Team 3
Allen Gurdus
Victor Sacristan
IEOR 140 Fall 12
Final Project Report

Introduction:

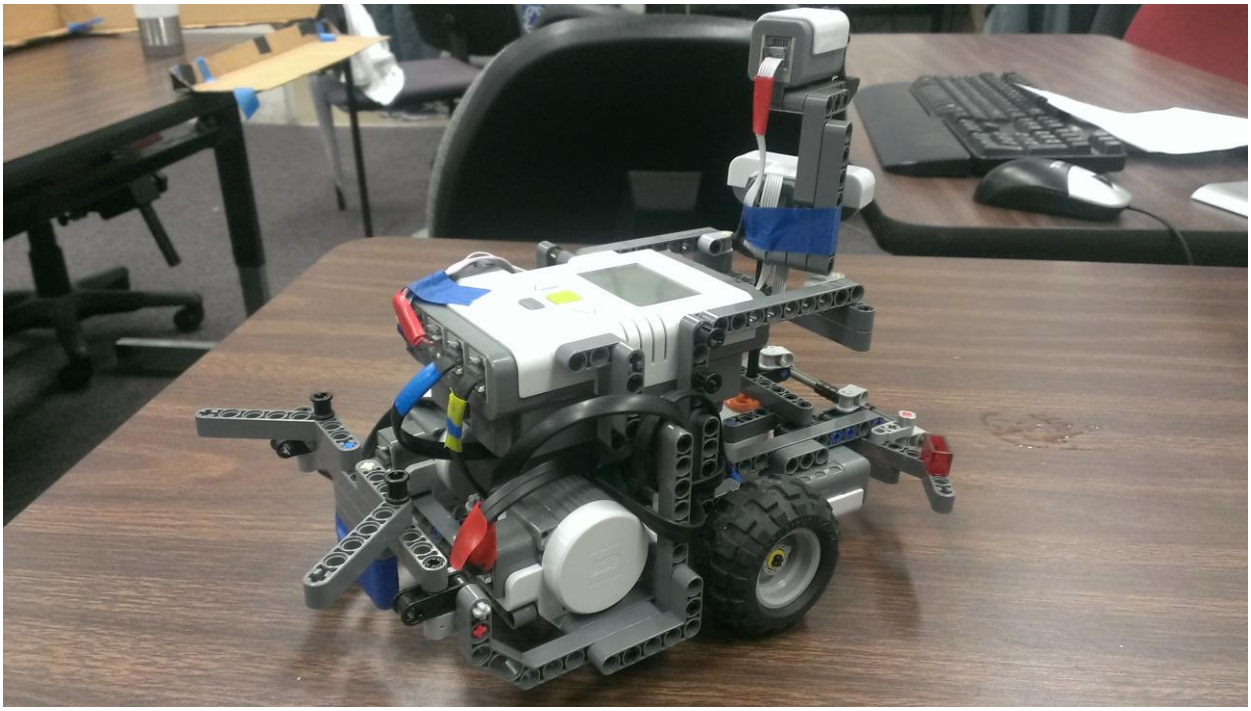
Approximately 120 person hours were spent on the final project. Allen and Victor shared all duties. The NXT java files can be found in the NXT Files/src folder and the PC java files can be found in the PC Files/src folder. A list of .java files and links to their javadocs can be found in the appendix.

Specifications:

All specifications were met for this project.

