < qandalaPirateManager.length ; ++i )
1735     {
1736        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1737        {
1738           signalPiarteAdapter.connect( goaNavyManager[j],
1739                                        qandalaPirateManager[i] );
1740        }
1741     }
1742
1743     for ( int i = 0 ; i < qandalaPirateManager.length ; ++i )
1744     {
1745        for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1746        {
1747           signalPiarteAdapter.connect( ioNavyManager[j],
1748                                        qandalaPirateManager[i] );
1749        }
1750     }
1751
1752     for ( int i = 0 ; i < aluulaPirateManager.length ; ++i )
1753     {
1754        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1755        {
1756           signalPiarteAdapter.connect( goaNavyManager[j],
```

```
1757                                    aluulaPirateManager[i] );
1758            }
1759        }
1760
1761        for ( int i = 0 ; i < aluulaPirateManager.length ; ++i )
1762        {
1763            for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1764            {
1765                signalPiarteAdapter.connect( ioNavyManager[j],
1766                                    aluulaPirateManager[i] );
1767            }
1768        }
1769
1770        for ( int i = 0 ; i < bargalPirateManager.length ; ++i )
1771        {
1772            for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1773            {
1774                signalPiarteAdapter.connect( goaNavyManager[j],
1775                                    bargalPirateManager[i] );
1776            }
1777        }
1778
1779        for ( int i = 0 ; i < bargalPirateManager.length ; ++i )
1780        {
1781            for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1782            {
1783                signalPiarteAdapter.connect( ioNavyManager[j],
1784                                    bargalPirateManager[i] );
1785            }
1786        }
1787
1788        for ( int i = 0 ; i < hafunPirateManager.length ; ++i )
1789        {
1790            for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1791            {
1792                signalPiarteAdapter.connect( goaNavyManager[j],
1793                                    hafunPirateManager[i] );
1794            }
1795        }
1796
1797        for ( int i = 0 ; i < hafunPirateManager.length ; ++i )
1798        {
1799            for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1800            {
1801                signalPiarteAdapter.connect( ioNavyManager[j],
1802                                    hafunPirateManager[i] );
1803            }
1804        }
1805
1806        for ( int i = 0 ; i < baylaPirateManager.length ; ++i )
1807        {
1808            for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1809            {
1810                signalPiarteAdapter.connect( goaNavyManager[j],
1811                                    baylaPirateManager[i] );
1812            }
1813        }
1814
1815        for ( int i = 0 ; i < baylaPirateManager.length ; ++i )
1816        {
1817            for ( int j = 0 ; j < ioNavyManager.length ; ++j )
```

```
1818        {
1819            signalPiarteAdapter.connect( ioNavyManager[j],
1820                                         baylaPirateManager[i] );
1821        }
1822    }
1823
1824    for ( int i = 0 ; i < eylPirateManager.length ; ++i )
1825    {
1826        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1827        {
1828            signalPiarteAdapter.connect( goaNavyManager[j],
1829                                         eylPirateManager[i] );
1830        }
1831    }
1832
1833    for ( int i = 0 ; i < eylPirateManager.length ; ++i )
1834    {
1835        for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1836        {
1837            signalPiarteAdapter.connect( ioNavyManager[j],
1838                                         eylPirateManager[i] );
1839        }
1840    }
1841
1842    for ( int i = 0 ; i < garacadPirateManager.length ; ++i )
1843    {
1844        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1845        {
1846            signalPiarteAdapter.connect( goaNavyManager[j],
1847                                         garacadPirateManager[i] );
1848        }
1849    }
1850
1851    for ( int i = 0 ; i < garacadPirateManager.length ; ++i )
1852    {
1853        for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1854        {
1855            signalPiarteAdapter.connect( ioNavyManager[j],
1856                                         garacadPirateManager[i] );
1857        }
1858    }
1859
1860    for ( int i = 0 ; i < hobyoPirateManager.length ; ++i )
1861    {
1862        for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1863        {
1864            signalPiarteAdapter.connect( goaNavyManager[j],
1865                                         hobyoPirateManager[i] );
1866        }
1867    }
1868
1869    for ( int i = 0 ; i < hobyoPirateManager.length ; ++i )
1870    {
1871        for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1872        {
1873            signalPiarteAdapter.connect( ioNavyManager[j],
1874                                         hobyoPirateManager[i] );
1875        }
1876    }
1877
1878    for ( int i = 0 ; i < harardherePirateManager.length ; ++i )
```

```
1879    {
1880       for ( int j = 0 ; j < goaNavyManager.length ; ++j )
1881       {
1882          signalPiarteAdapter.connect( goaNavyManager[j],
1883                                       harardherePirateManager[i] );
1884       }
1885    }
1886
1887    for ( int i = 0 ; i < harardherePirateManager.length ; ++i )
1888    {
1889       for ( int j = 0 ; j < ioNavyManager.length ; ++j )
1890       {
1891          signalPiarteAdapter.connect( ioNavyManager[j],
1892                                       harardherePirateManager[i] );
1893       }
1894    }
1895
1896    //Allows Merchants to send distress call to Navy
1897    Adapter merchantDistressAdapter = new Adapter( "RadioNavy,"
1898                                                   "RcvDistressCall" );
1899    for ( int i = 0 ; i < suezToOmanMerchantManager.length ; ++i )
1900    {
1901       for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1902       {
1903          merchantDistressAdapter.connect( suezToOmanMerchantManager[i],
1904                                           goaNavyManager[j] );
1905
1906       }
1907    }
1908
1909    for ( int i = 0 ; i < suezToOmanMerchantManager.length ; ++i )
1910    {
1911       for ( int j = 0 ; j < ioNavyMover.length ; ++j )
1912       {
1913          merchantDistressAdapter.connect( suezToOmanMerchantManager[i],
1914                                           ioNavyManager[j] );
1915
1916       }
1917    }
1918
1919    for ( int i = 0 ; i < suezToMaldivesMerchantManager.length ; ++i )
1920    {
1921       for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1922       {
1923          merchantDistressAdapter.connect(
1924                suezToMaldivesMerchantManager[i], goaNavyManager[j] );
1925
1926       }
1927    }
1928
1929    for ( int i = 0 ; i < suezToMaldivesMerchantManager.length ; ++i )
1930    {
1931       for ( int j = 0 ; j < ioNavyMover.length ; ++j )
1932       {
1933          merchantDistressAdapter.connect(
1934                suezToMaldivesMerchantManager[i], ioNavyManager[j] );
1935
1936       }
1937    }
1938
1939    for ( int i = 0 ; i < omanToSuezMerchantManager.length ; ++i )
```

```
1940        {
1941           for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1942           {
1943              merchantDistressAdapter.connect( omanToSuezMerchantManager[i],
1944                                               goaNavyManager[j] );
1945
1946           }
1947        }
1948
1949        for ( int i = 0 ; i < omanToSuezMerchantManager.length ; ++i )
1950        {
1951           for ( int j = 0 ; j < ioNavyMover.length ; ++j )
1952           {
1953              merchantDistressAdapter.connect( omanToSuezMerchantManager[i],
1954                                               ioNavyManager[j] );
1955
1956           }
1957        }
1958
1959        for ( int i = 0 ; i < omanToMaldivesMerchantManager.length ; ++i )
1960        {
1961           for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1962           {
1963              merchantDistressAdapter.connect(
1964                    omanToMaldivesMerchantManager[i],goaNavyManager[j] );
1965           }
1966        }
1967
1968        for ( int i = 0 ; i < omanToMaldivesMerchantManager.length ; ++i )
1969        {
1970           for ( int j = 0 ; j < ioNavyMover.length ; ++j )
1971           {
1972              merchantDistressAdapter.connect(
1973                    omanToMaldivesMerchantManager[i], ioNavyManager[j] );
1974           }
1975        }
1976
1977        for ( int i = 0 ; i < maldivesToSuezMerchantManager.length ; ++i )
1978        {
1979           for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1980           {
1981              merchantDistressAdapter.connect(
1982                    maldivesToSuezMerchantManager[i], goaNavyManager[j] );
1983           }
1984        }
1985
1986        for ( int i = 0 ; i < maldivesToSuezMerchantManager.length ; ++i )
1987        {
1988           for ( int j = 0 ; j < ioNavyMover.length ; ++j )
1989           {
1990              merchantDistressAdapter.connect(
1991                    maldivesToSuezMerchantManager[i], ioNavyManager[j] );
1992
1993           }
1994        }
1995
1996        for ( int i = 0 ; i < maldivesToOmanMerchantManager.length ; ++i )
1997        {
1998           for ( int j = 0 ; j < goaNavyMover.length ; ++j )
1999           {
2000              merchantDistressAdapter.connect(
```

