PCB Auto-routing

Computer Science Coursework

Edwin Shepherd

# Contents

[Contents 1](#_Toc1889758)

[Analysis 2](#_Toc1889759)

[The Problem 2](#_Toc1889760)

[What is routing 2](#_Toc1889761)

[Different types of routing 3](#_Toc1889762)

[Users, Limitations and Requirements 4](#_Toc1889763)

[Research 5](#_Toc1889764)

[Interview 5](#_Toc1889765)

[Perquisites and missing requirements 5](#_Toc1889766)

[Objectives 5](#_Toc1889767)

[Prototyping 7](#_Toc1889768)

[Objectives 7](#_Toc1889769)

[What I found out 7](#_Toc1889770)

[Data structure 8](#_Toc1889771)

[Documented Design 9](#_Toc1889772)

[Program overview 9](#_Toc1889773)

[A Star 9](#_Toc1889774)

[Highlevel Routing 10](#_Toc1889775)

[User Interface 11](#_Toc1889776)

[UML Digramoverview 13](#_Toc1889777)

[Data structures 14](#_Toc1889778)

[Bibliography 15](#_Toc1889779)

# Analysis

## The Problem

### What is routing

Electronics engineers develop circuits which are often printed on to a circuit board (PCB), keeping wiring tidy. However routing the wires is a very tedious task. Additionally the wiring must conform to rules or else malfunctions in the circuit may occur. For example, high current tracks must be of a set width, certain components cannot have tracks under them. This is takes more time and is difficult to ensure 100% accuracy.

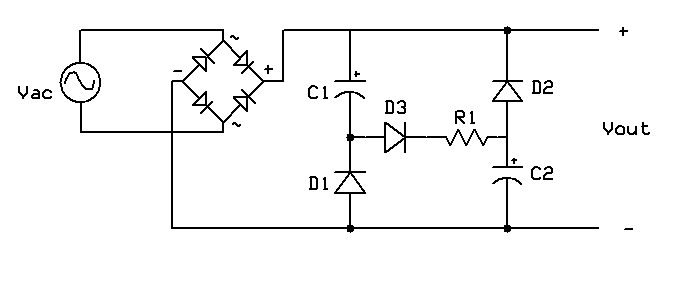
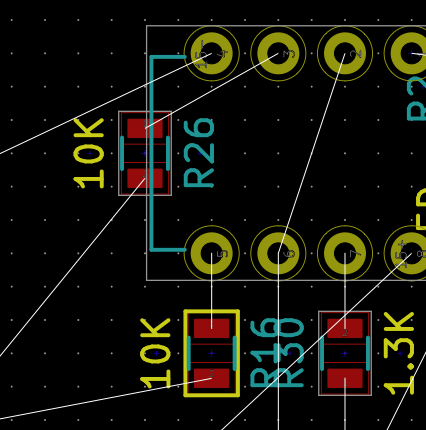
[](https://www.google.co.uk/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwjSqtfQ-NPdAhVIWBoKHcJ_DAsQjRx6BAgBEAU&url=https://en.wikipedia.org/wiki/File:Valley-fill_circuit_schematic_1.png&psig=AOvVaw1u6-1MKZyxtGEkY9-Jxwfn&ust=1537888868345021)To solve this problem PCB Routing Software is common place in PCB design tools. Its purpose is to automatically route all of the components and to perform “Design Rule Checking” DRC. PCB software usually has two modes one where the circuit is designed:

Figure 1 a schematic for a basic diode rectifier coupled with a valley-fill circuit - Image credit: WikiMedia available: <https://goo.gl/NJdaqG> , accessed 16:28 24/09/2018

[](https://www.google.co.uk/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=2ahUKEwj-0Oz0rqveAhUOThoKHfDXAwgQjRx6BAgBEAU&url=https://electronics.stackexchange.com/questions/200093/what-does-yellow-border-around-component-in-kicad-pcbnew-mean&psig=AOvVaw12652bQlXOE6Df1Zygvx7C&ust=1540892724447123)

Then another mode where the “nets”[[1]](#footnote-2) highlighted, these are the connections between components (in figure 2 they are blue) and then can be routed by the user (or by software). Finally it’s converted into a file that can be printed.