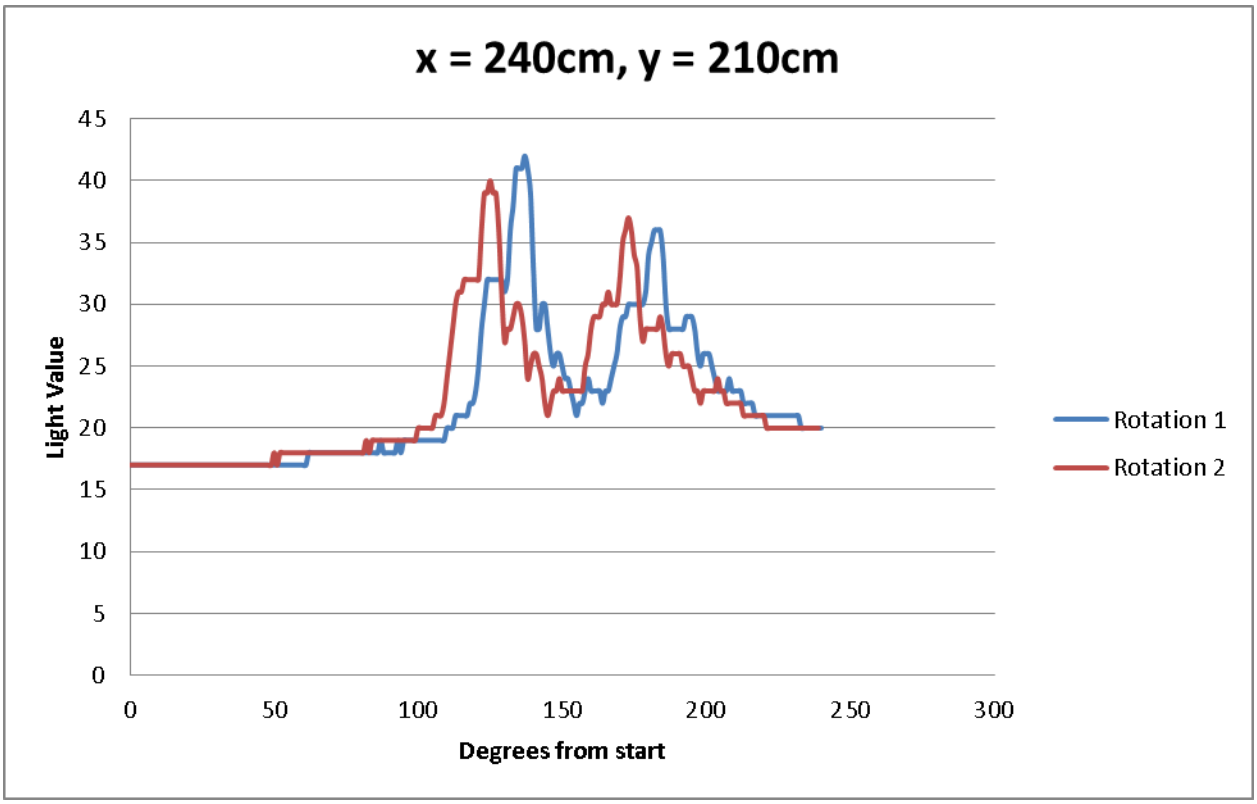
Hardware Design:

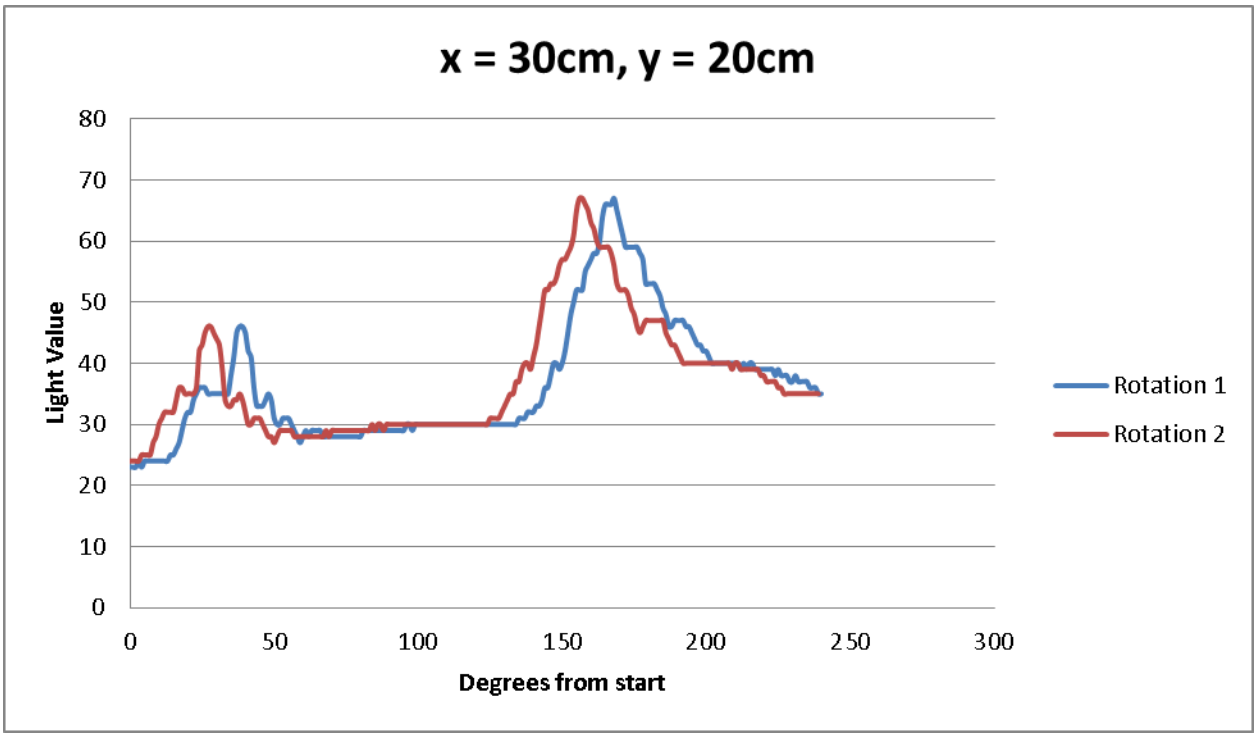
The robot is constructed with wheel and sensor motors as well as Ultrasonic, Light and two Bumper sensors. The Bumper sensors are placed at the front of the robot to detect objects in the robot's path. The Light sensor and Ultrasonic sensor are placed at the front of the robot and can rotate to any position to identify light or objects. A magnet was added to the back of the robot for picking up the "bombs."



