Wi-Fi Signal Strength Mapping Application

[1]Hariharan Sundaram, [2]Zixiao Li, [3]Abhinav Choudhary [4]Souvik Dinda

[1] Software Engineering Systems, Northeastern University
[2]Software Engineering Systems, Northeastern University
[3]Software Engineering Systems, Northeastern University
[4]Software Engineering Systems, Northeastern University

{sundaram.ha, li.zixia, choudhary.ab, dinda.s}@northeastern.edu

***Abstract*—Wi-Fi signal strength mapping application is a project that aims to provide the most ideal spot to place the Wi-Fi router, ensuring the entirety of the house receives an adequate reception.**
The system enables users to enter their floor plan as a PNG image, modify certain parameters like frequency and material of the walls and then pick a spot to place the router, the system then takes those inputs and generates a heatmap image which displays the flow and reach of the signals. Users can then save the image in their local directory and proceed to either change the router position or start from the beginning with new floor plan or updated parameters.
Currently, the system offers 2.4Ghz as frequency, and cement & wood as available wall materials. In the PNG image depicting the floor plan, the empty spaces depict air whereas thick black lines and borders depict the walls.
An equation called "Helmholtz equation" defines the propagation of the electromagnetic waves, it includes various parameters like wavelength, refractive index of the material, time, etc. In our program, the time independent version of the equation is solved to get the results.
The refractive index of the materials is in the form of a complex number where the imaginary part provides the absorption rate of different materials. The greater the imaginary part of the refractive index, the lower the absorption rate of the material. For example, the refractive index of air does not contain an imaginary part, because of which the absorption is close to nil. The absorption rate of wood is higher than that of cement because of which the same floor plan with similar router position will depict vastly different results. The results generated with cement as wall material would show a much wider reach when compared to wood as wall material.
The user interface and interaction are done utilizing the Java and JavaFx, whereas the complex calculations are solved in C++. After the user provides all the parameters, the program generates a background thread and sends over the values to the C++ program to be processed. The C++ program returns a sparse matrix with electromagnetic wave strength values which is then used to generate a heatmap. The results are sent back and the background thread is merged with the main thread to proceed with the UI and display the result.
In the end, the users can generate and save multiple images at different router positions with different materials and compare all the results to determine which is the perfect location to place their router in.

Keywords*—*Wi-Fi simulation/ JavaFX/ JNI/ eigen/ linear system/ sparse matrix

I. Problem Description

A common problem for people moving to a new large house is that the Wi-Fi router may not cover all corner of the house. In some remote room or some room built with special material will have very weak Wi-Fi signal. The only way to fix this problem is to constantly change the place of your router and test where is the best location, which is a very annoying process.

Our goal is to write an app that can simulate the Wi-Fi distribution in a room providing changeable parameters so that it can be used in different scenario. Another benefit of a simulation is we can not only see the Wi-Fi strength and we can also analyze why the signal is strong or weak.

II. Analysis (Related Work)

A. *Algorithm*

Our application need a floor plan picture as input and get a picture of Wi-Fi strength as output. The strength will be expressed by the brightness of a certain color. Here is an example of input, black part will represent concrete wall and white place will represent air.

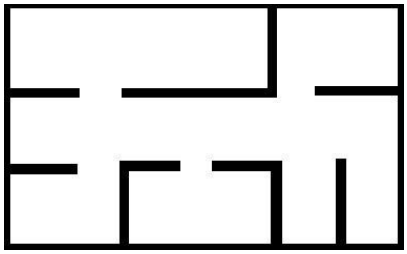

Figure 1. sample of input

To solve the problem above, we need to simulate using computation physics method. In our case, Wi-Fi signal is a electric-magnetic wave, so the equation we need to simulate is Helmholtz equation:

$$\frac{\partial E(x,y,t)^2}{\partial^2 x} + \frac{\partial E(x,y,t)^2}{\partial^2 y} - \frac{1}{c^2}\frac{\partial E(x,y,t)^2}{\partial^2 t} = f(x,y,t)$$

Here x means the x-label, y means the y-label and t means time. And E means the electric-magnetic wave strength, which is also known as the Wi-Fi signal strength. It is obvious our simulation is in a 2-dimension case.

Because light velocity is too fast for humans to notice, we can assume the final result is time-independent, which means the above equation can be write in a time independent form:

$$\left(\frac{\partial^2}{\partial^2 x}+\frac{\partial^2}{\partial^2 y}\right)E(x,y)+\frac{k^2}{n(x,y)^2}E(x,y)=f(x,y)$$

But computer can not solve an infinite small problem, we need to convert the above equation to a finite form:

$$\frac{E(i+1,j)+E(i-1,j)-2E(i,j)}{(\Delta x)^2}+\frac{E(i,j+1)+E(i,j-1)-2E(i,j)}{(\Delta y)^2}+\frac{k^2}{n(i,j)^2}E(i,j)=f(i,j)$$

In above equation i means the x-label index, j means y-label index. Notice i and j are both integer, which means we already change this problem into an finite form.