In this project I will solely focus on the challenges that are presented by the routing.

Figure 2 - Nets are shown in white straight lines. This screen shot is taken from KiCAD's PCBnew software. Image credit; Scanny available: <https://goo.gl/muyghy> date accessed 29/10/2018

### Different types of routing

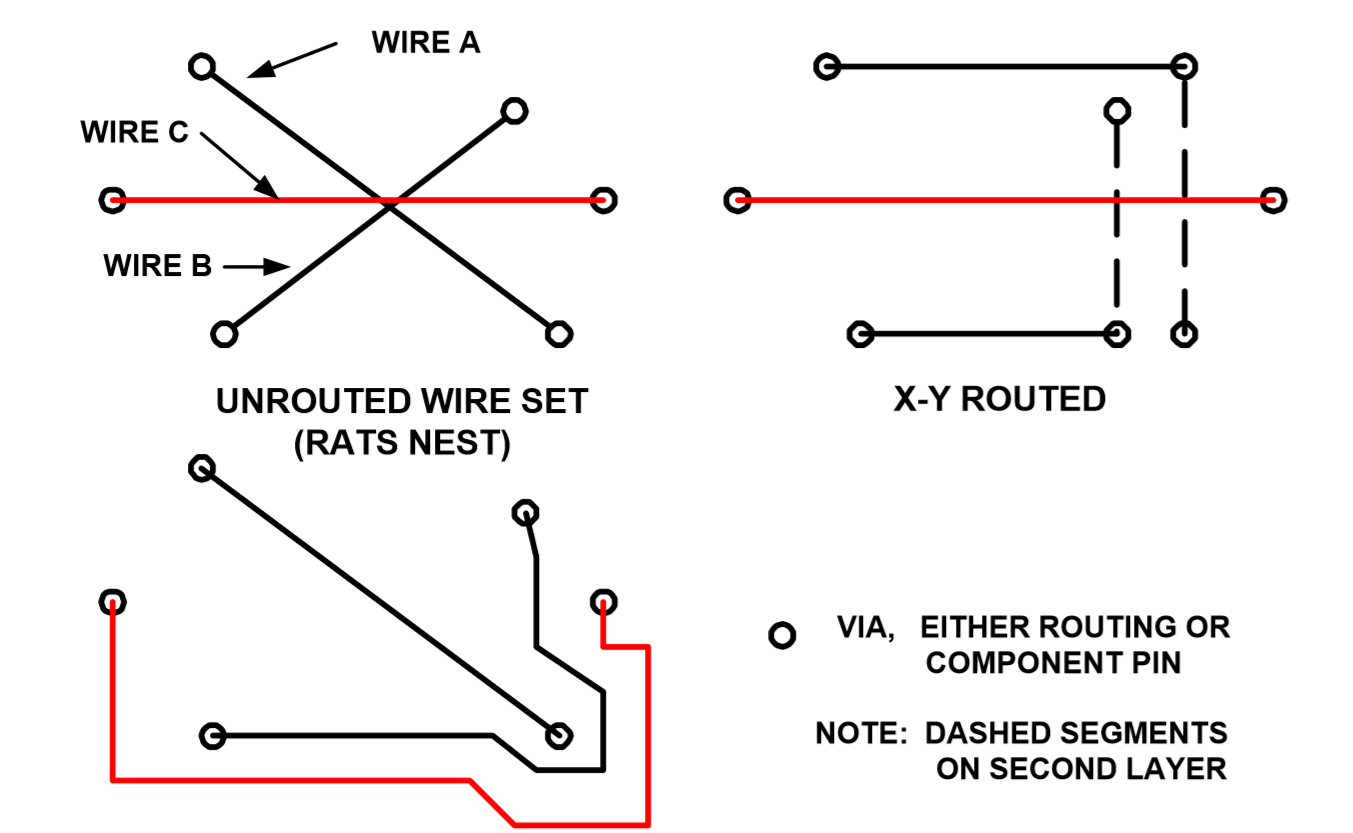
There are two main types of routing X-Y routing and Maze routing (LEE W. RITCHEY, 1999).

