Sensor Simulator

Sensor Simulator

Product Documentation

DAVID KRUTSKO
github.com/dkrutsko

ABHINAV SHUKLA
github.com/AbsMechanik

CONTENTS

INTRODUCTION

Wireless connectivity is established with either omnidirectional or directional antennae. An omnidirectional antenna transmits a signal in all directions (modelled as a circle) whereas a directional antenna (modelled as a circular sector) transmits a signal within a limited, predefined angle. It is interesting to observe that the energy consumed by an omnidirectional antenna, with range r, is proportional to about πr^2 whereas a directional antenna, with angle φ and range r, consumes energy proportional to $\varphi \cdot \frac{R^2}{2}$.

One of the biggest problems with the usage of omnidirectional antennae is interference. A signal arriving at such a sensor, within the proximity of several omnidirectional antennae, will interfere and degrade reception and potentially also result in a security risk (due to interference). Hence, replacing omnidirectional antennae with directional ones becomes more feasible for reasons of energy efficiency and potentially reduced interference (as well as security). This brings up an interesting problem to replace omnidirectional antennae with its directional counterparts. The main issue of concern, when such a replacement is being considered, is to understand how this may affect certain characteristics such as degree, diameter, path length, etc. of the resulting network. Such an analysis naturally leads one to question the trade-offs between various factors such as connectivity, diameter, interference, security risks, etc. when using directional antennae in constructing sensor networks.

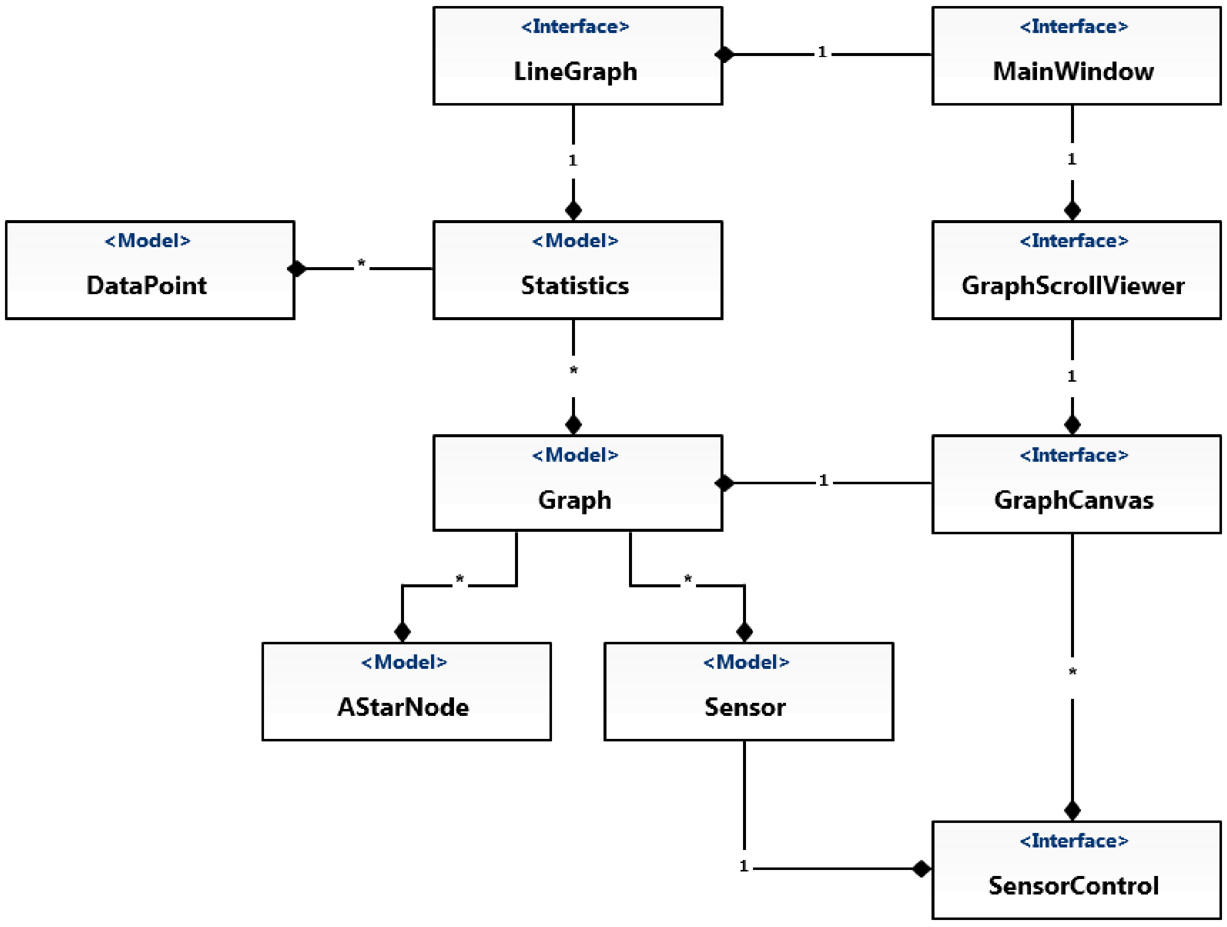
To address this issue, we decided to build a "sensor simulator" to determine if constructing a sensor network with strictly directional antennae is a feasible idea or not. A sensor network problem can be reduced to a simple undirected/directed discrete graph. We hope that several iterations of the simulator will help us better understand the nature of the underlying issues and hopefully give one the ability to perform various graphical analysis techniques and statistically predict the ratio of the hop counts made by using only directional antenna and omnidirectional antenna.

OVERVIEW

The sole purpose of this document is to present the reader with our class diagrams, implementation procedures and various test plans to ensure that the simulator system meets all quality requirements and performs as intended. This is meant to give the developers and testers a good idea of what faults/errors/bugs need to be addressed and how the system might behave under erroneous conditions. Each of the test cases focuses on a different style of testing, namely, Unit Testing, Integration Testing and System Testing and is accompanied by a screenshot from the application itself to ensure accuracy, integrity and traceability.

CLASS DIAGRAM

This section describes the structure of the overall system as well as the interaction between classes. The data is presented in a simplified fashion to better emphasise relationships throughout the system. Please refer to the next section in order to get more detailed information about the internal workings of individual classes.



CLASS STRUCTURE

This section describes the internal workings of classes detailed in the previous section. Each class is split up into one or more subgroups depending on the type of data being portrayed. Some information is omitted as it does not add to the overall representation of the class.

MAIN WINDOW

This class represents the main window; it controls the menu system and handles higher level input functionality for its children. It is also responsible for invoking statistical queries and displaying the results.

PROPERTIES

+ Topmost	: bool
+ DisplayMenu	: bool
+ DisplayModify	: bool
+ DisplayAbout	: bool
+ DisplayRandom	: bool
+ DisplayStats	: bool
+ DisplayLineGraph	: bool
+ DisplayClear	: bool

CONSTRUCTORS

+ WndMain	void

GRAPH SCROLL VIEWER

This class represents a scroll viewer which allows an instance of the graph canvas to be moved when the spacebar key is pressed. It also makes sure that the canvas stays centered in the same position after the window has been resized.