```
                maldivesToOmanMerchantManager[i], goaNavyManager[j] );

        }
    }

    for ( int i = 0 ; i < maldivesToOmanMerchantManager.length ; ++i )
    {
        for ( int j = 0 ; j < ioNavyMover.length ; ++j )
        {
            merchantDistressAdapter.connect(
                maldivesToOmanMerchantManager[i],ioNavyManager[j] );

        }
    }


//*****************End of Adapters for Simulation*******************************//

//*************Start of Property Change Listeners for Stats*********************//

    SimpleStatsTally elaayoDepartStat =
            new SimpleStatsTally("numberDepartedGOA");
        epc.addPropertyChangeListener("numberDepartedGOA,"
                                elaayoDepartStat);

        SimpleStatsTally qandalaDepartStat =
            new SimpleStatsTally("numberDepartedGOA");
        qpc.addPropertyChangeListener ( "numberDepartedGOA,"
                                qandalaDepartStat);

        SimpleStatsTally aluulaDepartStat =
            new SimpleStatsTally("numberDepartedGOA");
        apc.addPropertyChangeListener ( "numberDepartedGOA,"
                                aluulaDepartStat);

        SimpleStatsTally bargalDepartStat =
            new SimpleStatsTally("numberDepartedIO");
        bpc.addPropertyChangeListener ( "numberDepartedIO,"
                                bargalDepartStat);

        SimpleStatsTally hafunDepartStat =
            new SimpleStatsTally("numberDepartedIO");
        hpc.addPropertyChangeListener ( "numberDepartedIO,"
                                hafunDepartStat);

        SimpleStatsTally baylaDepartStat =
            new SimpleStatsTally("numberDepartedIO");
        baypc.addPropertyChangeListener ( "numberDepartedIO,"
                                  baylaDepartStat);

     SimpleStatsTally eylDepartStat =
            new SimpleStatsTally("numberDepartedIO");
        eylpc.addPropertyChangeListener ( "numberDepartedIO,"
                                  eylDepartStat);

    SimpleStatsTally garacadDepartStat =
            new SimpleStatsTally("numberDepartedIO");
        gpc.addPropertyChangeListener ( "numberDepartedIO,"
                                garacadDepartStat);

    SimpleStatsTally hobyoDepartStat =
```

```
2062            new SimpleStatsTally(“numberDepartedIO”);
2063        hobpc.addPropertyChangeListener ( “numberDepartedIO,”
2064                                    hobyoDepartStat);
2065
2066    SimpleStatsTally harardhereDepartStat =
2067            new SimpleStatsTally(“numberDepartedIO”);
2068        harpc.addPropertyChangeListener ( “numberDepartedIO,”
2069                                    harardhereDepartStat);
2070
2071    SimpleStatsTally goaNavyDetectionStat =
2072            new SimpleStatsTally( “numberPiratesDetected”);
2073    for (int i = 0; i < goaNavyManager.length; i++)
2074    {
2075        goaNavyManager[i].addPropertyChangeListener(
2076                “numberPiratesDetected,”goaNavyDetectionStat );
2077    }
2078
2079    SimpleStatsTally ioNavyDetectionStat =
2080            new SimpleStatsTally( “numberPiratesDetected”);
2081    for (int i = 0; i < ioNavyManager.length; i++)
2082    {
2083        ioNavyManager[i].addPropertyChangeListener(
2084                “numberPiratesDetected,”ioNavyDetectionStat );
2085    }
2086
2087    SimpleStatsTally elaayoAttemptStat =
2088            new SimpleStatsTally(“numberAttemptedAttacks”);
2089    for ( int i = 0; i < elaayoPirateManager.length; i++ )
2090    {
2091        elaayoPirateManager[i].addPropertyChangeListener(
2092                “numberAttemptedAttacks,” elaayoAttemptStat);
2093    }
2094
2095   SimpleStatsTally aluulaAttemptStat =
2096            new SimpleStatsTally(“numberAttemptedAttacks”);
2097    for ( int i = 0; i < aluulaPirateManager.length; i++ )
2098    {
2099
2100        aluulaPirateManager[i].addPropertyChangeListener(
2101                “numberAttemptedAttacks,” aluulaAttemptStat);
2102    }
2103
2104    SimpleStatsTally qandalaAttemptStat =
2105            new SimpleStatsTally(“numberAttemptedAttacks”);
2106    for ( int i = 0; i < qandalaPirateManager.length; i++ )
2107    {
2108
2109        qandalaPirateManager[i].addPropertyChangeListener(
2110                “numberAttemptedAttacks,” qandalaAttemptStat);
2111    }
2112
2113    SimpleStatsTally bargalAttemptStat =
2114            new SimpleStatsTally(“numberAttemptedAttacks”);
2115    for ( int i = 0; i < bargalPirateManager.length; i++ )
2116    {
2117        bargalPirateManager[i].addPropertyChangeListener(
2118                “numberAttemptedAttacks,” bargalAttemptStat);
2119    }
2120
2121    SimpleStatsTally hafunAttemptStat =
2122            new SimpleStatsTally(“numberAttemptedAttacks”);
```

```
2123    for ( int i = 0; i < hafunPirateManager.length; i++ )
2124    {
2125       hafunPirateManager[i].addPropertyChangeListener(
2126             “numberAttemptedAttacks,”
2127             hafunAttemptStat);
2128    }
2129
2130    SimpleStatsTally baylaAttemptStat =
2131             new SimpleStatsTally(“numberAttemptedAttacks”);
2132    for ( int i = 0; i < baylaPirateManager.length; i++ )
2133    {
2134       baylaPirateManager[i].addPropertyChangeListener(
2135             “numberAttemptedAttacks,” baylaAttemptStat);
2136    }
2137
2138    SimpleStatsTally eylAttemptStat =
2139             new SimpleStatsTally(“numberAttemptedAttacks”);
2140    for ( int i = 0; i < eylPirateManager.length; i++ )
2141    {
2142       eylPirateManager[i].addPropertyChangeListener(
2143             “numberAttemptedAttacks,” eylAttemptStat);
2144    }
2145
2146    SimpleStatsTally garacadAttemptStat =
2147             new SimpleStatsTally(“numberAttemptedAttacks”);
2148    for ( int i = 0; i < garacadPirateManager.length; i++ )
2149    {
2150       garacadPirateManager[i].addPropertyChangeListener(
2151             “numberAttemptedAttacks,” garacadAttemptStat);
2152    }
2153
2154    SimpleStatsTally hobyoAttemptStat =
2155             new SimpleStatsTally(“numberAttemptedAttacks”);
2156    for ( int i = 0; i < hobyoPirateManager.length; i++ )
2157    {
2158       hobyoPirateManager[i].addPropertyChangeListener(
2159             “numberAttemptedAttacks,” hobyoAttemptStat);
2160    }
2161
2162    SimpleStatsTally harardhereAttemptStat =
2163             new SimpleStatsTally(“numberAttemptedAttacks”);
2164    for ( int i = 0; i < harardherePirateManager.length; i++ )
2165    {
2166       harardherePirateManager[i].addPropertyChangeListener(
2167             “numberAttemptedAttacks,” harardhereAttemptStat);
2168    }
2169
2170    SimpleStatsTally stmDepartStat =
2171          new SimpleStatsTally(“numberDepartedPort”);
2172    stm.addPropertyChangeListener(“numberDepartedPort,”
2173          stmDepartStat);
2174
2175    SimpleStatsTally stoDepartStat =
2176          new SimpleStatsTally(“numberDepartedPort”);
2177    sto.addPropertyChangeListener(“numberDepartedPort,”
2178          stoDepartStat);
2179
2180    SimpleStatsTally mtsDepartStat =
2181          new SimpleStatsTally(“numberDepartedPort”);
2182       mts.addPropertyChangeListener ( “numberDepartedPort,”
2183                                   mtsDepartStat);
```