To explain what this equation means to people not familiar with mathematics in a brief way, what we going to do is to divide an input picture into lots of point, and calculate the signal strength at each point. For this purpose, a png file will be a good input, because png file is naturally divided into pixels. For example, if we have an input like Figure 2, and this box means a pixel in the png picture, then we will divide the floor plan into 4*7 boxes and calculate the Wi-Fi strength value of each box. And image we divide the whole png picture into small enough boxes, the result will be the a Wi-Fi strength distribution.

Figure 2. sample of png picture pixel

A classic approach is to covert this partial differentiation equation to a linear matrix operation, which computer is very good at solving. To explain this, we can write the equation we are going to solve as:

$$\left[\left(\frac{\partial^2}{\partial^2 x}+\frac{\partial^2}{\partial^2 y}\right)+\frac{k^2}{n(x,y)^2}\right]E(x,y)=f(x,y)$$

We can change the whole $\left[\left(\frac{\partial^2}{\partial^2 x}+\frac{\partial^2}{\partial^2 y}\right)+\frac{k^2}{n(x,y)^2}\right]$ into a matrix called A. And the $E(x,y)$ will be changed into a matrix called B. The right hand part $f(x,y)$ will be changed into a matrix called C. Then the differential equation above will be converted into a matrix calculation A*B=C. We can use Figure 3 to explain this.

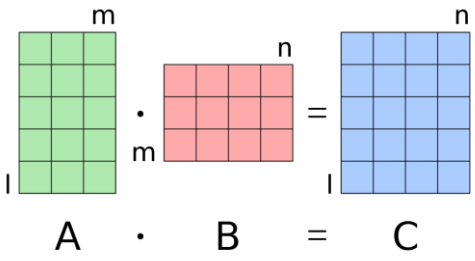

Figure 3. The matrix form of the differential equation

But this is a problem that computer is not good at solving, because this is a matrix-matrix operation. We need to change it into a matrix-vector operation, which is known as solving linear system [1]. To explain this, we will convert matrix A into another matrix A, matrix B into a vector x and matrix C into a vector y. Figure 4 is used to explain this.

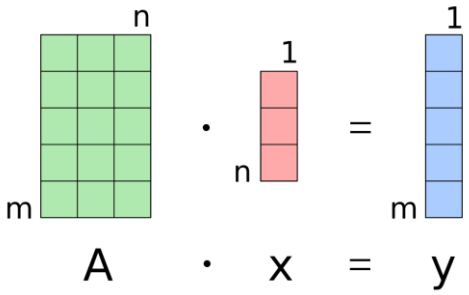

Figure 4. Linear system form of the differential equation

Because the mathematic operation in this part is too complicated and not the main thing in a software design, we just skip it and show the final result in Figure 5.

$$\begin{pmatrix} \vdots & \vdots & \vdots & \vdots & & \vdots & & \vdots & \vdots & \vdots & \vdots \\ \cdots & \frac{1}{\Delta x^2} & \cdots & \frac{1}{\Delta y^2} & -\frac{2}{\Delta x^2}-\frac{2}{\Delta y^2}+\frac{k^2}{n(i,j)^2} & \frac{1}{\Delta y^2} & \cdots & \frac{1}{\Delta x^2} & \cdots \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{pmatrix} \begin{pmatrix} \vdots \\ E(i-1,j) \\ \vdots \\ E(i,j-1) \\ E(i,j) \\ E(i,j+1) \\ \vdots \\ E(i+1,j) \\ \vdots \end{pmatrix} = \begin{pmatrix} \vdots \\ f(i-1,j) \\ \vdots \\ f(i,j-1) \\ f(i,j) \\ f(i,j+1) \\ \vdots \\ f(i+1,j) \\ \vdots \end{pmatrix}$$

Figure 5. Final linear system equation

A very important issue here is matrix A is a very giant matrix. For example, Figure 1 is a png picture with a resolution of 381*236, then the matrix A will be a matrix of 89916*89916 (381*236=89916). It is impossible for a normal PC computer with a 8GB or 16GB memory to store it. But what is lucky is that the matrix is a sparse matrix, and only 0.00625% of the matrix has none-zero value. Which means we can use a special

data structure known as sparse matrix to store it. And In this particular data structure, the memory will only store the none-zero value. Figure 6 is a picture of a sample sparse matrix of a 64*64 size. And the blue dots mean those place will none-zero value.