Experimental Work:

The first experiment conducted was a lightScan from two locations. This allowed us to observe the backlash error (see figures 1 and 2).





The second experiment required using the ultrasonic sensor to identify the distance to each wall at multiple points. At three separate points our robot returned the following distance values…

South Wall	North Wall
19	221
92	149
172	67

The sum of the values is usually about 4cm short of the width of the hall. To compensate for this, in the future we will add 2cm to every distance reading.

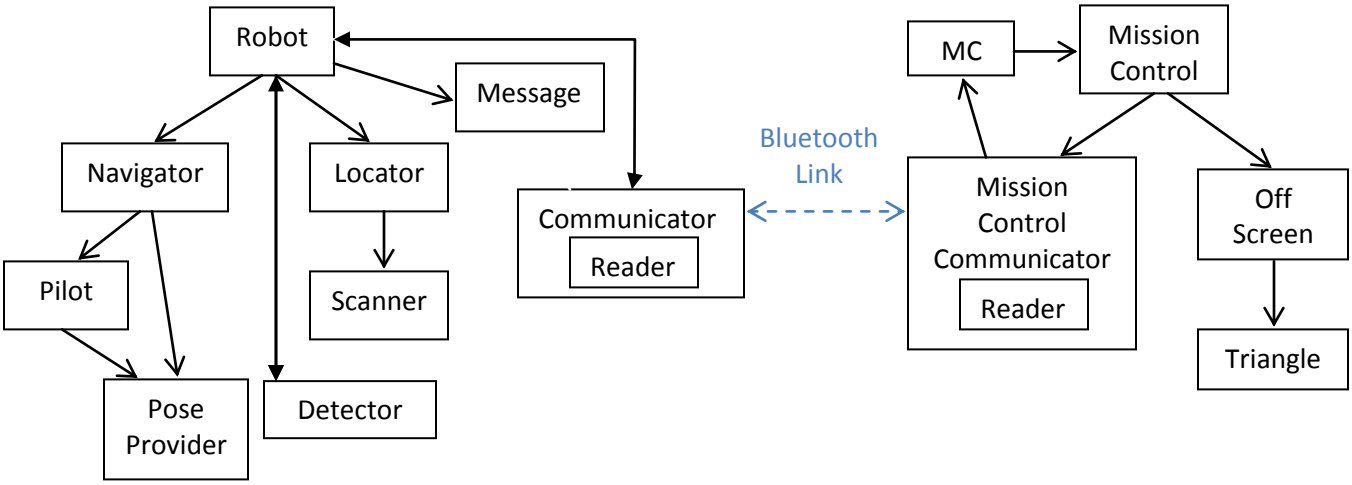
The third experiment finds the average and standard deviation of the beacon bearings returned by the lightScan method at both locations for the first experiment facing in the 90 and -90 directions.