```
2184
2185        SimpleStatsTally mtoDepartStat =
2186              new SimpleStatsTally("numberDepartedPort");
2187        mto.addPropertyChangeListener("numberDepartedPort ,"
2188              mtoDepartStat);
2189
2190        SimpleStatsTally otmDepartStat =
2191              new SimpleStatsTally("numberDepartedPort");
2192           otm.addPropertyChangeListener ( "numberDepartedPort,"
2193                                    otmDepartStat);
2194
2195        SimpleStatsTally otsDepartStat =
2196              new SimpleStatsTally("numberDepartedPort");
2197           ots.addPropertyChangeListener ( "numberDepartedPort,"
2198                                   otsDepartStat);
2199
2200 //**************End of Property Change Listeners for Stats*********************//
2201 //********************Start of Stats and Schedule Implementation***************//
2202        LinkedList goaDepartures = new LinkedList();
2203        LinkedList ioDepartures = new LinkedList();
2204        LinkedList numPiratesDetected = new LinkedList();
2205        LinkedList navalEffectivenessList = new LinkedList();
2206        LinkedList pirateAttemptList = new LinkedList();
2207        LinkedList pirateEffectiveness1List = new LinkedList();
2208        LinkedList pirateEffectiveness2List = new LinkedList();
2209        LinkedList merchantTransits = new LinkedList();
2210
2211        for ( int i = 0; i < 30; ++i )
2212        {
2213
2214           Schedule.setDecimalFormat("0.00");
2215           Schedule.setVerbose(false);
2216           Schedule.setEventSourceVerbose(false);
2217           Schedule.stopAtTime(simTime);
2218           elaayoDepartStat.reset ();
2219           qandalaDepartStat.reset ();
2220           aluulaDepartStat.reset ();
2221           bargalDepartStat.reset ();
2222           hafunDepartStat.reset ();
2223           baylaDepartStat.reset ();
2224           eylDepartStat.reset ();
2225           garacadDepartStat.reset ();
2226           hobyoDepartStat.reset ();
2227           harardhereDepartStat.reset ();
2228           totalNumDepartedGOA = 0;
2229           totalNumDepartedIO = 0;
2230           totalNumberPiratesDeparted = 0;
2231           goaNavyDetectionStat.reset ();
2232           ioNavyDetectionStat.reset ();
2233           totalNumberPiratesDetected = 0;
2234           navalEffectiveness = 0;
2235           elaayoAttemptStat.reset();
2236           qandalaAttemptStat.reset();
2237           aluulaAttemptStat.reset();
2238           bargalAttemptStat.reset();
2239           hafunAttemptStat.reset();
2240           baylaAttemptStat.reset();
2241           eylAttemptStat.reset();
2242           garacadAttemptStat.reset();
2243           hobyoAttemptStat.reset();
2244           harardhereAttemptStat.reset();
```

```
2245        stoDepartStat.reset();
2246        stmDepartStat.reset();
2247        otmDepartStat.reset();
2248        otsDepartStat.reset();
2249        mtoDepartStat.reset();
2250        mtsDepartStat.reset();
2251        totalNumberMerchantTransits = 0;
2252        totalAttemptedAttacks = 0;
2253        pirateEffectiveness1 = 0;
2254        Schedule.reset();
2255        Schedule.startSimulation();
2256
2257        totalNumberPiratesDetected = goaNavyDetectionStat.getCount()
2258                                   + ioNavyDetectionStat.getCount();
2259
2260        totalNumDepartedGOA = elaayoDepartStat.getCount()
2261              + aluulaDepartStat.getCount()
2262              + qandalaDepartStat.getCount();
2263
2264        totalNumDepartedIO = baylaDepartStat.getCount()
2265              + hafunDepartStat.getCount()
2266              + baylaDepartStat.getCount()
2267              + eylDepartStat.getCount()
2268              + garacadDepartStat.getCount()
2269              + hobyoDepartStat.getCount()
2270              + harardhereDepartStat.getCount();
2271
2272        totalNumberPiratesDeparted = totalNumDepartedGOA +
2273                                   totalNumDepartedIO;
2274
2275        System.out.println( "Total Number Pirates Detected: " +
2276              totalNumberPiratesDeparted );
2277
2278        navalEffectiveness = totalNumberPiratesDetected
2279              / totalNumberPiratesDeparted;
2280
2281        totalAttemptedAttacks = elaayoAttemptStat.getCount()
2282              + aluulaAttemptStat.getCount()
2283              + qandalaAttemptStat.getCount()
2284              + bargalAttemptStat.getCount()
2285              + hafunAttemptStat.getCount()
2286              + baylaAttemptStat.getCount()
2287              + eylAttemptStat.getCount()
2288              + garacadAttemptStat.getCount()
2289              + hobyoAttemptStat.getCount()
2290              + harardhereAttemptStat.getCount();
2291
2292        pirateEffectiveness1 = totalAttemptedAttacks
2293              / totalNumberPiratesDeparted;
2294
2295        totalNumberMerchantTransits = stoDepartStat.getCount()
2296              + stmDepartStat.getCount()
2297              + otmDepartStat.getCount()
2298              + otsDepartStat.getCount()
2299              + mtoDepartStat.getCount()
2300              + mtsDepartStat.getCount();
2301
2302        pirateEffectiveness2 =  totalAttemptedAttacks /
2303                              totalNumberMerchantTransits;
2304
2305        goaDepartures.add(totalNumDepartedGOA);
```

```
2306            ioDepartures.add(totalNumDepartedIO);
2307            merchantTransits.add ( totalNumberMerchantTransits );
2308            numPiratesDetected.add(totalNumberPiratesDetected);
2309            navalEffectivenessList.add(navalEffectiveness);
2310            pirateAttemptList.add(totalAttemptedAttacks);
2311            pirateEffectiveness1List.add(pirateEffectiveness1);
2312            pirateEffectiveness2List.add(pirateEffectiveness2);
2313
2314            System.out.println("Ellayo Numbers: " +
2315                  epc.getMyPirates ().size ());
2316            System.out.println("Ellayo Departures: " +
2317                  elaayoDepartStat.getCount());
2318            System.out.println( "Number Merchants: " + merchantTransits );
2319
2320        }
2321
2322        System.out.println( "Pirate Camp Operations Stats Output");
2323        System.out.println( "Goa Departures: " + goaDepartures );
2324        System.out.println( "IO Departures: " + ioDepartures );
2325        System.out.println( "Merchant Transits: " + merchantTransits);
2326        System.out.println( "Pirates Detected: " + numPiratesDetected );
2327        System.out.println( "Naval Effectiveness: " + navalEffectivenessList );
2328        System.out.println( "Attempted Attacks: " + pirateAttemptList );
2329        System.out.println( "Pirate Effectiveness 1: " +
2330                        pirateEffectiveness1List );
2331        System.out.println( "Pirate Effectiveness 2: " +
2332                        pirateEffectiveness2List );
2333
2334
2335 //**********************End of Schedule Implementation**************************//
2336 //*********************************END OF ASSEMBLY*************************************//
2337
2338    }
2339 }
2340
```

APPENDIX L. PLATFORM CLASS JAVA CODE