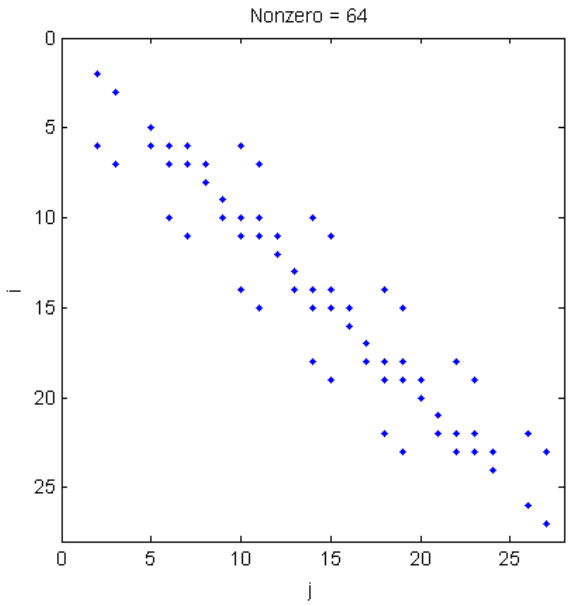

Figure 6. a sample sparse matrix of a 64*64 size

Now we successfully changed the Helmholtz equation into a problem computer can solve, which is known as solving linear system in computer science. And it is a problem of solving a matrix equation of A*x=b. And we know A and b, we need to get x.

B. *Difficulty encountered and how we solved it*

Because our algorithm is strongly related with the biggest difficulty we encountered, so we write a sub section to explain the difficulty and how we solved it.

In the linear system we need to solve, matrix A contains complex number. That is because Helmholtz equation contains a parameter called refractive index. For air, the value is 1. For concrete wall the value is 2.55-0.084i [2]. Here i means the imaginary part of the refractive index. And the imaginary part in physics represent the absorption effects the concrete wall has to Wi-Fi signal. And it is the reason why the Wi-Fi signal penetrating the concrete wall will be weaker.

But We faced huge difficulty to solve this problem. The matrix equation we need to solve need to be a sparse matrix contains complex number, which is a very high requirement. We searched several java packages to solve this but unfortunately none of them can solve this problem. And later we tried another two approaches, one is to call Julia from java and one is to integrate java with C++. That is because these two languages has much stronger support to solve linear system compared to java.

Here is a detailed process of how we tried to solve the problem.

1. Native java packages.

We searched for almost all open-source java packages that is used for sparse matrix calculation. And in this website [java-matrix.org)](java-matrix.org) [3] there is a list of java packages used for sparse matrix calculation and the function they support. Figure 6 is a screen shot of it.

	Colt	Commons Math	EJML	JAMA	jblas	JLinAlg	JMathArray	JMatrices	JSci	JScience
Current Version	1.2.0	3.2	0.25	1.0.3	1.2.3	0.6		0.6	1.5.2	4.3.1
Latest Release	2004	2013	2014	2012	2013	2009	2008	2004	2009	2007
License	BSD	Apache	Apache	PD	BSD	GPL	BSD	LGPL	LGPL	BSD
Supports Java 1.4	✓	✗	✗	✓	✗	✗	✗	✓	✓	✗
Supports Java 5	✓	✓	✗	✓	✓	✗	✓	✓	✓	✓
Supports Java 6	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Supports Java 7	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Supports Java 8	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
Stores Dense Data in Single Array	✓	✗	✓	✗	✓	✗	✗	✗	✗	✗
Stores Dense Data in 2D Array	✗	✓	✗	✓	✗	✓	✓	✓	✓	✓
Stores Dense Data in Block Storage	✗	✓	✓	✗	✗	✗	✗	✗	✗	✗
Stores Sparse Data	✓	✓	✗	✗	✗	✗	✗	✗	✗	✓

Figure 7.a list of java sparse matrix packages

And here is a list pf packages we tried:

Colt

Commons Math

EJML

Jblas

JSci

JScience

ojAlgo

Parallel Colt

Every package we tried announce that they can solve a linear system with sparse matrix containing complex number, but what we see is either they have strong limit on data structure or their solver class for sparse matrix containing complex number is an unfinished class. And this is a common thing in open-source packages. All members in our team agreed that java open-source packages cannot solve the problem.

2. Call Julia from java

After we believe java open-source packages do not support our need, we turned to connect java with Julia, because Julia is a language that is strong in mathematical operation. What we did is to call Julia from java using TCP communication. The process relies on a java package called juliacaller. The juliacaller work well if the data sent to Julia from java is small. We successfully call Julia function in Java. But when we send the whole sparse matrix in an array form into Julia, the application just got stuck and never recover. TCP communication itself does not have limit on how many data you send, so we guess a possible explanation is that the data sent to Julia will be pasted to keyboard first, and if the data is too big, the Julia in the backend just gets stuck.