PROPERTIES

+ bool IsTranslatable	: bool

CONSTRUCTORS

+ GraphScrollViewer	void

METHOD

+ ScrollToOffsetCenter	double, double	: void
+ ScrollToOffsetCenter	**Point**	: void

SENSOR CONTROL

This class represents a sensor which can be added to the canvas and eventually be drawn to the screen. It works alongside an instance of a sensor object and updates itself automatically whenever it detects a change. This class controls all the visual components of a sensor and indirectly interacts with the user.

CONSTRUCTORS

+ SensorControl	**Sensor**

METHODS

+ ToggleGraphView	**Sensor**, bool	: void
+ AddConnection	**Sensor**, **Sensor**	: void
+ RemoveAllConnections	void	: void

GRAPH CANVAS

This class is responsible for handling the core interaction between the user and the graph system. It interprets the users' actions and translates them to the graph which can be outputted to the screen. This class also interacts with the main window in order to reflect any actions done by the menu system.

EVENTS

+ SelectionPropertyChanged	: **Action**<**Sensor**>

PROPERTIES

+ CountSelected	: int
+ IsModifying	: bool
+ IsSelecting	: bool

CONSTRUCTORS

+ GraphCanvas	void

METHODS

+ GetSelectedProperty	byte	: **Nullable**<float>
+ SetSelectedProperty	byte, float	: **Nullable**<float>
+ RandomPathSource	void	: void
+ RandomPathTarget	void	: void
+ ReverseShortestPath	void	: void
+ SelectAll	void	: void
+ Clear	void	: void
+ ClearSelection	void	: void
+ ToggleGraphView	bool	: void
+ ToggleShortestPath	bool	: void
+ Randomize	**Range**, **Range**, **Range**, **Range**, bool	: bool

SENSOR

This class represents a sensor object which is responsible for defining all data pertaining to a sensor on a graph. In addition, it also allows other objects to be notified of any changes by subscribing to the desired events.

EVENTS

+ PropertyChanged	: **Action**<**Sensor**>
+ MouseOver	: **Action**<**Sensor**>
+ Selected	: **Action**<**Sensor**>

PROPERTIES

+ X	: float
+ Y	: float
+ Angle	: float
+ Range	: float
+ Orientation	: float
+ IsMouseOver	: bool
+ IsSelected	: bool
+ Connected	: **IEnumerable**<**Sensor**>
+ Control: SensorControl	

CONSTRUCTORS

+ Sensor	void
+ Sensor	float, float
+ Sensor	float, float, float, float, float

METHODS

+ GetProperty	byte	: **Nullable**<float>
+ SetProperty	byte, float	: void
+ Distance	**Sensor**	: float
+ Distance	float, float	: float
+ DistanceSquared	**Sensor**	: float
+ DistanceSquared	float, float	: float
+ AngleBetween	**Sensor**	: float
+ AngleBetween	float, float	: float
+ Contains	**Sensor**	: bool
+ Contains	float, float	: bool
+ AddConnection	**Sensor**	: void
+ RemoveAllConnections	void	: void

OBJECT

+ ToString	void	: string
+ Equals	**Sensor**	: bool
+ CompareTo	**Sensor**	: int
+ GetHashCode	void	: int
+ Equals	object	: bool

LINE GRAPH

This class represents a line graph which is responsible for displaying results following a statistical analysis. It also interacts with the user such that additional information about the tests conducted can be viewed.

CONSTRUCTORS		
+ LineGraph	void	
METHODS		
+ Update	**Statistics**	: void

A STAR NODE

This class represents a node which is used exclusively by the A* algorithm to determine the shortest path from one sensor to another. This class is only created for sensors visited during the pathfinding operation.

FIELDS		
+ Sensor	: **Sensor**	
+ Parent	: **AStarNode**	
+ Visited	: bool	
+ FScore	: float	
+ GScore	: float	
+ HScore	: float	
CONSTRUCTORS		
+ AStarNode	**Sensor**	
ICOMPARABLE		
+ CompareTo	**AStarNode**	: int

GRAPH

This class represents a graph which contains a list of sensors. It also defines all the algorithms necessary to query and manipulate the sensors in the graph.

PROPERTIES

+ Count	: int
+ IsEmpty	: bool

GRAPH

+ Add	**Sensor**	: void
+ Remove	**Sensor**	: void
+ Clear	void	: void

ALGORITHMS

+ SelectRandom	void	: **Sensor**
+ SelectAt	int	: **Sensor**
+ SensorAt	float, float, float	: **Sensor**
+ SensorsIn	float, float, float, float	: **IEnumerable**<**Sensor**>
+ SensorsIn	**LinkedList**<**Point**>	: **IEnumerable**<**Sensor**>
+ RecomputeConnections	void	: void
+ FindPath	**Sensor**, **Sensor**	: **Stack**<**Sensor**>

IENUMERABLE

+ GetEnumerator	void	: **IEnumerator**

STATISTICS

This class represents the statistical analysis portion of the system. It is responsible for running various tests based on the input given and generates a set of data points which can be plotted on a graph.

PROPERTIES

+ Iterations	: int
+ Trials	: **Range**
+ Sensors	: **Range**
+ Angle	: **Range**
+ Distance	: **Range**
+ Width	: int
+ Height	: int

CONSTRUCTORS

+ Statistics	**Range**, **Range**, **Range**, **Range**
+ Statistics	**Range**, **Range**, **Range**, **Range**, int, int

METHODS

+ GetDataPoint	int	: **DataPoint**
+ PerformTests	void	: bool

DATAPOINT

This class represents a single data point which is created when statistical analysis is conducted on a graph. Depending on the number of iterations conducted, there may be multiple sets of data points generated.

PROPERTIES

+ Ratio	: double
+ Trials	: int
+ Sensors	: int
+ Angle	: int
+ Range	: int
+ Missed	: int
+ AvgDiff	: double
+ AvgDir	: double
+ AvgOmni	: double

CONSTRUCTORS

+ DataPoint	void

TEST CASES

This section focuses on the functionality of the system itself by using several different testing strategies and methodologies as outlined in the project requirements.

UNIT TESTING

Unit testing focuses on the objects and subsystems, i.e. the foundation/building blocks of the software system. The key components of unit testing are. For this project, we focused on black box testing techniques, namely equivalence and boundary testing, which are described as follows:

- EQUIVALENCE TESTING: This black-box technique assumes that systems behave in similar ways for all members of a class. To test the behaviour associated with an equivalence class, we only need to test one member of the class

- BOUNDARY TESTING: This special case of equivalence testing focuses on the conditions at the boundary of the equivalence classes.

These tests were performed on the Sensor Simulator Application which handles requests such as creating a network of omnidirectional and directional networks, finding the shortest path, performing statistical analysis and calculating the number of hops required by a sensor to communicate with another sensor. Each of the above actions is performed by interacting with the Sensor Simulator interface which provides the following functionality:

- Random Dialog
- Statistics Dialog
- Graph View Dialog
- Shortest Path Dialog
- Remove All Dialog
- About Dialog
- Full-screen Dialog

This section describes the various unit test cases which were carried out to test the individual components and the boundary conditions of the components implemented in the system.

U1: Create Random Sensor Network (Random Dialog)

- **U1-C1:** Number of sensors must be entered. Attempt to leave the field blank
- **U1-C2:** Input a range of values for angles. Attempt to leave field blank
- **U1-C3:** Input a range of values for orientation angles. Attempt to leave field blank
- **U1-C4:** Input a range of values of transmission range. Attempt to leave field blank

The test cases outlined above tests the functionality of the Random Dialog component of the application.