```
1 package supplemental;
2
3 import java.awt.geom.Point2D;
4 import simkit.Priority;
5 import simkit.smd.BasicLinearMover;
6 import simkit.smd.Mover;
7
8 /**
9  * @version $Id$
10  * @author Chad R Hutchins & Arnie Buss
11  */
12 public class Platform extends BasicLinearMover {
13
14    private PlatformType type;
15    protected boolean isAlive;
16
17    public Platform( String name, Point2D initialLocation,
18                  double maxSpeed, PlatformType type )
19    {
20       super( name, initialLocation, maxSpeed );
21       this.setType( type );
22    }
23
24    /**
25     * @return the type
26     */
27    public PlatformType getType()
28    {
29       return type;
30    }
31
32    /**
33     * @param type the type to set
34     */
35    public void setType( PlatformType type )
36    {
37       this.type = type;
38    }
39
40    /**
41     * @return the isAlive
42     */
43    public boolean getIsAlive ()
44    {
45       return isAlive;
46    }
47
48    /**
49     * removes (just) mover
50     *
51     * @param mover dead Mover
52     */
53    public void doDie( Mover mover )
54    {
55       //isAlive = false;
56       this.removeMover( mover );
```

```
57      this.interruptAll();
58
59      waitDelay( “OrderStop,” 0.0, Priority.HIGH, mover );
60    }
61
62    /**
63     * If in movers set, remove. Stop listening to it, and interrupt all pending
64     * events with mover as an argument.
65     *
66     * @param mover Mover to be removed
67     */
68    public void removeMover( Mover mover )
69    {
70      mover.removeSimEventListener( this );
71      this.interruptAllWithArgs( mover );
72    }
73
74    @Override
75    public String toString()
76    {
77      return super.toString().
78            replaceAll( “BasicLinearMover,” “Platform” )
79           + “ “ + getType();
80    }
81 }
```

APPENDIX M. PLATFORM TYPE CLASS JAVA CODE

```
1 /*
2  * PlatformType.java
3  *
4  */
5 package supplemental;
6
7 /**
8  * All the different entity players in the scenario
9  *
10  * @version $Id: PlatformType.java 120 2012–11–15 23:36:37Z crhutchi $
11  * @author Chad R Hutchins
12  */
13 public enum PlatformType {
14     NAVY,
15     MERCHANT,
16     PIRATE
17 }
18
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX N. NAVY STATE JAVA CODE

```
1 /*
2  * NavyState.java
3  */
4 package supplemental;
5
6 /**
7  * Enums that describe the state of a navy ship while conducting counter-piracy
8  * operations
9  *
10  * @author Chad R Hutchins
11  * @version $Id: NavyState.java 112 2012–11–07 06:53:20Z crhutchi $
12  */
13 public enum NavyState {
14
15     DEAD_IN_WATER,
16     PATROLLING,
17     INTERCEPTING,
18     BOARDING,
19     RETURNING_TO_PATROL
20 }
21
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX O. PIRATE STATE JAVA CODE

```
1 /*
2  * PirateState.java
3  */
4 package supplemental;
5
6 /**
7  * Enums that describe the state of Somali pirates
8  *
9  * @version $Id:
10  * @author Chad R Hutchins
11  *
12  */
13 public enum PirateState {
14
15    WAITING_AT_BASE,
16    ENROUTE_TO_PATROL,
17    PATROLLING,
18    INTERCEPTING,
19    ATTACKING,
20    RETURNING_TO_BASE,
21    RETURNING_WITH_MERCHANT,
22    NAVY_BOARDED;
23
24 }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX P. MERCHANT STATE JAVA CODE

```
1 /*
2  * MerchantState.java
3  */
4 package supplemental;
5
6 /**
7  * Enums that describe the state of a merchant ship around the Horn Of Africa
8  *
9  * @version $Id:
10  * @author Chad R Hutchins
11  *
12  */
13 public enum MerchantState {
14
15    DEAD_IN_WATER,
16    TRANSITTING,
17    EVADING,
18    BEEN_ATTACKED,
19    HIJACKED;
20
21 }
```