3. JNI with C++

On deeper analysis, C++ was considered as the next option since Julia used C++ under the hood and it is a well agreed fact that code in C++ executes much faster than Java. Managing cross platform (Windows and MAC) setup for Java Native Interface (JNI), requiring external tools configuration in eclipse to generate header files for bridging Java and C++, was a challenge. The core logic was still taking more than 6 minutes to compute inspite of using C++ packages. This was accelerated by adding NDEBUG pre-processor flags and applying 03 optimization for speed flag to the C++ compiler.

III. System Design

After a detailed analysis and research, we started working on designing the flow of our application along with the required tools and object-oriented programming concepts that need to be implemented.

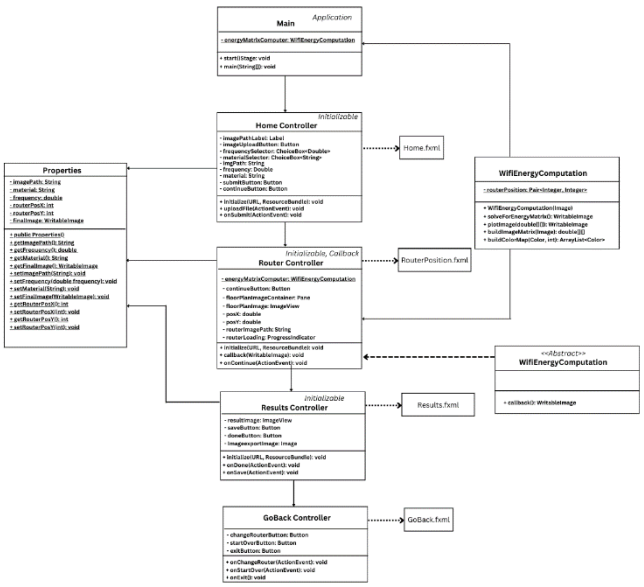

Figure 8. UML Class Diagram

After the first iteration of designing the UML, we started working on the flow of the application. It mainly comprises of the flow from user perspective, that is when a user lands on the home screen of our application what all options and actions user can perform to get the results and make the most of our application. We focused on keeping the flow as simple as possible so that even a layman can use the application easily.

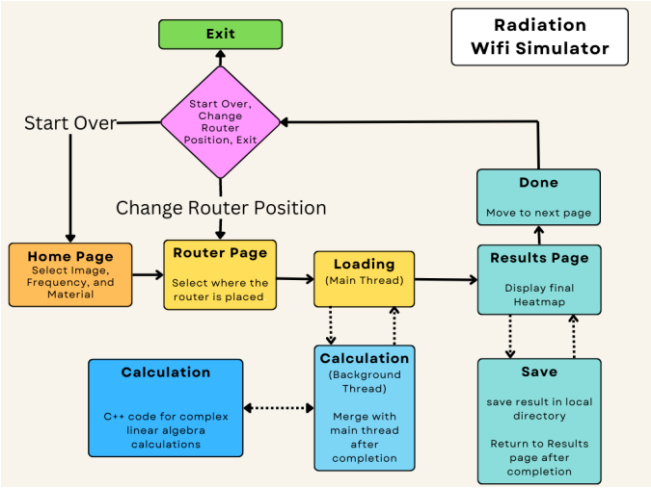

Figure 9. Flow Diagram

After the flow diagram the final part of designing was to create layout of the UI. For this we created wireframes of our UI screen. Below are the wireframes which were used as base in creating the application UI.

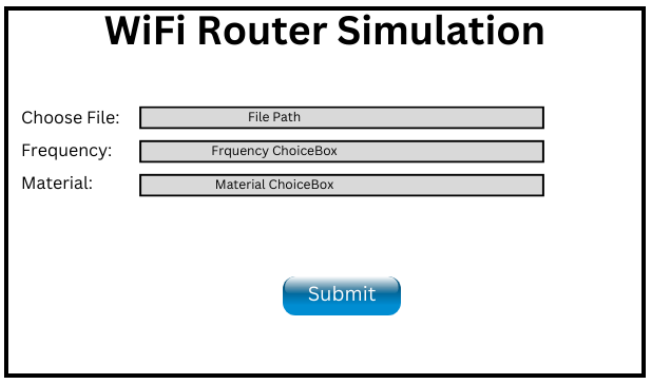

Figure 10. Home Screen Wireframe

In this screen, user can choose the floor plan which needs to be in .png format. Until the file is chosen, 'Submit' button shall be disabled Here user can also choose different types of materials of the wall and the frequency bandwidth of the Wi-Fi Router.