Test No.	Test ID	Initial State	Input	Expected Result
1	U1-C1-S1	Valid entry in field and display relevant description	Input a correct, reasonable integer value	Input is accepted and display random distribution of sensor nodes on the interface
2	U1-C1-S2	Valid entry in field and display relevant description	Leave field blank	No sensors are displayed and the dialog remains active
3	U1-C1-S3	Valid entry in field and display relevant description	Enter invalid data	Dialog does not accept invalid data and remains active until valid input is entered
4	U1-C2-S1	Valid entry in field and display relevant description	Input a correct range of reasonable integer values (e.g. 10-360)	Input is accepted
5	U1-C2-S2	Valid entry in field and display relevant description	Enter invalid data in field	Dialog does not accept invalid data and remains active until valid input is entered
6	U1-C3-S1	Valid entry in field and display relevant description	Input a correct range of reasonable integer values (e.g. 10-360)	Input is accepted
7	U1-C3-S2	Valid entry in field and display relevant description	Enter invalid data in any one field	Dialog does not accept invalid data and remains active until valid input is entered
8	U1-C4-S1	Valid entry in field and display relevant description	Input a correct, reasonable integer or correct range integer values	Input is accepted
9	U1-C4-S2	Valid entry in field and display relevant description	Enter invalid data in any one field	Dialog does not accept invalid data and remains active until valid input is entered

U2: Perform Statistical Analysis (Statistics Dialog)

- **U2-C1:** Number of trials must be entered. Attempt to leave blank or enter invalid input
- **U2-C2:** Input the number of sensors. Attempt to leave blank or invalid enter input
- **U2-C3:** Input a range of values for angles. Attempt to leave blank or invalid enter input
- **U2-C4:** Input a range of values of transmission range. Attempt to leave blank or enter input

The test cases outlined above tests the functionality of the Statistics Dialog component of the application.

Test No.	Test ID	Initial State	Input	Expected Result
1	U2-C1-S1	Valid entry in field and display relevant description	Input a correct, reasonable integer value	Input is accepted
2	U2-C1-S2	Valid entry in field and display relevant description	Leave field blank	Dialog remains active until valid data is entered
3	U2-C1-S3	Valid entry in field and display relevant description	Enter invalid data	Dialog does not accept invalid data and remains active until valid input is entered
4	U2-C2-S1	Valid entry in field and display relevant description	Input a correct range of reasonable integer values (e.g. 10-360)	Input is accepted
5	U2-C2-S2	Valid entry in field and display relevant description	Enter invalid data in field	Dialog does not accept invalid data and remains active until valid input is entered
6	U2-C3-S1	Valid entry in field and display relevant description	Input a correct range of reasonable integer values (e.g. 10-360)	Input is accepted
7	U2-C3-S2	Valid entry in field and display relevant description	Enter invalid data in any one field	Dialog does not accept invalid data and remains active until valid input is entered
8	U2-C4-S1	Valid entry in field and display relevant description	Input a correct, reasonable integer or correct range integer values	Input is accepted
9	U2-C4-S2	Valid entry in field and display relevant description	Enter invalid data in any one field	Dialog does not accept invalid data and remains active until valid input is entered

INTEGRATION TESTING

Unit test cases were useful for detecting faults within the individual components/subsystems. Once those faults have been eliminated, components are ready to be integrated into larger subsystems. This is where integration testing comes into play. This type of testing detects faults that have not been detected during unit testing by focusing on small groups of components. Initially, one starts by testing a minimum of two different components and then more components are added after the different tests have been performed. There are two major strategies for Integration testing:

- Horizontal integration testing strategies
- Vertical integration testing strategies

For the scope of this project, we decided to adopt the top down testing methodology, a style of horizontal integration testing. In Top Down testing, the components of the top layer are tested first. These components are then integrated with the components of the next layer down until all components in all layers are combined and tested as a whole. The following presents a list of all the components being used for testing purposes:

Test A – Sensor Simulator UI
Test B – Random Dialog
Test C – Statistics Dialog
Test D – Graph Dialog
Test E – Shortest Path Dialog
Test F – About Dialog
Test G – Full-Screen Dialog
Test H – Remove All Dialog

For simplicities sake, each test, as follows, has been labelled according to their alphabets.

TEST A

Test ID	**1**
Test Name	Launch Sensor Simulator Application
Component under test	UI
Purpose of test	To determine if application launches as expected and if it performs the required behaviour as implemented in the system
Input	S1 – Application is launched S2 – Application is closed S3 – User "right-clicks" the interface S4 – User holds "left-click" and moves mouse around the interface S5 – User "double left-clicks" the interface
Expected output	S1 – The UI is visible to the user S2 – The Application closes and does not save any data S3 – All the different dialogs are displayed on the interface which the user can interact with to perform various actions S4 – This initiates the "lasso" feature of the application. S5 – Creates a sensor

TEST A, B

Test ID	**2**
Test Name	Generate Random Sensor Network
Component under test	Random Dialog
Purpose of test	To determine if the dialog performs the correct actions as implemented in the system and how it behaves when invalid data is entered in the fields
Input	Click on Random button, display the relevant descriptions in the field and enter relevant data. Possible scenarios: S1 – Correct data is entered S2 – Invalid data is entered
Expected output	S1 – The dialog disappears and a randomly distributed network of sensors is generated and displays the coverage of each sensor with their various angles, orientation and transmission range. S2 – The dialog remains active prompting the user to enter valid data.

TEST A, C

Test ID	**3**
Test Name	Perform Statistical Analysis
Component under test	Statistics Dialog
Purpose of test	To determine if the dialog performs the correct actions as implemented in the system and how it behaves when invalid data is entered in the fields
Input	Click on Statistics button, display the relevant descriptions in the field and enter relevant data. Possible scenarios: S1 – Correct data is entered S2 – Invalid data is entered
Expected output	S1 – The dialog disappears followed by the display of a bar/line graph. The x-axis displays the number of trials. The y-axis displays the ratio of directional and omnidirectional hop counts. The graph also displays the average values for directional hop counts, omnidirectional hop counts when the mouse is hovered over each coordinate S2 – The dialog remains active prompting the user to enter valid data

TEST A, B, D

Test ID	**4**
Test Name	View Graphical mode
Component under test	Graph Dialog
Purpose of test	To determine if the dialog switches to a discrete graph view and displays the various connections between the sensor nodes
Input	Click on Graph button and make the sensor nodes static. Possible scenarios: S1 – User is now in Graph mode S2 – User attempts to move nodes S3 – User attempts to change/modify the properties of the sensor node
Expected output	S1 – The various angles and ranges disappears from the screen. The user now views the graph mode of the sensor network with all possible connections between its neighbours. S2 – The node remains static and does not move. S3 – The user is unable to change/modify the properties of the node.

TEST A, B, D, E

Test ID	**5**
Test Name	View Shortest Path
Component under test	Shortest Path Dialog
Purpose of test	To determine if the dialog displays the shortest path on the graph of the sensor network, and the hop count of the path
Input	User clicks on Shortest Path button, selects a starting point and selects an end point. User presses the “Enter” / “Return” key to select 2 random start/end points.
Expected output	The starting point is displayed as a “green” node and the end point is displayed as a “red” node. Initially, the hop count is displayed as “0” next to the “green” node. If a path exists, the path is displayed by a white line and the hop count is indicated next to the “green” node.

