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
Approved for public release; distribution is unlimited
**Title**

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

**Author**

Chad R. Hutchins

**Performing Organization**

Naval Postgraduate School
Monterey, CA 93943–5000

**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.

**Subject Terms**

Crowd-sourcing, Discrete Event Simulation (DES), MMOWGLI, Somali Piracy, Simkit, Viskit, Java, KML, X3D, X3D-Edit, OpenMap, Wicked Problems
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
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.
# TABLE OF CONTENTS

## I. INTRODUCTION

A. PROBLEM STATEMENT ................................................................. 1
B. OVERVIEW .................................................................................. 2
C. MINDSET AND APPROACH OF CURRENT COUNTER–PIRACY EFFORTS ................................................................. 3
D. MOTIVATION ................................................................................. 4
   1. Personal Experience ............................................................... 4
E. RESEARCH QUESTIONS AND OBJECTIVES ............................ 6
F. SCOPE OF THESIS ...................................................................... 6
G. THESIS ORGANIZATION ............................................................ 7

## II. BACKGROUND AND RELATED WORK

A. INTRODUCTION ............................................................................. 9
B. DISCRETE EVENT SIMULATION (DES) ........................................ 9
   1. Methodology ............................................................................. 9
   2. Simkit ....................................................................................... 12
   3. Viskit ....................................................................................... 15
C. VISUALIZATION ........................................................................... 18
   1. X3D–Edit .................................................................................. 18
   2. Keyhole Markup Language (KML) .......................................... 19
   3. OpenMap™, OpenStreetMap and OpenSeaMap ................. 20
   4. JAVA Swing Graphical User Interface (GUI) ....................... 21
D. PREVIOUS RESEARCH USING DES/SIMKIT MODELING ........ 21
   1. Viskit Modeling of ANTI–TERRORISM/FORCE PROTECTION (AT/FP) .......................................................... 22
   2. Simkit and GIS visualization ................................................... 22
E. MODELING AND SIMULATING MARITIME PIRACY ............ 22
   1. Agent Technology Center’s AgentC Project ......................... 23
   2. Piracy Attack Risk Surface (PARS) Model ......................... 24
   3. Piracy Asymmetric Naval Operations Patterns Modeling for Education and Analysis (PANOPEA) Project by Simulation Team .......................................................... 26
   4. Naval Postgraduate School (NPS) Research on Somali Piracy .... 27
F. SUMMARY ...................................................................................... 29

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

A. INTRODUCTION ............................................................................. 31
B. WHAT IS MMOWGLI? ................................................................. 31
C. TECHNICAL OVERVIEW ............................................................. 32
D. MMOWGLI GAME HISTORY ....................................................... 33
   1. Piracy MMOWGLI 2011–Open to Public ................................ 33
2. Piracy MMOWGLI 2012–Maritime Experts and Stakeholders Only ................................................................. 34
3. Energy MMOWGLI ................................................................................................................................. 35
4. EDGE Virtual Training Program (EVTP) MMOWGLI ............... 36
5. Business Innovation Initiative (BII) MMOWGLI .................. 36
6. Electromagnetic Maneuver (EM2) MMOWGLI....................... 37

E. MMOWGLI PORTAL ......................................................................................................................... 37
1. Piracy Portal ........................................................................................................................................ 38

F. SUMMARY ........................................................................................................................................... 39

IV. DETAILED PROBLEM DESCRIPTION ...................................................................................... 41
A. INTRODUCTION ................................................................................................................................. 41
B. PIRACY PROBLEMS AND CHALLENGES ............................................................................. 41
C. MODELING PIRACY AND COUNTER–PIRACY TACTICS .................................................. 43
1. Data Limitations ................................................................................................................................. 43
2. MMOWGLI Action Plans .................................................................................................................. 44
D. SUMMARY ........................................................................................................................................... 45

V. SIMULATION DESIGN AND MODELING ................................................................................ 47
A. INTRODUCTION ................................................................................................................................. 47
B. SIMULATION DESIGN ....................................................................................................................... 47
C. SIMKIT ENTITIES .............................................................................................................................. 48
1. Pirate Mover Manager ....................................................................................................................... 48
2. Navy Ship Mover Manager .............................................................................................................. 49
3. Merchant Ship Mover Manager ....................................................................................................... 50
4. Adjudicator ....................................................................................................................................... 51
D. SIMKIT PROCESSES ...................................................................................................................... 52
1. Pirate Departure Processes ............................................................................................................... 52
2. Pirate Camps ................................................................................................................................... 52
3. Merchant Ship Departure Processes ............................................................................................... 52
4. Merchant Ship Port of Origin .......................................................................................................... 53
E. SIMKIT SCENARIO ASSEMBLIES ............................................................................................. 53
1. Defense Scenario One: Transit Lane Patrols ..................................................................................... 54
2. Defense Scenario Two: Naval Quarantine ......................................................................................... 54
3. Defense Scenario Three: Pirate Camp Operations ......................................................................... 56
F. JAVA SUPPLEMENTAL CLASSES ........................................................................................... 57
G. DETAILED DESCRIPTION OF VISUALIZATION IMPLEMENTATION .................................... 58
1. X3D-Edit and KML ........................................................................................................................... 58
2. Open-source Geographical Information Systems (GIS) ................................................................. 62
3. Java Swing ...................................................................................................................................... 63
H. SUMMARY ........................................................................................................................................... 64

VI. SIMULATION ANALYSIS ............................................................................................................. 65
A. INTRODUCTION ................................................................................................................................. 65
B. SIMULATION ANALYSIS .................................................................................................................. 65
C. SUMMARY ........................................................................................................................................... 68
VII. CONCLUSION AND RECOMMENDATIONS .................................................................69
   A. RECOMMENDATIONS FOR COUNTER-PIRACY STRATEGY .........................69
   B. RECOMMENDATIONS FOR FUTURE WORK .....................................................70
   C. FINAL THOUGHTS AND CONSIDERATIONS .................................................71

APPENDIX A. PIRATE MOVER MANAGER JAVA CODE ..............................................73
APPENDIX B. NAVY MOVER MANAGER JAVA CODE .............................................89
APPENDIX C. MERCHANT MOVER MANAGER JAVA CODE ..................................97
APPENDIX D. BAYLA PIRATE DEPARTURE PROCESS JAVA CODE .....................107
APPENDIX E. BAYLA PIRATE CAMP JAVA CODE ..................................................111
APPENDIX F. SUEZ TO OMAN MERCHANT DEPARTURE JAVA CODE ...............115
APPENDIX G. SUEZ TO OMAN ORIGIN PORT JAVA CODE .....................................119
APPENDIX H. MMOWGLI ACTION PLAN 16: TRANSIT LANE PATROLS BY
   INTERNATIONAL NAVIES ......................................................................................123
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 ..................................................................................................125
APPENDIX J. MMOWGLI ACTION PLAN 9: PIRATE CAMP OPERATIONS
   ACTION PLAN 9 .....................................................................................................133
APPENDIX K. PIRATE CAMP OPERATIONS SIMKIT ASSEMBLY .........................137
APPENDIX L. PLATFORM CLASS JAVA CODE .......................................................175
APPENDIX M. PLATFORM TYPE CLASS JAVA CODE ...........................................177
APPENDIX N. NAVY STATE JAVA CODE ...............................................................179
APPENDIX O. PIRATE STATE JAVA CODE .............................................................181
APPENDIX P. MERCHANT STATE JAVA CODE .....................................................183
APPENDIX Q. OPENMAP™ SIMULATION LAYER JAVA CODE .........................185
APPENDIX R. JAVA SWING SANDBOX FRAME IMPLEMENTATION CODE
   SNIPPET ...............................................................................................................193
APPENDIX S. JAVA SWING WAYPOINT BUILDER JAVA CODE .........................195
APPENDIX T. MOUSE LISTENER JAVA CODE .......................................................197
LIST OF REFERENCES ...............................................................................................199
INITIAL DISTRIBUTION LIST ..................................................................................203
LIST OF FIGURES

Figure 1. The logic for the Next Event Algorithm for Discrete Event Simulation (DES) (From Buss, 2011) ........................................................................................................10

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) ........................................................................................................11

Figure 3. A Depiction of the SimEvenListener Pattern for a DES system (From Discrete Event Simulation Modeling by Dr. Arnold Buss) ........................................................................................................12

Figure 4. A simple GUI featuring a Property Change Frame displaying Detection and Undetection events. .............................................................................................................................................13

Figure 5. A graphical depiction of a Simkit Cookie cutter sensor model. From (Buss & Sanchez, 2005). Moving sensors are also possible ..........14

Figure 6. Arrival Process event graph using Viskit.........................................................15

Figure 7. Viskit XML output of an ArrivalProcess. Viskit displays the XML in two views, a tree graph and standard XML format. .................................................................................................................16

Figure 8. Viskit Java source code of an ArrivalProcess autogenerated from XML........17

Figure 9. A screen snapshot of X3D-Edit with Xj3D browser displaying Hello World scene (From X3D-Edit Home Page) .............................................................................................................................................19

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 .............................................................................................................................................21

Figure 11. AgentC Google Earth visualization of risk modeling. (From Agent Technology Center’s AgentC website, March 15, 2013) .................................................................................................................24

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.) .................................................................................................................25

Figure 13. Oceans Beyond Piracy’s Independent Assessment (From Oceans Beyond Piracy website, February 15, 2013). .................................................................................................................................................34

Figure 14. The MMOWGLI Portal Home Page is the home for all current and past MMOWGLI games. (From MMOWGLI Portal, February 4, 2013) .................................................................38

Figure 15. The MMOWGLI Piracy Portal Welcome Page is the start point for accessing Piracy MMOWGLI. (From MMOWGLI Portal, February 4, 2013) .................................................................................................................................................39

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). .................................................................................................................................................45

Figure 17. PirateMoverManager Viskit Event Graph shows the modeled behavior of a Somali pirate .................................................................................................................................................49

Figure 18. NavyMoverManager Viskit Event Graph shows the behavior modeled for a navy vessel conducting counter-piracy operations .................................................................................................................................................50

Figure 19. MerchantMoverManager Viskit Event Graph shows the modeled behavior of a merchant vessel transiting from its port of origin to its destination .................................................................51
Figure 20. Visual depiction of a Pirate Departure Process and Pirate Camp SimEventListener Pattern ..........................................................52
Figure 21. Merchant Departure Process and Merchant Origin Port SimEventListener Pattern ........................................................................................................53
Figure 22. Illustration of Transit Lane Patrol (From Piracy MMOWGLI 2012 Action Plan Report, February 10, 2012) ........................................................................................................54
Figure 23. Illustration of a 200NM Naval Quarantine off the Southern coast of Somalia (From Piracy MMOWGLI 2012 Action Plan Report, February 10, 2012) ........................................................................................................55
Figure 24. An illustration of Pirate Camp Operations modeled for this thesis ............57
Figure 25. X3D-Edit with PiratePath.kml and the KML Palette ........................................59
Figure 26. Pirate Path History of single pirate viewed in Google Earth .............................60
Figure 27. Pirate Successful Attack History for one simulation replication viewed in Google Earth ........................................................................................................61
Figure 28. OpenMap™ GUI with Simulation Layer Implemented ..................................62
Figure 29. Histogram comparing the results of the Naval Effectiveness MOE of each defense scenario .................................................................67
Figure 30. Histogram comparing the results of the Pirate Effectiveness MOE for each defense scenario .................................................................67
LIST OF TABLES

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 .................................................................33

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 .............................................................................................36

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 ..........................37

Table 4. Comparison of the Naval Effectiveness MOE simulation results among all three defense scenarios ..........................................................................................................................66

Table 5. Comparison of the Pirate Effectiveness MOE simulation results among all three defense scenarios ..........................................................................................................................66
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>3D</td>
<td>Three Dimensional</td>
</tr>
<tr>
<td>AMISOM</td>
<td>African Union Mission in Somalia</td>
</tr>
<tr>
<td>API</td>
<td>Application Programming Interface</td>
</tr>
<tr>
<td>APS</td>
<td>African Partnership Station</td>
</tr>
<tr>
<td>ATC</td>
<td>Agent Technology Center</td>
</tr>
<tr>
<td>BII</td>
<td>Business Innovation Initiative</td>
</tr>
<tr>
<td>CDS</td>
<td>Commander Destroyer Squadron</td>
</tr>
<tr>
<td>CNMOC</td>
<td>Commander Naval Meteorology and Oceanography Command</td>
</tr>
<tr>
<td>CTF</td>
<td>Commander Task Force</td>
</tr>
<tr>
<td>DES</td>
<td>Discrete Event Simulation</td>
</tr>
<tr>
<td>DHS</td>
<td>Department of Homeland Security</td>
</tr>
<tr>
<td>DOE</td>
<td>Design of Experiments</td>
</tr>
<tr>
<td>DTS</td>
<td>Discrete Time Simulation</td>
</tr>
<tr>
<td>EEZ</td>
<td>Economic Exclusion Zone</td>
</tr>
<tr>
<td>EU</td>
<td>European Union</td>
</tr>
<tr>
<td>EVT</td>
<td>EDGE Virtual Training</td>
</tr>
<tr>
<td>FEL</td>
<td>Future Event List</td>
</tr>
<tr>
<td>GPS</td>
<td>Global Positioning System</td>
</tr>
<tr>
<td>GUI</td>
<td>Graphical User Interface</td>
</tr>
<tr>
<td>GWT</td>
<td>Google Web Toolkit</td>
</tr>
<tr>
<td>HOA</td>
<td>Horn of Africa</td>
</tr>
<tr>
<td>HTTP</td>
<td>Hypertext Transfer Protocol</td>
</tr>
<tr>
<td>HSDL</td>
<td>Homeland Security Digital Library</td>
</tr>
<tr>
<td>IA–CGF</td>
<td>Intelligent Agent Simulation Computer Generated Force</td>
</tr>
<tr>
<td>ICC</td>
<td>International Chamber of Commerce</td>
</tr>
<tr>
<td>IFTF</td>
<td>Institute for the Future</td>
</tr>
<tr>
<td>IMB</td>
<td>International Maritime Bureau</td>
</tr>
<tr>
<td>IRTC</td>
<td>Internationally Recommended Traffic Corridor</td>
</tr>
<tr>
<td>Abbreviation</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>JCA</td>
<td>Joint Campaign Analysis</td>
</tr>
<tr>
<td>MMOWGLI</td>
<td>Massive Multiplayer Online War–game Leveraging the Internet</td>
</tr>
<tr>
<td>MOE</td>
<td>Measure of Effectiveness</td>
</tr>
<tr>
<td>NAVO</td>
<td>Naval Oceanographic Command</td>
</tr>
<tr>
<td>NCIS</td>
<td>Naval Criminal Investigative Service</td>
</tr>
<tr>
<td>NEC C2 M2</td>
<td>Network Centric Command and Control Maturity Models</td>
</tr>
<tr>
<td>NMCI</td>
<td>Navy/Marine Corps Intranet</td>
</tr>
<tr>
<td>NPS</td>
<td>Naval Postgraduate School</td>
</tr>
<tr>
<td>OBP</td>
<td>Oceans Beyond Piracy</td>
</tr>
<tr>
<td>ONR</td>
<td>Office of Naval Research</td>
</tr>
<tr>
<td>OSA</td>
<td>Open System Architecture</td>
</tr>
<tr>
<td>PANOPEA</td>
<td>Piracy Asymmetric Naval Operations Patterns modeling for Education and Analysis</td>
</tr>
<tr>
<td>PARS</td>
<td>Pirate Attack Risk Surface</td>
</tr>
<tr>
<td>PPS</td>
<td>Piracy Performance Surface</td>
</tr>
<tr>
<td>PPSN</td>
<td>Piracy Performance Surface Model</td>
</tr>
<tr>
<td>RF</td>
<td>Royalty Free</td>
</tr>
<tr>
<td>SWDG</td>
<td>Surface Warfare Development Group</td>
</tr>
<tr>
<td>TCP</td>
<td>Transmission Control Protocol</td>
</tr>
<tr>
<td>VV&amp;A</td>
<td>Verification, Validation, and Accreditation</td>
</tr>
<tr>
<td>X3D</td>
<td>Extensible 3D Graphics Language</td>
</tr>
<tr>
<td>XML</td>
<td>Extensible Markup Language</td>
</tr>
</tbody>
</table>
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.”

—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 MMOWLG1. 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.
II. BACKGROUND AND RELATED WORK

“Conformity is the jailer of freedom and the enemy of growth”

—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).

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.

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](image)

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 upded 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).
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 sensor. 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.](image)
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](unnamed.png)

**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.).
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 programming 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://OpenMaptm.bbn.com), [http://www.openstreetmap.org](http://www.openstreetmap.org), and [http://www.openseamap.org](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.](image)

**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.


**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).
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:  

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 operate. 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). PANOEPA models pirate activity around the Horn of Africa in an effort to evaluate various Network Centric Command and Control Maturity Models (NEC C2 M2). PANOEPA 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 PANOEPA 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.
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:

• “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.


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.
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

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 Protocol (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,

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

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dates</td>
<td>31-May-3 June, 21-23 June, 5-8 July</td>
<td>7-9 November</td>
<td>10-13 November</td>
<td>-</td>
</tr>
<tr>
<td>Days duration</td>
<td>11</td>
<td>3</td>
<td>3.5</td>
<td>18</td>
</tr>
<tr>
<td>Signups</td>
<td>16,000</td>
<td>31,000</td>
<td>31,000</td>
<td>31,000</td>
</tr>
<tr>
<td>Invitees</td>
<td>2,200</td>
<td>7,500</td>
<td>7,500</td>
<td>15,000</td>
</tr>
<tr>
<td>Players</td>
<td>832</td>
<td>920</td>
<td>413</td>
<td>~2,100</td>
</tr>
<tr>
<td>Signup %</td>
<td>30.7%</td>
<td>12.3%</td>
<td>5.5%</td>
<td>14%</td>
</tr>
<tr>
<td># Idea Cards</td>
<td>5142</td>
<td>5608</td>
<td>4228</td>
<td>14,978</td>
</tr>
<tr>
<td># Action Plans</td>
<td>28</td>
<td>18</td>
<td>22</td>
<td>68</td>
</tr>
<tr>
<td># Game Master Accounts</td>
<td>29</td>
<td>50</td>
<td>46</td>
<td>~60</td>
</tr>
</tbody>
</table>

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](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 professional. 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](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](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](https://portal.mmowgli.nps.edu/energy) and [https://mmowgli.nps.edu/energy/reports](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.

<table>
<thead>
<tr>
<th>energyMMOWGLI</th>
<th>piracyMMOWGLI 2012</th>
<th>evtp: Edge Virtual Training Program</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dates</td>
<td>21-27 May 2012</td>
<td>18 June - present, ongoing</td>
</tr>
<tr>
<td>Days duration</td>
<td>5</td>
<td>Long-term</td>
</tr>
<tr>
<td>Signups</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Invitees</td>
<td>797</td>
<td>200+</td>
</tr>
<tr>
<td>Players</td>
<td>561</td>
<td>115</td>
</tr>
<tr>
<td>Signup %</td>
<td>70.4%</td>
<td>Slow increase</td>
</tr>
<tr>
<td># Idea Cards</td>
<td>5121</td>
<td>432</td>
</tr>
<tr>
<td># Action Plans</td>
<td>37</td>
<td>8</td>
</tr>
<tr>
<td># Game Master Accounts</td>
<td>47</td>
<td>10</td>
</tr>
</tbody>
</table>

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](https://portal.mmowgli.nps.edu/bii) and [https://mmowgli.nps.edu/bii/reports](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](https://portal.mmowgli.nps.edu/em2) and [https://mmowgli.nps.edu/em2/reports](https://mmowgli.nps.edu/em2/reports).

### 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.

![MMOWGLI Players Portal](image)

**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](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!
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.”

– 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.
V. SIMULATION DESIGN AND MODELING

“All models are wrong, some are useful.”

–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

---

tasks, such as “Attack” or “Evade.” Each entity has a sensor 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.
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.