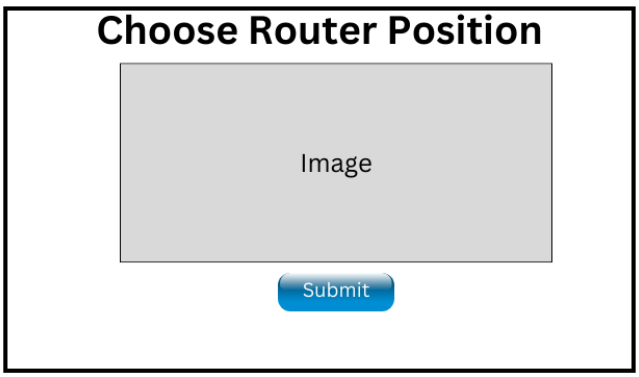

Figure 11. Choosing Router Position Wireframe

After choosing the router position user shall get the result which comprises the heatmap of Wi-Fi Signal strength. This result should be on top of the original floor plan image.

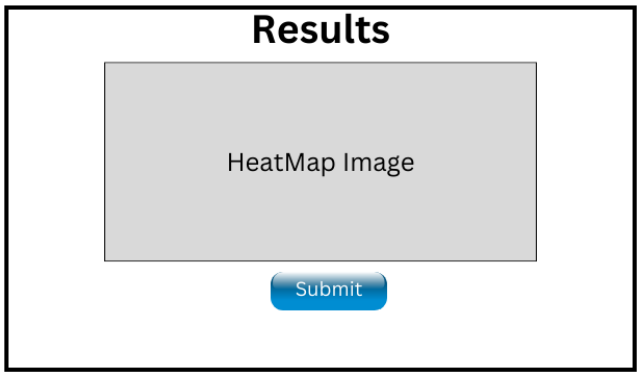

Figure 12. Result Screen Wireframe

And finally, the user can change the router position and recalculate the result or go to the home page and choose a new floor plan or can exit the application.

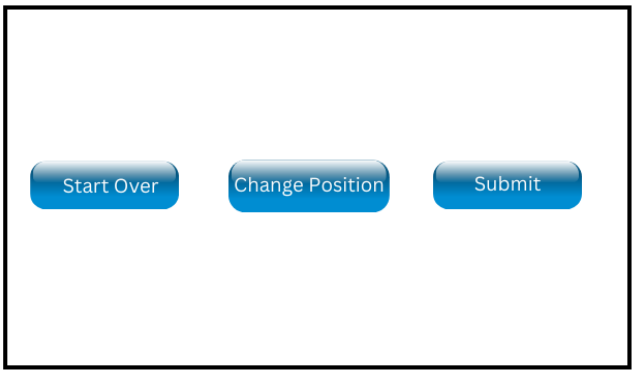

Figure 13. Final Screen Wireframe

IV. Implementation

A. C++ Compiler and Eclipse IDE Setup

A C++ compiler such as MinGW is required to be installed in Windows operating systems. This compiler needs to be made available throughout the system by adding it to System Environment. MAC has GCC toolchains installed by default. Eclipse C/C++ IDE CDT needs to be installed to support development of C++ project.

B. Eigen C++ library

Eigen C++ template library for linear algebra(version 3.4.0) has to be downloaded, unzipped and added to an accessible location. The path to the Eigen headers folder in the downloaded package needs to be added to C++ compiler includes. Also, the includes headers path to JDK needs to be added to the C++ compiler includes.

C. Signal Strength Computation Algorithm

The core logic for Wi-Fi signal strength computation has been implemented using the Eigen Linear Algebra library. The complex sparse matrices use the SparseMatrix<double> class from SparseCore package. An array of triplets having sparse matrix coordinates and values is built. This is pushed into the sparse matrix in one operation using setFromTriplets function. This accelerated the sparse matrix construction process which was very slow otherwise with regular iteration. The sparse matrix linear equation was solved using SparseLU<SparseMatrix<complex<double>>>
Decomposition solver and the resulting vector was reshaped into a two-dimensional image matrix using the reshaped function which is a column major vector to matrix converter.

D. Java Native Interface (JNI)

A class having the interfacing methods, EnergyComputationAlgoNative was created on the Java project. The interfacing function has the keyword ‘native’ to help java compiler understand that the function needs to be added to JNI bridging headers. External tools configuration needs to be added specifying the path to Java compiler, the destination of the bridging header file and the source file having the bridging functions with native keywords in the Java project. On running the external tools configuration, JNI bridging header file is generated. Implementation is written for this header on C++ project where the input parameter objects are mapped to C++ data structures. In our case, a two-dimensional double matrix, double values and integers were the parameters passed. This bridging function should also map the C++ data structures back to JNI objects while returning from the function.