TEST A, F

Test ID	**6**
Test Name	Display Application Information
Component under test	About dialog
Purpose of test	To determine if the dialog presents the relevant information as implemented in the system
Input	User clicks the “?” button
Expected output	The dialog displays the logo and the names of the authors of the application.

TEST A, G

Test ID	**7**
Test Name	Full Screen View
Component under test	Full screen Dialog
Purpose of test	To determine if the dialog behaves correctly as implemented in the system
Input	User clicks the full screen button
Expected output	The dialog displays the logo and the names of the authors of the application.

TEST A, B, D, H

Test ID	**8**
Test Name	Remove All
Component under test	Remove All Dialog
Purpose of test	To determine if the dialog removes all instances of the sensor nodes on the interface
Input	User clicks the remove all button
Expected output	A dialog box pops up and prompts the user for a response

SYSTEM TESTING

While Unit and Integration testing focus on finding faults in individual components and the interfaces between components, a thorough system testing ensures that the complete system complies with the functional and non-functional requirements, as outlined in the document requirements. To meet the requirements of this project, we performed the following tests:

- **FUNCTIONAL TESTING**: A black-box technique, where the test cases are, generally speaking, derived from Use Case Models. In our case, we obtained our test cases from the requirements outlined in the document itself.

- PERFORMANCE TESTING: These tests were conducted to ensure it met our design goals.

- PILOT TESTING: This was performed by the various team members to ensure the system worked properly across different platforms. It also helped us to perform usability tests and to ensure that the system was used the way it was meant to be used and that the behaviour of the system was the same for everybody.
-
- INSTALLATION TESTING: Even though the system specifications were not provided to us, it was easy to perform an installation test since the main goal of this test is to ensure that the system works as intended when ported from the development environment (our personal computers) to the target environment (SCS lab machines).

For the purposes of this document, we only highlight the functional testing of the system since performance was not one of the main criteria due to the theoretical nature of the simulator. Pilot testing simply insured that the application was compatible across various systems and that the user experience was more or less the same throughout after performing a few usability tests to ensure the integrity of the application.

Test Case No.	**1**
Test Case Name	GenerateRandomNetwork_CommonCase
Entry Condition	The application and the UI are active and the user clicks the “Random” button on the interface
Flow of Events	1. The user opens the random dialog and inputs the relevant data in the textboxes 2. The UI shows the changes in the fields. 3. The user hits “enter”/”return” or clicks “done”. 4. The UI displays a random distribution of sensor nodes. 5. Every sensor displays its coverage area (shades of gray)
Exit Condition	The dialog box on the interface disappears and a random distribution of sensor networks are displayed on the interface

Test Case No.	**2**
Test Case Name	StatisticalAnalysis_CommonCase
Entry Condition	The application and the UI are active and the user clicks the “Statistics” button on the interface
Flow of Events	1. The user opens the statistics dialog and inputs the relevant data in the textboxes 2. The UI displays the changes in the fields 3. The user hits “enter”/”return” or clicks “done”. 4. The UI displays a line graph chart
Exit Condition	1. The user clicks outside the graph region 2. The dialog box disappears 3. The user clicks outside the graph region 4. The graph disappears

Test Case No.	**3**
Test Case Name	FindShortestPath_CommonCase
Entry Condition	The application and UI are active. The user has created/generated instances of a sensor network The user is in graph view
Flow of Events	1. The user selects a starting point and is indicated by a green node 2. Initial hop count is displayed next to the starting point 3. The user selects an end point 4. If a path exists, the path is traced and the hop count is updated accordingly 5. No path exists, but the start/end points are displayed 6. The user can choose to select 2 points at random by pressing the “enter”/”return” key
Exit Condition	The path is displayed on the interface

Test Case No.	**4**
Test Case Name	SensorRemoval_CommonCase
Entry Condition	The user clicks the remove all dialog
Flow of Events	1. The user launches the remove all dialog 2. The dialog prompts the user for an action 3. The user selects the appropriate action
Exit Condition	The dialog disappears

INTERFACE

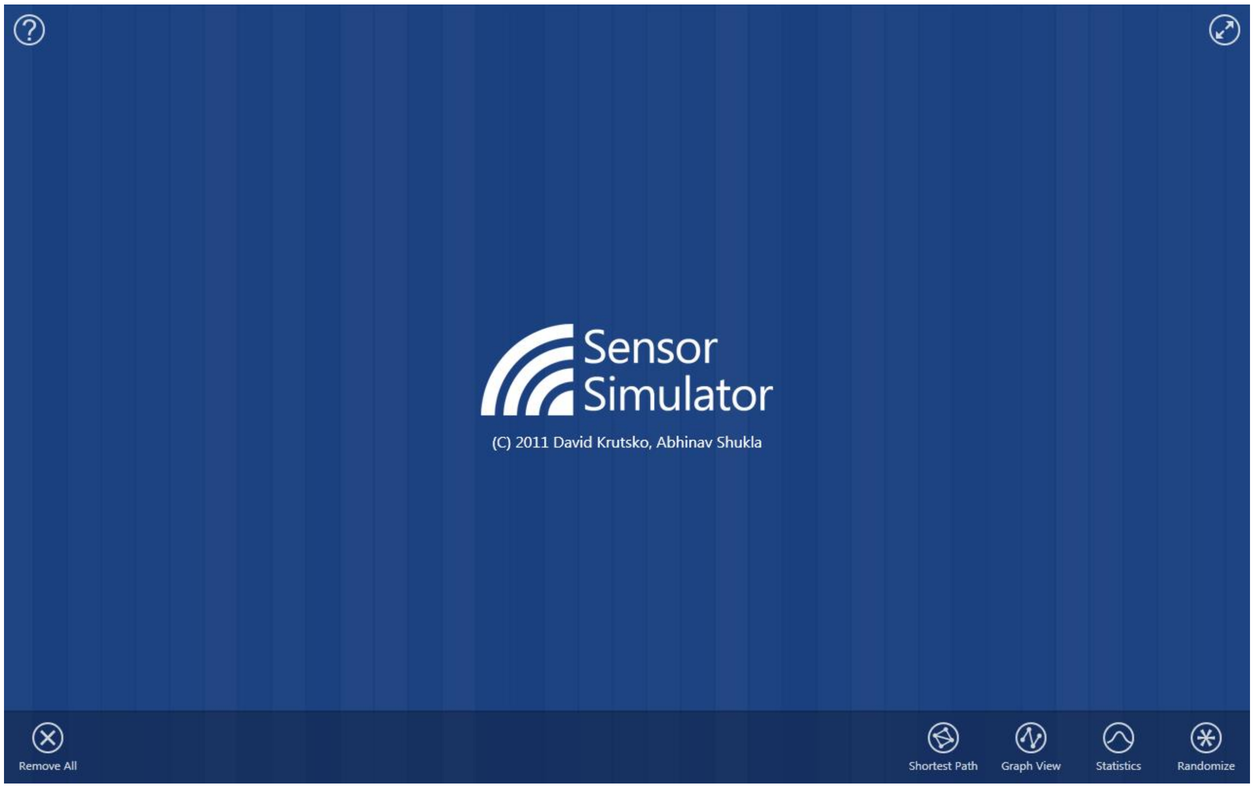
One of the major criteria of this project is to build an interface which the user can interact with. While it is extremely important to be able to perform an accurate analysis, it is also important for the user to be able to use the simulator in a user friendly manner. We decided to use the Windows Presentation Foundation (WPF) to achieve our goals. The main advantage is that WPF employs XAML, a derivative of XML, to define and link various UI elements. This helps the developer to focus more on the UI object models as opposed to using fancy, possibly complicated, animations. Since its part of the .NET Framework, we decided to build the system itself in C#.