	46	160		158	217		36	355		172	221
	45	159		158	217		37	354		172	219
	45	160		158	217		37	354		172	219
	46	160		158	217		37	355		172	219
	45	159		158	217		37	355		172	219
	45	159		158	217		37	356		172	219
	46	159		158	217		37	355		172	218
	45	159		158	217		37	355		172	218
Avg.	45.375	159.375		158	217		36.875	354.875		172	219
SD	0.517549	0.517549		0	0		0.353553	0.64087		0	0.92582

For the third experiment, the fixPosition() method was tested eight times in each direction for two different points. All of our experiments yielded results up to specification with two exceptions. The beacon differences did not have a standard deviation of 1.5 or less at x=30, y=35. This was due to the light sensor being at a very different location relative to the bearings when facing each direction. The second exception was at x=240, y = 180 when using the ultrasonic sensor on the south wall instead of the closer north wall. Due to the distance, the sensor could not consistently get the right distance and the rest of the values were erratic as a result.

All of the data for the experiments can be found in the reports folder.

Problem Analysis/Software Design:



The Robot (built in milestone 4 and edited for 5 and 6) carries out the primary tasks of the milestone with commands received from Mission Control. The Robot calls on the Locator method fixPosition() to determine its pose using the Ultrasonic Sensor and the Light Sensor to determine the distance of the nearest wall and bearing of the two light beacons. The Robot calls on Navigator to carry out all movement for the robot using the travel(), rotateTo() and navigate() methods. The Navigator calls on the Pose Provider and various DifferentialPilot methods to move the robot and to track its position respectively.

The Robot class also has two methods used when picking up the bomb that were built for milestone 6; pickBomb() and checkBomb(). There are also buttons added to the GUI for calling on these methods. The pickBomb() method scans the area directly in front of it slowly with the ultrasonic sensor by calling on the bomb() method in Scanner. The angle at which the closest distance was found is recorded. Then, the robot travels half the distance to the bomb, turns around and backs up into the bomb until it picks it up with the magnet. Another check is conducted by rotating the ultrasonic sensor 180 degrees and making sure the distance to the nearest object is correct. Additional checks for the bomb can be made at any time from MissionControl.

The Detector class (built during milestone 6) is a simplified version of the Detector from Project 3. It only utilizes touch sensors to identify objects in the robot's path. The Detector runs in its own thread and calls the avoider() method in Robot if either touch sensor is pressed at any time. The avoider() method calls the stop() method and then moves the robot backwards 15cm.

A new Scanner was built (milestone 2) for the final project based on the Scanner from Project 3. This scanner has two primary methods; lightScan(int startAngle, finishAngle) and distanceAt(int angle). lightScan repeats a scan between the start and finish once in each direction to identify the angle of both light beacons using the light sensor. This is done to correct the backlash error by taking the average of the two identified angles of each peak. lightScan returns both of these angles. distanceAt rotates to the given angle and pings using the ultrasonic sensor and returns the distance. Another method which was built for milestone 6, bomb(), is called on by the Robot to slowly scan a small area twice to identify the angle of the shortest distance. This is enacted by mission control when it has identified a bomb.

The Locator (milestone 3) was built off a template provided by Roger Glassey. The Locator's main function is to determine its position based on current pose. This is carried out by the primary method fixPosition(). fixPosition calls on the scanBeacons() method in Locator which then calls on lightScan() method in Scanner for determining the angle of beacons relative to the robot's position.

The Locator class has one primary function which is to determine the position of the robot. It uses an initial position that is provided by the tester which calls on the ButtonCounter for user input. The ButtonCounter returns a pose created using user input. This pose is sent to the Locator using the updatePose() method in Locator.

Then, the primary method in Locator, fixPosition() is called. fixPosition() calls on getBearings() to find the relative bearing towards the beacons as well as the wall distance and angle to the wall. The beacon bearings are identified using the lightScan() method in Scanner. lightScan() uses the LightSensor to identify the two peaks of light intensity. The distance to the wall and angle to the wall are found using the checkWall() method. fixPosition() then calculates the x-value of the robot's location using the y-value found with checkWall() and the angle between the beacons. A boolean is set if the first wall check returns a value off by more than 10 of what was expected (the Pose Y). Since the beacon scan starts from the opposite side if the farther wall is scanned, this Boolean notifies the Locator that the recorded beacon positons should be switched.

The lightScan() method uses heading and direction of scan to determine which beacon matches which identified light peak. For example, if the heading is zero and the scan direction is clockwise, the beacon on the north wall will be scanned first and last while the two middle scans will be the beacon on the south wall.