Header files were also written on the C++ project for code segregation. The actual algorithm was written inside another C++ class called EnergyComputationAlgo. An instance object of this class was created and the function for computing was invoked from the JNI bridging function.

The C++ compiler needs to be configured to generate dll or dylib shared dynamic libraries (based on the OS) instead of the standard executable file. Also, NDEBUG flag has to be added to C++ pre-processor and -O3 optimisation needs to be added to the compiler flags in the C++ compiler settings. Upon building the C++ project, the generated dynamic library is saved to the configured destination.

E. Java JNI Integration

The generated C++ dynamic library is copied and added to the Java project workspace. This library is dynamically loaded using the System.load function. Whenever a call to the function with native keyword (in our case computeWifiEnergy) is made, Java dynamically loads this library and calls the matching

function on the JNI bridging header. This in turn invokes the C++ function and we retrieve the results. The resulting matrix values are rescaled to a range of 1-100 in order to map them to different color values. A color map of hundred colors using a base color specified is created and final matrix is built by mapping this to the matrix. This matrix is converted back to an image by PixelWriter.

F. Multithreading

The energy computation algorithm is a heavy process as there is a matrix decomposition operation involved. This makes it a slow process and running it on main thread causes the UI to freeze. In order to make UX better, the heavy computation was switched to a background thread using the CalculationThread class which implements Runnable interface. Once the calculation operation is complete and a response is received from the C++ dynamic library, we have to continue the UI operation through a callback function. For this, a Callback interface is written and the class calling this thread operation(RouterController) implements the callback interface function. The callback function is invoked from inside Platform.runLater block to run the UI operations again on the main thread.

G. JavaFX UI

The UI is created using .fxml files and .java controller files. The fxml files are updated using Scene Builder by dragging and dropping various elements. These elements have been assigned an Id, and linked to certain controller methods at various events like mouse click. Saperate controller files are created corresponding to each page and each .fxml file. The .fxml files are linked to their respective controller files and a .css file to implement design of the application. A "Properties" class file is created which contains static data fields and methods, the different controller use the static getter and setter methods to update or retrieve the values present in the data fields. Finally, everything is wrapped using a scene and stage. The movement between different screens is achieved by updating the scenes with the root of different .fxml files and setting the stage with the new scene.

V. Evaluation

Here is some screen shots of our application in real run time.

Figure 14. home page

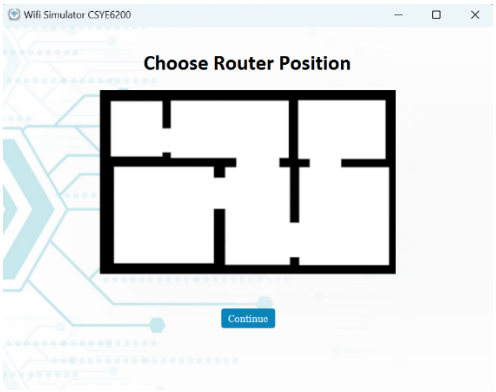

Figure 15. after chose floor plan picture

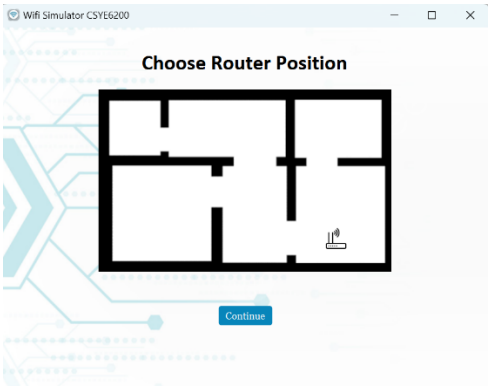

Figure 16. after chose Wi-Fi router position

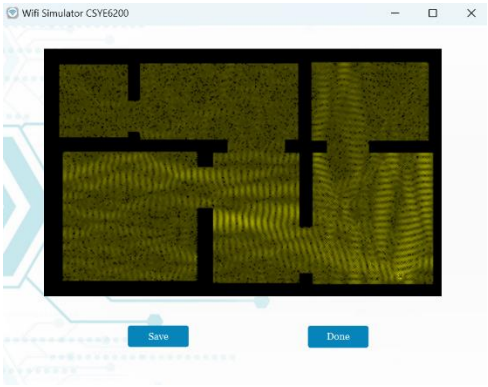

Figure 17. result

Figure 18. choice after saving your result

VII. Discussion (Reflection)

In this section we will show some sample runs and analyze their meaning in reality and in physics. And our analyze will prove our application is right because it show the real effect of physical world.