THIS PAGE INTENTIONALLY LEFT BLANK

APPENDIX Q. OPENMAP™ SIMULATION LAYER JAVA CODE

```
1 package oldStuff;
 2 /*
 3  * Java imports
 4  */
 5
 6 import com.bbn.openmap.Layer;
 7 import com.bbn.openmap.event.LayerStatusEvent;
 8 import com.bbn.openmap.event.MapMouseListener;
 9 import com.bbn.openmap.event.ProjectionEvent;
10 import com.bbn.openmap.omGraphics.OMCircle;
11 import com.bbn.openmap.omGraphics.OMGraphicList;
12 import com.bbn.openmap.omGraphics.OMText;
13 import com.bbn.openmap.proj.Projection;
14 import java.awt.Color;
15 import java.awt.event.ActionEvent;
16 import java.awt.event.ActionListener;
17 import java.awt.event.MouseEvent;
18 import java.awt.geom.Point2D;
19 import javax.swing.JButton;
20 import javax.swing.JPanel;
21 import simkit.SimEvent;
22 import simkit.SimEventListener;
23
24 /**
25  * This is an OpenMap layer for simulating entities on a map.
26  *
27  * @author Murat Gunal Modified by Chad R Hutchins
28  */
29 public class SimulationLayer extends Layer implements SimEventListener,
30                                         MapMouseListener,
31                                         //ModEventListener,
32                                         ActionListener {
33
34    OMText text1, text2;
35    OMCircle[] circle;
36    OMCircle circle1, circle2, circle3, circle4, circle5, circle6, circle7,
37          circle8, circle9, circle10, circle11, circle12, circle13, circle14,
38          circle15, circle16, circle17, circle18;//, circle19;
39    OMCircle moverCircle1;
40    OMGraphicList graphicList;
41    // friendly;
42    private JButton runButton = new JButton( "R U N   S I M U L A T I O N" );
43    public Projection proj;
44    public OpenMapDemo scn;
45    public int detectionCounter = 0;
46
47    public SimulationLayer()
48    {
49       scn = new OpenMapDemo();
50
51       graphicList = new OMGraphicList();
52
53       Point2D pirateIO = scn.getLocationIoPirateMover( 0 );
54       circle1 = new OMCircle( ( float ) pirateIO.getX(),
55                               ( float ) pirateIO.getY(),
56                               scn.nmToDeg( 1, 15.0 )); //12NM
57       circle1.setLinePaint( Color.RED );
```

```
58 //      moverCircle1 = new OMCircle( ( float ) pirateIO.getX(),
59 //                                   ( float ) pirateIO.getY(), 3, Length.METER );
60 //      moverCircle1.setFillPaint( Color.RED );
61
62        Point2D pirateGOA = scn.getLocationGoaPirateMover( 0 );
63        circle2 = new OMCircle( ( float ) pirateGOA.getX(),
64                                ( float ) pirateGOA.getY(),
65                                scn.nmToDeg( 1, 15.0 ) ); //12NM
66        circle2.setLinePaint( Color.RED );
67
68 //     Point2D pirateGOA2 = scn.getLocationGoaPirateMover( 0 );
69 //     circle19 = new OMCircle( ( float ) pirateGOA2.getX(),
70 //                              ( float ) pirateGOA2.getY(),
71 //                              scn.nmToDeg( 1, 5.0f ) ); //12NM
72 //     circle19.setLinePaint( Color.RED );
73
74        Point2D navyIoPB6 = scn.getLocationIoNavyMover( 0 );
75        circle16 = new OMCircle( ( float ) navyIoPB6.getX(),
76                                 ( float ) navyIoPB6.getY(),
77                                 scn.nmToDeg( 1, 20.0 ) ); //25NM
78        circle16.setLinePaint( Color.BLUE );
79
80        Point2D navyIoPB7 = scn.getLocationIoNavyMover( 1 );
81        circle17 = new OMCircle( ( float ) navyIoPB7.getX(),
82                                 ( float ) navyIoPB7.getY(),
83                                 scn.nmToDeg( 1, 20.0f ) ); //25NM
84        circle17.setLinePaint( Color.BLUE );
85
86        Point2D navyIoPB8 = scn.getLocationIoNavyMover( 2 );
87        circle3 = new OMCircle( ( float ) navyIoPB8.getX(), ( float ) navyIoPB8.
88              getY(),
89                              scn.nmToDeg( 1, 20.0f ) ); //25NM
90        circle3.setLinePaint( Color.BLUE );
91
92        Point2D navyIoPB9 = scn.getLocationIoNavyMover( 3 );
93        circle6 = new OMCircle( ( float ) navyIoPB9.getX(), ( float ) navyIoPB9.
94              getY(),
95                              scn.nmToDeg(  1, 20.0f ) ); //25NM
96        circle6.setLinePaint( Color.BLUE );
97
98        Point2D navyIoPB10 = scn.getLocationIoNavyMover( 4 );
99        circle7 = new OMCircle( ( float ) navyIoPB10.getX(),
100                                ( float ) navyIoPB10.getY(),
101                                scn.nmToDeg(  1, 20.0f ) ); //25NM
102       circle7.setLinePaint( Color.BLUE );
103
104       Point2D navyIoPB11 = scn.getLocationIoNavyMover( 5 );
105       circle8 = new OMCircle( ( float ) navyIoPB11.getX(),
106                               ( float ) navyIoPB11.getY(),
107                               scn.nmToDeg(  1, 20.0f ) ); //25NM
108       circle8.setLinePaint( Color.BLUE );
109
110       Point2D navyIoPB12 = scn.getLocationIoNavyMover( 6 );
111       circle9 = new OMCircle( ( float ) navyIoPB12.getX(),
112                               ( float ) navyIoPB12.getY(),
113                               scn.nmToDeg(  1, 20.0f ) ); //25NM
114       circle9.setLinePaint( Color.BLUE );
115
116       Point2D navyIoPB13 = scn.getLocationIoNavyMover( 7 );
117       circle10 = new OMCircle( ( float ) navyIoPB13.getX(),
118                                ( float ) navyIoPB13.getY(),
```

```
119                     scn.nmToDeg(  1, 20.0f ) ); //25NM
120     circle10.setLinePaint( Color.BLUE );
121
122     Point2D navyGoaPB1 = scn.getLocationGoaNavyMover( 0 );
123     circle11 = new OMCircle( ( float ) navyGoaPB1.getX(),
124                     ( float ) navyGoaPB1.getY(),
125                     scn.nmToDeg(  1, 20.0f ) ); //25NM
126     circle11.setLinePaint( Color.BLUE );
127
128     Point2D navyGoaPB2 = scn.getLocationGoaNavyMover( 1 );
129     circle12 = new OMCircle( ( float ) navyGoaPB2.getX(),
130                     ( float ) navyGoaPB2.getY(),
131                     scn.nmToDeg(  1, 20.0f ) ); //25NM
132     circle12.setLinePaint( Color.BLUE );
133
134     Point2D navyGoaPB3 = scn.getLocationGoaNavyMover( 2 );
135     circle13 = new OMCircle( ( float ) navyGoaPB3.getX(),
136                     ( float ) navyGoaPB3.getY(),
137                     scn.nmToDeg(  1, 20.0f) ); //25NM
138     circle13.setLinePaint( Color.BLUE );
139
140     Point2D navyGoaPB4 = scn.getLocationGoaNavyMover( 3 );
141     circle14 = new OMCircle( ( float ) navyGoaPB4.getX(),
142                     ( float ) navyGoaPB4.getY(),
143                     scn.nmToDeg(  1, 20.0f ) ); //25NM
144     circle14.setLinePaint( Color.BLUE );
145
146     Point2D navyGoaPB5 = scn.getLocationGoaNavyMover( 4 );
147     circle15 = new OMCircle( ( float ) navyGoaPB5.getX(),
148                     ( float ) navyGoaPB5.getY(),
149                     scn.nmToDeg(  1, 20.0f ) ); //25NM
150     circle15.setLinePaint( Color.BLUE );
151
152     Point2D merchantSB = scn.getLocationSbMerchant( 0 );
153     circle4 = new OMCircle( ( float ) merchantSB.getX(),
154                     ( float ) merchantSB.getY(),
155                     0.33459801    ); //25NM
156     circle4.setLinePaint( Color.MAGENTA );
157
158     Point2D merchantNB = scn.getLocationNbMerchant( 0 );
159     circle5 = new OMCircle( ( float ) merchantNB.getX(),
160                     ( float ) merchantNB.getY(),
161                     scn.nmToDeg(  1, 20.0 ) ); //25NM
162     circle5.setLinePaint( Color.MAGENTA );
163
164     //for ( int i = 0 ; i < scn.ioPirateMover.length ; ++i )
165
166     graphicList.add( circle1 );
167     graphicList.add( circle2 );
168     graphicList.add( circle3 );
169     graphicList.add( circle4 );
170     graphicList.add( circle5 );
171     graphicList.add( circle6 );
172     graphicList.add( circle7 );
173     graphicList.add( circle8 );
174     graphicList.add( circle9 );
175     graphicList.add( circle10 );
176     graphicList.add( circle11 );
177     graphicList.add( circle12 );
178     graphicList.add( circle13 );
179     graphicList.add( circle14 );
```

```
180     graphicList.add( circle15 );
181     graphicList.add( circle16 );
182     graphicList.add( circle17 );
183     //grafikList.add( moverCircle1 );
184   }
185
186   @Override
187   public void processSimEvent( SimEvent e )
188   {
189     fireStatusUpdate( LayerStatusEvent.START_WORKING );
190
191     if ( e.getEventName().
192           equals( "Ping" ) )
193     {
194       OMCircle tempCirc1 = ( OMCircle ) graphicList.getOMGraphicAt( 0 );
195       OMCircle tempCirc2 = ( OMCircle ) graphicList.getOMGraphicAt( 1 );
196       OMCircle tempCirc3 = ( OMCircle ) graphicList.getOMGraphicAt( 2 );
197       OMCircle tempCirc4 = ( OMCircle ) graphicList.getOMGraphicAt( 3 );
198       OMCircle tempCirc5 = ( OMCircle ) graphicList.getOMGraphicAt( 4 );
199       OMCircle tempCirc6 = ( OMCircle ) graphicList.getOMGraphicAt( 5 );
200       OMCircle tempCirc7 = ( OMCircle ) graphicList.getOMGraphicAt( 6 );
201       OMCircle tempCirc8 = ( OMCircle ) graphicList.getOMGraphicAt( 7 );
202       OMCircle tempCirc9 = ( OMCircle ) graphicList.getOMGraphicAt( 8 );
203       OMCircle tempCirc10 = ( OMCircle ) graphicList.getOMGraphicAt( 9 );
204       OMCircle tempCirc11 = ( OMCircle ) graphicList.getOMGraphicAt( 10 );
205       OMCircle tempCirc12 = ( OMCircle ) graphicList.getOMGraphicAt( 11 );
206       OMCircle tempCirc13 = ( OMCircle ) graphicList.getOMGraphicAt( 12 );
207       OMCircle tempCirc14 = ( OMCircle ) graphicList.getOMGraphicAt( 13 );
208       OMCircle tempCirc15 = ( OMCircle ) graphicList.getOMGraphicAt( 14 );
209       OMCircle tempCirc16 = ( OMCircle ) graphicList.getOMGraphicAt( 15 );
210       OMCircle tempCirc17 = ( OMCircle ) graphicList.getOMGraphicAt( 16 );
211       //OMCircle tempCirc19 = ( OMCircle ) graphicList.getOMGraphicAt( 17 );
212
213       tempCirc1.setLatLon( ( float ) scn.getLocationIoPirateMover( 0 ).
214             getX(),
215                   ( float ) scn.getLocationIoPirateMover( 0 ).
216             getY() );
217
218       tempCirc2.setLatLon( ( float ) scn.getLocationGoaPirateMover( 0 ).
219             getX(),
220                   ( float ) scn.getLocationGoaPirateMover( 0 ).
221             getY() );
222
223       tempCirc3.setLatLon( ( float ) scn.getLocationIoNavyMover( 2 ).
224             getX(),
225                   ( float ) scn.getLocationIoNavyMover( 2 ).
226             getY() );
227
228       tempCirc4.setLatLon( ( float ) scn.getLocationSbMerchant( 0 ).
229             getX(),
230                   ( float ) scn.getLocationSbMerchant( 0 ).
231             getY() );
232
233       tempCirc5.setLatLon( ( float ) scn.getLocationNbMerchant( 0 ).
234             getX(),
235                   ( float ) scn.getLocationNbMerchant( 0 ).
236             getY() );
237
238       tempCirc6.setLatLon( ( float ) scn.getLocationIoNavyMover( 3 ).
239             getX(),
240                   ( float ) scn.getLocationIoNavyMover( 3 ).
```

```
241                 getY() );
242
243         tempCirc7.setLatLon( ( float ) scn.getLocationIoNavyMover( 4 ).
244                 getX(),
245                         ( float ) scn.getLocationIoNavyMover( 4 ).
246                 getY() );
247
248         tempCirc8.setLatLon( ( float ) scn.getLocationIoNavyMover( 5 ).
249                 getX(),
250                         ( float ) scn.getLocationIoNavyMover( 5 ).
251                 getY() );
252
253         tempCirc9.setLatLon( ( float ) scn.getLocationIoNavyMover( 6 ).
254                 getX(),
255                         ( float ) scn.getLocationIoNavyMover( 6 ).
256                 getY() );
257
258         tempCirc10.setLatLon( ( float ) scn.getLocationIoNavyMover( 7 ).
259                 getX(),
260                          ( float ) scn.getLocationIoNavyMover( 7 ).
261                 getY() );
262
263         tempCirc11.setLatLon( ( float ) scn.getLocationGoaNavyMover( 0 ).
264                 getX(),
265                          ( float ) scn.getLocationGoaNavyMover( 0 ).
266                 getY() );
267
268         tempCirc12.setLatLon( ( float ) scn.getLocationGoaNavyMover( 1 ).
269                 getX(),
270                          ( float ) scn.getLocationGoaNavyMover( 1 ).
271                 getY() );
272
273         tempCirc13.setLatLon( ( float ) scn.getLocationGoaNavyMover( 2 ).
274                 getX(),
275                          ( float ) scn.getLocationGoaNavyMover( 2 ).
276                 getY() );
277
278         tempCirc14.setLatLon( ( float ) scn.getLocationGoaNavyMover( 3 ).
279                 getX(),
280                          ( float ) scn.getLocationGoaNavyMover( 3 ).
281                 getY() );
282
283         tempCirc15.setLatLon( ( float ) scn.getLocationGoaNavyMover( 4 ).
284                 getX(),
285                          ( float ) scn.getLocationGoaNavyMover( 4 ).
286                 getY() );
287
288         tempCirc16.setLatLon( ( float ) scn.getLocationIoNavyMover( 0 ).
289                 getX(),
290                          ( float ) scn.getLocationIoNavyMover( 0 ).
291                 getY() );
292
293         tempCirc17.setLatLon( ( float ) scn.getLocationIoNavyMover( 1 ).
294                 getX(),
295                          ( float ) scn.getLocationIoNavyMover( 1 ).
296                 getY() );
297
298 //          tempCirc19.setLatLon( ( float ) scn.getLocationGoaPirateMover( 1 ).
299 //                  getX(),
300 //                           ( float ) scn.getLocationGoaPirateMover( 1 ).
301 //                  getY() );
```

```
302
303         tempCirc1.generate( proj );
304         tempCirc2.generate( proj );
305         tempCirc3.generate( proj );
306         tempCirc4.generate( proj );
307         tempCirc5.generate( proj );
308         tempCirc6.generate( proj );
309         tempCirc7.generate( proj );
310         tempCirc8.generate( proj );
311         tempCirc9.generate( proj );
312         tempCirc10.generate( proj );
313         tempCirc11.generate( proj );
314         tempCirc12.generate( proj );
315         tempCirc13.generate( proj );
316         tempCirc14.generate( proj );
317         tempCirc15.generate( proj );
318         tempCirc16.generate( proj );
319         tempCirc17.generate( proj );
320         //tempCirc19.generate( proj );
321      }
322      if ( e.getEventName().
323            equals( "Detection" ) )
324      {
325         System.out.println(
326               "_____" + getSimTime() );
327         detectionCounter++;
328      }
329      if ( proj != null )
330      {
331         ( ( OMGraphicList ) graphicList ).project( ( Projection ) proj, true );
332      }
333      repaint();
334      fireStatusUpdate( LayerStatusEvent.FINISH_WORKING );
335   }
336
337   @Override
338   public String[] getMouseModeServiceList()
339   {
340      // TODO Auto-generated method stub
341      return null;
342   }
343
344   @Override
345   public boolean mouseClicked( MouseEvent arg0 )
346   {
347      // TODO Auto-generated method stub
348      return false;
349   }
350
351   @Override
352   public boolean mouseDragged( MouseEvent arg0 )
353   {
354      // TODO Auto-generated method stub
355      return false;
356   }
357
358   @Override
359   public void mouseEntered( MouseEvent arg0 )
360   {
361      // TODO Auto-generated method stub
362   }
```

```
363
364   @Override
365   public void mouseExited( MouseEvent arg0 )
366   {
367      // TODO Auto-generated method stub
368   }
369
370   @Override
371   public void mouseMoved()
372   {
373      // TODO Auto-generated method stub
374   }
375
376   @Override
377   public boolean mouseMoved( MouseEvent arg0 )
378   {
379      // TODO Auto-generated method stub
380      return false;
381   }
382
383   @Override
384   public boolean mousePressed( MouseEvent arg0 )
385   {
386      // TODO Auto-generated method stub
387      return false;
388   }
389
390   @Override
391   public boolean mouseReleased( MouseEvent arg0 )
392   {
393      // TODO Auto-generated method stub
394      return false;
395   }
396
397   @Override
398   public void projectionChanged( ProjectionEvent e )
399   {
400      proj = e.getProjection();
401      System.out.println( "projection Changed" );
402      ( ( OMGraphicList ) graphicList ).project( e.getProjection(), true );
403
404      repaint();
405   }
406
407   public void paint( java.awt.Graphics g )
408   {
409      if ( graphicList.size() > 0 )
410      {
411         graphicList.render( g );
412      }
413      fireStatusUpdate( LayerStatusEvent.FINISH_WORKING );
414   }
415
416   public void findAndInit( Object someObj )
417   {
418      /*
419       * if (someObj instanceof DenizSim.myLayer ){
420       * //System.out.println("myLayer is added !!!!!!!!"); //myLayer myL=
421       * (myLayer)someObj; }
422       */
423   }
```

```
424
425    public double getSimTime()
426    {
427       return scn.getSimTime();
428    }
429    //A GUI for the layer
430
431    @Override
432    public java.awt.Component getGUI()
433    {
434       JPanel returnPanel = new JPanel();
435
436       final PingThread2 pt = new PingThread2( 0.1, 100, false );
437
438       pt.addSimEventListener( this );
439       for ( int i = 0 ; i < scn.ioPirateMover.length ; ++i )
440       {
441          scn.ioPirateMover[i].addSimEventListener( this );
442       }
443
444       for ( int i = 0 ; i < scn.ioNavyMover.length ; ++i )
445       {
446          scn.ioNavyMover[i].addSimEventListener( this );
447       }
448
449       for ( int i = 0 ; i < scn.ioPirateSensor.length ; ++i )
450       {
451          scn.ioPirateSensor[i].addSimEventListener( this );
452       }
453
454       for ( int i = 0 ; i < scn.ioPirateSensor.length ; ++i )
455       {
456          scn.ioNavySensor[i].addSimEventListener( this );
457       }
458
459       runButton.addActionListener( new ActionListener() {
460
461          @Override
462          public void actionPerformed( ActionEvent e )
463          {
464             scn.startScenario();
465             pt.startPinging();
466          }
467       } );
468
469       returnPanel.add( runButton );
470
471       return returnPanel;
472    }
473 }
```