To communicate with Mission Control, the Robot class calls on the Message and Communicator classes (the entire communications system was built for milestone 4). The Robot sends information to Mission Control by calling on the send() method in Communicator. Communicator receives coded messages from the Mission Control Communicator over Bluetooth link. The Communicator has a subclass Reader that runs in its own thread so it can constantly receive information from Mission Control Communicator. Communicator sends Message objects to Robot which holds Messages in an ArrayList. Messages, like information communicated over Bluetooth, contain a header and an array of floats. The header is an integer that is coded as a header by the ENUM class when the message is received by the robot. The ENUM header corresponds to an action. stop() messages are carried out immediately and clear the ArrayList.

The MissionControlCommunicator is constructed similarly to the Communicator, but runs on the PC and is utilized by the Mission Control. It also has a Reader subclass running in its own thread so it can constantly receive information. Information is sent from MisionControlCommunicator to MissionControl through an interface, MC.java. All information sent between communicators are sent with an integer header and float array.

MissionControl is a GUI that presents information received from the robot to the user and also sends information from the user to the robot the MIssionControlCommunicator. MissionControl calls on the OffScreen class to create a grid that graphically shows the robot and its path. OffScreen calls on

Graphics to draw the grid, robot and path. The robot is represented by a triangle built by the Triangle class which is called on by Offscreen. The GUI can be seen in the screenshot in the appendix.
For this milestone, the Robot was given the ability to fix its position if an obstacle is blocking the nearest wall, but the far wall is unblocked. The robot also now has a mapping function in which it scans for obstacles and walls parallel to its path of motion. This information is communicated to MissionControl which in turn sends the information to OffScreen. OffScreen uses the information to create a map which the robot can later follow.

The mapping functions (built for milestone 5) are enabled by buttons on the GUI (MapL, MapR and Ping). MapL and MapR turn the scanner to the left and right (respectively) of the robots path and record distance with the ultrasonic sensor. This information is communicated to the GUI every 100 milliseconds. Lines are drawn between values close enough to each other to draw walls on the Off Screen grid.

Difficult/Interesting/Challenging:

The most interesting part of completing this project was creating the fix position software. Using trigonometry and the two light beacons to identify location is a fairly easy concept that becomes very complicated and abstract when being programmed. This was also the most difficult part of the project as it required a lot of trial and error and extra time outside of class to complete.

Bomb Recovery Challenges:

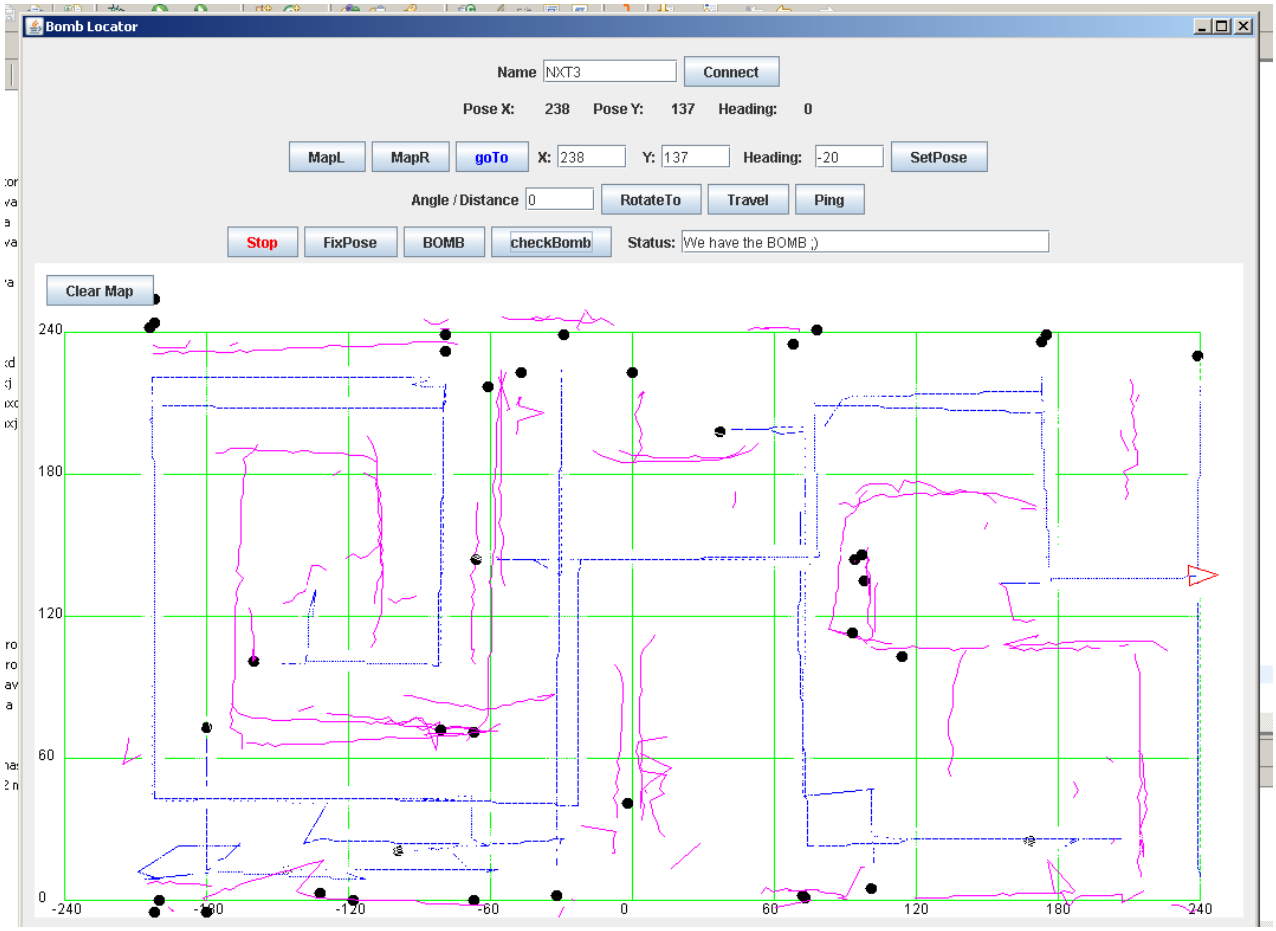
For the most part, our bomb recovery went smoothly. However, at about x = -150, y = 10 we were not able to get a proper heading for our robot. This resulted in bumping into the wall a number of times and exacerbating the issue. Eventually we were able to fix our pose by mapping to the right of the robot and adjusting the heading based on how slanted the wall was. After that we called the fix position method a couple times. The map is particularly congested at this location as a result of our attempts to fix the problem.

After we got a reliable pose there were no issues the rest of the way. The bomb was located and picked up with no issue. Returning to the origin point required minimal mapping, pinging and pose fixing.

Appendix:

Screenshot

Black dots are ping results. Pink lines are walls. The blue lines are the robot path.



NXT Files (found in the NXT Files/src folder)
Test file: Tester.java
Scanner: Scanne.java
Locator: Locator.java
Message.java
Communicator.java
Robot.java
Detector.java

PC Files (found in the PC Files/src folder)
Mission Control: MissionControl.java
Mission Control Communicator: MissionControlCommunicator.java
MC interface: MC.java
Off Screen Grid: OffScreen.java
Triangle: Triangle.java

JavaDocs
file:///Z:/TEAMS/Team3/FinalProject /NXT Files/doc/Detector.html
file:///Z:/TEAMS/Team3/FinalProject /NXT Files/doc/Scanner.html
file:///Z:/TEAMS/Team3/FinalProject/NXT Files/doc/Locator.html
file:///Z:/TEAMS/Team3/FinalProject/NXT Files/doc/Message.html

file:///Z:/TEAMS/Team3/FinalProject/NXT Files/doc/Communicator.html
file:///Z:/TEAMS/Team3/FinalProject/NXT Files/doc/Robot.html
file:///Z:/TEAMS/Team3/FinalProject/NXT Files/doc/Tester.html

file:///Z:/TEAMS/Team3/FinalProject/PC Files/doc/MissionControl.html
file:///Z:/TEAMS/Team3/FinalProject/PC Files/doc/MissionControlCommunicator.html
file:///Z:/TEAMS/Team3/FinalProject/PC Files/doc/MC.html
file:///Z:/TEAMS/Team3/FinalProject/PC Files/doc/OffScreen.html
file:///Z:/TEAMS/Team3/FinalProject/PC Files/doc/Triangle.html