This section is a step by step guide which will help familiarize the user with every aspect of the application. This will allow the user to take full advantage of all its powerful features. This section is split up into four subsections, each describing a set of topics.

- THE MENU: Introduces the user to the menu as well as preliminary topics.
- EDITOR: Discusses the basics of editing and moving around the canvas
- ADVANCED: Introduces more advanced operations such as randomize
- STATISTICAL ANALYSIS: Discusses the built-in statistical analysis tool

New users are encouraged to read this section from beginning to end as the layout is designed to introduce concepts at a steady pace and in the correct order.

THE MENU



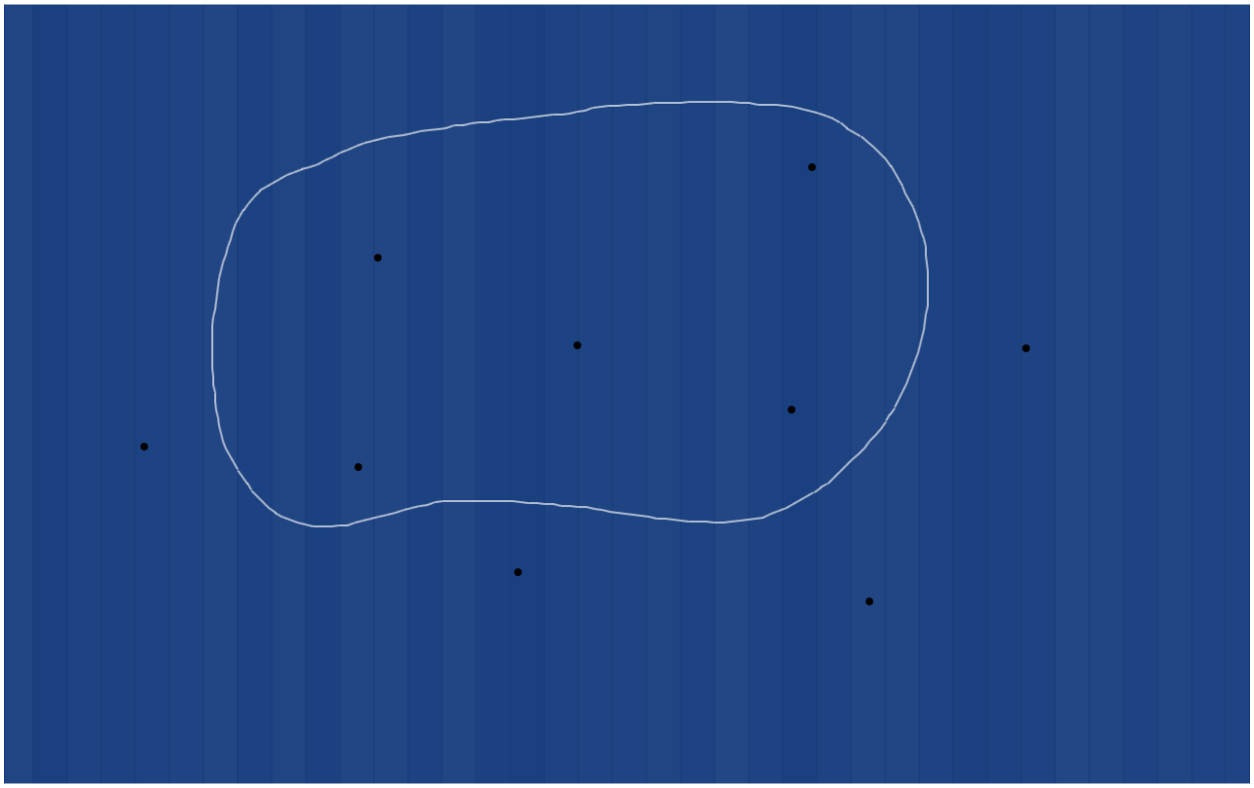
- When the application is started, you will be presented with a blue background; you can access the menu right away by pressing the right mouse button while hovering over the application. Press any mouse button, or ESCAPE key, to hide it.

- Pressing TAB when in a menu or submenu will shift your keyboard focus from one element to another. Focus will be rotated based on your current view, similar to any other application a user may be familiar with.

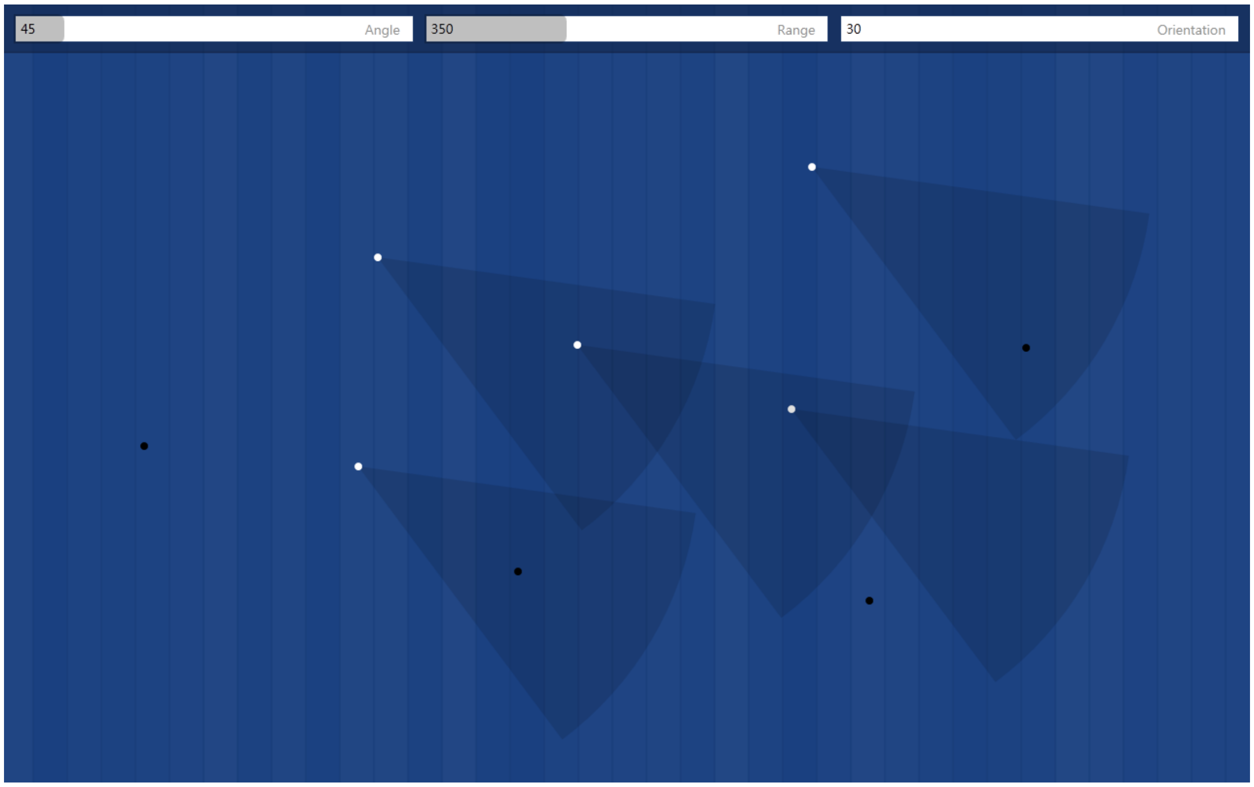
- There are five buttons on the main menu, located at the bottom of the screen along with an additional two at top.