![NavyMoverManager Viskit Event Graph](image)

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.](image)

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](image)

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](image)

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](image)

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 University, Monterey Bay’s Panetta Institute of Health and Human Services and Public Policy. The thesis is an applied policy report that gives three alternatives for countering piracy:

2. Provide methods of alternative livelihood for Somalis, including a moratorium on fishing in the Somalia EEZ (Law, 2011, pp 24-26),

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 IMPLEMENTATION

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:

```java
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.

![X3D-Edit with PiratePath.kml and the KML Palette](image)

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("<description>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.
2. **Open-source Geographical Information Systems (GIS)**

Since OpenMap™ and Open Street Map are both open-source they are appealing platforms to learn and connect Simkit to. Another benefit of OpenMap™ 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 OpenMap™ 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.

![OpenMap™ GUI with Simulation Layer Implemented](image)

**Figure 28.** OpenMap™ 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 OpenMap™ 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{\text{number pirates detected}}{\text{number pirates departed camp}}
- Pirate Effectiveness = \frac{\text{number successful pirate attacks}}{\text{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.

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Pirate Camp Operations</th>
<th>Naval Quarantine</th>
<th>Transit Lane Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.90</td>
<td>0.54</td>
<td>0.40</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.06</td>
<td>0.07</td>
<td>0.06</td>
</tr>
<tr>
<td>Standard Error.</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Pirate Camp Operations</th>
<th>Naval Quarantine</th>
<th>Transit Lane Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.05</td>
<td>0.14</td>
<td>0.16</td>
</tr>
<tr>
<td>Standard Deviation</td>
<td>0.04</td>
<td>0.06</td>
<td>0.06</td>
</tr>
<tr>
<td>Standard Error.</td>
<td>0.01</td>
<td>0.01</td>
<td>0.01</td>
</tr>
</tbody>
</table>

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.
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.
APPENDIX A. PIRATE MOVER MANAGER JAVA CODE

```java
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 supplemenal.PirateState;
15 import supplemental.Platform;
16 import supplemental.PlatformType;
17
18 /**
19 * Models the behavior of a Somali Pirate.
20 *
21 * @version Sld: PirateMoverManager.java 199 2013–03–03 06:10:24Z crutchi $*
22 * @author Chad R Hutchins
23 */
24 public class PirateMoverManager extends SimEntityBase {
25
26 /**
27 * Parameters. Contains Setters and Getters
28 *
29 */
30 private Platform myMover;
31 private CookieCutterSensor sensor;
32 private Point2D baseLocation;
33 private RandomVariate[] pathGenerator;
34 private RandomVariate[] patrolBoxGenerator;
```
private double timeOnPatrol;
private PlatformType platformType;
private DiscreteRandomVariate attackDecision;
private DiscreteRandomVariate successOrFailGenerator;
private RandomVariate[] unsuccessfulAttackTime;
private Point2D patrolBoxStartX;
private Point2D patrolBoxStartY;
private Point2D nextPathWaypoint;
/**
 * State Variables. Contains only getters, no setters.
 */
protected PirateState myMovementState;
protected double numberAttemptedAttacks; //number of attempted attacks
protected double numberSuccessfulAttacks; //number of successful attacks
protected double numberUnsuccessfulAttacks; //number of unsuccessful attacks
protected double numberMerchantsDetected; //number of merchants detected
protected double numberDetectedBeforeAction; //
protected boolean isAlive; //
/**
 * String constant for firePropertyChange modification of the state
 * variable, not visible outside this class
 */
private final String MY_MOVEMENT_STATE = "myMovementState";
private final String NUMBER_ATTEMPTED_ATTACKS = "numberAttemptedAttacks";
private final String NUMBER_MERCHANTS_DETECTED = "numberMerchantsDetected";
private final String NUMBER_UNSUCCESSFUL_ATTACKS =
    "numberUnsuccessfulAttacks";
private final String NUMBER_SUCCESSFUL_ATTACKS =
    "numberSuccessfulAttacks";
private final String IS_ALIVE = "isAlive";
/**
 * String constant for waitDelay method scheduling, visible to other classes
 */
protected final String MOVE_TO = "MoveTo";
protected final String SEARCH_FOR_MERCHANTS = "SearchForMerchants";
protected final String DECIDE_COA = "DecideCOA";
protected final String ATTACK = "Attack";
protected final String RETURN_TO_PIRATE_CAMP = "ReturnToPirateCamp";
protected final String STOP = “Stop”;
protected final String BOARDED_BY_NAVY = “BoardedByNavy”;
protected final String DIE = “Die”;
/**
 * String constant for all other cases.
 */
protected final String MERCHANT = “Merchant”;

// Local patrolbox distance coordinates
double scale = 0.5;
double localDistance = 10 * scale; // 10NM
double transitSpeed = 10 * scale;
double searchSpeed;
double successfulAttackTimeDelay;
double timeOfNavyBoarding = 2.0;
 /**
 * Default Constructor
 */
public PirateMoverManager()
{
    // Does not set anything
}

/**
 * Main constructor: Sets mover, sensor, base location, path, patrol box, attack decision random variate, and success or Fail random variate.
 */
public PirateMoverManager( Platform myMover,
    CookieCutterSensor sensor,
    Point2D baseLocation,
    ...
RandomVariate[] pathGenerator,
DiscreteRandomVariate attackDecision,
RandomVariate[] unsuccessfulAttackTime)
{
    this.setMyMover( myMover );
    this.setSensor( sensor );
    this.setBaseLocation( baseLocation );
    this.setPathGenerator( pathGenerator );
    this.setAttackDecision( attackDecision );
    this.setUnsuccessfulAttackTime( unsuccessfulAttackTime );
}

/**
 * Reset: Resets state variables at end of each replication
 */
@Override
public void reset()
{
    super.reset();
    myMovementState = PirateState.WAITING_AT_BASE;
    myMover.setInitialLocation( baseLocation );
    numberAttemptedAttacks = 0;
    numberSuccessfulAttacks = 0;
    numberUnsuccessfulAttacks = 0;
    numberMerchantsDetected = 0;
    isAlive = true;
}

/**
 * Run: FirePropertyChange for all state variables in reset method
 */
public void doRun()
{
    firePropertyChange( MY_MOVEMENT_STATE, getMyMovementState() );
    // firePropertyChange( NUMBER_ATTEMPTED_ATTACKS,
    //                      getNumberAttemptedAttacks() );
    firePropertyChange( NUMBER_SUCCESSFUL_ATTACKS,
                        getNumberSuccessfulAttacks() );
    firePropertyChange( NUMBER_UNSUCCESSFUL_ATTACKS,
getNumberUnsuccessfulAttacks() );
firePropertyChange( NUMBER_MERCHANTS_DETECTED,
    getNumberMerchantsDetected() );
}
/**
 * LeavePirateIoPirateCamp Event: Changes myMovementState to
 * ENROUTE_TO_PATROL, generates the next way points, and schedules MoveTo
 * event for pirates departing from pirate camps in the Indian Ocean side of
 * Somalia.
 */
public void doLeaveIoPirateCamp()
{
    PirateState oldMyMovementState = getMyMovementState();
    myMovementState = PirateState.ENROUTE_TO_PATROL;
    firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
        getMyMovementState() );
    RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
    transitSpeedGenerator[0] = RandomVariateFactory.getInstance("Uniform," 8 * scale, 12 * scale);
    transitSpeed = transitSpeedGenerator[0].generate();
    myMover.setMaxSpeed( transitSpeed );
    nextPathWaypoint = new Point2D.Double(
        getPathGenerator()[0].generate(),
        getPathGenerator()[1].generate() );
    waitDelay( MOVE_TO, 0.0, nextPathWaypoint );
}
/**
 * LeavePirateGoaPirateCamp Event: Changes myMovementState to
 * ENROUTE_TO_PATROL, generates the next way points, and schedules MoveTo
 * event for pirates departing from pirate camps in the Gulf of Aden side of
 * Somalia.
 */
public void doLeaveGoaPirateCamp()
PirateState oldMyMovementState = getMyMovementState();
myMovementState = PirateState.ENROUTE_TO_PATROL;
firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState, getMyMovementState() );

RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
transitSpeedGenerator[0] = RandomVariateFactory.getInstance("Uniform," 8 * scale, 12 * scale);

transitSpeed = transitSpeedGenerator[0].generate();
myMover.setMaxSpeed( transitSpeed );

nextPathWaypoint = new Point2D.Double( getPathGenerator()[0].generate(), getPathGenerator()[1].generate() );
waitDelay( MOVE_TO, 0.0, nextPathWaypoint );

/**
 * EndMove Event: Generates nextWayPoint and if myMovementState is
 * PATROLLING it schedules MoveTo. If myMovementState is ENROUTE_TO_PATROL
 * it schedules SEARCH_FOR_MERCHANTS.
 *
 * @param mover
 */

class doEndMove( Platform mover )
{
    double xVal = nextPathWaypoint.getX();
    double yVal = nextPathWaypoint.getY();

    RandomVariate[] localPatrolBoxGenerator = new RandomVariate[2];
    localPatrolBoxGenerator[0] = RandomVariateFactory.getInstance("Uniform,"
                                 ( xVal - localDistance ),
                                 ( xVal + localDistance ),
                                 ( yVal - localDistance ),
                                 ( yVal + localDistance ));

}
Point2D nextWaypoint = new Point2D.Double(
    localPatrolBoxGenerator[0].generate(),
    localPatrolBoxGenerator[1].generate() );

if ( myMovementState == PirateState.ENROUTE_TO_PATROL )
{  
    waitDelay( SEARCH_FOR_MERCHANTS, 0.0, nextWaypoint );
}

if ( myMovementState == PirateState.PATROLLING )
{  
    waitDelay( MOVE_TO, 0.0, nextWaypoint );
}

/**
 * SearchForMerchants Event: Changes myMovementState to PATROLLING. 
 * Generates patrolBox to hunt for merchant ships, and schedules MOVE_TO 
 * with nextWaypoint in patrol box.
 */

public void doSearchForMerchants( Point2D nextWaypoint )
{
    PirateState oldMyMovementState = getMyMovementState();
    myMovementState = PirateState.PATROLLING;
    firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState, 
        getMyMovementState() );

    RandomVariate[] searchSpeedGenerator = new RandomVariate[2];
    searchSpeedGenerator[0] = RandomVariateFactory.
        getInstance("Uniform", 2 * scale, 8 * scale);
    searchSpeed = searchSpeedGenerator[0].generate();
    myMover.setMaxSpeed( searchSpeed );

    double xVal = nextWaypoint.getX();
    double yVal = nextWaypoint.getY();
RandomVariate[] localPatrolBoxGenerator = new RandomVariate[2];
localPatrolBoxGenerator[0] = RandomVariateFactory.getInstance("Uniform",
                        xVal - localDistance,
xVal + localDistance);
localPatrolBoxGenerator[1] = RandomVariateFactory.getInstance("Uniform",
yVal - localDistance,
yVal + localDistance);

Point2D nextPatrolWaypoint = new Point2D.Double(
                        localPatrolBoxGenerator[0].generate(),
                        localPatrolBoxGenerator[1].generate());

waitDelay( MOVE_TO, 0.0, nextPatrolWaypoint);

//IO pirates: Fuel is a RV from 2 weeks - 2 months
if ( myMover.getInitialLocation().getY() <= 300.0 ) {
    RandomVariate[] lowFuelIOGenerator = new RandomVariate[1];
    lowFuelIOGenerator[0] = RandomVariateFactory.getInstance("Uniform",
                              336.0, 1460.0);
    double lowFuelIO = ((lowFuelIOGenerator[0].generate()) -
                        (getEventList().getSimTime()));
    if (lowFuelIO < 0)
        lowFuelIO = 12.0;
    waitDelay ( RETURN_TO_PIRATE_CAMP, lowFuelIO, Priority.HIGH );
}

//GOA pirates: Fuel is a RV from 3 days - 3 weeks
if ( myMover.getInitialLocation().getY() > 300.0 ) {

RandomVariate[] lowFuelGOAGenerator = new RandomVariate[1];
lowFuelGOAGenerator[0] = RandomVariateFactory.getInstance("Uniform", 72.0, 504.0);

double lowFuelGOA = ((lowFuelGOAGenerator[0].generate()) - (getEventList().getSimTime()));

if (lowFuelGOA < 0)
    lowFuelGOA = 12.0;

// If fuel is low go back to camp
waitDelay(RETURN_TO_PIRATE_CAMP, lowFuelGOA, Priority.HIGH);

/**
 * ReturnToPirateCamp Event: Changes myMovementState to RETURNING_TO_BASE.
 * Schedules MOVE_TO with baseLocation coordinate.
 */
public void doReturnToPirateCamp()
{
    PirateState oldMyMovementState = getMyMovementState();
    myMovementState = PirateState.RETURNING_TO_BASE;
    firePropertyChange(MY_MOVEMENT_STATE, oldMyMovementState, getMyMovementState());

    RandomVariate[] transitSpeedGenerator = new RandomVariate[2];
    transitSpeedGenerator[0] = RandomVariateFactory.getInstance("Uniform", 8 * scale, 12 * scale);

    transitSpeed = transitSpeedGenerator[0].generate();
    myMover.setMaxSpeed(transitSpeed);
    waitDelay(MOVE_TO, 0.0, Priority.HIGH, myMover.getInitialLocation());
}
/**
 * Detection Event: Detects any mover within the sensor range. If contact is
 * a Merchant and the merchant hasn’t been detected before increments
 * numberMerchantsDetected. Schedules DecideCOA. Adds merchant to list of
 * detectedMerchants.
 */

public void doDetection( Platform contact )
{
    LinkedList<Platform> detectedMerchants = new LinkedList();

    if ( ( contact.getType() == PlatformType.MERCHANT )
         &&
         !detectedMerchants.contains( contact ) )
    {
        System.out.println( "Detected a Merchant" );
        detectedMerchants.add( contact );
        numberMerchantsDetected = getNumberMerchantsDetected() + 1;
        waitDelay( DECIDE_COA, 0.0, Priority.HIGH, contact );
    }

    /**
     * DecideCOA Event: generates attack decision based on Bernoulli random
     * variable. If choice does not equal 1 the decision is to attack, and
     * cancels (interrupts) prior MOVE_TO events and schedules ATTACK event. If
     * choice equals 1 then the decision is not to attack. This logic is based
     * on size of merchant, weather, and various statistics.
     */
* @param target

public void doDecideCOA( Platform contact )
{
    int choice = attackDecision.generateInt();
    // System.out.println( “Attack Decision: “ + choice );
    if ( choice == 0 )
    {
        // System.out.println( “Decided not to attack” );
        double xValue = myMover.getCurrentLocation().getX();
        double yValue = myMover.getCurrentLocation().getY();
        RandomVariate[] localPatrolBoxGenerator = new RandomVariate[ 2 ];
        localPatrolBoxGenerator[0] = RandomVariateFactory.getInstance( “Uniform,”
            xValue - localDistance,
            xValue + localDistance );
        localPatrolBoxGenerator[1] = RandomVariateFactory.getInstance( “Uniform,”
            yValue - localDistance,
            yValue + localDistance );
        Point2D nextWaypoint = new Point2D.Double(
            localPatrolBoxGenerator[0].generate(),
            localPatrolBoxGenerator[1].generate() );
        waitDelay( SEARCH_FOR_MERCHANTS, 0.0, nextWaypoint );
    }
    if ( choice == 1 )
    {
        // System.out.println( “Decided to Attack!!” );
        waitDelay( ATTACK, 0.0, Priority.HIGH, myMover, contact );
    }
}
public void doAttack( Platform myMover, Platform contact )
{
    PirateState oldMovementState = getMyMovementState();
    myMovementState = PirateState.ATTACKING;

    double oldNumberAttemptedAttacks = getNumberAttemptedAttacks();
    numberAttemptedAttacks = getNumberAttemptedAttacks() + 1;

    //        System.out.println("I am attacking yer ship!!!");
    firePropertyChange( MY_MOVEMENT_STATE, oldMovementState, getMyMovementState() );
    firePropertyChange( NUMBER_ATTEMPTED_ATTACKS, oldNumberAttemptedAttacks, getNumberAttemptedAttacks() );
}

public void doUnsuccessfulAttack()
{
    double oldNumberUnSuccessfulAttacks = getNumberUnsuccessfulAttacks();
    numberUnsuccessfulAttacks = getNumberUnsuccessfulAttacks() + 1;

    //        System.out.println("My attack has been foiled!!");
    double timeOfAttack = unsuccessfulAttackTime[0].generate();

System.out.println("Duration of Pirate Attack: "+timeOfAttack);

waitDelay(SEARCH_FOR_MERCHANTS, timeOfAttack, Priority.HIGH);

firePropertyChange(NUMBER_UNSUCCESSFUL_ATTACKS, oldNumberUnSuccessfulAttacks, 
                    getNumberOfUnsuccessfulAttacks());

/**
 * A successful attack equals a successful hijacking. Increments
 * numberSuccessfulAttacks. Schedules returnToPirateCamp.
 */

double oldNumberSuccessfulAttacks = getNumberOfSuccessfulAttacks();
numberSuccessfulAttacks = getNumberOfSuccessfulAttacks() + 1;

PirateState oldMovementState = getMyMovementState();

myMovementState = PirateState.RETURNING_WITH_MERCHANT;

firePropertyChange(MY_MOVEMENT_STATE, oldMovementState, 
                    getMyMovementState());

System.out.println("I got me a ship... aaarrgghhh!!");

RandomVariate[] successfulAttackTimeGenerator = new RandomVariate[2];
successfulAttackTimeGenerator[0] = RandomVariateFactory.getInstance("Uniform", 1.0, 3.0);

successfulAttackTimeDelay = successfulAttackTimeGenerator[0].generate();

firePropertyChange(NUMBER_SUCCESSFUL_ATTACKS, oldNumberSuccessfulAttacks, 
                    getNumberOfSuccessfulAttacks());

waitDelay(STOP, 0.0, Priority.HIGH);
waitDelay( RETUN_TO_PIRATE_CAMP, successfulAttackTimeDelay, Priority.HIGH);

/**
 * DetectedByNavy Event: Is triggered when a Navy vessel detects it... this
 * is setup in main class via adapter. Schedules STOP event and
 * BOARDED_BY_NAVY event.
 *
 * @param contact
 */
public void doDetectedByNavy( Platform contact, double boardingTime )
{
    // System.out.println( “Contact:” + contact );
    // System.out.println( “Pirate Speed after detection: “ + myMover.getCurrentSpeed() );
    waitDelay( STOP, 0.0, Priority.HIGH );
    contact.waitDelay( BOARDED_BY_NAVY, 0.0, Priority.HIGH, boardingTime );
}

/**
 * BoardedByNavy Event: Changes myMovementState to NAVY_BOARDED. If pirate
 * is attacking when detected schedule DIE event. In all other conditions
 * schedule pirate to RETURN_TO_CAMP.
 *
 * @param boarding
 */
public void doBoardedByNavy( double boardingTime )
{
    PirateState oldMyMovementState = getMyMovementState();
    myMovementState = PirateState.NAVY_BOARDED;
    firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState, getMyMovementState() );
    // System.out.println( “I’m being boarded” );
}
if ( oldMyMovementState == PirateState.ENROUTE_TO_PATROL
|| oldMyMovementState == PirateState.RETURNING_TO_BASE )
{
    waitDelay( RETURN_TO_PIRATE_CAMP, boardingTime,
                Priority.HIGH);
    System.out.println( "DETECTED AND RELEASED TO CAMP" );
}
if ( oldMyMovementState == PirateState.ATTACKING ||
     oldMyMovementState == PirateState.PATROLLING )
{
    sensor.interruptAll();
    myMover.interruptAll();
    myMover.removeMover(myMover);
    waitDelay( DIE, 0.0, Priority.HIGHEST, sensor );
    waitDelay( DIE, 0.0, Priority.HIGHEST, myMover);
    myMover.removeMover(myMover);
    System.out.println( "DETECTED AND APPREHENDED" );
}
/**
 * Returns a String containing the type of Player.
 */
@Override
public String toString()
{
    return "I am a (" + myMover.getType() + ")";
}
THIS PAGE INTENTIONALLY LEFT BLANK
APPENDIX B. NAVY MOVER MANAGER JAVA CODE

```java
/*
 * NavyShipMoverManager.java
 */
package entities;

import java.awt.geom.Point2D;
import java.util.LinkedList;
import simkit.Priority;
import simkit.SimEntityBase;
import simkit.random.RandomVariate;
import simkit.random.RandomVariateFactory;
import simkit.smd.CookieCutterSensor;
import supplemental.NavyState;
import supplemental.Platform;
import supplemental.PlatformType;

/**
 * Models the behavior of a Navy ship on patrol in the Indian Ocean and Gulf of Aden
 */
public class NavyShipMoverManager extends SimEntityBase {