A. *Deflection behavior and wave path*

Figure 19 is a sample run that perfectly shows the reflection behavior and the path the signal transmits. We put the Wi-Fi router in the left corner room, and you can clearly see the room has very strong signal. That is because the signal is deflected over and over again between that walls and most of the signal just remain in the room.

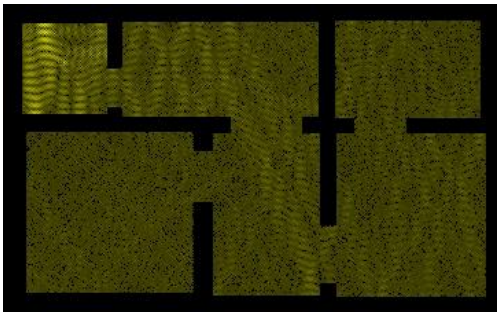

Figure 19. sample run frequency:2.4GHz/ material: concrete

And you can see how the signal goes into other rooms. The most part of signal just go through the air in straight line into other rooms. But it is hard to see it. So we added figure 11 to tell you which path the signal goes. You can compare figure 19 and figure 20.

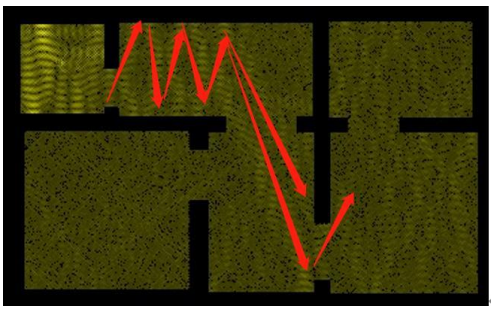

Figure 20.the transmission path of Figure 10

B. *Refraction behavior*

Our application also shows how Wi-Fi signal's refraction behavior. can penetrate concrete walls. But definitely the signal will be much weaker after penetrating the wall.
Figure 21 is the result of putting the Wi-Fi router in a totally sealed room. You can see there is still some signal in other rooms, which proves the signal can penetrate the concrete wall.

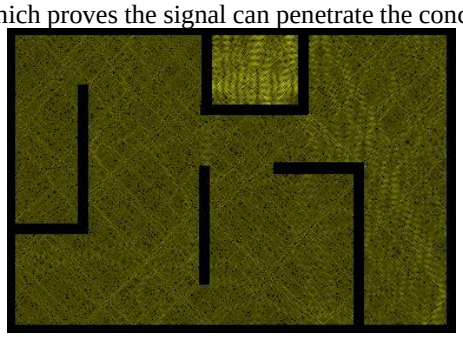

Figure 21. put the Wi-Fi router in a sealed room

B. *Analyzing house structure*

Our application also shows why some house structure is not good for signal to transmit.

Figure 22 is a very classic American house. It is a corridor in the center and house is distributed in two sides. We put the Wi-Fi router in the corridor.

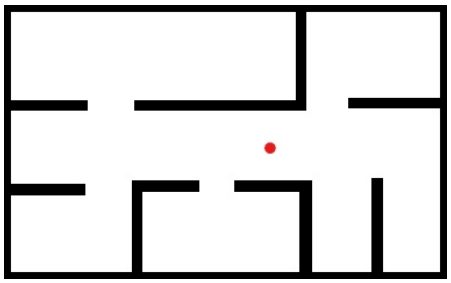

Figure 22.sample room with corridor

And the result is in figure 23. You can see most of the signal is restricted in the corridor. In fact such structure is not good for Wi-Fi signal to transmit.

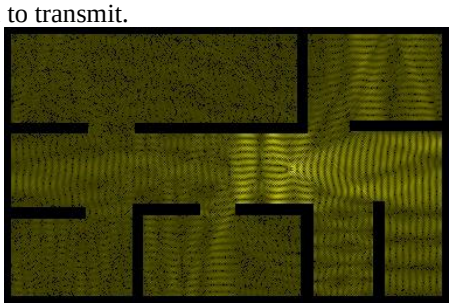

Figure 23.sample room with corridor

We can see this in figure 24. Even if we put Wi-Fi router before the door of a room, the result is signal can only cover two rooms. What suggestion we can give to the house owner of such structure is to buy two routers…

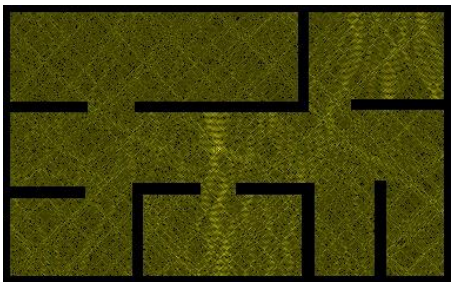

Figure 24. put Wi-Fi router before the door

VIII. Conclusions and Future Work

A. *Advantage of using our application*

Our application can give a reliable result and the result can be saved to users' computer correctly. User can run several time and compare which result is the best easily find a best place to put their router.