APPENDIX R. JAVA SWING SANDBOX FRAME IMPLEMENTATION CODE SNIPPET

```
2218 //**************Start of Sandbox with background image implementation***********//
2219
2220        //Allows for background image
2221        BufferedImage img = null;
2222        File file = new File( "images/test.PNG" );
2223        System.out.println( file.exists() );
2224        img = ImageIO.read( file );
2225
2226        //Scale for background image
2227        double scale = 1.0;
2228
2229        //More scaling
2230        int rescaledWidth = ( int ) ( img.getWidth() * scale );
2231        int rescaledHeight = ( int ) ( img.getHeight() * scale );
2232        BufferedImage resizedImage = new BufferedImage( rescaledWidth,
2233                                              rescaledHeight, img.
2234              getType() );
2235
2236        AffineTransform scaleTransform =
2237           AffineTransform.getScaleInstance( scale, scale );
2238
2239        Graphics2D g = resizedImage.createGraphics();
2240        g.drawImage( img, scaleTransform, null );
2241
2242        //Sandbox for simulation
2243        SandboxFrame sandboxFrame = new SandboxFrame();
2244        Sandbox2 sandbox = sandboxFrame.getSandbox();
2245
2246        //Sets the background image to the appropriate scale
2247        sandbox.setBackroundImage( resizedImage );
2248
2249        sandboxFrame.setSize( resizedImage.getWidth(),
2250                        resizedImage.getHeight() + 100 );
2251
2252        //Sets orgin based on resized image
2253        sandbox.setOrigin( new Point2D.Double( 0.0, resizedImage.getHeight() ));
2254        sandbox.setDrawAxes( true );
2255
2256        //Listener for moust points
2257        sandbox.addMouseMotionListener(
2258              new MouseLocationListener( sandboxFrame ) );
2259
2260        //Window for collecting waypoint data
2261        JFrame wayPointFrame = new JFrame();
2262        wayPointFrame.setSize( 300, 100 );
2263        wayPointFrame.setLocation( ( int ) sandboxFrame.getLocation().
2264              getX() + sandboxFrame.getWidth(),
2265                           ( int ) sandboxFrame.getLocation().
2266              getY() );
2267        WaypointBuilder wayPointBuilder = new WaypointBuilder();
2268        JScrollPane jscrollPane = new JScrollPane( wayPointBuilder );
2269        wayPointFrame.getContentPane().
2270              add( jscrollPane );
2271        wayPointFrame.setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );
2272        wayPointBuilder.addPropertyChangeListener( new PathBuilder() );
```

```
2273
2274        //Add listener to your mouse which allows ability to click the mouse at
2275        //a given point in the Sandbox and get the x and y values.
2276        sandbox.addMouseListener( wayPointBuilder );
2277
2278        sandboxFrame.setVisible( true );
2279        wayPointFrame.setVisible( true );
2280 //**************End of Background and Sandbox implementation******************//
```