    /**
     * Parameters: Contains getters and setters
     */
    private Platform myMover;
    private Point2D startLocation;
    private RandomVariate[] patrolBoxGenerator;
    private double maxSpeed;
    private PlatformType platformType;
    private CookieCutterSensor sensor;
}
```
public static final double EPSILON = 1.0E-5;

//Scales all distances and speeds for Java Swing. This works for this
//particular set of simulations. You need to ensure proper scale of any
//area other than the exact same location as this sim.
double scale = 0.5;
double patrolSpeed = 8 * scale;

/**
 * State Variables: Contains only getters, no setters.
 */
protected NavyState myMovementState;
protected double timeOnPatrol;
protected double numberPiratesDetected;
protected double numberDistressCallRcv;
protected Platform target;
protected Point2D interceptPoint;
protected double timeOfBoarding;

/**
 * String constant for firePropertyChange modification of the state
 * variable, not visible outside this class
 */
private final String MY_MOVEMENT_STATE = "myMovementState";
private final String TARGET = "target";
private final String INTERCEPT_POINT = "interceptPoint";
private final String NUMBER_PIRATES_DETECTED = "numberPiratesDetected";
private final String NUMBER_DISTRESS_CALL_RCV = "numberDistressCallRcv";

/**
 * String constant for waitDelay method scheduling, visible to other classes
 */
protected final String MOVE_TO = "MoveTo";
protected final String START_PATROLLING = "StartPatrolling";
protected final String STOP = "Stop";
protected final String SIGNAL_PIRATE = "SignalPirate";
protected final String START_INTERCEPT = "Start Intercept";

/**
 * String constant for all other cases.
 */
protected final String PIRATE = "Pirate";
Main constructor: Sets mover, sensor, starting location, and patrol box, id, and max speed of ship

@param myMover
@param sensor
@param startLocation
@param patrolBoxGenerator
@param maxSpeed

public NavyShipMoverManager( Platform myMover,
CookieCutterSensor sensor,
Point2D startLocation,
RandomVariate[] patrolBoxGenerator,
double maxSpeed )
{
    this.setMyMover( myMover );
    this.setSensor( sensor );
    this.setStartLocation( startLocation );
    this.setPatrolBoxGenerator( patrolBoxGenerator );
    this.setMaxSpeed( maxSpeed );
}

Default Constructor

public NavyShipMoverManager()
{
    //Does not set anything
}

Reset: Resets state variables at end of each replication

@Override
public void reset()
{
    super.reset();
myMovementState = NavyState.DEAD_IN_WATER;
numberPiratesDetected = 0;
numberDistressCallRcv = 0;
myMover.setInitialLocation(startLocation);
this.target = null;
this.interceptPoint = Platform.NaP;
}
/**
* Run: FirePropertyChange for all state variables in reset method.
* Schedules StarPatrolling.
*
*/
public void doRun()
{
    firePropertyChange(MY_MOVEMENT_STATE, getMyMovementState());
    firePropertyChange(TARGET, getTarget());
    firePropertyChange(INTERCEPT_POINT, getInterceptPoint());
    waitDelay(START_PATROLLING, 0.0, Priority.HIGH);
}
/**
* StartPatrolling: Changes state to PATROLLING, generates next way point in
* patrol box, and schedules MoveTo.
*
*/
public void doStartPatrolling()
{
    NavyState oldMyMovementState = getMyMovementState();
    myMovementState = NavyState.PATROLLING;
    firePropertyChange(MY_MOVEMENT_STATE, oldMyMovementState,
                        getMyMovementState());
    myMover.setMaxSpeed(patrolSpeed);
    Point2D nextWaypoint = new Point2D.Double(
        patrolBoxGenerator[0].generate(),
        patrolBoxGenerator[1].generate());
    waitDelay(MOVE_TO, 0.0, nextWaypoint);
/**
 * EndMove: Generates nextWayPoint and if myMovementState is PATROLLING it
 * schedules MoveTo. If myMovementState is INTERCEPTING it schedules MoveTo
 * with intercept point
 * @param mover
 */
public void doEndMove( Platform mover )
{
    Point2D nextWaypoint = new Point2D.Double(
        patrolBoxGenerator[0].generate(),
        patrolBoxGenerator[1].generate() );

        if ( myMovementState == NavyState.PATROLLING )
        {
            waitDelay( MOVE_TO, 0.0, nextWaypoint );
        }
}

/**
 * Detection Event: Detects any mover within the sensor range. If it is a
 * pirate while PATROLLING it increments numberPiratesDetected, adds pirate
 * to list of detected pirates, stops the ship, and signals the pirate
 * (which stops the pirate vessel) by an adapter in main class. Schedules
 * StartPatrolling after a determined amount of time via a random variate.
 * Schedules: Stop, SignalPirate, and Start Patrolling.
 * @param contact
 */
public void doDetection( Platform contact )
{
    double oldNumberPiratesDetected = getNumberPiratesDetected();
    LinkedList detectedPirates = new LinkedList();

    if ( contact.getType() == PlatformType.PIRATE &&
        myMovementState == NavyState.PATROLLING &&
        !detectedPirates.contains( contact ) )
        { detectedPirates.add( contact );

    oldNumberPiratesDetected = getNumberPiratesDetected();
    LinkedList detectedPirates = new LinkedList();

    if ( contact.getType() == PlatformType.PIRATE &&
        myMovementState == NavyState.PATROLLING &&
        !detectedPirates.contains( contact ) )
        { detectedPirates.add( contact );

73    }
202	numberPiratesDetected = getNumberPiratesDetected() + 1;
203
tFirePropertyChange( NUMBER_PIRATES_DETECTED,
204	oldNumberPiratesDetected,
205	numberPiratesDetected() );
206
207	RandomVariate[] timeOfBoardingGenerator = new RandomVariate[ 2 ];
208
timeOfBoardingGenerator[0] = RandomVariateFactory.
209	getInstance ( "Uniform," 1.0, 3.0 );
210
timeOfBoarding = timeOfBoardingGenerator[0].generate ();
211
ccontact.waitDelay( "OrderStop," 0.0, Priority.HIGHEST, contact );
212	navyState oldMyMovementState = getMyMovementState ();
213
tmyMovementState = NavyState.BOARDING;
214
tFirePropertyChange ( MY_MOVEMENT_STATE, oldMyMovementState,
215
tgetMyMovementState () );
216
twaitDelay( STOP, 0.0, Priority.HIGHER, myMover );
217	waitDelay( SIGNAL_PIRATE, 0.0, Priority.HIGHER, contact,
218	
ttimeOfBoarding );
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242

public void doSignalPirate( Platform contact, double boardingTime )
{
// System.out.println( "I see you!!!" );
//Does nothing but signals to pirate
RcvDistressCall: Receives call from Merchant using adapter in main class.

If Navy within 40NM increments numberDistressCallRcv. Assumes helo on board and can respond to distress in less than 30 min.

@param caller

public void doRcvDistressCall( Platform caller )

{ // System.out.println( “Caller: “ + caller );
  // System.out.println( “Here I come to save the day!!” );

  double upperBoundCallerX = caller.getCurrentLocation().getX() + 20;
  double lowerBoundCallerX = caller.getCurrentLocation().getX() - 20;
  double upperBoundCallerY = caller.getCurrentLocation().getY() + 20;
  double lowerBoundCallerY = caller.getCurrentLocation().getY() - 20;

  if (( myMover.getCurrentLocation().getX() <= upperBoundCallerX   &&
       myMover.getCurrentLocation().getX() >= lowerBoundCallerX  ) &&
      ( myMover.getCurrentLocation().getY() <=  upperBoundCallerY &&
       myMover.getCurrentLocation().getY() >= lowerBoundCallerY  )   )
  {
    double oldNumberDistressCallRcv = getNumberDistressCallRcv();
    numberDistressCallRcv = getNumberDistressCallRcv() + 1;

    // System.out.println(“Navy Received Distress Call: “ + myMover +
    // “ From: “ + caller);
    firePropertyChange(NUMBER_DISTRESS_CALL_RCV,
    oldNumberDistressCallRcv,
    getNumberDistressCallRcv());
  }
}
* Returns a String containing the type of Player.
 */

@Override
public String toString()
{
    return "(" + myMover.getType() + ")";
}

/*******************REMOVED ALL SETTERS AND GETTERS******************//
APPENDIX C. MERCHANT MOVER MANAGER JAVA CODE

```java
/*
 * MerchantShipMoverManager.java
 */
package entities;

import java.awt.geom.Point2D;
import java.util.LinkedList;
import java.util.ListIterator;
import simkit.Priority;
import simkit.SimEntityBase;
import simkit.random.RandomVariate;
import simkit.smd.CookieCutterSensor;
import supplemental.MerchantState;
import supplemental.Platform;
import supplemental.PlatformType;

/**
 * Models the behavior of merchant traffic in the GOA and Indian Ocean.
 * @version $Id: MerchantShipMoverManager.java 70 2012-07-11 15:48:44Z crhutchi$
 * @author Chad R Hutchins
 **/ public class MerchantShipMoverManager extends SimEntityBase {

 /**
 * Parameters. Contains Setters and Getters
 **/
 private Platform myMover;
 private CookieCutterSensor sensor;
 private Point2D startLocation;
 private RandomVariate[] pathGenerator;
 private PlatformType platformType;
 private LinkedList<Point2D> wayPoint;

 /**
 * State Variables. Contains only getters, no setters.
 */
```
```java
/* */
protected MerchantState myMovementState;
protected ListIterator<Point2D> nextWayPointIter;
protected double numberPiratesEncountered;
protected double numberPiratesEvaded;
protected double numberHijacked;
protected double numberSuccessfulTransits;
protected Point2D wayPointOne;
protected Point2D wayPointTwo;
protected Point2D wayPointThree;
protected Point2D wayPointFour;
protected boolean isAlive;

private double scale = 0.5;
private double transitSpeed = 15 * scale;

/* */
* String constant for firePropertyChange modification of the state
* variable, not visible outside this class
/* */
private final String MY_MOVEMENT_STATE = "myMovementState";
private final String NUMBER_PIRATES_ENCOUNTERED =
  "numberPiratesEncountered";
private final String NUMBER_PIRATES_EVADED = "numberPiratesEvaded";
private final String NUMBER_HIJACKED = "numberHijacked";
private final String NUMBER_SUCCESSFUL_TRANSITS =
  "numberSuccessfulTransits";
private final String NEXT_WAY_POINT = "nextWaypoint";
private final String IS_ALIVE = "isAlive";

/* */
* String constant for waitDelay method scheduling, visible to other classes
/* */
protected final String MOVE_TO = "MoveTo";
protected final String STOP = "Stop";
protected final String ORDER_STOP = "OrderStop";
protected final String RADIO_NAVY = "RadioNavy";
protected final String DIE = "Die";

/* */
* String constant for all other cases.
```
protected final String PIRATE = "Pirate";

/** Main constructor. Sets mover, sensor, starting location, and path */
* @param myMover
* @param sensor
* @param startLocation
* @param pathGenerator
*/
public MerchantShipMoverManager( Platform myMover,
                                CookieCutterSensor sensor,
                                Point2D startLocation,
                                RandomVariate[] pathGenerator
) {
    this.setMyMover( myMover );
    this.setSensor( sensor );
    this.setStartLocation( startLocation );
    this.setPathGenerator( pathGenerator );
}

/** Default constructor */
public MerchantShipMoverManager() {
}

/** Reset: Resets state variables at end of each replication */
@Override
public void reset() {
    super.reset();
    myMovementState = MerchantState.DEAD_IN_WATER;
    numberPiratesEncountered = 0;
    numberPiratesEvaded = 0;
    numberHijacked = 0;
    wayPoint = new LinkedList<>();
myMover.setInitialLocation( startLocation );

wayPointOne = new Point2D.Double(
    getPathGenerator()[0].generate(),
    getPathGenerator()[1].generate() );

wayPointTwo = new Point2D.Double(
    getPathGenerator()[2].generate(),
    getPathGenerator()[3].generate() );

wayPointThree = new Point2D.Double(
    getPathGenerator()[4].generate(),
    getPathGenerator()[5].generate() );

wayPointFour = new Point2D.Double(
    getPathGenerator()[6].generate(),
    getPathGenerator()[7].generate() );

isAlive = true;

}
wayPoint.add( 0, wayPointOne );
wayPoint.add( 1, wayPointTwo );
wayPoint.add( 2, wayPointThree );
wayPoint.add( 3, wayPointFour );

nextWayPointIter = getWayPoint().
                listIterator();
Point2D nextWaypoint = nextWayPointIter.hasNext() ? nextWayPointIter.
                next() : null;

firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
                getMyMovementState());
firePropertyChange( NEXT_WAY_POINT, nextWaypoint );

if ( nextWaypoint != null )
    {
        waitDelay( MOVE_TO, 0.0, nextWaypoint );
    }

/**
 * End Move: Checks if at the end of the path. If not it schedules MoveTo,
 * if it is at the end of that path it stops the merchant.
 * @param mover
 */
public void doEndMove( Platform mover )
{
    Point2D next = nextWayPointIter.hasNext() ?
                nextWayPointIter.next() : null;
    firePropertyChange( NEXT_WAY_POINT, next );
    if ( myMovementState == MerchantState.TRANSITTING )
        {
            if ( next != null )
                {
                    waitDelay(MOVE_TO, 0.0, next);
                }
if (next == null) {
    waitDelay( STOP, 0.0, myMover );
    double oldNumberSuccessfulTransits = getNumberSuccessfulTransits();
    numberSuccessfulTransits = numberSuccessfulTransits + 1;
    firePropertyChange(NUMBER_SUCCESSFUL_TRANSITS,
                        oldNumberSuccessfulTransits,
                        numberSuccessfulTransits);
}
if (myMovementState == MerchantState.HIJACKED) {
    //System.out.println("Merchant Location: " +
    //myMover.getCurrentLocation());
}

/** Detection: Detects any mover within the sensor range. If it is a pirate
 * the merchant will radio the Navy, increment numberPiratesEncountered, and
 * add the pirate to detectedPirates list.
 * @param contact */
public void doDetection( Platform contact ) {
    LinkedList detectedPirates = new LinkedList();
    if ( contact.getType() == PlatformType.PIRATE &&
        !detectedPirates.contains(contact) &&
        (myMovementState == MerchantState.TRANSITTING ||
         myMovementState == MerchantState.EVADING)) {
        detectedPirates.add(contact);
        double oldNumberPiratesEncountered = getNumberPiratesEncountered();
numberPiratesEncountered = getNumberPiratesEncountered() + 1;

//System.out.println( "I see you Pirate! " + contact );

waitDelay( RADIO_NAVY, 0.0, Priority.HIGHER, this.myMover );

firePropertyChange( NUMBER_PIRATES_ENCOUNTERED,
    oldNumberPiratesEncountered,
    getNumberPiratesEncountered() );

/**
 * RadioNavy: Signals nearest Navy vessel for help. This is done via an adapter in the "main" file.
 * @param merchant
 */

public void doRadioNavy( Platform merchant )
{
    MerchantState oldMyMovementState = getMyMovementState();
    myMovementState = MerchantState.EVADING;

    //System.out.println( "Help me!!!!!" );
    //Send message to nearest Navy vessel

    firePropertyChange( MY_MOVEMENT_STATE, oldMyMovementState,
     getMyMovementState() );
}

/**
 * EvadeSuccessfully: Increments numberPiratesEvaded. Merchant continues on voyage.
 ***/

public void doEvadeSuccessfully()
{
    double oldNumberPiratesEvaded = getNumberPiratesEvaded();
    numberPiratesEvaded = getNumberPiratesEvaded() + 1;

    firePropertyChange( NUMBER_PIRATES_EVADED,
oldNumberPiratesEvaded,
getNumberPiratesEvaded() );

/**
 * Hijacked: Increments numberHijacked. Takes merchant back to pirate base
camp for ransom negotiations.
**/
public void doHijacked( Platform pirate )
{
    double oldNumberHijacked = getNumberHijacked();
    numberHijacked = getNumberHijacked() + 1;
    MerchantState oldState = getMyMovementState();
    myMovementState = MerchantState.HIJACKED;
    isAlive = false;
    myMover.setIsAlive( isAlive );
    waitDelay( STOP, 0.0, Priority.HIGH );
    double pirateCampX = pirate.getInitialLocation().getX();
    double pirateCampY = pirate.getInitialLocation().getY();
    //If pirate Camp is on GOA
    if( pirateCampY > 285 )
    {
        double hijackedIOMerchantX;
        double hijackedIOMerchantY;
        hijackedIOMerchantX = pirateCampY + 5;
        hijackedIOMerchantY = pirateCampX;
        Point2D merchantIOHijackLocation = new Point2D.Double(  
            hijackedIOMerchantX, hijackedIOMerchantY );
        waitDelay( MOVE_TO, 2.0, merchantIOHijackLocation );
    }
    //IF pirate camp is on Indian Ocean
    else
{  
    double hijackedGOAMerchantX;
    double hijackedGOAMerchantY;
    hijackedGOAMerchantX = pirateCampX + 5;
    hijackedGOAMerchantY = pirateCampY;
    Point2D merchantGOAHijackLocation = new Point2D.Double(
        hijackedGOAMerchantX, hijackedGOAMerchantY);
    waitDelay(MOVE_TO, 2.0, merchantGOAHijackLocation);
}

firePropertyChange(NUMBER_HIJACKED, oldNumberHijacked,
    getNumberHijacked());
firePropertyChange(MY_MOVEMENT_STATE, oldState, getMyMovementState());

/**
 * Returns a String containing the type of Player.
 * *
 * @Override
 * public String toString()
 {  
    return "I am a (" + myMover.getType() + ")";
}

//**************************REMOVED ALL SETTERS AND GETTERS**************************//
APPENDIX D. BAYLA PIRATE DEPARTURE PROCESS JAVA CODE.

```java
package process;

import simkit.SimEntityBase;
import simkit.random.RandomVariate;

/**
 * Generates departure times for pirates leaving the Gulf of Aden(GOA).
 * 
 * @version $Id: BaylaPirateDepartureProcess.java 168 2013-02-14 06:59:16Z crhutchi $
 * 
 * @author Chad R Hutchins
 *
 * public class BaylaPirateDepartureProcess extends SimEntityBase {

 /**
 * Parameters. Contains Setters and Getters
 **/
 private RandomVariate IoDepartureTimeGenerator; //Generates departure times

 /**
 * State Variables. Contains only getters, no setters.
 **/
 protected int numberDepartedIO;

 /**
 * String constant for firePropertyChange modification of the state
 * * variable, not visible outside this class
 **/
 private final String NUMBER_DEPARTED_IO = “numberDepartedIO”;

 /**
 * String constant for waitDelay method scheduling, visible to other classes
 **/
 protected final String DEPART = “Depart”;
```
public BaylaPirateDepartureProcess(RandomVariate rv) {
    this.setIoDepartureTimeGenerator(rv);
}

@Override
public void reset() {
    super.reset();
    numberDepartedIO = 0;
}

/**
 * Reset Event: resets all state variables after each replication.
 */

public void doRun() {
    firePropertyChange(NUMBER_DEPARTED_IO, getNumberDepartedIO());
    waitDelay(DEPART, IoDepartureTimeGenerator.generate());
}

/**
 * LeaveGoaPirateCamp Event: increments numberDepartedSB and schedules
 * it's self with delay of departureTime.
 */

public void doDepart() {
    int oldState = getNumberDepartedIO();
    numberDepartedIO = getNumberDepartedIO() + 1;
    firePropertyChange(NUMBER_DEPARTED_IO, oldState,
...
getNumberDepartedIO();

    //**Comment for visual testing***/
    waitDelay(DEPART, IoDepartureTimeGenerator.generate());
} }

/**
   * @return the IoDepartureTimeGenerator
   */
public RandomVariate getIoDepartureTimeGenerator()
{
    return IoDepartureTimeGenerator;
}

/**
   * @param IoDepartureTimeGenerator the IoDepartureTimeGenerator to set
   */
public void setIoDepartureTimeGenerator(
    RandomVariate goaDepartureTimeGenerator )
{
    this.IoDepartureTimeGenerator = goaDepartureTimeGenerator;
}

/**
   * @return the numberDepartedIO
   */
public int getNumberDepartedIO()
{
    return numberDepartedIO;
}
APPENDIX E. BAYLA PIRATE CAMP JAVA CODE.

```java
package process;

import entities.PirateMoverManager;
import java.util.Arrays;
import java.util.LinkedList;
import simkit.Priority;
import simkit.SimEntityBase;

public class BaylaPirateCamp extends SimEntityBase {
    private PirateMoverManager[] pirateMM;
    protected LinkedList<PirateMoverManager> myPirates;
    protected int numberDepartedIO;

    public BaylaPirateCamp(PirateMoverManager[] pirateMM)
```
```java
    { 
        this.setPirateMM(pirateMM);
        this.myPirates = new LinkedList<PirateMoverManager>();
    }

    /**
     * Reset Event: resets all state variables after each replication.
     */
    @Override
    public void reset() {
        super.reset();
        numberDepartedIO = 0;
        myPirates.clear();
        myPirates.addAll(Arrays.asList(pirateMM));
    }

    public void doRun() {
        //firePropertyChange( NUMBER_DEPARTED_IO, getNumberDepartedIO() );
    }

    public void doDepart() {
        if (!myPirates.isEmpty()) {
            //System.out.println("myPirate size: " + myPirates.size());
            waitDelay(LEAVE, 0.0);
        }
    }

    public void doLeave() {
        PirateMoverManager p = myPirates.removeFirst();
        p.waitDelay(LEAVE_IO_PIRATE_CAMP, 0.0, Priority.HIGH);
        int oldState = getNumberDepartedIO();
        numberDepartedIO = getNumberDepartedIO() + 1;
        firePropertyChange( NUMBER_DEPARTED_IO, oldState,
                            getNumberDepartedIO() );
```
113
79 // System.out.println(
80 // "Number Pirate Departures from Bayla " +
81 // getNumberDepartedIO() );
82
83 }
84
85 /**<
86 * @return the myPirates
87 */
88 public LinkedList<PirateMoverManager> getMyPirates() {
89     return myPirates;
90 }
91
92 /**<
93 * @return the numberDepartedIO
94 */
95 public int getNumberDepartedIO() {
96     return numberDepartedIO;
97 }
98
99 /**<
100 * @return the pirateMM
101 */
102 public PirateMoverManager[] getPirateMM() {
103     return pirateMM.clone();
104 }
105
106 /**<
107 * @param pirateMM the pirateMM to set
108 */
109 public void setPirateMM(PirateMoverManager[] pirateMM) {
110     this.pirateMM = pirateMM.clone();
111 }
112
113
114 }
package process;

import simkit.SimEntityBase;
import simkit.random.RandomVariate;

public class SuezToOmanMerchantDepartureProcess extends SimEntityBase {
    private RandomVariate merchantDepartureTimeGenerator;
    protected int numberDeparted;
    private final String NUMBER_DEPARTED = "numberDeparted";
}
protected final String DEPART = “Depart”;

/**
 * Main constructor. Sets merchantDepartureTimeGenerator.
 *
 * @param rv The RandomVariate instance for DepartureTimeGeneratorSB times
 */
public SuezToOmanMerchantDepartureProcess( RandomVariate rv )
{
    this.setMerchantDepartureTimeGenerator( rv );
}

/**
 * Reset Event: resets all state variables after each replication.
 */
@Override
public void reset()
{
    super.reset();
    numberDeparted = 0;
}

/**
 * Run Event: Initial event - put on event list at the start of e run.
 * State Transition: in reset() Schedule: First LeaveCampIo event with
 * departureTime delay
 */
public void doRun()
{
    firePropertyChange( NUMBER_DEPARTED, getNumberDeparted() );
    waitDelay( DEPART, merchantDepartureTimeGenerator.generate() );
}

/**
 * Depart Event: increments numberDeparted and schedules
 * it's self with delay of departureTime.
 */
public void doDepart()
{
    int oldState = getNumberDeparted();
numberDeparted = getNumberDeparted() + 1;
firePropertyChange( NUMBER_DEPARTED, oldState,
    getNumberDeparted() );

//**Comment for visual testing**//
waitDelay( DEPART, merchantDepartureTimeGenerator.generate() );
}
/**
 * @return the merchantDepartureTimeGenerator
 */
public RandomVariate getMerchantDepartureTimeGenerator()
{
    return merchantDepartureTimeGenerator;
}

/**
 * @param merchantDepartureTimeGenerator the merchantDepartureTimeGenerator
 * to set
 */
public void setMerchantDepartureTimeGenerator(
    RandomVariate merchantDepartureTimeGenerator )
{
    this.merchantDepartureTimeGenerator = merchantDepartureTimeGenerator;
}

/**
 * @return the numberDeparted
 */
public int getNumberDeparted()
{
    return numberDeparted;
}
APPENDIX G. SUEZ TO OMAN ORIGIN PORT JAVA CODE

```java
/*
 * SuezToOmanOriginPort.java
 */

package process;

import entities.MerchantShipMoverManager;
import java.util.Arrays;
import java.util.LinkedList;
import simkit.Priority;
import simkit.SimEntityBase;

/**
 * Port of Origin for merchants sailing from Suez to Oman.
 *
 * @author Chad R Hutchins
 */
public class SuezToOmanOriginPort extends SimEntityBase {
    private MerchantShipMoverManager[] merchantMM;

    protected LinkedList<MerchantShipMoverManager> myMerchants;
    protected int numberDepartedPort;

    /**
     * String constant for firePropertyChange modification of the state
     * variable, not visible outside this class
     **/
    private final String NUMBER_DEPARTED_PORT = "numberDepartedPort";