- The first button, located in the top left, will display the application logo and authors while the button in the top right, will toggle the FullScreen mode. The rest of the buttons will be described in greater detail in later sections.

EDITOR



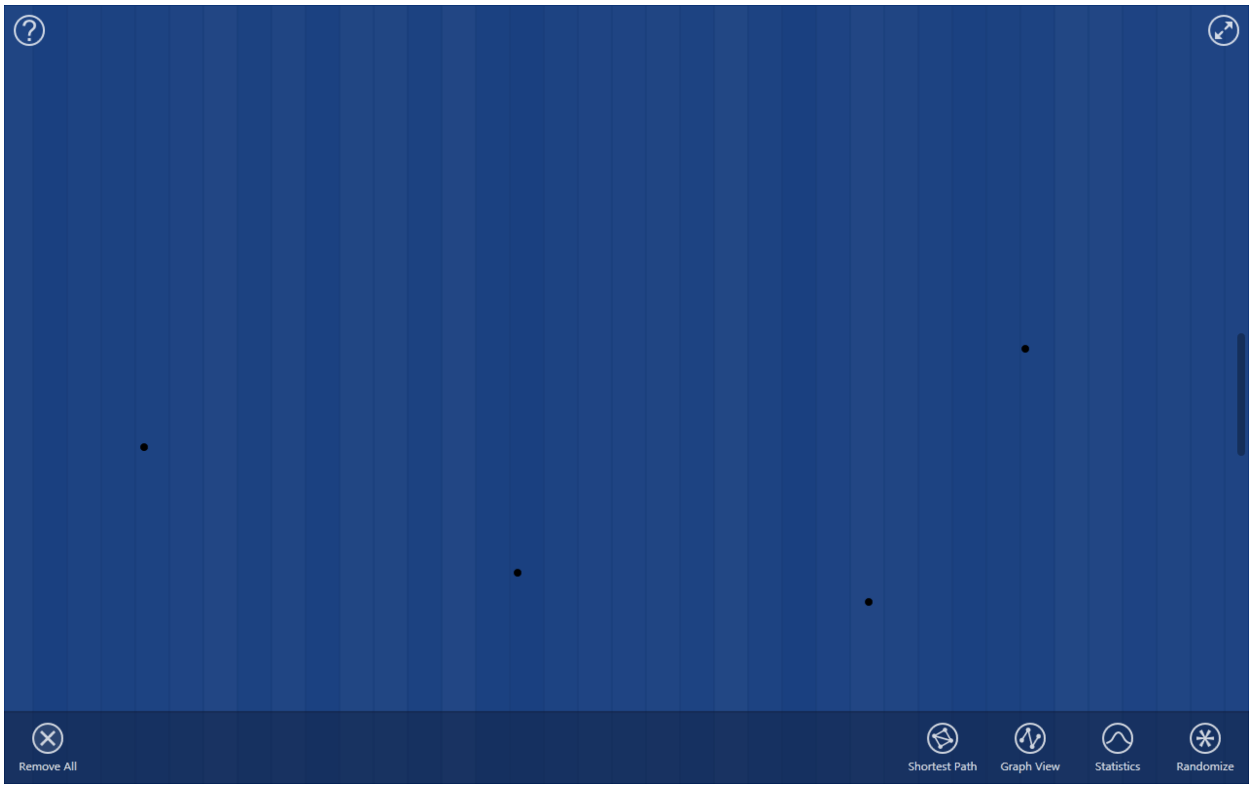
- To create a sensor, simply double left click anywhere on the editor.
- Selecting sensors can be done in multiple ways. You can individually select sensors by left clicking on them or you can hold down CONTROL key and left click together to select multiple sensors at once. Additionally, you can select all the sensors, displayed on the editor, by holding down CONTROL key and the A key. A user can also select the sensor(s) by using the lasso feature in the editor by pressing and holding the left mouse button over an empty region and dragging the mouse in the editor view. Letting go of the left mouse button will select all sensors inside the lasso. Left click over an empty region to deselect any currently selected sensors.
- In order to move a sensor, click and hold the left mouse button over a sensor and drag the mouse. You can move multiple sensors at once by selecting more sensors.



- Sensors can be modified by pressing the ENTER key when one or more sensors is selected. This initiates the editor mode which allows the user to edit the attributes of sensors. In editor mode there is a bar at the top which will display the current properties of selected sensors such as angle, orientation and transmission range

- While in editor mode, the user can change/modify the range an orientation by holding down the left mouse button over a node and dragging the mouse left or right. Holding down the right mouse button and dragging the mouse left or right will modify the angle of the sensor. If multiple sensors are selected, they have the same value as indicated in the figure above

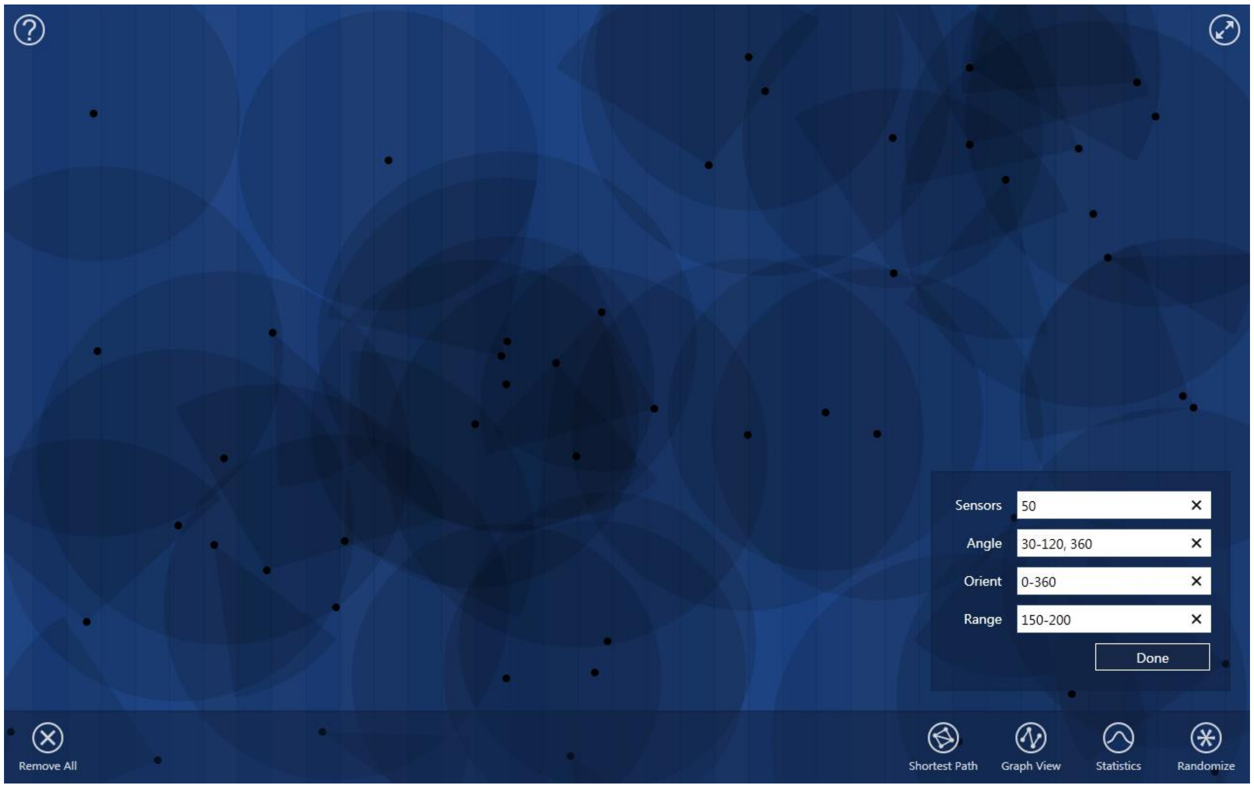
- A user can also modify sensor properties by using the Textbox feature. One has to simply hold down the left mouse button over one of the TextBoxes at the top of the window and drag left or right with the mouse to modify the attribute. Left clicking on the TextBox allows the user to edit values directly.