Figure 3 - A diagram to show the different methods of routing. Image Credit LEE W. RITCHEY, 1999 Accessed: 21/11/2018 Available <http://www.speedingedge.com/PDF-Files/pcbrouters.pdf>

Initially, I will focus on maze routing as it the preferred method of routing by industry, due to the fact that X-Y routing is almost always impossible with less than 4 layers due to the component footprints disallowing straight lines to be drawn. However as the project progresses adding support for X-Y routing or even creating a hybrid router would have its uses. The routes can move between the layers of the board using what are called “via’s”, which are essentially just holes in the board allowing a current to pass through.

### Users, Limitations and Requirements

Auto-routing is a tool to make electronic design easier for engineers to design a board. The user for the system, who I have reached out to, Stuart Jessop, should be able to use the final version to carry out the following tasks:

* Import a Net either by manually entering the coordinates of the pads or by parsing a file
* Set design constraints as detailed in the “interview”
* Route the board:
  + This will involve finding a route from two points
  + Moving on to the next two points
  + If routing impossible ripping up and retrying
* Export the routed file in a format that can be printed (SVG[[2]](#footnote-3))

Limitations:

* The software must run on a school computer. I have to demonstrate this to my user at school and I don’t have my own laptop so I either need to compile to a “.exe” or use a language with a run-time environment on the schools machines.
* My own coding abilities, I should do this project in a language that I am very familiar with such as node.js or python
  + Need to perform I/O in those languages which could difficult
* Time – This project needs to be done in the time allocated in the time allocated for the project in my computer science course.

## Research

### Interview

After speaking with my user there are a range of areas that my project could support and they were ranked according to value. The list below is ranked.

1. Different track widths
2. Keep out areas
3. Thermal sinks[[3]](#footnote-4)
4. Isometric routing[[4]](#footnote-5)
5. Integration with EDA (Electronic Design Application) software.

### Perquisites and missing requirements

After my interview with my user I investigated other useful features which auto-routers have which are perquisites to being able to implement the features requested. The definition for what auto-routing is on Wikipedia is “*the routing step adds wires needed to properly connect the placed components while obeying all*[*design rules*](https://en.wikipedia.org/wiki/Design_rules)” (Wikimedia Foundation, 2018). This definition clearly points out a few areas which my project would need to support before the requirements suggested by my user:

1. Being able to input data
2. Being able to find a route between two points
3. The route such does not interfere with other routes
4. Ability to export files

Other features which auto routers sell themselves on is the ability to route across multiple layers (Altium LLC, 2018), support for X-Y routing and for nets with multiple contacts. Auto-routers also prize themselves on their speed.

### Objectives

The requirements listed above are quite vague and need to be made more precise.

Being able to find a route between two points– The route should not waste board space, this means that it is likely to be the shortest possible route and so is “efficient”.

The route such does not interfere with other routes– Interfere here means cause a short or a break in another track.

The other requirements have been reworded and ordered as bellow:

1. User can input the positions of the points into the system and what routes they belong to.
2. Finding an efficient routes between two points
3. Being able to route between multiple points
4. Program can export result in SVG format.
5. Being able to rip up bad decisions to make improvements.
6. Ensuring that the routes do not violate any the DRC requirements listed in order bellow:
   1. Shorts and breaks do not occur
   2. No tracks cross “keep out area’s”
   3. Areas of the board can be specified as floods.
7. Support for nets with multiple contacts

Extensions

1. Support for different track widths
2. Isometric Routing
3. Integration with EDA[[5]](#footnote-6) software, KiCAD[[6]](#footnote-7)

A general requirement for all of the above steps is that the auto-router should be as fast as I can make it.