    /**
     * String constant for waitDelay method scheduling, visible to other classes
     **/
    protected final String LEAVE = "Leave";
    protected final String START_TRANSIT = "StartTransit";

    public SuezToOmanOriginPort( MerchantShipMoverManager[] merchantMM )
```
```java
{  
    this.setMerchantMM(merchantMM);
    this.myMerchants = new LinkedList<MerchantShipMoverManager>();
}
/**
 * Reset Event: resets all state variables after each replication.
 */
@Override
public void reset()
{
    super.reset();
    numberDepartedPort = 0;
    myMerchants.clear();
    myMerchants.addAll(Arrays.asList(getMerchantMM()));
}

public void doRun()
{
    //firePropertyChange( NUMBER_DEPARTED_PORT, getNumberDepartedPort() );
}

public void doDepart()
{
    if (!myMerchants.isEmpty())
    {
        waitDelay(LEAVE, 0.0);
    }
}

public void doLeave()
{
    MerchantShipMoverManager m = myMerchants.removeFirst();
    m.waitDelay(START_TRANSIT, 0.0, Priority.HIGH);
    int oldState = getNumberDepartedPort();
    numberDepartedPort = getNumberDepartedPort() + 1;
    firePropertyChange( NUMBER_DEPARTED_PORT, oldState,
                        getNumberDepartedPort() );
```
System.out.println("Number Merchant Ship Departures from SuezToMaldives Port "+
                getNumberDepartedPort());
}

/**
 * @return the myMerchants
 */
public LinkedList<MerchantShipMoverManager> getMyMerchants() {
    return myMerchants;
}

/**
 * @return the numberDepartedPort
 */
public int getNumberDepartedPort() {
    return numberDepartedPort;
}

/**
 * @return the merchantMM
 */
public MerchantShipMoverManager[] getMerchantMM() {
    return merchantMM.clone();
}

/**
 * @param merchantMM the merchantMM to set
 */
public void setMerchantMM(MerchantShipMoverManager[] merchantMM) {
    this.merchantMM = merchantMM.clone();
}
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
(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
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.
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.
<table>
<thead>
<tr>
<th></th>
<th>Date: Time:</th>
<th>Author:</th>
<th>Message:</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monday, 25 June 2012 11:36:38-PDT</td>
<td>LawDawg:</td>
<td>Would you include the UN in this?</td>
</tr>
<tr>
<td>2</td>
<td>Saturday, 30 June 2012 09:20:19-PDT</td>
<td>EdwardPreble:</td>
<td>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.</td>
</tr>
<tr>
<td>3</td>
<td>Friday, 6 July 2012 10:39:43-PDT</td>
<td>gm_donb:</td>
<td>Needed: openly available maps of where ships are being held for ransom, and tracks taken when captured ships are brought back to Somalia by the pirates.</td>
</tr>
<tr>
<td>4</td>
<td>Tuesday, 17 July 2012 10:54:38-PDT</td>
<td>LawDawg:</td>
<td>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.</td>
</tr>
<tr>
<td>5</td>
<td>Monday, 23 July 2012 16:16:30-PDT</td>
<td>LawDawg:</td>
<td><a href="http://www.esa.int/esaEO/SEMATD8X73H_index_0.html">http://www.esa.int/esaEO/SEMATD8X73H_index_0.html</a> This article explores how environmental conditions limit pirate activity. Conclusions show that wave height and pirate attacks were correlated as well as wind speed and pirate activity. (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.</td>
</tr>
<tr>
<td>6</td>
<td>Tuesday, 24 July 2012 13:35:56-PDT</td>
<td>WillyRobert:</td>
<td>interesting and something that we can without a doubt simulate! Thanks.</td>
</tr>
<tr>
<td>7</td>
<td>Sunday, 29 July 2012 11:46:21-PDT</td>
<td>WillyRobert:</td>
<td>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??</td>
</tr>
<tr>
<td>8</td>
<td>Sunday, 29 July 2012 11:47:44-PDT</td>
<td>WillyRobert:</td>
<td>I'll hopefully have a pic up within the next day or so to give example of what I'm thinking.</td>
</tr>
<tr>
<td>9</td>
<td>Thursday, 2 August 2012 09:32:52-PDT</td>
<td>LawDawg:</td>
<td>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 vigilance on the part of naval ships enforcing the quarantine. As for those who don’t report or check in, enter at your own risk.</td>
</tr>
<tr>
<td>10</td>
<td>Thursday, 2 August 2012 09:33:30-PDT</td>
<td>LawDawg:</td>
<td>This could all be enforced through insurance rates as well...something to consider.</td>
</tr>
<tr>
<td>11</td>
<td>Thursday, 2 August 2012 11:56:04-PDT</td>
<td>LawDawg:</td>
<td>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 implications for placement of naval vessels.</td>
</tr>
</tbody>
</table>
gm_donb: The maps of pirate camps don’t really pertain to this plan. They should be in separate plans for each pirate camp.

gm_donb: Multiple separate idea cards and action plans have been spun off for each pirate camp.

WillyRobert: Will the patrols in the IRTC remain the same? Or will they more of a quarantine role as well?

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.

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?

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.

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
https://mmowgli.nps.edu/piracy/images/9/SomaliaPirateCamps.png
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

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
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

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)
public class PirateCampOperations {
    public static void main(String[] args) {
        // Simulation specific constants
        double simTime = 730.0; // 1 Month
        double scaleDistance = 0.5; // scales the distances in the simulation
        double pirateMaxSpeed = 15 * scaleDistance;
        double pirateVisualSensorRange = 15 * scaleDistance;
        double navyMaxSpeed = 30.0 * scaleDistance;
        double navySurfaceRadarRange = 25 * scaleDistance;
        double merchantSurfaceRadarRange = 25 * scaleDistance;
        double merchantMaxSpeed = 20 * scaleDistance;

        // Probability Distribution Constants
        double elaayoInterarrivalTimeLambda = 150.0;
        double qandalaInterarrivalTimeLambda = 100.0;
        double aluulaInterarrivalTimeLambda = 150.0;
        double bargalInterarrivalTimeLambda = 150.0;
        double hafunInterarrivalTimeLambda = 100.0;
        double baylaInterarrivalTimeLambda = 100.0;
        double EylInterarrivalTimeLambda = 100.0;
        double GaracadInterarrivalTimeLambda = 100.0;
        double HobyoInterarrivalTimeLambda = 100.0;
        double HarardhereInterarrivalTimeLambda = 100.0;

        // Pirate Camp Assembly
        int numElaayoyPirates = 6;
        int numQandalaPirates = 8;
        int numAluulaPirates = 6;
        int numBargalPirates = 6;
        int numHafunPirates = 8;
        int numBaylaPirates = 6;
        int numEylPirates = 6;
        int numGaracadPirates = 8;
        int numHobyoPirates = 6;
        int numHarardherePirates = 8;
        int numSuezToOmanMerchants = 370;
        int numSuezToMaldivesMerchants = 370;
        int numOmanToSuezMerchants = 370;
        int numOmanToMaldivesMerchants = 370;
        int numMaldivesToSuezMerchants = 370;
        int numMaldivesToOmanMerchants = 370;

        // Merchant Assembly
        int numIoNavyShips = 7;
        int numGoaNavyShips = 3;
        int numOmanToSuezShips = 370;
        int numOmanToMaldivesShips = 370;
        int numMaldivesToSuezShips = 370;
        int numMaldivesToOmanShips = 370;
    }
}

APPENDIX K. PIRATE CAMP OPERATIONS SIMKIT ASSEMBLY

1 /*
2  * PirateCampOperations.java
3  */
4 //****************Imports Removed******************************************/
58
59 /**
60  *
61  *
62  *
63  *
64  *
65  *
66  *
67  *
68  *
69  */
70 public class PirateCampOperations {
71     public static void main(String[] args) {
72         // Simulation specific constants
73         double simTime = 730.0; // 1 Month
74         double scaleDistance = 0.5; // scales the distances in the simulation
75
76         // Pirate Camp Assembly
77         int numElaayoyPirates = 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         int numSuezToOmanMerchants = 370;
88         int numSuezToMaldivesMerchants = 370;
89         int numOmanToSuezMerchants = 370;
90         int numOmanToMaldivesMerchants = 370;
91         int numMaldivesToSuezMerchants = 370;
92         int numMaldivesToOmanMerchants = 370;
93
94         // Merchant Assembly
95         int numIoNavyShips = 7;
96         int numGoaNavyShips = 3;
97         int numOmanToSuezShips = 370;
98         int numOmanToMaldivesShips = 370;
99         int numMaldivesToSuezShips = 370;
100        int numMaldivesToOmanShips = 370;
101        double merchantSurfaceRadarRange = 25 * 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;
double baylaInterarrivalTimeLambda = 150.0;
double eylInterarrivalTimeLambda = 150.0;
double garacadInterarrivalTimeLambda = 100.0;
double hobyoInterarrivalTimeLambda = 150.0;
double harardhereInterarrivalTimeLambda = 100.0;
double stoInterarrivalTimeLambda = 1.2;
double stmInterarrivalTimeLambda = 2.21;
double otsInterarrivalTimeLambda = 2.22;
double otmInterarrivalTimeLambda = 2.23;
double mtsInterarrivalTimeLambda = 2.24;
double mtoInterarrivalTimeLambda = 2.25;
probOfAttackingDecision = 0.75;
minUnsuccessfulAttackTime = 0.1;
maxUnsuccessfulAttackTime = 0.75;

//*****Constants FOR INITIAL LOCATIONS OF ENTITIES***********************///

Point2D pirateCampElaayo = new Point2D.Double(306.0, 301.0);
Point2D pirateCampQandala = new Point2D.Double(339.0, 310.0);
Point2D pirateCampAluula = new Point2D.Double(367.0, 323.0);
Point2D pirateCampBargal = new Point2D.Double(379.0, 300.0);
Point2D pirateCampHafun = new Point2D.Double(384.0, 273.0);
Point2D pirateCampBayla = new Point2D.Double(370.0, 240.0);
Point2D pirateCampEyl = new Point2D.Double(345.0, 183.0);
Point2D pirateCampGaracad = new Point2D.Double(322.0, 155.0);
Point2D pirateCampHobyo = new Point2D.Double(305.0, 103.0);
Point2D pirateCampHarardhere = new Point2D.Double(283.0, 79.0);

Point2D initialLocationNavyPB1 = new Point2D.Double(294.0, 325.0);
Point2D initialLocationNavyPB2 = new Point2D.Double(331.0, 337.0);
Point2D initialLocationNavyPB3 = new Point2D.Double(365.0, 346.0);
Point2D initialLocationNavyPB4 = new Point2D.Double(408.0, 313.0);
Point2D initialLocationNavyPB5 = new Point2D.Double(410.0, 276.0);
Point2D initialLocationNavyPB6 = new Point2D.Double(396.0, 243.0);
Point2D initialLocationNavyPB7 = new Point2D.Double(370.0, 185.0);
Point2D initialLocationNavyPB8 = new Point2D.Double(344.0, 154.0);
Point2D initialLocationNavyPB9 = new Point2D.Double(330.0, 101.0);
Point2D initialLocationNavyPB10 = new Point2D.Double(310.0, 76.0);

Point2D initialLocationMerchantSuezToMaldives =
    new Point2D.Double(145.0, 345.0);
Point2D initialLocationMerchantSuezToOman =
    new Point2D.Double(145.0, 345.0);
Point2D initialLocationMerchantMaldivesToSuez =
    new Point2D.Double(1135.0, 250.0);
Point2D initialLocationMerchantMaldivesToOman =
    new Point2D.Double(1135.0, 250.0);
Point2D initialLocationMerchantOmanToMaldives =
    new Point2D.Double(655.0, 725.0);
Point2D initialLocationMerchantOmanToSuez =
    new Point2D.Double(655.0, 725.0);

Point2D minLatGoaPiratePath = 145.00;
Point2D maxLatGoaPiratePath = 465.00;
Point2D minLonGoaPiratePath = 340.0;
Point2D maxLonGoaPiratePath = 460.0;
Point2D minLatPiratePath = 400.0;
double maxLatIoPiratePath = 1060.0;
double minLonIoPiratePath = 0.0;
double maxLonIoPiratePath = 720.0;
double minLatGoaAndIoPiratePath = 145.0;
double maxLatGoaAndIoPiratePath = 1060.0;
double minLonGoaAndIoPiratePath = 0.0;
double maxLonGoaAndIoPiratePath = 720.0;

///**Merchant Paths**///

double minLatSuezToMaldivesMerchantWaypoint1 = 170.00;
double maxLatSuezToMaldivesMerchantWaypoint1 = 194.0;
double minLonSuezToMaldivesMerchantWaypoint1 = 320.0;
double maxLonSuezToMaldivesMerchantWaypoint1 = 328.0;
double minLatSuezToMaldivesMerchantWaypoint2 = 425.0;
double maxLatSuezToMaldivesMerchantWaypoint2 = 450.0;
double minLonSuezToMaldivesMerchantWaypoint2 = 390.0;
double maxLonSuezToMaldivesMerchantWaypoint2 = 415.0;
double minLatSuezToMaldivesMerchantWaypoint3 = 1055.0;
double maxLatSuezToMaldivesMerchantWaypoint3 = 1090.0;
double minLonSuezToMaldivesMerchantWaypoint3 = 250.0;
double maxLonSuezToMaldivesMerchantWaypoint3 = 265.0;
double minLatSuezToMaldivesMerchantWaypoint4 = 1115.0;
double maxLatSuezToMaldivesMerchantWaypoint4 = 1140.0;
double minLonSuezToMaldivesMerchantWaypoint4 = 220.0;
double maxLonSuezToMaldivesMerchantWaypoint4 = 260.0;
double minLatSuezToOmanMerchantWaypoint1 = 170.00;
double maxLatSuezToOmanMerchantWaypoint1 = 194.0;
double minLonSuezToOmanMerchantWaypoint1 = 320.0;
double maxLonSuezToOmanMerchantWaypoint1 = 328.0;
double minLatSuezToOmanMerchantWaypoint2 = 425.0;
double maxLatSuezToOmanMerchantWaypoint2 = 450.0;
double minLonSuezToOmanMerchantWaypoint2 = 390.0;
double maxLonSuezToOmanMerchantWaypoint2 = 415.0;
double minLatSuezToOmanMerchantWaypoint3 = 625.0;
double maxLatSuezToOmanMerchantWaypoint3 = 645.0;
double minLonSuezToOmanMerchantWaypoint3 = 515.0;
double maxLonSuezToOmanMerchantWaypoint3 = 530.0;
double minLatSuezToOmanMerchantWaypoint4 = 685.0;
double maxLatSuezToOmanMerchantWaypoint4 = 700.0;
double minLonSuezToOmanMerchantWaypoint4 = 720.0;
double maxLonSuezToOmanMerchantWaypoint4 = 725.0;
double minLatMaldivesToSuezMerchantWaypoint1 = 1065.0;
double maxLatMaldivesToSuezMerchantWaypoint1 = 1090.00;
double minLonMaldivesToSuezMerchantWaypoint1 = 265.0;
double maxLonMaldivesToSuezMerchantWaypoint1 = 280.0;
double minLatMaldivesToSuezMerchantWaypoint2 = 425.0;
double maxLatMaldivesToSuezMerchantWaypoint2 = 460.0;
double minLonMaldivesToSuezMerchantWaypoint2 = 410.0;
double maxLonMaldivesToSuezMerchantWaypoint2 = 420.0;
double minLatMaldivesToSuezMerchantWaypoint3 = 170.0;
double maxLatMaldivesToSuezMerchantWaypoint3 = 200.0;
double minLonMaldivesToSuezMerchantWaypoint3 = 325.0;
double maxLonMaldivesToSuezMerchantWaypoint3 = 340.0;
double minLatMaldivesToSuezMerchantWaypoint4 = 140.0;
double maxLatMaldivesToSuezMerchantWaypoint4 = 155.0;
double minLonMaldivesToSuezMerchantWaypoint4 = 330.0;
double maxLonMaldivesToSuezMerchantWaypoint4 = 350.0;
double minLatMaldivesToOmanMerchantWaypoint1 = 1065.0;
double maxLatMaldivesToOmanMerchantWaypoint1 = 1090.0;
double minLonMaldivesToOmanMerchantWaypoint1 = 265.0;
double maxLonMaldivesToOmanMerchantWaypoint1 = 280.0;
double minLatMaldivesToOmanMerchantWaypoint2 = 890.0;
double maxLatMaldivesToOmanMerchantWaypoint2 = 900.0;
double minLonMaldivesToOmanMerchantWaypoint2 = 500.0;
double maxLonMaldivesToOmanMerchantWaypoint2 = 515.0;
double minLatMaldivesToOmanMerchantWaypoint3 = 695.0;
double maxLatMaldivesToOmanMerchantWaypoint3 = 715.0;
double minLonMaldivesToOmanMerchantWaypoint3 = 700.0;
double maxLonMaldivesToOmanMerchantWaypoint3 = 720.0;
double minLatMaldivesToOmanMerchantWaypoint4 = 685.0;
double maxLatMaldivesToOmanMerchantWaypoint4 = 700.0;
double minLonMaldivesToOmanMerchantWaypoint4 = 725.0;
double maxLonMaldivesToOmanMerchantWaypoint4 = 730.0;
double minLatOmanToMaldivesMerchantWaypoint1 = 700.0;
double maxLatOmanToMaldivesMerchantWaypoint1 = 720.0;
double minLonOmanToMaldivesMerchantWaypoint1 = 685.0;
double maxLonOmanToMaldivesMerchantWaypoint1 = 695.0;
double minLatOmanToMaldivesMerchantWaypoint2 = 620.0;
double maxLatOmanToMaldivesMerchantWaypoint2 = 635.0;
double minLonOmanToMaldivesMerchantWaypoint2 = 530.0;
double maxLonOmanToMaldivesMerchantWaypoint2 = 545.0;
double minLatOmanToMaldivesMerchantWaypoint3 = 170.0;
double maxLatOmanToMaldivesMerchantWaypoint3 = 200.0;
double minLonOmanToMaldivesMerchantWaypoint3 = 325.0;
double maxLonOmanToMaldivesMerchantWaypoint3 = 340.0;
double minLatOmanToMaldivesMerchantWaypoint4 = 140.0;
double maxLatOmanToMaldivesMerchantWaypoint4 = 155.0;
double minLonOmanToMaldivesMerchantWaypoint4 = 335.0;
double maxLonOmanToMaldivesMerchantWaypoint4 = 350.0;

//**********PROBABILITY DISTRIBUTIONS FOR ENTIRE SIMULATION**********/

// Arrival and Departure Processes
*
* TODO: Discuss this distribution
*/
RandomVariate elaayoInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," elaayoInterarrivalTimeLambda);
RandomVariate qandalaInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," qandalaInterarrivalTimeLambda);
RandomVariate aluulaInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," aluulaInterarrivalTimeLambda);
RandomVariate bargalInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," bargalInterarrivalTimeLambda);
RandomVariate hafunInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," hafunInterarrivalTimeLambda);
RandomVariate baylaInterarrivalTime = RandomVariateFactory.
getInstance("Poisson," baylaInterarrivalTimeLambda);
getInstance("Poisson," baylaInterarrivalTimeLambda);
RandomVariate eylInterarrivalTime = RandomVariateFactory.getInstance("Poisson," eylInterarrivalTimeLambda);
RandomVariate garacadInterarrivalTime = RandomVariateFactory.getInstance("Poisson," garacadInterarrivalTimeLambda);
RandomVariate hobyoInterarrivalTime = RandomVariateFactory.getInstance("Poisson," hobyoInterarrivalTimeLambda);
RandomVariate harardhereInterarrivalTime = RandomVariateFactory.getInstance("Poisson," harardhereInterarrivalTimeLambda);
RandomVariate stoMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," stoInterarrivalTimeLambda);
RandomVariate stmMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," stmInterarrivalTimeLambda);
RandomVariate otsMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," otsInterarrivalTimeLambda);
RandomVariate otmMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," otmInterarrivalTimeLambda);
RandomVariate mtsMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," mtsInterarrivalTimeLambda);
RandomVariate mtoMerchantInterarrivalTime = RandomVariateFactory.getInstance("Poisson," mtoInterarrivalTimeLambda);

/***Applies to both IO and GOA pirates***/

/*/ This distribution attempts to capture whether or not the pirate will
attack a detected merchant vessel. We attempt to capture the types
of vessels the proportion of the types of vessels that traverse
around the Horn of Africa and weather factors. However, without any
real data, which no one will ever have, this is just an educated
guess, and the best COA is to either say 50/50 or that the pirates
are more likely to attack than not.
*/
DiscreteRandomVariate attackDecision =
    RandomVariateFactory
        .getDiscreteRandomVariateInstance("Bernoulli,
            probOfAttackingDecision);

//Random variable for how long an attack on a merchant takes
RandomVariate[] unsuccessfulAttackTime = new RandomVariate[1];
unsuccessfulAttackTime[0] =
    RandomVariateFactory.getInstance("Uniform,
        minUnsuccessfulAttackTime,
        maxUnsuccessfulAttackTime);

/***GOA Pirates probability distributions***/
RandomVariate[] goaPiratePathGenerator = new RandomVariate[2];
goaPiratePathGenerator[0] =
    RandomVariateFactory.getInstance("Uniform,
        minLatGaoPiratePath,
        maxLatGaoPiratePath);
goaPiratePathGenerator[1] =
    RandomVariateFactory.getInstance("Uniform,
        minLonGaoPiratePath,
        maxLonGaoPiratePath);

/***IO Pirates probability distributions***/
/*
* TODO: Discuss this distribution
RandomVariate[] ioPiratePathGenerator = new RandomVariate[2];
ioPiratePathGenerator[0] = RandomVariateFactory.getInstance("Uniform,"
minLatIoPiratePath,
maxLatIoPiratePath);
minLonIoPiratePath,
maxLonIoPiratePath);

RandomVariate[] bargalPiratePathGenerator = new RandomVariate[2];
bargalPiratePathGenerator[0] = RandomVariateFactory.getInstance("Uniform,"
minLatGoaAndIoPiratePath,
maxLatGoaAndIoPiratePath);
minLonGoaAndIoPiratePath,
maxLonGoaAndIoPiratePath);

DiscreteRandomVariate successOrFailGenerator = RandomVariateFactory.getDiscreteRandomVariateInstance("Bernoulli," 0.26);

double totalNumDepartedGOA = 0;
double totalNumDepartedIO = 0;
double totalNumberPiratesDeparted = 0;
double numberOfGoaPiratesDetected = 0;
double numberOfIoPiratesDetected = 0;
double totalNumberPiratesDetected = 0;
double numberAttemptedAttacksEllayoPirate = 0;
double numberAttemptedAttacksQandalaPirate = 0;
double numberAttemptedAttacksAluulaPirate = 0;
double numberAttemptedAttacksBargalPirate = 0;
double numberAttemptedAttacksHafunPirate = 0;
double numberAttemptedAttacksBaylaPirate = 0;
double numberAttemptedAttacksEylPirate = 0;
double numberAttemptedAttacksGaracadPirate = 0;
double numberAttemptedAttacksHobyoPirate = 0;
double numberAttemptedAttacksHarardherePirate = 0;
double totalAttemptedAttacks = 0;
double totalNumberSuccessfulHijacksStM = 0;
double totalNumberSuccessfulHijacksStO = 0;
double totalNumberSuccessfulHijacksOtM = 0;
double totalNumberSuccessfulHijacksOtS = 0;
double totalNumberSuccessfulHijacksMtS = 0;
double totalNumberSuccessfulHijacksMtO = 0;
415 double totalNumberSuccessfulHijacks = 0;
416 double numberOfSOMerchantTransits = 0;
417 double numberOfSMMerchantTransits = 0;
418 double numberOfOMerchantTransits = 0;
419 double numberOfOMMerchantTransits = 0;
420 double numberOfTotalMerchantTransits = 0;
421 double numberOfOSSuccessfulTransits = 0;
422 double numberOfSSSuccessfullTransits = 0;
423 double numberOfOMSuccessfulTransits = 0;
424 double numberOfOSSuccessfullTransits = 0;
425 double numberOfMMSuccessfullTransits = 0;
426 double numberOfMOSuccessfulTransits = 0;
427 double numberOfTotalMerchantTransits = 0;
428 double navalEffectiveness = 0;
429 double pirateEffectiveness1 = 0;
430 double pirateEffectiveness2 = 0;
431 double merchantSuccessRate = 0;

432 //******************START OF PIRATE ENTITIES*******************************/
433 PlatformType typePirate = PlatformType.PIRATE;
434 //******************START OF GOA Pirates****************************/
435 //*****************START OF ELAAYO PIRATE IMPLEMENTATION***********************/
436 ElaayoPirateDepartureProcess elaayoDepartureTimeProcess =
437 new ElaayoPirateDepartureProcess(elaayoInterarrivalTime);
438 Platform[] elaayoPirateMover = new Platform[ numElaayoPirates ];
439 for (int i = 0; i < numElaayoPirates; ++i) {
440     elaayoPirateMover[i] = new Platform("Pirate-Ellaayo" + i,
441         pirateCampElaayo,
442         pirateMaxSpeed, typePirate);
443     System.out.println("Pirate: " + elaayoPirateMover[0].paramString());
444 }
445 CookieCutterSensor[] elaayoPirateSensor =
446 new CookieCutterSensor[elaayoPirateMover.length];
447 for (int i = 0; i < elaayoPirateMover.length; ++i) {
448     elaayoPirateSensor[i] = new CookieCutterSensor(elaayoPirateMover[i],
449         pirateVisualSensorRange);
450 }
451 PirateMoverManager[] elaayoPirateManager =
452 new PirateMoverManager[elaayoPirateMover.length];
453 for (int i = 0; i < elaayoPirateMover.length; ++i) {
454     elaayoPirateManager[i] =
455         new PirateMoverManager(elaayoPirateMover[i],
456             elaayoPirateSensor[i],
457             pirateCampElaayo,
458             goaPiratePathGenerator,
459             attackDecision,
460             unsuccessfulAttackTime);
461 }
462 ElaayoPirateCamp epc = new ElaayoPirateCamp( elaayoPirateManager );
463 elaayoDepartureTimeProcess.addSimEventListener(epc);
QandalaPirateDepartureProcess qandalaDepartureTimeProcess =
    new QandalaPirateDepartureProcess(qandalaInterarrivalTime);

Platform[] qandalaPirateMover = new Platform[ numQandalaPirates ];
for (int i = 0; i < numQandalaPirates; ++i) {
    qandalaPirateMover[i] = new Platform("Pirate-Qandala" + i,
        pirateCampQandala,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + qandalaPirateMover[0].paramString());

CookieCutterSensor[] qandalaPirateSensor =
    new CookieCutterSensor[qandalaPirateMover.length];