APPENDIX S. JAVA SWING WAYPOINT BUILDER JAVA CODE

```
1 package util;
2
3 import animate.Sandbox;
4 import java.awt.event.MouseEvent;
5 import java.awt.event.MouseListener;
6 import java.awt.geom.Point2D;
7 import java.util.ArrayList;
8 import javax.swing.DefaultListModel;
9 import javax.swing.JList;
10 import javax.swing.JPanel;
11 import javax.swing.JScrollPane;
12
13 /**
14  * @version $Id: WaypointBuilder.java 51 2012–06–16 05:20:29Z crhutchi $
15  * @author ahbuss
16  */
17 public class WaypointBuilder extends JPanel implements MouseListener {
18
19    private JList waypointsList;
20    private DefaultListModel waypointListModel;
21
22    public WaypointBuilder() {
23       this.waypointListModel = new DefaultListModel();
24       this.waypointsList = new JList(waypointListModel);
25       this.waypointsList.setVisibleRowCount(10);
26       JScrollPane jscrollPane = new JScrollPane(this.waypointsList);
27       this.add(this.waypointsList);
28    }
29
30    @Override
31    public void mouseClicked(MouseEvent me) {
32       Object source = me.getSource();
33       if (source instanceof Sandbox) {
34          Sandbox sb = (Sandbox) source;
35          double x = me.getX() - sb.getOrigin().getX();
36          double y = sb.getOrigin().getY() - me.getY();
37          Point2D.Double newPoint = new Point2D.Double(x, y);
38          waypointListModel.addElement(newPoint);
39          firePropertyChange("waypoint," null, newPoint);
40       }
41    }
42
43    @Override
44    public void mousePressed(MouseEvent me) {
45    }
46
47    @Override
48    public void mouseReleased(MouseEvent me) {
49    }
50
51    @Override
52    public void mouseEntered(MouseEvent me) {
53    }
54
55    @Override
```

```
56     public void mouseExited(MouseEvent me) {
57     }
58
59 }
```

APPENDIX T. MOUSE LISTENER JAVA CODE

```
1 package util;
2
3 import java.awt.event.MouseEvent;
4 import java.awt.event.MouseMotionListener;
5 import java.awt.geom.Point2D;
6 import simkit.smd.animate.SandboxFrame;
7
8 /**
9  * @version $Id: MouseLocationListener.java 51 2012–06–16 05:20:29Z crhutchi $
10  * @author ahbuss
11  */
12 public class MouseLocationListener implements MouseMotionListener {
13
14    private SandboxFrame sandboxFrame;
15
16    private Point2D origin;
17
18    public MouseLocationListener(SandboxFrame sandboxFrame) {
19       this.setSandboxFrame(sandboxFrame);
20    }
21
22    @Override
23    public void mouseDragged(MouseEvent me) {
24    }
25
26    @Override
27    public void mouseMoved(MouseEvent me) {
28       sandboxFrame.setStatus(me.getX()+ ," " + me.getY() + "  =>  " +
29                (me.getX() - origin.getX()) + ," " + (origin.getY() - me.getY()));
30    }
31
32    /**
33     * @return the sandboxFrame
34     */
35    public SandboxFrame getSandboxFrame() {
36       return sandboxFrame;
37    }
38
39    /**
40     * @param sandboxFrame the sandboxFrame to set
41     */
42    public void setSandboxFrame(SandboxFrame sandboxFrame) {
43       this.sandboxFrame = sandboxFrame;
44       this.origin = sandboxFrame.getSandbox().getOrigin();
45    }
46
47 }
```

THIS PAGE INTENTIONALLY LEFT BLANK

LIST OF REFERENCES

Alrowaie, A. (2011). *The effect of time–advance mechanism in modeling and simulation* (Doctoral Dissertation, Naval Postgraduate School). Retrieved from http://calhoun.nps.edu/public/handle/10945/10798

Bowden, A., & Basnet, S. (2011). *The economic cost of Somalia piracy, 2011*. Retrieved from Oceans Beyond Piracy website: http://oceansbeyondpiracy.org/sites/default/files/economic_cost_of_piracy_2011.pdf

Brutzman, D. (2011). *MMOWGLI massive multiplayer online wargame leveraging the Internet: New capabilities for crowd sourcing innovation* [PDF document]. Retrieved from http://portal.mmowgli.nps.edu/c/wiki/get_page_attachment?p_l_id=34310&nodeId=17497&title=Communications+and+Outreach&fileName=Communications+and+Outreach%2FMmowgliOverviewNps26June2011.pdf

Buss, A. (2011). *Discrete event simulation modeling* [Word document]. Retrieved from http://diana.nps.edu/mv4302/Handouts/Discrete%20Event%20Simulation%20Modeling.docx

Buss, A. H., & Sanchez, P. J. (Eds.). (2005). *Simple movement and detection in discrete event simulation.* Proceeding of the 2005 Winter Simulation Conference

Buss, A. (n.d) *Viskit: Rapid modeling of event graph components* [PDF document]. Retrieved from http://diana.nps.edu/Viskit/presentations/Viskit.pdf

Camillus, J.C. (2008). *Strategy as a wicked problem*. Harvard Business Review, May 2008. Retrieved from hbr.org/2008/05/strategy-as-a-wicked-problem/ar/1

Clausewitz, C. V. (1984). *On war* (Howard, M. E., & Paret, P, Trans.). Princeton, N.J.: Princeton University Press (Original work published 1780–1831).