## Prototyping

### Objectives

From the prototyping period I needed answers to the following questions:

* What language should I use?
  + If it’s not Delphi how do I get it to work on the school PC’s
  + How do I compile/run code
* How hard is this problem:
  + What solution and algorithms can I implement?
  + Is it of the right difficultly level for my coursework?
* What should a basic outline of the classes for my project look like?

### What I found out

#### Language

I do not want to learn a new language for my project and of the languages which I know I feel most confident in python or JS, graphical output is hard in python and so I chose to use JS. The schools computers have node.js installed which means that I can write my code in a text editor and execute it using PowerShell.

#### Algorithms

In-order to perform my prototyping I needed a basic awareness of the available algorithms. My problem is effectively how to solve a maze as if I can find a route between two points then that route just becomes another “wall” of the maze for the remaining routes. This means that I can use any maze solving algorithm. There are two main types of routing algorithms which apply “breadth-first” and “depth-first” (Zhou, 2000). Breadth-first algorithms are slower, require more resources but often find better routes as they will explore all possible routes over just prioritising one route. Depth-first algorithms attempt a single and likely path which is much faster (particularly for long tracks[[7]](#footnote-8)). For prototyping I wanted to show that it would be possible and so I chose a breadth first approach, however for my final project I may want to use a hybrid algorithm.

For my prototyping I chose to implement the “Lee-algorithm” (Techie Delight, 2018). Which works by propagating a wave out from one of the nodes leaving behind it a record of how long it to the wave to reach each point. When the wave hits the other contact it is then able to route to the point by tracing back through the record of distances selecting for the smallest distance to the start on the way back.

### Data structure

There are three main entities that my program needs to be able to handle as shown in figure 4. So during the prototyping period I created a class diagram as shown in figure 5

Figure 4 - An entity relationship diagram showing how the data about the board is related.

Point

Board

Net

Route

No route area

This class structure divides the problem into multiple clear sections. The Nets are represented by an object in which there are four integer properties which store the start and end position of the points. This object is contained within a board object which is then manipulated by a Board Router Object. The Net router object never gets instantiated but instead contains all of the functions associated with routing one net; the board router inherits all of these functions and then adds functions to prioritize which nets to route first and which nets to rip up. The route is represented as an array in the BoardRouter Class, which also contains the no-route areas

Net

NetRouter

BoardRouter

Board

Figure 5 - A class diagram showing my OOP model at the end of the prototyping period

# Documented Design

## Program overview

### A Star

To route individual nets I model the board as a grid which can then be modelled as a graph. From this point all common graph traversal algorithms can be applied.

I use my own implementation of the A\* path finding algorithm. At the core of this implementation there is a Heap, which stores all of the nodes by their value where and node.h is a heuristic value in this case determined by the Euclidian distance between the start and end node and node.g is the shortest known distance to that node.

The algorithm pops values from the heap and adds their neighbours to the heap until it runs out of nodes or reaches the end point.

Pseudo code is based upon my netrouter.route function.

1. toCheck = **new** Heap(sortedBy item.f)
3. startCell.g = 0
4. startCell.f = 0
5. startCell.checked = True
7. toCheck.push(startCell)
9. **while** toCheck not empty {
10. cell = toCheck.pop()
12. **if** (cell = end) {
13. **while** (cell != start) {
14. trace = current.push
15. current = current.**super**
16. }
17. **return** trace
18. }
20. neighbours = cell.getneighbours()
22. **for** n **in** neighbour {
23. ng = cell.g + 1;
25. **if** (ng < n.g or neighbour not checked) {
26. n.**super** = cell
27. n.checked = **true**
29. n.g = ng;
30. n.h = EuclideanDist(neighbour, end)
31. n.f = n.g + n.h
33. toCheck.push(neighbour)
34. }
35. }
36. }

### Highlevel Routing

When deciding on the order in which to route the nets I want to try and minimize interference. Therefore the nets are placed into a queue in which they are initially sorted by Manhattan length. The pseudo code for my board routing looks like this.