for (int i = 0; i < qandalaPirateMover.length; ++i) {
    qandalaPirateSensor[i] =
        new CookieCutterSensor(qandalaPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] qandalaPirateManager =
    new PirateMoverManager[qandalaPirateMover.length];
for (int i = 0; i < qandalaPirateMover.length; ++i) {
    qandalaPirateManager[i] =
        new PirateMoverManager(qandalaPirateMover[i],
            qandalaPirateSensor[i],
            pirateCampQandala,
            goaPiratePathGenerator,
            attackDecision,
            unsuccessfulAttackTime);
}

QandalaPirateCamp qpc = new QandalaPirateCamp(qandalaPirateManager);
qandalaDepartureTimeProcess.addSimEventListener(qpc);

//*****************END OF QANDALA PIRATE IMPLEMENTATION***********************//
//****************START OF ALUULA PIRATE IMPLEMENTATION***********************//

AluulaPirateDepartureProcess aluulaDepartureTimeProcess =
    new AluulaPirateDepartureProcess(aluulaInterarrivalTime);

Platform[] aluulaPirateMover = new Platform[ numAluulaPirates ];
for (int i = 0; i < numAluulaPirates; ++i) {
    aluulaPirateMover[i] = new Platform("Pirate-Aluula" + i,
        pirateCampAluula,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + aluulaPirateMover[0].paramString());

CookieCutterSensor[] aluulaPirateSensor =
    new CookieCutterSensor[aluulaPirateMover.length];
for (int i = 0; i < aluulaPirateMover.length; ++i) {
    aluulaPirateSensor[i] =
        new CookieCutterSensor(aluulaPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] aluulaPirateManager =
new PirateMoverManager[aluulaPirateMover.length];
for (int i = 0; i < aluulaPirateMover.length; i++) {
    aluulaPirateManager[i] =
        new PirateMoverManager(aluulaPirateMover[i],
                                aluulaPirateSensor[i],
                                pirateCampAluula,
                                goaPiratePathGenerator,
                                attackDecision,
                                unsuccessfulAttackTime);
}

AluulaPirateCamp apc = new AluulaPirateCamp(aluulaPirateManager);
aluulaDepartureTimeProcess.addSimEventListener(apc);

*****************END OF ALUULA PIRATE IMPLEMENTATION*************************/
************************END OF GOA Pirates************************************
*******************START OF IO Pirates***************************************/

*****************START OF BARGAL Pirate Implemenation***************************/
BargalPirateDepartureProcess bargalDepartureTimeProcess=
    new BargalPirateDepartureProcess(bargalInterarrivalTime);
Platform[] bargalPirateMover = new Platform[numBargalPirates ];
for (int i = 0; i < numBargalPirates; i++) {
    bargalPirateMover[i] = new Platform("Pirate-Bargal" + i,
                                        pirateCampBargal,
                                        pirateMaxSpeed, typePirate);
}
System.out.println("Pirate: " + bargalPirateMover[0].paramString());
CookieCutterSensor[] bargalPirateSensor =
    new CookieCutterSensor[bargalPirateMover.length];
for (int i = 0; i < bargalPirateMover.length; i++) {
    bargalPirateSensor[i] =
        new CookieCutterSensor(bargalPirateMover[i],
                                pirateVisualSensorRange);
}
PirateMoverManager[] bargalPirateManager =
    new PirateMoverManager[bargalPirateMover.length];
for (int i = 0; i < bargalPirateMover.length; i++) {
    bargalPirateManager[i] =
        new PirateMoverManager(bargalPirateMover[i],
                                bargalPirateSensor[i],
                                pirateCampBargal,
                                bargalPiratePathGenerator,
                                attackDecision,
                                unsuccessfulAttackTime);
}
BargalPirateCamp bpc = new BargalPirateCamp(bargalPirateManager);
bargalDepartureTimeProcess.addSimEventListener(bpc);

*****************END OF BARGAL PIRATE IMPLEMENTATION***************************/
****************************START OF HAFUN PIRATE IMPLEMENTATION*******************/
HafunPirateDepartureProcess hafunDepartureTimeProcess=
    new HafunPirateDepartureProcess(hafunInterarrivalTime);
Platform[] hafunPirateMover = new Platform[numHafunPirates ];
for (int i = 0; i < numHafunPirates; i++) {
hafunPirateMover[i] = new Platform("Pirate-Hafun" + i,
    pirateCampHafun,
    pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + hafunPirateMover[0].paramString());

CookieCutterSensor[] hafunPirateSensor =
new CookieCutterSensor[hafunPirateMover.length];
for (int i = 0; i < hafunPirateMover.length; ++i) {
    hafunPirateSensor[i] =
        new CookieCutterSensor(hafunPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] hafunPirateManager =
    new PirateMoverManager[hafunPirateMover.length];
for (int i = 0; i < hafunPirateMover.length; ++i) {
    hafunPirateManager[i] =
        new PirateMoverManager(hafunPirateMover[i],
            hafunPirateSensor[i],
            pirateCampHafun,
            ioPiratePathGenerator,
            attackDecision,
            unsuccessfulAttackTime);
}

HafunPirateCamp hpc = new HafunPirateCamp(hafunPirateManager);
    hafunDepartureTimeProcess.addSimEventListener(hpc);

//*****************END OF HAFUN PIRATE IMPLEMENTATION*************************
//****************START OF BAYLA PIRATE IMPLEMENTATION*************************

BaylaPirateDepartureProcess baylaDepartureTimeProcess=
    new BaylaPirateDepartureProcess(baylaInterarrivalTime);

Platform[] baylaPirateMover = new Platform[ numBaylaPirates ];
for (int i = 0; i < numBaylaPirates; ++i) {
    baylaPirateMover[i] = new Platform("Pirate-Bayla" + i,
        pirateCampBayla,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + baylaPirateMover[0].paramString());

CookieCutterSensor[] baylaPirateSensor =
    new CookieCutterSensor[baylaPirateMover.length];
for (int i = 0; i < baylaPirateMover.length; ++i) {
    baylaPirateSensor[i] =
        new CookieCutterSensor(baylaPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] baylaPirateManager =
    new PirateMoverManager[baylaPirateMover.length];
for (int i = 0; i < baylaPirateMover.length; ++i) {
    baylaPirateManager[i] =
        new PirateMoverManager(baylaPirateMover[i],
            baylaPirateSensor[i],
            pirateCampBayla,
            ioPiratePathGenerator,
            attackDecision,
BaylaPirateCamp baypc = new BaylaPirateCamp(baylaPirateManager);
bayDepPartTimeProcess.addSimEventListener(baypc);

BaylaPirateCamp baypc = new BaylaPirateCamp(baylaPirateManager);
bayDepPartTimeProcess.addSimEventListener(baypc);

//******************END OF BAYLA PIRATE IMPLEMENTATION**************************/

EylPirateDepartureProcess eylDepPartTimeProcess=
    new EylPirateDepartureProcess(eylInterarrivalTime);

EylPirateDepartureProcess eylDepPartTimeProcess=
    new EylPirateDepartureProcess(eylInterarrivalTime);

Platform[] eylPirateMover = new Platform[ numEylPirates ];
for (int i = 0; i < numEylPirates; ++i) {
    eylPirateMover[i] = new Platform("Pirate-Eyl" + i,
        pirateCampEyl,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + eylPirateMover[0].paramString());

CookieCutterSensor[] eylPirateSensor =
    new CookieCutterSensor[eylPirateMover.length];
for (int i = 0; i < eylPirateMover.length; ++i) {
    eylPirateSensor[i] =
        new CookieCutterSensor(eylPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] eylPirateManager =
    new PirateMoverManager[eylPirateMover.length];
for (int i = 0; i < eylPirateMover.length; ++i) {
    eylPirateManager[i] =
        new PirateMoverManager(eylPirateMover[i],
            eylPirateSensor[i],
            pirateCampEyl,
            ioPiratePathGenerator,
            attackDecision,
            unsuccessfulAttackTime);
}

EylPirateCamp eylpc = new EylPirateCamp(eylPirateManager);
eylDepPartTimeProcess.addSimEventListener(eylpc);

EylPirateCamp eylpc = new EylPirateCamp(eylPirateManager);
eylDepPartTimeProcess.addSimEventListener(eylpc);

GaracadPirateDepartureProcess garacadDepPartTimeProcess=
    new GaracadPirateDepartureProcess(garacadInterarrivalTime);

GaracadPirateDepartureProcess garacadDepPartTimeProcess=
    new GaracadPirateDepartureProcess(garacadInterarrivalTime);

Platform[] garacadPirateMover = new Platform[ numGaracadPirates ];
for (int i = 0; i < numGaracadPirates; ++i) {
    garacadPirateMover[i] = new Platform("Pirate-Garacad" + i,
        pirateCampGaracad,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + garacadPirateMover[0].paramString());

CookieCutterSensor[] garacadPirateSensor =
    new CookieCutterSensor[garacadPirateMover.length];
for (int i = 0; i < garacadPirateMover.length; ++i) {
    garacadPirateSensor[i] =
        new CookieCutterSensor(garacadPirateMover[i],
            garacadPirateSensor.length);
new CookieCutterSensor(garacadPirateMover[i], pirateVisualSensorRange);

PirateMoverManager[] garacadPirateManager =
    new PirateMoverManager[garacadPirateMover.length];
for (int i = 0; i < garacadPirateMover.length; ++i) {
    garacadPirateManager[i] =
        new PirateMoverManager(garacadPirateMover[i],
            garacadPirateSensor[i],
            pirateCampGaracad,
            ioPiratePathGenerator,
            attackDecision,
            unsuccessfulAttackTime);
}

GaracadPirateCamp gpc = new GaracadPirateCamp(garacadPirateManager);
    garacadDepartureTimeProcess.addSimEventListener(gpc);

//*****************END OF GARACAD PIRATE IMPLEMENTATION***********************//

HobyoPirateDepartureProcess hobyoDepartureTimeProcess=
    new HobyoPirateDepartureProcess(hobyoInterarrivalTime);
Platform[] hobyoPirateMover = new Platform[ numHobyoPirates ];
for (int i = 0; i < numHobyoPirates; ++i) {
    hobyoPirateMover[i] = new Platform("Pirate-Hobyo" + i,
        pirateCampHobyo,
        pirateMaxSpeed, typePirate);
}

System.out.println("Pirate: " + hobyoPirateMover[0].paramString());

CookieCutterSensor[] hobyoPirateSensor =
    new CookieCutterSensor[hobyoPirateMover.length];
for (int i = 0; i < hobyoPirateMover.length; ++i) {
    hobyoPirateSensor[i] =
        new CookieCutterSensor(hobyoPirateMover[i],
            pirateVisualSensorRange);
}

PirateMoverManager[] hobyoPirateManager =
    new PirateMoverManager[hobyoPirateMover.length];
for (int i = 0; i < hobyoPirateMover.length; ++i) {
    hobyoPirateManager[i] =
        new PirateMoverManager(hobyoPirateMover[i],
            hobyoPirateSensor[i],
            hobyoPirateMover[i],
            hobyoPirateSensor[i],
            hobyoPirateMover[i],
            hobyoPirateSensor[i],
            hobyoPirateMover[i],
            hobyoPirateSensor[i],
            attackDecision,
            unsuccessfulAttackTime);
}

HobyoPirateCamp hobpc = new HobyoPirateCamp(hobyoPirateManager);
    hobyoDepartureTimeProcess.addSimEventListener(hobpc);

//*****************END OF HOBYO PIRATE IMPLEMENTATION***********************//

HarardherePirateDepartureProcess harardhereDepartureTimeProcess=
    new HarardherePirateDepartureProcess(harardhereInterarrivalTime);
Platform[] harpc = new Platform[ numHaradherePirates ];
for (int i = 0; i < numHaradherePirates; ++i) {
    haradherePirateMover[i] = new Platform("Pirate-Haradhere" + i,
        pirateCampHaradhere,
        pirateMaxSpeed, typePirate);
}
System.out.println("Pirate: " + haradherePirateMover[0].paramString());
CookieCutterSensor[] haradherePirateSensor =
    new CookieCutterSensor[haradherePirateMover.length];
for (int i = 0; i < haradherePirateMover.length; ++i) {
    haradherePirateSensor[i] =
        new CookieCutterSensor(haradherePirateMover[i],
            pirateVisualSensorRange);
}
PirateMoverManager[] haradherePirateManager =
    new PirateMoverManager[haradherePirateMover.length];
for (int i = 0; i < haradherePirateMover.length; ++i) {
    haradherePirateManager[i] =
        new PirateMoverManager( haradherePirateMover[i],
            haradherePirateSensor[i],
            pirateCampHaradhere,
            ioPiratePathGenerator,
            attackDecision,
            unsuccessfulAttackTime);
}
HaradherePirateCamp harpc = new HaradherePirateCamp(
    haradherePirateManager);
haradhereDepartureTimeProcess.addSimEventListener(harpc);

//*****************END OF HARARDHERE PIRATE IMPLEMENTATION******************//
//************************END OF IO PIRATES***********************************//
//*******************END OF PIRATE IMPLEMENTATION*****************************//
//*******************START OF NAVY IMPLEMENTATION*****************************//
PlatformType typeNavy = PlatformType.NAVY;

//**Navy Patrolling in Indian Ocean**/
RandomVariate[] navyPatrolBox1Generator = new RandomVariate[ 2 ];
navyPatrolBox1Generator[0] = RandomVariateFactory.getInstance("Uniform,"
    290.00,
    300.00 );
navyPatrolBox1Generator[1] = RandomVariateFactory.getInstance("Uniform,
    325.00,
    328.00 );

//Navy patrol points Box 2
RandomVariate[] navyPatrolBox2Generator = new RandomVariate[ 2 ];
navyPatrolBox2Generator[0] = RandomVariateFactory.getInstance("Uniform,
    326.00,
    336.00 );
navyPatrolBox2Generator[1] = RandomVariateFactory.getInstance("Uniform,
    335.00,
    338.00 );
RandomVariate[ ] navyPatrolBox3Generator = new RandomVariate[ 2 ];
navyPatrolBox3Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
360.00, 370.00 );
navyPatrolBox3Generator[1] = RandomVariateFactory.getINSTANCE("Uniform,
344.00, 347.00 );

RandomVariate[ ] navyPatrolBox4Generator = new RandomVariate[ 2 ];
navyPatrolBox4Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
407.00, 410.00 );
navyPatrolBox4Generator[1] = RandomVariateFactory.getINSTANCE("Uniform,
310.00, 320.00 );

RandomVariate[ ] navyPatrolBox5Generator = new RandomVariate[ 2 ];
navyPatrolBox5Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
408.00, 410.00 );
navyPatrolBox5Generator[1] = RandomVariateFactory.getINSTANCE("Uniform,
270.00, 280.00 );

RandomVariate[ ] navyPatrolBox6Generator = new RandomVariate[ 2 ];
navyPatrolBox6Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
395.00, 398.00 );
navyPatrolBox6Generator[1] = RandomVariateFactory.getINSTANCE("Uniform,
238.00, 248.00 );

RandomVariate[ ] navyPatrolBox7Generator = new RandomVariate[ 2 ];
navyPatrolBox7Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
363.00, 366.00 );
navyPatrolBox7Generator[1] = RandomVariateFactory.getINSTANCE("Uniform,
180.00, 190.00 );

RandomVariate[ ] navyPatrolBox8Generator = new RandomVariate[ 2 ];
navyPatrolBox8Generator[0] = RandomVariateFactory.getINSTANCE("Uniform,
342.00, 345.00 );
navyPatrolBox8Generator[1] = RandomVariateFactory.getInstance("Uniform",
150.00,
160.00);

//Navy patrol points in IO Box 9
RandomVariate[] navyPatrolBox9Generator = new RandomVariate[2];
navyPatrolBox9Generator[0] = RandomVariateFactory.getInstance("Uniform",
322.00,
325.00);
navyPatrolBox9Generator[1] = RandomVariateFactory.getInstance("Uniform",
96.00,
106.00);

//Navy patrol points Box 10
RandomVariate[] navyPatrolBox10Generator = new RandomVariate[2];
navyPatrolBox10Generator[0] = RandomVariateFactory.getInstance("Uniform",
301.00,
304.00);
navyPatrolBox10Generator[1] = RandomVariateFactory.getInstance("Uniform",
71.00,
81.00);

Platform[] ioNavyMover = new Platform[numIoNavyShips];
ioNavyMover[0] = new Platform("IO Navy-6," initialLocationNavyPB4,
navyMaxSpeed, typeNavy);
ioNavyMover[1] = new Platform("Navy-7," initialLocationNavyPB5,
navyMaxSpeed, typeNavy);
ioNavyMover[2] = new Platform("Navy-8," initialLocationNavyPB6,
navyMaxSpeed, typeNavy);
navyMaxSpeed, typeNavy);
ioNavyMover[4] = new Platform("Navy-10," initialLocationNavyPB8,
navyMaxSpeed, typeNavy);
ioNavyMover[5] = new Platform("Navy-11," initialLocationNavyPB9,
navyMaxSpeed, typeNavy);
ioNavyMover[6] = new Platform("Navy-12," initialLocationNavyPB10,
navyMaxSpeed, typeNavy);

CookieCutterSensor[] ioNavySensor =
new CookieCutterSensor[numIoNavyShips];
ioNavySensor[0] = new CookieCutterSensor(ioNavyMover[0],
navySurfaceRadarRange);
ioNavySensor[1] = new CookieCutterSensor(ioNavyMover[1],
navySurfaceRadarRange);
navySurfaceRadarRange);
navySurfaceRadarRange);
navySurfaceRadarRange);
navySurfaceRadarRange);
navySurfaceRadarRange);

NavyShipMoverManager[] ioNavyManager =
new NavyShipMoverManager(numIoNavyShips);

ioNavyManager[0] = new NavyShipMoverManager(ioNavyMover[0],
ioNavySensor[0], initialLocationNavyPB4,
navyPatrolBox4Generator, navyMaxSpeed);

ioNavyManager[1] = new NavyShipMoverManager(ioNavyMover[1],
ioNavySensor[1], initialLocationNavyPB5,
navyPatrolBox5Generator, navyMaxSpeed);

ioNavySensor[2], initialLocationNavyPB6,
navyPatrolBox6Generator, navyMaxSpeed);

ioNavyManager[3] = new NavyShipMoverManager(ioNavyMover[3],
ioNavySensor[3], initialLocationNavyPB7,
navyPatrolBox7Generator, navyMaxSpeed);

ioNavyManager[4] = new NavyShipMoverManager(ioNavyMover[4],
ioNavySensor[4], initialLocationNavyPB8,
navyPatrolBox8Generator, navyMaxSpeed);

ioNavyManager[5] = new NavyShipMoverManager(ioNavyMover[5],
ioNavySensor[5], initialLocationNavyPB9,
navyPatrolBox9Generator, navyMaxSpeed);

ioNavyManager[6] = new NavyShipMoverManager(ioNavyMover[6],
ioNavySensor[6], initialLocationNavyPB10,
navyPatrolBox10Generator, navyMaxSpeed);

System.out.println(“ioNavyManager Length: “ +
ioNavyManager.length);

//**Navy Patrols in the Gulf of Aden**/
Platform[] goaNavyMover = new Platform[numGoaNavyShips];
goaNavyMover[0] = new Platform(“IO Navy-1,” initialLocationNavyPB1,
navyMaxSpeed, typeNavy);
goaNavyMover[1] = new Platform(“Navy-2,” initialLocationNavyPB2,
navyMaxSpeed, typeNavy);
goaNavyMover[2] = new Platform(“Navy-3,” initialLocationNavyPB3,
navyMaxSpeed, typeNavy);

CookieCutterSensor[] goaNavySensor =
new CookieCutterSensor[numGoaNavyShips];
goaNavySensor[0] = new CookieCutterSensor(goaNavyMover[0],
navySurfaceRadarRange);
goaNavySensor[1] = new CookieCutterSensor(goaNavyMover[1],
navySurfaceRadarRange);
navySurfaceRadarRange);

NavyShipMoverManager[] goaNavyManager =
new NavyShipMoverManager[numGoaNavyShips];
goaNavyManager[0] = new NavyShipMoverManager(goaNavyMover[0],
goaNavySensor[0], initialLocationNavyPB1,
navyPatrolBox1Generator, navyMaxSpeed);
goaNavyManager[1] = new NavyShipMoverManager(goaNavyMover[1],
goaNavySensor[1], initialLocationNavyPB2,
navyPatrolBox2Generator, navyMaxSpeed);
goaNavyManager[2] = new NavyShipMoverManager(goaNavyMover[2],
goaNavySensor[2], initialLocationNavyPB3,
navyPatrolBox3Generator, navyMaxSpeed);

System.out.println(“goaNavyManager length: “ +
goaNavyManager.length);

//%%%%%%%%%%%%%%%%%%END OF NAVY IMPLEMENTATION%%%%%%%%%%%%%%%%%%%//
//**********START OF MERCHANT SHIP IMPLEMENTATION***********************//
PlatformType typeMerchant = PlatformType.MERCHANT;
//Creates instance of ArrivalProcess w/ interarrival time passed in
SuezToMaldivesMerchantDepartureProcess stmDepartureTimeProcess = new
      SuezToMaldivesMerchantDepartureProcess(
          strmMerchantInterarrivalTime);

//**********START OF SUEZ TO MALDIVES MERCHANT SHIP IMPLEMENTATION*************//
RandomVariate[] suezToMaldivesMerchantPathGenerator =
      new RandomVariate[8];
   suezToMaldivesMerchantPathGenerator[0] = RandomVariateFactory.GetInstance(
      "Uniform,"
      minLatSuezToMaldivesMerchantWaypoint1,
      maxLatSuezToMaldivesMerchantWaypoint1);
      "Uniform,"
      minLonSuezToMaldivesMerchantWaypoint1,
      maxLonSuezToMaldivesMerchantWaypoint1);
      "Uniform,"
      minLatSuezToMaldivesMerchantWaypoint2,
      maxLatSuezToMaldivesMerchantWaypoint2);
      "Uniform,"
      minLonSuezToMaldivesMerchantWaypoint2,
      maxLonSuezToMaldivesMerchantWaypoint2);
      "Uniform,"
      minLatSuezToMaldivesMerchantWaypoint3,
      maxLatSuezToMaldivesMerchantWaypoint3);
      "Uniform,"
      minLonSuezToMaldivesMerchantWaypoint3,
      maxLonSuezToMaldivesMerchantWaypoint3);
      "Uniform,"
      minLatSuezToMaldivesMerchantWaypoint4,
      maxLatSuezToMaldivesMerchantWaypoint4);
      "Uniform,"
      minLonSuezToMaldivesMerchantWaypoint4,
      maxLonSuezToMaldivesMerchantWaypoint4);
Platform [] suezToMaldivesMerchantMover =
      new Platform[numSuezToMaldivesMerchants];
for (int i = 0; i < suezToMaldivesMerchantMover.length; ++i) {
   suezToMaldivesMerchantMover[i] =
      new Platform("Merchant: SuezToMaldives " + i,
                     initialLocationMerchantSuezToMaldives,
                     merchantMaxSpeed, typeMerchant);
}
CookieCutterSensor[] suezToMaldivesMerchantSensor =
      new CookieCutterSensor[suezToMaldivesMerchantMover.length];
for (int i = 0; i < suezToMaldivesMerchantMover.length; ++i) {
   suezToMaldivesMerchantSensor [i] =
      new CookieCutterSensor(suezToMaldivesMerchantMover[i],

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 );

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,
maxLonSuezToOmanMerchantWaypoint4);
Platform[] suezToOmanMerchantMover =
    new Platform[numSuezToOmanMerchants];
for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
{
suezToOmanMerchantMover[i] =
    new Platform("Merchant: SuezToOman " + i,
    initialLocationMerchantSuezToOman,
    merchantMaxSpeed, typeMerchant);
}

CookieCutterSensor[] suezToOmanMerchantSensor =
    new CookieCutterSensor[suezToOmanMerchantMover.length];
for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
{
suezToOmanMerchantSensor[i] =
    new CookieCutterSensor(suezToOmanMerchantMover[i],
    merchantSurfaceRadarRange);
}

MerchantShipMoverManager[] suezToOmanMerchantManager =
    new MerchantShipMoverManager[suezToOmanMerchantMover.length];
for (int i = 0; i < suezToOmanMerchantMover.length; ++i)
{
suezToOmanMerchantManager[i] =
    new MerchantShipMoverManager(
    suezToOmanMerchantMover[i],
    suezToOmanMerchantSensor[i],
    initialLocationMerchantSuezToOman,
    suezToOmanMerchantPathGenerator);
}
SuezToOmanOriginPort sto = new SuezToOmanOriginPort( suezToOmanMerchantManager );
stoDepartureTimeProcess.addSimEventListener( sto );

//************END OF SUEZ TO OMAN MERCHANT IMPLEMENTATION******************//
//***********START OF MALDIVES TO SUEZ MERCHANT SHIP IMPLEMENTATION***********//
MaldivesToSuezMerchantDepartureProcess mtsDepartureTimeProcess = new MaldivesToSuezMerchantDepartureProcess(
    mtsMerchantInterarrivalTime );
RandomVariate[] maldiveToSuezMerchantPathGenerator =
    new RandomVariate[ 8 ];
maldiveToSuezMerchantPathGenerator[0] = RandomVariateFactory.getInstance(“Uniform,”
    minLatMaldivesToSuezMerchantWaypoint1,
    maxLatMaldivesToSuezMerchantWaypoint1 );
    minLonMaldivesToSuezMerchantWaypoint1,
    maxLonMaldivesToSuezMerchantWaypoint1 );
    minLatMaldivesToSuezMerchantWaypoint2,
    maxLatMaldivesToSuezMerchantWaypoint2 );
    minLonMaldivesToSuezMerchantWaypoint2,
minLatMaldivesToSuezMerchantWaypoint3, 
maxLatMaldivesToSuezMerchantWaypoint3 );

minLonMaldivesToSuezMerchantWaypoint3, 
maxLonMaldivesToSuezMerchantWaypoint3 );

minLatMaldivesToSuezMerchantWaypoint4, 
maxLatMaldivesToSuezMerchantWaypoint4 );

minLonMaldivesToSuezMerchantWaypoint4, 
maxLonMaldivesToSuezMerchantWaypoint4 );

Platform [] maldivesToSuezMerchantMover =
new Platform[numMaldivesToSuezMerchants];

for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
{
    maldivesToSuezMerchantMover[i] =
        new Platform("Merchant: MaldivesToSuez " + i, 
            initialLocationMerchantMaldivesToSuez, 
            merchantMaxSpeed, typeMerchant );
}

CookieCutterSensor[] maldivesToSuezMerchantSensor =
new CookieCutterSensor[maldivesToSuezMerchantMover.length];

for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
{
    maldivesToSuezMerchantSensor [i] =
        new CookieCutterSensor(maldivesToSuezMerchantMover[i], 
            merchantSurfaceRadarRange);