- Sensors can be deleted by pressing the DELETE key when one or more sensors are selected. Additionally, all sensors can be removed at once by pressing the Remove All button, located in the menu at the bottom of the screen. If one or more sensors in selected this button will change to Delete which can be used to delete selected sensors.

- To view the current position in the canvas, press and hold down SPACEBAR key. This triggers the Scrollbar feature which is displayed on the left and bottom parts of the screen. Holding down the left mouse button and dragging will allow you the user to perform panning.

ADVANCED



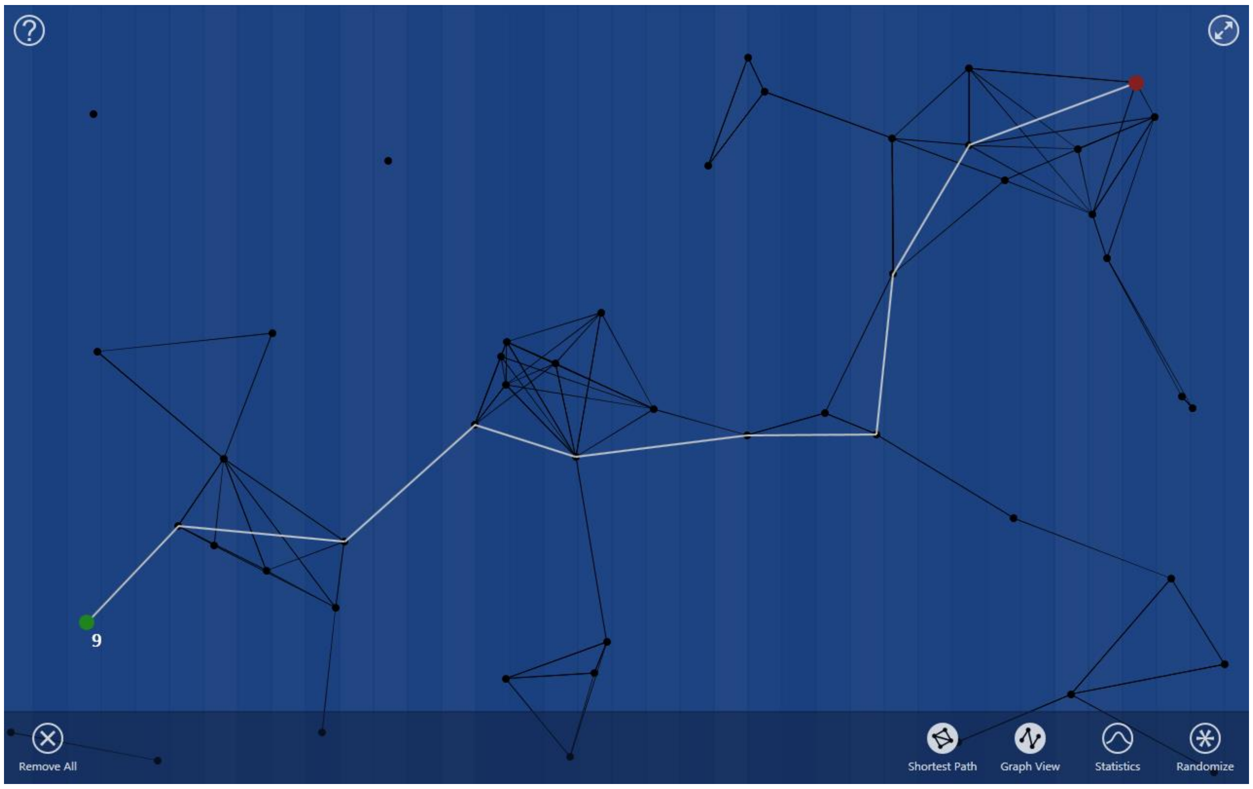
Sensors 50
Angle 30-120, 360
Orient 0-360
Range 150-200

- The Randomize button, located in the menu, can be used to generate a specified number of sensors with randomized attributes. Pressing the Randomize button displays a submenu allowing the user to enter various attributes (with constraints) to customize the randomization. Any field left blank will default it to zero.

- Data inside the randomize submenu can be entered as single numbers (e.g. 33), or as ranges (e.g. 23-42). Values can also be separated by commas resulting in even greater flexibility (e.g. 30-120, 150, 180-320, 360).

- If an input error occurs (e.g. Sensors is left blank) then no action will occur and the submenu will still be visible. If everything is successful, the submenu will disappear and the user is presented with a random sensor network graph on the canvas.

- Sensors will be generated within the users' current view of the canvas. This means that the bigger the application window, the more spread out the sensors will be.