Commander Destroyer Squandron Twenty-Six (2012). *Tactical Bulletin SUW-12–01: Anti-piracy and counter-piracy operation.* Released for public/partner use. Retrieved from http://portal.mmowgli.nps.edu/c/wiki/get_page_attachment?p_l_id=33393&nodeId=10773&title=Information+Sources&fileName=Information+Sources%2FSUW-12–01.pdf

Gunal, M. M. (2010). *Notes on naval simulation*. Retrieved from http://www.hospitalsimulation.info/gunal/eBook/NotesOnNavalSimulations.v.0.2.pdf

Harney, J. W. (2003). *Analyzing anti–terrorist tactical effectiveness of picket boats for force protection of Navy ships using X3D graphics and agent–based simulation* (Master's thesis, Naval Postgraduate School). Retrieved from *http://calhoun.nps.edu/public/bitstream/handle/10945/1105/03Mar_Harney.pdf?sequence=1*

Haywood, R., & Spivak, R. (2012). *Maritime piracy* (Global Institutions Series). New York, NY: Routledge.

ICC International Maritime Bureau (2013). *Piracy and armed robbery against ships. Report for the period 1 January – 31 December 2012*. Retrieved from http://www.icc–ccs.org/piracy–reporting–centre/request–piracy–report

Jakob, M., Vanek, O., Hrstka, O., Bosansky, B., & Pechoucek, M. (2011, December 31). *Adversarial modeling and reasoning in the maritime domain. Year 3 Report.* Retrieved from http://agents.felk.cvut.cz/cgi–bin/docarc/docarc.pl/document/411/reportY3.pdf

Jensen, G. & Cook, M. (2010). *Gaming for innovation: An open source approach to generating insight.* ONR Director of Innovation Newsletter, Volume 5, pp 8 – 10. Retrieved from: http://portal.mmowgli.nps.edu/c/wiki/get_page_attachment?p_l_id=34310&nodeId=17497&title=Communications+and+Outreach&fileName=Communications+and+Outreach%2FOnrOfficeInnovationNewsletterJune2010.MmowgliArticle.pdf

Konstam, A. (2008). *Piracy: The complete history*. United Kingdom: Osprey Publishing.

Law, A. (2007). *Simulation modeling & analysis* (4th ed.). New York, NY: McGraw–Hill Companies.

Law, R. L. (2011). *Maritime piracy off the coast of Somalia* (Master's thesis California State University, Monterey Bay). Retrieved from: https://portal.mmowgli.nps.edu/c/wiki/get_page_attachment?p_l_id=33393&nodeId=10773&title=Masters+Theses&fileName=Masters+Theses%2FLawThesisSomaliPiracyCsumbPanettaInstituteJune2011.pdf

Mack, P. V. (2000). *THORN: A study in designing a usable interface for a geo–referenced discrete event simulation* (Master's thesis, Naval Postgraduate School). Retrieved from http://calhoun.nps.edu/public/bitstream/handle/10945/9410/00Sep_Mack.pdf?sequence=1

Mahan, A. T. (1918). *The influence of sea power upon history, 1660 – 1783* (12th ed.). Boston: Little, Brown.

Major, W. F., Kline, J., & Fricker, R. D (September 2012). Accessing counter–piracy tactics: Is it better to fight or flee?. *MORS PHALANX, 45, no. 3,* 22 – 24. Retrieved from http://www.mors.org/UserFiles/file/Phalanx/MORS_Phalanx_SEPT2012_web.pdf

MMOWGLI Players Portal. (n.d.). About MMOWGLI. Retrieved from https://portal.mmowgli.nps.edu/game–wiki/–/wiki/PlayerResources/About+MMOWGLI

Ohab, J. (2011). *MMOWGLI: An experiment in generating collective intelligence. Retrieved from* Armed with Science website: http://science.dodlive.mil/2011/04/28/mmowgli–an–experiment–in–generating–collective–intelligence

Roberts, N. (2000). *Wicked problems and network approaches to resolution.* International Public Management Review, 1,1 pp. 1 – 32.

Schruben, L. (1983). *Simulation modeling with event graphs*. Communications of the ACM. 26: 957–963.

Seguin, J. M. (2007). *Simulating candidate missions for a novel glider unmanned underwater vehicle* (Master's thesis, Naval Postgraduate School). Retrieved from http://calhoun.nps.edu/public/bitstream/handle/10945/3664/07Mar_Seguin.pdf?sequence=1

Shapiro, A., "Taking diplomatic action against piracy." Remarks to the Global Maritime Information Sharing Symposium, National Defense University, Washington, D.C. September 16, 2009. Retrieved from http://www.state.gov/t/pm/rls/rm/129258.htm

Slootmaker, L. A. (2011). *Countering piracy with the next generation piracy performance surface model* (Master's Thesis, Naval Postgraduate School). Retrieved from http://calhoun.nps.edu/public/bitstream/handle/10945/5747/11Mar_Slootmaker.pdf

Sullivan, P. J. (2006). *Evaluating the effectiveness of waterside security alternatives for force protection of Navy ships and installations using X3D graphics and agent–based modeling* (Master's thesis, Naval Postgraduate School). Retrieved from http://calhoun.nps.edu/public/bitstream/handle/10945/2645/06Sep_Sullivan.pdf?sequence=1

U.S. Library of Congress, Congressional Research Service. (2010). *Piracy: A legal definition* by R. Chuck Mason. (CRS Report No. R41455). Washington DC: Office of Congressional Information and Publishing. Retrieved from http://www.fas.org/sgp/crs/misc/R41455.pdf

Wernecke, J. (2009). *The KML handbook: Geographic visualization for the web*. Upper Saddle River, NJ: Pearson Education.

X3D–Edit authoring tool for extensible 3D (X3D) graphics (2013, January 5). Retrieved from Savage Developers Guide website: https://savage.nps.edu/X3D–Edit

INITIAL DISTRIBUTION LIST

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California

3. Admiral Cecil D. Haney, USN
   U.S. Pacific Fleet
   Pearl Harbor, Hawaii

4. Vice Admiral Mark I. Fox
   Office of the Chief of Naval Operations
   Washington, District of Columbia

5. Donna Hopkins
   U.S. Department of State
   Washington, District of Columbia

6. Jon Huggins
   Oceans Beyond Piracy
   Broomfield, Colorado

7. Jens Vestergaard Madsen
   Oceans Beyond Piracy
   Broomfield, Colorado

8. Dr. Peter Chalk
   RAND Corporation
   Santa Monica, California

9. Cyrus Mody
   ICC International Maritime Bureau
   London, United Kingdom

10. Officer In Charge
    Maritime Liaison Office
    Manama, Bahrain

11. Dr. Jim Hansen
    Navy Research Laboratory
    Monterey, California

12. Dr. John Arquilla
Naval Postgraduate School
Monterey, California

13. Wayne Hughes, Capt. USN (Ret)
Naval Postgraduate School
Monterey, California

14. Jeffrey Kline, Capt. USN (Ret.)
Naval Postgraduate School
Monterey, California

15. Dr. Eva Regnier
Naval Postgraduate School
Monterey, California

16. Dr. Donald Brutzman
Naval Postgraduate School
Monterey, California

17. Dr. Arnold Buss
Naval Postgraduate School
Monterey, California

18. Terry Norbraten
Naval Postgraduate School
Monterey, California

19. Lyla Englehorn
Naval Postgraduate School
Monterey, California

20. Rebeca Law
Naval Postgraduate School
Monterey, California

21. Wendy Walsh
Naval Postgraduate School
Monterey, California

22. Peter Pham
Atlantic Council
Washington, D.C.

23. Bronwyn Bruton
Atlantic Council
Washington, District of Columbia

24. Dan Burns
Naval Postgraduate School
Monterey, California

25. Captain James Wyatt
U.S. Pacific Fleet
Pearl Harbor, Hawaii