}

MerchantShipMoverManager [] maldivesToSuezMerchantManager =
new MerchantShipMoverManager[maldivesToSuezMerchantMover.length];

for (int i = 0; i < maldivesToSuezMerchantMover.length; ++i)
{
    maldivesToSuezMerchantManager[i] =
        new MerchantShipMoverManager ( 
            maldivesToSuezMerchantMover[i], 
            maldivesToSuezMerchantSensor[i], 
            initialLocationMerchantMaldivesToSuez, 
            maldivesToSuezMerchantPathGenerator );
}

MaldivesToSuezOriginPort mts = new MaldivesToSuezOriginPort( maldivesToSuezMerchantManager );
mtsDepartureTimeProcess.addSimEventListener( mts );

//************END OF MALDIVES TO SUEZ MERCHANT IMPLEMENTATION******************/
//***********START OF MALDIVES TO OMAN MERCHANT SHIP IMPLEMENTATION***********/
MaldivesToOmanMerchantDepartureProcess mtoDepartureTimeProcess = new
RandomVariate[] maldivesToOmanMerchantPathGenerator =
    new RandomVariate[8];
maldivesToOmanMerchantPathGenerator[0] = RandomVariateFactory.getInstance("Uniform,
    minLatMaldivesToOmanMerchantWaypoint1,
    maxLatMaldivesToOmanMerchantWaypoint1 );
maldivesToOmanMerchantPathGenerator[1] = RandomVariateFactory.getInstance("Uniform,
    minLonMaldivesToOmanMerchantWaypoint1,
    maxLonMaldivesToOmanMerchantWaypoint1 );
maldivesToOmanMerchantPathGenerator[2] = RandomVariateFactory.getInstance("Uniform,
    minLatMaldivesToOmanMerchantWaypoint2,
    maxLatMaldivesToOmanMerchantWaypoint2 );
maldivesToOmanMerchantPathGenerator[3] = RandomVariateFactory.getInstance("Uniform,
    minLonMaldivesToOmanMerchantWaypoint2,
    maxLonMaldivesToOmanMerchantWaypoint2 );
maldivesToOmanMerchantPathGenerator[4] = RandomVariateFactory.getInstance("Uniform,
    minLatMaldivesToOmanMerchantWaypoint3,
    maxLatMaldivesToOmanMerchantWaypoint3 );
maldivesToOmanMerchantPathGenerator[5] = RandomVariateFactory.getInstance("Uniform,
    minLonMaldivesToOmanMerchantWaypoint3,
    maxLonMaldivesToOmanMerchantWaypoint3 );
maldivesToOmanMerchantPathGenerator[6] = RandomVariateFactory.getInstance("Uniform,
    minLatMaldivesToOmanMerchantWaypoint4,
    maxLatMaldivesToOmanMerchantWaypoint4 );
maldivesToOmanMerchantPathGenerator[7] = RandomVariateFactory.getInstance("Uniform,
    minLonMaldivesToOmanMerchantWaypoint4,
    maxLonMaldivesToOmanMerchantWaypoint4 );
Platform[] maldivesToOmanMerchantMover =
    new Platform[numMaldivesToOmanMerchants];
for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
    {
        maldivesToOmanMerchantMover[i] =
            new Platform("Merchant: MaldivesToOman “ + i,
                initialLocationMerchantMaldivesToOman,
                merchantMaxSpeed, typeMerchant );
    }
CookieCutterSensor[] maldivesToOmanMerchantSensor =
    new CookieCutterSensor[maldivesToOmanMerchantMover.length];
for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
    {
        maldivesToOmanMerchantSensor[i] =
            new CookieCutterSensor(maldivesToOmanMerchantMover[i],
                merchantSurfaceRadarRange);
    }
MerchantShipMoverManager[] maldivesToOmanMerchantManager =
    new MerchantShipMoverManager[maldivesToOmanMerchantMover.length];
for (int i = 0; i < maldivesToOmanMerchantMover.length; ++i)
    {
        maldivesToOmanMerchantManager[i] =

157
new MerchantShipMoverManager {
    maldivesToOmanMerchantMover[i],
    maldivesToOmanMerchantSensor[i],
    initialLocationMerchantMaldivesToOman,
    maldivesToOmanMerchantPathGenerator );
}

MaldivesToOmanOriginPort mto = new MaldivesToOmanOriginPort( maldivesToOmanMerchantManager );
mtoDepartureTimeProcess.addSimEventListener( mto );

//************END OF MALDIVES TO OMAN MERCHANT IMPLEMENTATION*****************//
//***********START OF OMAN TO MALDIVES MERCHANT SHIP IMPLEMENTATION***********//
//Creates Instance of ArrivalProcess w/ interarrival time passed in
OmanToMaldivesMerchantDepartureProcess otmDepartureTimeProcess = new OmanToMaldivesMerchantDepartureProcess( otmMerchantInterarrivalTime );

RandomVariate[] omanToMaldivesMerchantPathGenerator =
    new RandomVariate[8 ];
omanToMaldivesMerchantPathGenerator[0] = RandomVariateFactory.getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint1,
maxLatOmanToMaldivesMerchantWaypoint1 );
omanToMaldivesMerchantPathGenerator[1] = RandomVariateFactory.getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint1,
maxLonOmanToMaldivesMerchantWaypoint1 );
omanToMaldivesMerchantPathGenerator[2] = RandomVariateFactory.getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint2,
maxLatOmanToMaldivesMerchantWaypoint2 );
omanToMaldivesMerchantPathGenerator[3] = RandomVariateFactory.getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint2,
maxLonOmanToMaldivesMerchantWaypoint2 );
omanToMaldivesMerchantPathGenerator[4] = RandomVariateFactory.getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint3,
maxLatOmanToMaldivesMerchantWaypoint3 );
omanToMaldivesMerchantPathGenerator[5] = RandomVariateFactory.getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint3,
maxLonOmanToMaldivesMerchantWaypoint3 );
omanToMaldivesMerchantPathGenerator[6] = RandomVariateFactory.getInstance( "Uniform," minLatOmanToMaldivesMerchantWaypoint4,
maxLatOmanToMaldivesMerchantWaypoint4 );
omanToMaldivesMerchantPathGenerator[7] = RandomVariateFactory.getInstance( "Uniform," minLonOmanToMaldivesMerchantWaypoint4,
maxLonOmanToMaldivesMerchantWaypoint4 );

Platform[] omanToMaldivesMerchantMover =
    new Platform[numOmanToMaldivesMerchants];
for ( int i = 0; i < omanToMaldivesMerchantMover.length; ++i )
    omanToMaldivesMerchantMover[i] =
        new Platform( "Merchant: OmanToMaldives " + i,
        initialLocationMerchantOmanToMaldives, merchantMaxSpeed, typeMerchant );

CookieCutterSensor[] omanToMaldivesMerchantSensor =
    new CookieCutterSensor[omanToMaldivesMerchantMover.length];
for ( int i = 0; i < omanToMaldivesMerchantMover.length; ++i )
    {
omanToMaldivesMerchantSensor [i] =
    new CookieCutterSensor(omanToMaldivesMerchantMover[i],
        merchantSurfaceRadarRange);
    }

MerchantShipMoverManager [] omanToMaldivesMerchantManager =
    new MerchantShipMoverManager[omanToMaldivesMerchantMover.length];
for (int i = 0; i < omanToMaldivesMerchantMover.length; ++i)
{
    omanToMaldivesMerchantManager[i] =
        new MerchantShipMoverManager ( 
            omanToMaldivesMerchantMover[i],
            omanToMaldivesMerchantSensor[i],
            initialLocationMerchantOmanToMaldives,
            omanToMaldivesMerchantPathGenerator );
}

OmanToMaldivesOriginPort otm = new
    OmanToMaldivesOriginPort( omanToMaldivesMerchantManager );
    otmDepartureTimeProcess.addSimEventListener( otm );

//************END OF OMAN TO MALDIVES MERCHANT IMPLEMENTATION*****************
//***********START OF OMAN TO SUEZ MERCHANT SHIP IMPLEMENTATION***************

omanToSuezMerchantDepartureProcess otsDepartureTimeProcess = new
    OmanToSuezMerchantDepartureProcess(
        otsMerchantInterarrivalTime );

omanToSuezMerchantPathGenerator =
    new RandomVariate[ 8 ];
omanToSuezMerchantPathGenerator[0] = RandomVariateFactory.getInstance( 
    "Uniform," 
    minLatOmanToSuezMerchantWaypoint1, 
    maxLatOmanToSuezMerchantWaypoint1 );
omanToSuezMerchantPathGenerator[1] = RandomVariateFactory.getInstance( 
    "Uniform," 
    minLonOmanToSuezMerchantWaypoint1, 
    maxLonOmanToSuezMerchantWaypoint1 );
    "Uniform," 
    minLatOmanToSuezMerchantWaypoint2, 
    maxLatOmanToSuezMerchantWaypoint2 );
    "Uniform," 
    minLonOmanToSuezMerchantWaypoint2, 
    maxLonOmanToSuezMerchantWaypoint2 );
    "Uniform," 
    minLatOmanToSuezMerchantWaypoint3, 
    maxLatOmanToSuezMerchantWaypoint3 );
    "Uniform," 
    minLonOmanToSuezMerchantWaypoint3, 
    maxLonOmanToSuezMerchantWaypoint3 );
    "Uniform," 
    minLatOmanToSuezMerchantWaypoint4, 
    maxLatOmanToSuezMerchantWaypoint4 );
omanToSuezMerchantPathGenerator[7] = RandomVariateFactory.getInstance(
“Uniform,”

Platform[] omanToSuezMerchantMover =
    new Platform[numOmanToSuezMerchants];
for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
{
    omanToSuezMerchantMover[i] =
        new Platform("Merchant: OmanToSuez " + i,
                        initialLocationMerchantOmanToSuez,
                        merchantMaxSpeed, typeMerchant);
}

CookieCutterSensor[] omanToSuezMerchantSensor =
    new CookieCutterSensor[omanToSuezMerchantMover.length];
for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
{
    omanToSuezMerchantSensor[i] =
        new CookieCutterSensor(omanToSuezMerchantMover[i],
                            merchantSurfaceRadarRange);
}

MerchantShipMoverManager[] omanToSuezMerchantManager =
    new MerchantShipMoverManager[omanToSuezMerchantMover.length];
for (int i = 0; i < omanToSuezMerchantMover.length; ++i)
{
    omanToSuezMerchantManager[i] =
        new MerchantShipMoverManager (omanToSuezMerchantMover[i],
                                        omanToSuezMerchantSensor[i],
                                        initialLocationMerchantOmanToSuez,
                                        omanToSuezMerchantPathGenerator);
}

OmanToSuezOriginPort ots = new OmanToSuezOriginPort(omanToSuezMerchantManager);
otsDepartureTimeProcess.addSimEventListener(ots);

//************END OF OMAN TO SUEZ MERCHANT IMPLEMENTATION*********************//
//****************END OF MERCHANT SHIP IMPLEMENTATION*************************//
//*****************START OF ADUDICATOR IMPLEMENTATION*************************//
Adjudicator adj = new Adjudicator(successOrFailGenerator);

//***************Referees, Mediators, and EventListeners*********************//
//Create a SensorMoverReferee
SensorMoverReferee smr = new SensorMoverReferee();
//Add a mediator for each senor and mediator
smr.addMediator( CookieCutterSensor.class, Platform.class,
                new CookieCutterMediator() );
adj.addSimEventListener( smr );
for ( int i = 0 ; i < elaayoPirateMover.length ; ++i )
{
    elaayoPirateMover[i].addSimEventListener( smr );
    elaayoPirateManager[i].addSimEventListener( smr );
    elaayoPirateSensor[i].addSimEventListener( elaayoPirateManager[i] );
}
for ( int i = 0 ; i < qandalaPirateMover.length ; ++i )
{
    qandalaPirateMover[i].addSimEventListener( smr );
    qandalaPirateManager[i].addSimEventListener( smr );
    qandalaPirateSensor[i].addSimEventListener( smr );
    qandalaPirateSensor[i].addSimEventListener(qandalaPirateManager[i]);
}

for ( int i = 0 ; i < aluulaPirateMover.length ; ++i )
{
    aluulaPirateMover[i].addSimEventListener( smr );
    aluulaPirateManager[i].addSimEventListener( smr );
    aluulaPirateSensor[i].addSimEventListener( smr );
    aluulaPirateSensor[i].addSimEventListener( aluulaPirateManager[i] );
}

for ( int i = 0 ; i < bargalPirateMover.length ; ++i )
{
    bargalPirateMover[i].addSimEventListener( smr );
    bargalPirateManager[i].addSimEventListener( smr );
    bargalPirateSensor[i].addSimEventListener( smr );
    bargalPirateSensor[i].addSimEventListener( bargalPirateManager[i] );
}

for ( int i = 0 ; i < hafunPirateMover.length ; ++i )
{
    hafunPirateMover[i].addSimEventListener( smr );
    hafunPirateManager[i].addSimEventListener( smr );
    hafunPirateSensor[i].addSimEventListener( smr );
    hafunPirateSensor[i].addSimEventListener( hafunPirateManager[i] );
}

for ( int i = 0 ; i < baylaPirateMover.length ; ++i )
{
    baylaPirateMover[i].addSimEventListener( smr );
    baylaPirateManager[i].addSimEventListener( smr );
    baylaPirateSensor[i].addSimEventListener( smr );
    baylaPirateSensor[i].addSimEventListener( baylaPirateManager[i] );
}

for ( int i = 0 ; i < eylPirateMover.length ; ++i )
{
    eylPirateMover[i].addSimEventListener( smr );
    eylPirateManager[i].addSimEventListener( smr );
    eylPirateSensor[i].addSimEventListener( smr );
    eylPirateSensor[i].addSimEventListener( eylPirateManager[i] );
}

for ( int i = 0 ; i < garacadPirateMover.length ; ++i )
{
    garacadPirateMover[i].addSimEventListener( smr );
    garacadPirateManager[i].addSimEventListener( smr );
    garacadPirateSensor[i].addSimEventListener( smr );
    garacadPirateSensor[i].addSimEventListener(garacadPirateManager[i]);
}

for ( int i = 0 ; i < hobyoPirateMover.length ; ++i )
{
hobyoPirateMover[i].addSimEventListener( smr );
hobyoPirateManager[i].addSimEventListener( smr );
hobyoPirateSensor[i].addSimEventListener( smr );
hobyoPirateSensor[i].addSimEventListener( hobyoPirateManager[i] );
}
for ( int i = 0 ; i < harardherePirateMover.length ; ++i )
{
    harardherePirateMover[i].addSimEventListener( smr );
    harardherePirateManager[i].addSimEventListener( smr );
    harardherePirateSensor[i].addSimEventListener( smr );
    harardherePirateSensor[i].addSimEventListener( harardherePirateManager[i] );
}
for ( int i = 0 ; i < goaNavyMover.length ; ++i )
{
    goaNavyMover[i].addSimEventListener( smr );
    goaNavyManager[i].addSimEventListener( smr );
    goaNavySensor[i].addSimEventListener( smr );
    goaNavySensor[i].addSimEventListener( goaNavyManager[i] );
}
for ( int i = 0 ; i < ioNavyMover.length ; ++i )
{
    ioNavyMover[i].addSimEventListener( smr );
    ioNavyManager[i].addSimEventListener( smr );
    ioNavySensor[i].addSimEventListener( smr );
    ioNavySensor[i].addSimEventListener( ioNavyManager[i] );
}
for ( int i = 0 ; i < suezToOmanMerchantMover.length ; ++i )
{
    suezToOmanMerchantMover[i].addSimEventListener( smr );
    suezToOmanMerchantManager[i].addSimEventListener( smr );
    suezToOmanMerchantSensor[i].addSimEventListener( smr );
    suezToOmanMerchantSensor[i].addSimEventListener( suezToOmanMerchantManager[i] );
}
for ( int i = 0 ; i < omanToSuezMerchantMover.length ; ++i )
{
    omanToSuezMerchantMover[i].addSimEventListener( smr );
    omanToSuezMerchantManager[i].addSimEventListener( smr );
    omanToSuezMerchantSensor[i].addSimEventListener( smr );
    omanToSuezMerchantSensor[i].addSimEventListener( omanToSuezMerchantManager[i] );
}
for ( int i = 0 ; i < omanToMaldivesMerchantMover.length ; ++i )
{
    omanToMaldivesMerchantMover[i].addSimEventListener( smr );
    omanToMaldivesMerchantManager[i].addSimEventListener( smr );
    omanToMaldivesMerchantSensor[i].addSimEventListener( smr );
    omanToMaldivesMerchantSensor[i].addSimEventListener( omanToSuezMerchantManager[i] );
}
omanToMaldivesMerchantManager[i].addSimEventListener( smr );
omanToMaldivesMerchantSensor[i].addSimEventListener( smr );
omanToMaldivesMerchantSensor[i].addSimEventListener( omanToMaldivesMerchantManager[i] );
}
for ( int i = 0 ; i < maldivesToSuezMerchantMover.length ; ++i )
{
maldivesToSuezMerchantMover[i].addSimEventListener( smr );
maldivesToSuezMerchantManager[i].addSimEventListener( smr );
maldivesToSuezMerchantSensor[i].addSimEventListener( smr );
maldivesToSuezMerchantSensor[i].addSimEventListener( maldivesToSuezMerchantManager[i] );
}
for ( int i = 0 ; i < maldivesToOmanMerchantMover.length ; ++i )
{
maldivesToOmanMerchantMover[i].addSimEventListener( smr );
maldivesToOmanMerchantManager[i].addSimEventListener( smr );
maldivesToOmanMerchantSensor[i].addSimEventListener( smr );
maldivesToOmanMerchantSensor[i].addSimEventListener( maldivesToOmanMerchantManager[i] );
}
for ( int i = 0 ; i < eylPirateManager.length ; ++i )
{
decision.connect( eylPirateManager[i], adj);
}
for ( int i = 0 ; i < qandalaPirateManager.length ; ++i )
{
decision.connect( qandalaPirateManager[i], adj);
}
for ( int i = 0 ; i < aluulaPirateManager.length ; ++i )
{
decision.connect( aluulaPirateManager[i], adj);
}
for ( int i = 0 ; i < bargalPirateManager.length ; ++i )
{
decision.connect( bargalPirateManager[i], adj);
}
for ( int i = 0 ; i < hafunPirateManager.length ; ++i )
{
decision.connect( hafunPirateManager[i], adj);
}
for ( int i = 0 ; i < baylaPirateManager.length ; ++i )
{
decision.connect( baylaPirateManager[i], adj);
}
decision.connect( eylPirateManager[i], adj);

for(int i = 0; i < garacadPirateManager.length; ++i)
{
    decision.connect( garacadPirateManager[i], adj);
}

for(int i = 0; i < hobyoPirateManager.length; ++i)
{
    decision.connect( hobyoPirateManager[i], adj);
}

for(int i = 0; i < harardherePirateManager.length; ++i)
{
    decision.connect( harardherePirateManager[i], adj);
}

for(int i = 0; i < elaayoPirateManager.length; ++i)
{
    for(int j = 0; j < goaNavyManager.length; ++j)
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                     elaayoPirateManager[i] );
    }
    for(int i = 0; i < elaayoPirateManager.length; ++i)
    {
        for(int j = 0; j < ioNavyManager.length; ++j)
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          elaayoPirateManager[i] );
        }
    }
    for(int i = 0; i < qandalaPirateManager.length; ++i)
    {
        for(int j = 0; j < goaNavyManager.length; ++j)
        {
            signalPiarteAdapter.connect( goaNavyManager[j],
                                          qandalaPirateManager[i] );
        }
    }
    for(int i = 0; i < qandalaPirateManager.length; ++i)
    {
        for(int j = 0; j < ioNavyManager.length; ++j)
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          qandalaPirateManager[i] );
        }
    }
    for(int i = 0; i < aluulaPirateManager.length; ++i)
    {
        for(int j = 0; j < ioNavyManager.length; ++j)
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          aluulaPirateManager[i] );
        }
    }
}

//**Allows Navy vessels to signal pirates when detections occur**/
Adapter signalPiarteAdapter = new Adapter( "SignalPirate,"
                                         "DetectedByNavy");
for( int i = 0; i < elaayoPirateManager.length ; ++i )
{
    for( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                     elaayoPirateManager[i] );
    }
    for( int i = 0 ; i < elaayoPirateManager.length ; ++i )
    {
        for( int j = 0 ; j < ioNavyManager.length ; ++j )
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          elaayoPirateManager[i] );
        }
    }
    for( int i = 0 ; i < qandalaPirateManager.length ; ++i )
    {
        for( int j = 0 ; j < goaNavyManager.length ; ++j )
        {
            signalPiarteAdapter.connect( goaNavyManager[j],
                                          qandalaPirateManager[i] );
        }
    }
    for( int i = 0 ; i < qandalaPirateManager.length ; ++i )
    {
        for( int j = 0 ; j < ioNavyManager.length ; ++j )
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          qandalaPirateManager[i] );
        }
    }
    for( int i = 0 ; i < aluulaPirateManager.length ; ++i )
    {
        for( int j = 0 ; j < ioNavyManager.length ; ++j )
        {
            signalPiarteAdapter.connect( ioNavyManager[j],
                                          aluulaPirateManager[i] );
        }
    }
}
aluulaPirateManager[i] );
}

for ( int i = 0 ; i < aluulaPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                               aluulaPirateManager[i] );
    }
}

for ( int i = 0 ; i < bargalPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                   bargalPirateManager[i] );
    }
}

for ( int i = 0 ; i < bargalPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                                   bargalPirateManager[i] );
    }
}

for ( int i = 0 ; i < hafunPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                   hafunPirateManager[i] );
    }
}

for ( int i = 0 ; i < hafunPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                                   hafunPirateManager[i] );
    }
}

for ( int i = 0 ; i < baylaPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                   baylaPirateManager[i] );
    }
}

for ( int i = 0 ; i < baylaPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
for ( int i = 0 ; i < eylPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                eylPirateManager[i] );
    }
}

for ( int i = 0 ; i < garacadPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                garacadPirateManager[i] );
    }
}

for ( int i = 0 ; i < hobyoPirateManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( goaNavyManager[j],
                                hobyoPirateManager[i] );
    }
}

for ( int i = 0 ; i < harardherePirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                                harardherePirateManager[i] );
    }
}

for ( int i = 0 ; i < harardherePirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                                harardherePirateManager[i] );
    }
}
{ for ( int j = 0 ; j < goaNavyManager.length ; ++j )
    signalPiarteAdapter.connect( goaNavyManager[j],
                           harardherePirateManager[i] );
}

for ( int i = 0 ; i < harardherePirateManager.length ; ++i )
{
    for ( int j = 0 ; j < ioNavyManager.length ; ++j )
    {
        signalPiarteAdapter.connect( ioNavyManager[j],
                               harardherePirateManager[i] );
    }
}

// Allows Merchants to send distress call to Navy
Adapter merchantDistressAdapter = new Adapter( "RadioNavy,"
                       "RevDistressCall" );

for ( int i = 0 ; i < suezToOmanMerchantManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyMover.length ; ++j )
    {
        merchantDistressAdapter.connect( suezToOmanMerchantManager[i],
                                            goaNavyManager[j] );
    }
}

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

for ( int i = 0 ; i < suezToMaldivesMerchantManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyMover.length ; ++j )
    {
        merchantDistressAdapter.connect( suezToMaldivesMerchantManager[i],
                                            goaNavyManager[j] );
    }
}

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

for ( int i = 0 ; i < omanToSuezMerchantManager.length ; ++i )

for ( int j = 0 ; j < goaNavyMover.length ; ++j )
{
    merchantDistressAdapter.connect( omanToSuezMerchantManager[i],
goaNavyManager[j] );
}

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

for ( int i = 0 ; i < omanToMaldivesMerchantManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyMover.length ; ++j )
    {
        merchantDistressAdapter.connect( omanToMaldivesMerchantManager[i],goaNavyManager[j] );
    }
}

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

for ( int i = 0 ; i < maldivesToSuezMerchantManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyMover.length ; ++j )
    {
        merchantDistressAdapter.connect( maldivesToSuezMerchantManager[i], goaNavyManager[j] );
    }
}

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

for ( int i = 0 ; i < maldivesToOmanMerchantManager.length ; ++i )
{
    for ( int j = 0 ; j < goaNavyMover.length ; ++j )
    {
        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 =
    new SimpleStatsTally("numberDepartedIO");

new SimpleStatsTally("numberDepartedIO");
hobpc.addPropertyChangeListener ( "numberDepartedIO," hobyoDepartStat);

SimpleStatsTally harardhereDepartStat =
    new SimpleStatsTally("numberDepartedIO");
harpc.addPropertyChangeListener ( "numberDepartedIO," harardhereDepartStat);

SimpleStatsTally goaNavyDetectionStat =
    new SimpleStatsTally("numberPiratesDetected");
for (int i = 0; i < goaNavyManager.length; i++)
    goaNavyManager[i].addPropertyChangeListener( "numberPiratesDetected," goaNavyDetectionStat );

SimpleStatsTally ioNavyDetectionStat =
    new SimpleStatsTally("numberPiratesDetected");
for (int i = 0; i < ioNavyManager.length; i++)
    ioNavyManager[i].addPropertyChangeListener( "numberPiratesDetected," ioNavyDetectionStat );

SimpleStatsTally elaayoAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < elaayoPirateManager.length; i++ )