B. *Problems not solved*

Initially we tried to give users the freedom to choose which Wi-Fi frequency they want to simulate. Because the real Wi-Fi frequency bands is 2.4 GHz and 5 GHz. But our application can not give a right result of 5 GHz signals. Figure 25 is a result of setting Wi-Fi frequency to 5GHz.

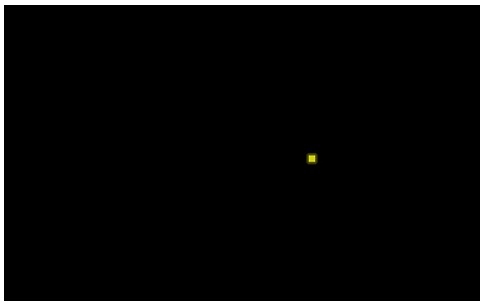

Figure 25. set Wi-Fi frequency to 5GHz simulation breaks

We analyzed why our application break down in this scenario. That is because signal with 5GHz frequency will have a wave length of 6cm, and in our simulation a pixel represents a 3cm*3cm room. But when the wave length is too close to the resolution, some physical part of our simulation will be fundamentally wrong. If we want to solve this, we need to let a pixel represent a smaller cell. But that will make the sparse matrix we calculate even larger. The run time will explode in a n^2 way. So unless we can run our application in a super computer, or the run time will be much longer if we want better accuracy.

C. *Future scope*

For future scope we want to give user more choice of Wi-Fi frequency, material and number of routers. And we also want to improve our algorithm to accept input such as figure 26. We can read more color and let them represent more material. For example, black means concrete wall and red means wooden doors. This will add more accuracy and do not sacrifice run time.

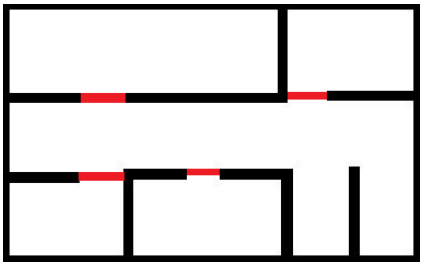

Figure 26. sample input with two materials, concrete and wood

IX. Job Assignment

- Zixiao Li: Designed the parse matrix solving algorithm, writing several java packages testing files and implementing connecting java and Julia using TCP communication and help finding a fast solver in C++
- Hariharan Sundaram: Java Native Interface(JNI) implementation with bridging headers, Eclipse configuration for External Tools, C++ compiler and project settings.

Implemented C++ core computation algorithm with Eigen package integration. Added Java native class for JNI bridging.

- Abhinav Choudhary: UI screens (fxml and controller files), Movement between various screens, Uploading and Saving of the images, Migration of C++ code in windows, Multi-threading, Java jar implementation.
- Souvik: UML Class Diagram, Wireframes, ImageView implementation for picking router position, HeatMap creation class (Expects result matrix and creates a WritableImage consisting of HeatMap on top of original image), added CSS and styling of UI, added custom icon.

X. How to run our app

Steps to run Java Jar file (Windows)

Step 1: Open command prompt (cmd) at the location where the jar file is located.
Step 2: Enter the following command:

java --module-path "location-of-javafx-sdk\lib" --add-modules javafx.controls,javafx.fxml,javafx.swing -jar nameOfExecutable.jar

For example:
java --module-path "D:\Northeastern\CSYE6200\Source\javafx-sdk-19\lib" --add-modules javafx.controls,javafx.fxml,javafx.swing -jar WifiSimulationExecutable.jar

References

[1] Nicholas J. Giordano, “Computational Physics”

[2] https://jasmcole.com/2014/08/25/helmhurts/?from=groupmessage&isappinstalled=0&scene=1&clicktime=1584852482&enterid=1584852482.

[3] K. Sato, T. Manabe, J. Polivka, T. Ihara, Y. Kasashima and K. Yamaki, "Measurement of the complex refractive index of concrete at 57.5 GHz," in IEEE Transactions on Antennas and Propagation, vol. 44, no. 1, pp. 35-40, Jan. 1996, doi: 10.1109/8.477526.

[4] Dense and Sparse Matrix Libraries for Java: An Overview (java-matrix.org)

[5] https://eigen.tuxfamily.org/index.php?title=Main_Page

[6] https://nbviewer.org/gist/fredo-dedup/31ae1b6017833e9a18f8#Setting-the-up-the-data

[7] https://www.youtube.com/playlist?list=PLWchVAowvRxCOFW7iJuWuMPNRf2eJ-IsO