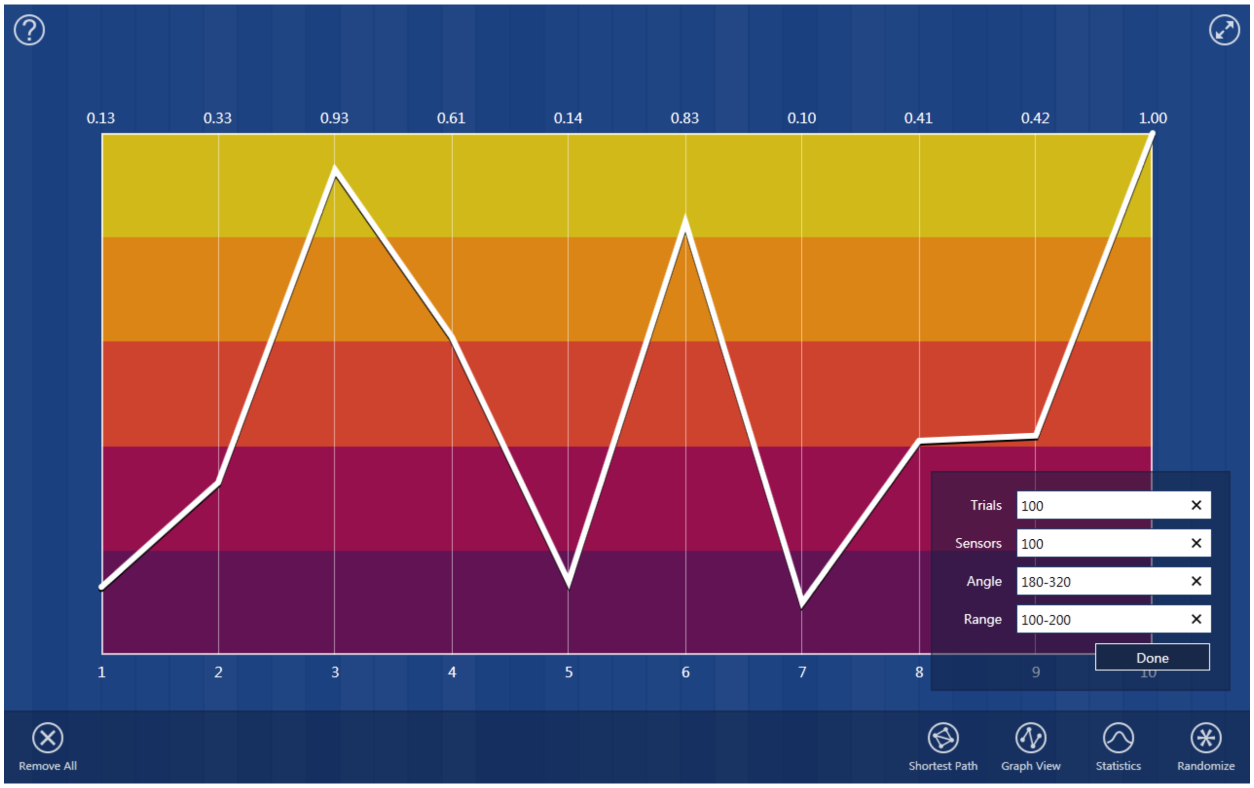


- Toggling the Graph View, located in the menu, will transition the graph to display sensor connections (based on directional and omnidirectional behaviour) as opposed to the sensor attributes. Selecting and hovering over sensors will display their attributes. This feature essentially displays the resulting Directed Graph.

- In order to find shortest path between two sensors the user must first toggle the Shortest Path button, located in the menu. While in this mode, pressing the left mouse button over a sensor will set it as the source (start point). Holding down SHIFT key and left clicking a sensor (at the same time) will set it as the target (end point). Once both the source and target have been set, the shortest path is computed and displayed as a white line. The number of hops is displayed below the source node (starting point).

- Pressing R key, while in shortest path mode, will flip the source and target nodes, reversing the path. Pressing ENTER key will choose two random nodes to be the source and target allowing you to find a random shortest path.

STATISTICAL ANALYSIS



Trials 100

Sensors 100

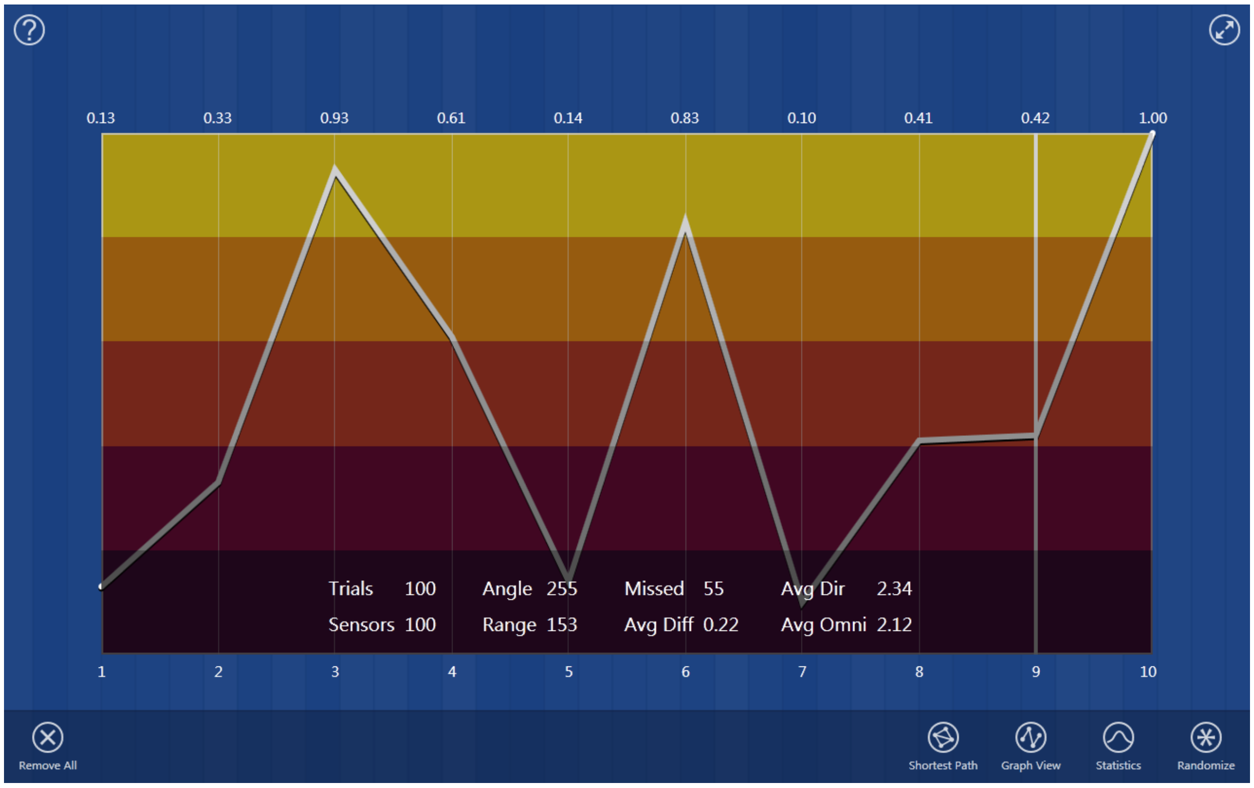
Angle 180-320

Range 100-200

- Statistical analysis can be performed quickly and efficiently using the Statistics button, located in the menu. Pressing this button will bring up a submenu allowing you to enter various attributes (with constraints) to customize the tests. Any field left blank will default it to zero.

- Data can be entered as numbers or ranges and can also be separated by commas (similar to the Randomize function).

- When the results of a statistical analysis are available, a line graph will pop up in the center of the screen displaying the data. The values at the bottom represent the number of iterations conducted (The number of iterations is always ten). The values at the top represent the ratios per iteration (Average number of Directional hops over the Average number of omnidirectional hops).



Trials 100 Angle 255 Missed 55 Avg Dir 2.34
Sensors 100 Range 153 Avg Diff 0.22 Avg Omni 2.12

- If the user hovers their mouse over the graph, additional information will be displayed at the bottom. This information is based on the iteration you have currently selected. Moving the mouse around will allow you to select a different iteration.

- Statistical averages can be run over and over again quickly since the menu will not automatically hide itself when the results have been computed.

- The menu and the graph can be hidden by either left clicking on an empty area in the background (similar to deselecting sensors) or by pressing the ESCAPE key.