    elaayoPirateManager[i].addPropertyChangeListener( "numberAttemptedAttacks," elaayoAttemptStat);

SimpleStatsTally aluulaAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < aluulaPirateManager.length; i++ )
    aluulaPirateManager[i].addPropertyChangeListener( "numberAttemptedAttacks," aluulaAttemptStat);

SimpleStatsTally qandalaAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < qandalaPirateManager.length; i++ )
    qandalaPirateManager[i].addPropertyChangeListener( "numberAttemptedAttacks," qandalaAttemptStat);

SimpleStatsTally bargalAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < bargalPirateManager.length; i++ )
    bargalPirateManager[i].addPropertyChangeListener( "numberAttemptedAttacks," bargalAttemptStat);

SimpleStatsTally hafunAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");

for ( int i = 0; i < hafunPirateManager.length; i++ )
{
    hafunPirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            hafunAttemptStat);
}

SimpleStatsTally baylaAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < baylaPirateManager.length; i++ )
{
    baylaPirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            baylaAttemptStat);
}

SimpleStatsTally eylAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < eylPirateManager.length; i++ )
{
    eylPirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            eylAttemptStat);
}

SimpleStatsTally garacadAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < garacadPirateManager.length; i++ )
{
    garacadPirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            garacadAttemptStat);
}

SimpleStatsTally hobyoAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < hobyoPirateManager.length; i++ )
{
    hobyoPirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            hobyoAttemptStat);
}

SimpleStatsTally harardhereAttemptStat =
    new SimpleStatsTally("numberAttemptedAttacks");
for ( int i = 0; i < harardherePirateManager.length; i++ )
{
    harardherePirateManager[i].addPropertyChangeListener("numberAttemptedAttacks,"
            harardhereAttemptStat);
}

SimpleStatsTally stmDepartStat =
    new SimpleStatsTally("numberDepartedPort");
stm.addPropertyChangeListener("numberDepartedPort,"
            stmDepartStat);

SimpleStatsTally stoDepartStat =
    new SimpleStatsTally("numberDepartedPort");
sto.addPropertyChangeListener("numberDepartedPort,"
            stoDepartStat);

SimpleStatsTally mtsDepartStat =
    new SimpleStatsTally("numberDepartedPort");
mts.addPropertyChangeListener("numberDepartedPort,"
            mtsDepartStat);
SimpleStatsTally mtoDepartStat =
    new SimpleStatsTally("numberDepartedPort");
    mto.addPropertyChangeListener("numberDepartedPort ,
        mtoDepartStat);
SimpleStatsTally otmDepartStat =
    new SimpleStatsTally("numberDepartedPort");
    otm.addPropertyChangeListener ( "numberDepartedPort,
        otmDepartStat);
SimpleStatsTally otsDepartStat =
    new SimpleStatsTally("numberDepartedPort");
    ots.addPropertyChangeListener ( "numberDepartedPort,
        otsDepartStat);

//*************End of Property Change Listeners for Stats*********************/
//*******************Start of Stats and Schedule Implementation*****************

LinkedList goaDepartures = new LinkedList();
LinkedList ioDepartures = new LinkedList();
LinkedList numPiratesDetected = new LinkedList();
LinkedList navalEffectivenessList = new LinkedList();
LinkedList pirateAttemptList = new LinkedList();
LinkedList pirateEffectiveness1List = new LinkedList();
LinkedList pirateEffectiveness2List = new LinkedList();
LinkedList merchantTransits = new LinkedList();

    for ( int i = 0; i < 30; ++i )
    {
        Schedule.setDecimalFormat("0.00");
        Schedule.setVerbose(false);
        Schedule.setEventSourceVerbose(false);
        Schedule.stopAtTime(simTime);
        elaayoDepartStat.reset ();
        qandalaDepartStat.reset ();
        aluulaDepartStat.reset ();
        bargalDepartStat.reset ();
        hafunDepartStat.reset ();
        baylaDepartStat.reset ();
        eylDepartStat.reset ();
        garacadDepartStat.reset ();
        hobyoDepartStat.reset ();
        harardhereDepartStat.reset ();
        totalNumDepartedGOA = 0;
        totalNumDepartedIO = 0;
        totalNumberPiratesDeparted = 0;
        goaNavyDetectionStat.reset ();
        ioNavyDetectionStat.reset ();
        totalNumberPiratesDetected = 0;
        navalEffectiveness = 0;
        elaayoAttemptStat.reset();
        qandalaAttemptStat.reset();
        aluulaAttemptStat.reset();
        bargalAttemptStat.reset();
        hafunAttemptStat.reset();
        baylaAttemptStat.reset();
        eylAttemptStat.reset();
        garacadAttemptStat.reset();
        hobyoAttemptStat.reset();
        harardhereAttemptStat.reset();
stoDepartStat.reset();
stmDepartStat.reset();
otmDepartStat.reset();
otsDepartStat.reset();
mutexDepartStat.reset();
mutsDepartStat.reset();
totalNumberMerchantTransits = 0;
totalAttemptedAttacks = 0;
pirateEffectiveness1 = 0;
Schedule.reset();
Schedule.startSimulation();

totalNumberPiratesDetected = goaNavyDetectionStat.getCount()
     + ioNavyDetectionStat.getCount();
totalNumDepartedGOA = elaayoDepartStat.getCount()
     + aluulaDepartStat.getCount()
     + qandalaDepartStat.getCount();
totalNumDepartedIO = baylaDepartStat.getCount()
     + hafunDepartStat.getCount()
     + baylaDepartStat.getCount()
     + eylDepartStat.getCount()
     + garacadDepartStat.getCount()
     + hobyoDepartStat.getCount()
     + harardhereDepartStat.getCount();
totalNumberPiratesDeparted = totalNumDepartedGOA +
     totalNumDepartedIO;

System.out.println( "Total Number Pirates Detected: " +
     totalNumberPiratesDeparted );

navalEffectiveness = totalNumberPiratesDetected
     / totalNumberPiratesDeparted;
totalAttemptedAttacks = elaayoAttemptStat.getCount()
     + aluula AttemptStat.getCount()
     + qandalaAttemptStat.getCount()
     + bargalAttemptStat.getCount()
     + hafunAttemptStat.getCount()
     + baylaAttemptStat.getCount()
     + eylAttemptStat.getCount()
     + garacadAttemptStat.getCount()
     + hobyoAttemptStat.getCount()
     + harardhereAttemptStat.getCount();
pirateEffectiveness1 = totalAttemptedAttacks
     / totalNumberPiratesDeparted;
totalNumberMerchantTransits = stoDepartStat.getCount()
     + stmDepartStat.getCount()
     + otmDepartStat.getCount()
     + otsDepartStat.getCount()
     + mutexDepartStat.getCount()
     + mutsDepartStat.getCount();
pirateEffectiveness2 = totalAttemptedAttacks /
     totalNumberMerchantTransits;
goaDepartures.add(totalNumDepartedGOA);
ioDepartures.add(totalNumDepartedIO);
merchantTransits.add ( totalNumberMerchantTransits );
numPiratesDetected.add(totalNumberPiratesDetected);
navalEffectivenessList.add(navalEffectiveness);
pirateAttemptList.add(totalAttemptedAttacks);
pirateEffectiveness1List.add(pirateEffectiveness1);
pirateEffectiveness2List.add(pirateEffectiveness2);

System.out.println("Ellayo Numbers: “ +
epc.getMyPirates ().size ());
System.out.println("Ellayo Departures: “ +
elayoDepartStat.getCount());
System.out.println( “Number Merchants: “ + merchantTransits);
}
}
System.out.println("Pirate Camp Operations Stats Output");
System.out.println("Goa Departures: “ + goaDepartures );
System.out.println( “IO Departures: “ + ioDepartures );
System.out.println( “Merchant Transits: “ + merchantTransits);
System.out.println( “Pirates Detected: “ + numPiratesDetected );
System.out.println("Naval Effectiveness: “ + navalEffectivenessList );
System.out.println( “Attempted Attacks: “ + pirateAttemptList );
System.out.println( “Pirate Effectiveness 1: “ +
pirateEffectiveness1List );
System.out.println( “Pirate Effectiveness 2: “ +
pirateEffectiveness2List );

//*********************End of Schedule Implementation**************************/
//****************************END OF ASSEMBLY*********************************//
}
package supplemental;

import java.awt.geom.Point2D;
import simkit.Priority;
import simkit.smd.BasicLinearMover;
import simkit.smd.Mover;

public class Platform extends BasicLinearMover {

    private PlatformType type;
    protected boolean isAlive;

    public Platform( String name, Point2D initialLocation,
                      double maxSpeed, PlatformType type )
    {
        super( name, initialLocation, maxSpeed );
        this.setType( type );
    }

    public PlatformType getType()
    {
        return type;
    }

    public void setType( PlatformType type )
    {
        this.type = type;
    }

    public boolean getIsAlive()
    {
        return isAlive;
    }

    public void doDie( Mover mover )
    {
        //isAlive = false;
        this.removeMover( mover );
    }
}
this.interruptAll();

waitDelay("OrderStop," 0.0, Priority.HIGH, mover);
}

/**
 * If in movers set, remove. Stop listening to it, and interrupt all pending
 * events with mover as an argument.
 *
 * @param mover Mover to be removed
 */

public void removeMover( Mover mover )
{
    mover.removeSimEventListener( this);
    this.interruptAllWithArgs( mover);
}

@Override

public String toString()
{
    return super.toString()
        .replaceAll( "BasicLinearMover," "Platform"")
        + " " + getType();
}
package supplemental;

public enum PlatformType {
  NAVY,
  MERCHANT,
  PIRATE
}
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  DEAD_IN_WATER,
15  PATROLLING,
16  INTERCEPTING,
17  BOARDING,
18  RETURNING_TO_PATROL
19 }
20 */
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   WAITING_AT_BASE,
15   ENROUTE_TO_PATROL,
16   PATROLLING,
17   INTERCEPTING,
18   ATTACKING,
19   RETURNING_TO_BASE,
20   RETURNING_WITH_MERCHANT,
21   NAVY_BOARDED;
22
23 }
package supplemental;

/**
 * Enums that describe the state of a merchant ship around the Horn Of Africa
 * @version 1.0.0
 * @author Chad R Hutchins
 */

public enum MerchantState {
    DEAD_IN_WATER, TRANSITTING, EVADING, BEEN_ATTACKED, HIJACKED;
}
import com.bbn.openmap.Layer;
import com.bbn.openmap.event.LayerStatusEvent;
import com.bbn.openmap.event.MapMouseListener;
import com.bbn.openmap.event.ProjectionEvent;
import com.bbn.openmap.openmap.omGraphics.OMCircle;
import com.bbn.openmap.openmap.omGraphics.OMGraphicList;
import com.bbn.openmap.openmap.omGraphics.OMText;
import com.bbn.openmap.proj.Projection;
import java.awt.Color;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.MouseEvent;
import java.awt.geom.Point2D;
import javax.swing.JButton;
import javax.swing.JPanel;
import simkit.SimEvent;
import simkit.SimEventListener;

public class SimulationLayer extends Layer implements SimEventListener,
                                    MapMouseListener,
                                    //ModEventListener,
                                    ActionListener {

    OMText text1, text2;
    OMCircle[] circle;
    OMCircle circle1, circle2, circle3, circle4, circle5, circle6, circle7,
    circle8, circle9, circle10, circle11, circle12, circle13, circle14,
    circle15, circle16, circle17, circle18, circle19;
    OMCircle moverCircle1;
    OMGraphicList graphicList;
    //friendly:
    private JButton runButton = new JButton( "RUN SIMULATION" );
    public Projection proj;
    public OpenMapDemo scn;
    public int detectionCounter = 0;

    public SimulationLayer() {
        scn = new OpenMapDemo();
        graphicList = new OMGraphicList();
        Point2D pirateIO = scn.getLocationIoPirateMover( 0 );
        circle1 = new OMCircle( ( float ) pirateIO.getX(),
                               ( float ) pirateIO.getY(),
                               scn.mmToDeg( 1, 15.0 )); //12NM
        circle1.setLinePaint( Color.RED );
moverCircle1 = new OMCircle( ( float ) pirateIO.getX(),
( float ) pirateIO.getY(), 3, Length.METER );
moverCircle1.setFillPaint( Color.RED );

Point2D pirateGOA = scn.getLocationGoaPirateMover( 0 );
circle2 = new OMCircle( ( float ) pirateGOA.getX(),
( float ) pirateGOA.getY(),
scn.nmToDeg( 1, 15.0f ) ); //12NM
circle2.setLinePaint( Color.RED );

Point2D pirateGOA2 = scn.getLocationGoaPirateMover( 0 );
circle19 = new OMCircle( ( float ) pirateGOA2.getX(),
( float ) pirateGOA2.getY(),
scn.nmToDeg( 1, 5.0f ) ); //12NM
circle19.setLinePaint( Color.RED );

Point2D navyIoPB6 = scn.getLocationIoNavyMover( 0 );
circle16 = new OMCircle( ( float ) navyIoPB6.getX(),
( float ) navyIoPB6.getY(),
scn.nmToDeg( 1, 20.0 ) ); //25NM
circle16.setLinePaint( Color.BLUE );

Point2D navyIoPB7 = scn.getLocationIoNavyMover( 1 );
circle17 = new OMCircle( ( float ) navyIoPB7.getX(),
( float ) navyIoPB7.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle17.setLinePaint( Color.BLUE );

Point2D navyIoPB8 = scn.getLocationIoNavyMover( 2 );
circle3 = new OMCircle( ( float ) navyIoPB8.getX(),
( float ) navyIoPB8.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle3.setLinePaint( Color.BLUE );

Point2D navyIoPB9 = scn.getLocationIoNavyMover( 3 );
circle6 = new OMCircle( ( float ) navyIoPB9.getX(),
( float ) navyIoPB9.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle6.setLinePaint( Color.BLUE );

Point2D navyIoPB10 = scn.getLocationIoNavyMover( 4 );
circle7 = new OMCircle( ( float ) navyIoPB10.getX(),
( float ) navyIoPB10.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle7.setLinePaint( Color.BLUE );

Point2D navyIoPB11 = scn.getLocationIoNavyMover( 5 );
circle8 = new OMCircle( ( float ) navyIoPB11.getX(),
( float ) navyIoPB11.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle8.setLinePaint( Color.BLUE );

Point2D navyIoPB12 = scn.getLocationIoNavyMover( 6 );
circle9 = new OMCircle( ( float ) navyIoPB12.getX(),
( float ) navyIoPB12.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle9.setLinePaint( Color.BLUE );

Point2D navyIoPB13 = scn.getLocationIoNavyMover( 7 );
circle10 = new OMCircle( ( float ) navyIoPB13.getX(),
( float ) navyIoPB13.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
Point2D navyGoaPB1 = scn.getLocationGoaNavyMover( 0 );
circle11 = new OMCircle( ( float ) navyGoaPB1.getX(),
( float ) navyGoaPB1.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle11.setLinePaint( Color.BLUE );

Point2D navyGoaPB2 = scn.getLocationGoaNavyMover( 1 );
circle12 = new OMCircle( ( float ) navyGoaPB2.getX(),
( float ) navyGoaPB2.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle12.setLinePaint( Color.BLUE );

Point2D navyGoaPB3 = scn.getLocationGoaNavyMover( 2 );
circle13 = new OMCircle( ( float ) navyGoaPB3.getX(),
( float ) navyGoaPB3.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle13.setLinePaint( Color.BLUE );

Point2D navyGoaPB4 = scn.getLocationGoaNavyMover( 3 );
circle14 = new OMCircle( ( float ) navyGoaPB4.getX(),
( float ) navyGoaPB4.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle14.setLinePaint( Color.BLUE );

Point2D navyGoaPB5 = scn.getLocationGoaNavyMover( 4 );
circle15 = new OMCircle( ( float ) navyGoaPB5.getX(),
( float ) navyGoaPB5.getY(),
scn.nmToDeg( 1, 20.0f ) ); //25NM
circle15.setLinePaint( Color.BLUE );

Point2D merchantSB = scn.getLocationSbMerchant( 0 );
circle4 = new OMCircle( ( float ) merchantSB.getX(),
( float ) merchantSB.getY(),
0.33459801 ); //25NM
circle4.setLinePaint( Color.MAGENTA );

Point2D merchantNB = scn.getLocationNbMerchant( 0 );
circle5 = new OMCircle( ( float ) merchantNB.getX(),
( float ) merchantNB.getY(),
scn.nmToDeg( 1, 20.0 ) ); //25NM
circle5.setLinePaint( Color.MAGENTA );

//for ( int i = 0 ; i < scn.ioPirateMover.length ; ++i )
graphicList.add( circle1 );
graphicList.add( circle2 );
graphicList.add( circle3 );
graphicList.add( circle4 );
graphicList.add( circle5 );
graphicList.add( circle6 );
graphicList.add( circle7 );
graphicList.add( circle8 );
graphicList.add( circle9 );
graphicList.add( circle10 );
graphicList.add( circle11 );
graphicList.add( circle12 );
graphicList.add( circle13 );
graphicList.add( circle14 );
graphicList.add(circle15);
graphicList.add(circle16);
graphicList.add(circle17);
//grafikList.add(moverCircle1);
}

@Override
public void processSimEvent(SimEvent e)
{
    fireStatusUpdate(LayerStatusEvent.START_WORKING);
    if (e.getName().equals("Ping"))
    {
        OMCircle tempCirc1 = (OMCircle)graphicList.getOMGraphicAt(0);
        OMCircle tempCirc2 = (OMCircle)graphicList.getOMGraphicAt(1);
        OMCircle tempCirc3 = (OMCircle)graphicList.getOMGraphicAt(2);
        OMCircle tempCirc4 = (OMCircle)graphicList.getOMGraphicAt(3);
        OMCircle tempCirc5 = (OMCircle)graphicList.getOMGraphicAt(4);
        OMCircle tempCirc6 = (OMCircle)graphicList.getOMGraphicAt(5);
        OMCircle tempCirc7 = (OMCircle)graphicList.getOMGraphicAt(6);
        OMCircle tempCirc8 = (OMCircle)graphicList.getOMGraphicAt(7);
        OMCircle tempCirc9 = (OMCircle)graphicList.getOMGraphicAt(8);
        OMCircle tempCirc10 = (OMCircle)graphicList.getOMGraphicAt(9);
        OMCircle tempCirc11 = (OMCircle)graphicList.getOMGraphicAt(10);
        OMCircle tempCirc12 = (OMCircle)graphicList.getOMGraphicAt(11);
        OMCircle tempCirc13 = (OMCircle)graphicList.getOMGraphicAt(12);
        OMCircle tempCirc14 = (OMCircle)graphicList.getOMGraphicAt(13);
        OMCircle tempCirc15 = (OMCircle)graphicList.getOMGraphicAt(14);
        OMCircle tempCirc16 = (OMCircle)graphicList.getOMGraphicAt(15);
        OMCircle tempCirc17 = (OMCircle)graphicList.getOMGraphicAt(16);
        //OMCircle tempCirc19 = (OMCircle)graphicList.getOMGraphicAt(17);
        tempCirc1.setLatLon((float)scn.getLocationIoPirateMover(0).getX(),
                          (float)scn.getLocationIoPirateMover(0).getY());
        tempCirc2.setLatLon((float)scn.getLocationIoPirateMover(0).getX(),
                          (float)scn.getLocationIoPirateMover(0).getY());
        tempCirc3.setLatLon((float)scn.getLocationIoNavyMover(2).getX(),
                          (float)scn.getLocationIoNavyMover(2).getY());
        tempCirc4.setLatLon((float)scn.getLocationSbMerchant(0).getX(),
                          (float)scn.getLocationSbMerchant(0).getY());
        tempCirc5.setLatLon((float)scn.getLocationNbMerchant(0).getX(),
                          (float)scn.getLocationNbMerchant(0).getY());
        tempCirc6.setLatLon((float)scn.getLocationIoNavyMover(3).getX(),
                          (float)scn.getLocationIoNavyMover(3).getY());
tempCirc7.setLatLon( ( float ) scn.getLocationIoNavyMover( 4 ).
getX(),
( float ) scn.getLocationIoNavyMover( 4 ).
getY() );

tempCirc8.setLatLon( ( float ) scn.getLocationIoNavyMover( 5 ).
getX(),
( float ) scn.getLocationIoNavyMover( 5 ).
getY() );

tempCirc9.setLatLon( ( float ) scn.getLocationIoNavyMover( 6 ).
getX(),
( float ) scn.getLocationIoNavyMover( 6 ).
getY() );

tempCirc10.setLatLon( ( float ) scn.getLocationIoNavyMover( 7 ).
getX(),
( float ) scn.getLocationIoNavyMover( 7 ).
getY() );

tempCirc11.setLatLon( ( float ) scn.getLocationGoaNavyMover( 0 ).
getX(),
( float ) scn.getLocationGoaNavyMover( 0 ).
getY() );

tempCirc12.setLatLon( ( float ) scn.getLocationGoaNavyMover( 1 ).
getX(),
( float ) scn.getLocationGoaNavyMover( 1 ).
getY() );

tempCirc13.setLatLon( ( float ) scn.getLocationGoaNavyMover( 2 ).
getX(),
( float ) scn.getLocationGoaNavyMover( 2 ).
getY() );

tempCirc14.setLatLon( ( float ) scn.getLocationGoaNavyMover( 3 ).
getX(),
( float ) scn.getLocationGoaNavyMover( 3 ).
getY() );

tempCirc15.setLatLon( ( float ) scn.getLocationGoaNavyMover( 4 ).
getX(),
( float ) scn.getLocationGoaNavyMover( 4 ).
getY() );

tempCirc16.setLatLon( ( float ) scn.getLocationIoNavyMover( 0 ).
getX(),
( float ) scn.getLocationIoNavyMover( 0 ).
getY() );

tempCirc17.setLatLon( ( float ) scn.getLocationIoNavyMover( 1 ).
getX(),
( float ) scn.getLocationIoNavyMover( 1 ).
getY() );

// tempCirc19.setLatLon( ( float ) scn.getLocationGoaPirateMover( 1 ).
// getX(),
// ( float ) scn.getLocationGoaPirateMover( 1 ).
// getY() );

189
tempCirc1.generate( proj );
tempCirc2.generate( proj );
tempCirc3.generate( proj );
tempCirc4.generate( proj );
tempCirc5.generate( proj );
tempCirc6.generate( proj );
tempCirc7.generate( proj );
tempCirc8.generate( proj );
tempCirc9.generate( proj );
tempCirc10.generate( proj );
tempCirc11.generate( proj );
tempCirc12.generate( proj );
tempCirc13.generate( proj );
tempCirc14.generate( proj );
tempCirc15.generate( proj );
tempCirc16.generate( proj );
tempCirc17.generate( proj );
//tempCirc19.generate( proj );
}
if ( e.getEventName().
equals( "Detection" ) )
{
    System.out.println(
        "_______________________________________________________" + getSimTime() );
detectionCounter++;
}
if ( proj != null )
{
    ( ( OMGraphicList ) graphicList ).project( ( Projection ) proj, true );
}
repaint();
fireStatusUpdate( LayerStatusEvent.FINISH_WORKING );

@Override
public String[] getMouseModeServiceList() {
    // TODO Auto-generated method stub
    return null;
}

@Override
public boolean mouseClicked( MouseEvent arg0 ) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public boolean mouseDragged( MouseEvent arg0 ) {
    // TODO Auto-generated method stub
    return false;
}

@Override
public void mouseEntered( MouseEvent arg0 ) {
    // TODO Auto-generated method stub
}
@Override
class XMappedObject implements StateAware {
    private boolean dirty = false;
    private String name;
    
    public void mouseExited(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }
    
    public void mouseMoved(MouseEvent arg0) {
        // TODO Auto-generated method stub
    }
    
    @Override
    public boolean mouseMoved(MouseEvent arg0) {
        // TODO Auto-generated method stub
        return false;
    }
    
    @Override
    public boolean mousePressed(MouseEvent arg0) {
        // TODO Auto-generated method stub
        return false;
    }
    
    @Override
    public boolean mouseReleased(MouseEvent arg0) {
        // TODO Auto-generated method stub
        return false;
    }
    
    @Override
    public void projectionChanged(ProjectionEvent e) {
        proj = e.getProjection();
        System.out.println("projection Changed");
        (OMGraphicList) graphicList).project(e.getProjection(), true);
        repaint();
    }
    
    public void paint(java.awt.Graphics g) {
        if (graphicList.size() > 0 )
        {
            graphicList.render(g);
        }
        fireStatusUpdate(LayerStatusEvent.FINISH_WORKING);
    }
    
    public void findAndInit(Object someObj) {
        /*
         * if (someObj instanceof DenizSim.myLayer ){
         *   System.out.println("myLayer is added !!!!!!!!"); //myLayer myL=
         *   (myLayer)someObj; }
         */
    }
}
public double getSimTime()
{
    return scn.getSimTime();
}

// A GUI for the layer

@Override
public java.awt.Component getGUI()
{
    JPanel returnPanel = new JPanel();
    final PingThread2 pt = new PingThread2(0.1, 100, false);
    pt.addSimEventListener(this);
    for (int i = 0; i < scn.ioPirateMover.length; ++i)
    {
        scn.ioPirateMover[i].addSimEventListener(this);
    }
    for (int i = 0; i < scn.ioNavyMover.length; ++i)
    {
        scn.ioNavyMover[i].addSimEventListener(this);
    }
    for (int i = 0; i < scn.ioPirateSensor.length; ++i)
    {
        scn.ioPirateSensor[i].addSimEventListener(this);
    }
    for (int i = 0; i < scn.ioNavySensor.length; ++i)
    {
        scn.ioNavySensor[i].addSimEventListener(this);
    }
    runButton.addActionListener(new ActionListener() {

        @Override
        public void actionPerformed(ActionEvent e)
        {
            scn.startScenario();
            pt.startPinging();
        }
    });
    returnPanel.add(runButton);
    return returnPanel;
}
APPENDIX R. JAVA SWING SANDBOX FRAME IMPLEMENTATION CODE SNIPPET

```java
//*************Start of Sandbox with background image implementation**********/

//Allows for background image
BufferedImage img = null;
File file = new File( "images/test.PNG" );
System.out.println( file.exists() );
img = ImageIO.read( file );

//Scale for background image
double scale = 1.0;

//More scaling
int rescaledWidth = ( int ) ( img.getWidth() * scale );
int rescaledHeight = ( int ) ( img.getHeight() * scale );
BufferedImage resizedImage = new BufferedImage( rescaledWidth, rescaledHeight, img.getType() );

AffineTransform scaleTransform = AffineTransform.getScaleInstance( scale, scale );

Graphics2D g = resizedImage.createGraphics();
g.drawImage( img, scaleTransform, null );

//Sandbox for simulation
SandboxFrame sandboxFrame = new SandboxFrame();
Sandbox2 sandbox = sandboxFrame.getSandbox();

//Sets the background image to the appropriate scale
sandbox.setBackroundImage( resizedImage );

sandboxFrame.setSize( resizedImage.getWidth(), resizedImage.getHeight() + 100 );

sandbox.setOrigin( new Point2D.Double( 0.0, resizedImage.getHeight() ) );
sandbox.setDrawAxes( true );

//Listener for moust points
sandbox.addMouseListener( new MouseMotionListener( sandboxFrame ) );

//Window for collecting waypoint data
JFrame wayPointFrame = new JFrame();
wayPointFrame.setSize( 300, 100 );
wayPointFrame.setLocation( ( int ) sandboxFrame.getLocation().getX() + sandboxFrame.getWidth(),
( int ) sandboxFrame.getLocation().getY() );
WaypointBuilder wayPointBuilder = new WaypointBuilder();
JScrollPane jscrollPane = new JScrollPane( wayPointBuilder );
wayPointFrame.getContentPane().add( jscrollPane );
wayPointFrame.setDefaultCloseOperation( JFrame.DO_NOTHING_ON_CLOSE );
wayPointBuilder.addPropertyChangeListener( new PathBuilder() );
```

193
//Add listener to your mouse which allows ability to click the mouse at
//a given point in the Sandbox and get the x and y values.
sandbox.addMouseListener( wayPointBuilder );
sandboxFrame.setVisible( true );
wayPointFrame.setVisible( true );

//*************End of Background and Sandbox implementation*************//
package util;

import animate.Sandbox;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.geom.Point2D;
import java.util.ArrayList;
import javax.swing.DefaultListModel;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;

/**
 * @version $Id: WaypointBuilder.java 51 2012–06–16 05:20:29Z crhutchi $
 * @author ahhuss
 */
public class WaypointBuilder extends JPanel implements MouseListener {
    private JList waypointsList;
    private DefaultListModel waypointListModel;

    public WaypointBuilder() {
        this.waypointListModel = new DefaultListModel();
        this.waypointsList = new JList(waypointListModel);
        this.waypointsList.setVisibleRowCount(10);
        JScrollPane jscrollPane = new JScrollPane(this.waypointsList);
        this.add(this.waypointsList);
    }

    @Override
    public void mouseClicked(MouseEvent me) {
        Object source = me.getSource();
        if (source instanceof Sandbox) {
            Sandbox sb = (Sandbox) source;
            double x = me.getX() - sb.getOrigin().getX();
            double y = sb.getOrigin().getY() - me.getY();
            Point2D.Double newPoint = new Point2D.Double(x, y);
            waypointListModel.addElement(newPoint);
            firePropertyChange("waypoint", null, newPoint);
        }
    }

    @Override
    public void mousePressed(MouseEvent me) {
    }

    @Override
    public void mouseReleased(MouseEvent me) {
    }

    @Override
    public void mouseEntered(MouseEvent me) {
    }

    @Override
    public void mouseExited(MouseEvent me) {
    }

    @Override
    public void mouseMoved(MouseEvent me) {
    }

    @Override
    public void mouseDragged(MouseEvent me) {
    }

    @Override
    public void mouseWheelMoved(MouseEvent me) {
    }
}

195
public void mouseExited(MouseEvent me) {
}
APPENDIX T. MOUSE LISTENER JAVA CODE

```java
package util;

import java.awt.event.MouseEvent;
import java.awt.event.MouseMotionListener;
import java.awt.geom.Point2D;
import simkit.smd.animate.SandboxFrame;

/**
 * 
 * @version $Id: MouseLocationListener.java 51 2012–06–16 05:20:29Z crhutchi$
 * 
 * @author ahbuss
 */
public class MouseLocationListener implements MouseMotionListener {

private SandboxFrame sandboxFrame;
private Point2D origin;

public MouseLocationListener(SandboxFrame sandboxFrame) {
    this.setSandboxFrame(sandboxFrame);
}

@Override
public void mouseDragged(MouseEvent me) {
}

@Override
public void mouseMoved(MouseEvent me) {
    sandboxFrame.setStatus(me.getX() + " " + me.getY() + "  =>  " +
                        (me.getX() - origin.getX()) + " " + (origin.getY() - me.getY()));
}

/**
 * @return the sandboxFrame
 */
public SandboxFrame getSandboxFrame() {
    return sandboxFrame;
}

/**
 * @param sandboxFrame the sandboxFrame to set
 */
public void setSandboxFrame(SandboxFrame sandboxFrame) {
    this.sandboxFrame = sandboxFrame;
    this.origin = sandboxFrame.getSandbox().getOrigin();
}
}
```
LIST OF REFERENCES


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