1. toRoute.sortByManhattan()
3. **while** (toRoute.length > 0) {
4. **try** {
5. currentNet = toRoute.shift()
6. tracks.push(route(currentNet))
7. } **catch** (err) {
8. currentNet.errorCount++
9. **if** (currentNet.errorCount < threshold) {
10. netsToClear = getNetsToClear(currentNet)
11. ripup(netsToClear)
13. toRoute.unshift(netsToClear)
14. toRoute.unshift(currentNet)
15. }
16. }
17. }

The algorithm will try to route the nets in order of shortest first. If however it finds that a net is impossible to route then it checks to make sure that this net hasn’t failed many times already (maybe it is just impossible) and then removes the tracks that might be causing the problem and tries again.

#### get nets To Clear

The nets to be removed are calculated by drawing an L between the two end points and ripping up and nets that cross this L-shape. In the bellow diagram the blue net would be removed by not the green net.

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |
|  |  |  |  |  |  |  |  |  |

### User Interface

#### The Board

One of my design constraints was that I had to be able to be able to run this on a school computer, which vastly limits my JS runtime environments to either the edge 15 browser or the node.js V6. For development purposes I wanted to be able to use vs-code’s inbuilt debugging tools and so for this reason I decided to use node with as much code as I could possibly move server side.

All of the UI is displayed in the browser and the user interacts with the board via and AJAX connection to the server where the routing and construction of the DOM takes place. Here is the basic outline of how this works:

<http://www.plantuml.com/plantuml/png/TO-nJWCn44Jx-ugfg8Ze8uGYWQ12I4JGE-S9jkIpm-wQaB_7XaDS4SgHvis-tQWviJgbuJP5PiFg1blA1mLh2DyhrH1iIfTGEoKgNFRGKUB1z9UvNYqPFPHMEKOBFPGg8tK8kHXHUcU6rdZHbc8sYXjDVuXTSUBFkUnfOy2Lb6fSLA7cZ6sbHNzQdFaZxsFJwTmilwinUMnV7yxfFo_rCu-zXtsH-Te5ADBIH5NtzjzV1z-TB11tJuzmVSv5-vwfIb5aVXhQqx7aOSFiwvI-0000> D:\Documents\My Documents (Edwin)\Downloads\TO-nJWCn44Jx-ugfg8Ze8uGYWQ12I4JGE-S9jkIpm-wQaB_7XaDS4SgHvis-tQWviJgbuJP5PiFg1blA1mLh2DyhrH1iIfTGEoKgNFRGKUB1z9UvNYqPFPHMEKOBFPGg8tK8kHXHUcU6rdZHbc8sYXjDVuXTSUBFkUnfOy2Lb6fSLA7cZ6sbHNzQdFaZxsFJwTm.emf

To update the DOM without making everything unresponsive for the user an asynchronous request is performed to the server in the following format.

1. async update() {
2. //Fetch the SVG DOM from the server but store the value as a promise
3. let resGrid = fetch( body: JSON.stringify(requestContent) )
5. //Await for the promise to be fullied then assign the DOM
6. let response = await resGrid
8. //Convert the response into objects that we can manipulate
9. response = JSON.parse(response)
10. **this**.grid.innerHTML = response.board
11. **if** (response.errors.length > 0) {
12. show response.errors
13. }
14. }

The state of the client side board is represented using a series of objects which are then converted into a JSON string for transmission in the body of the request. This is then received by the server which parses the JSON, performs the routing, builds an SVG representation of the board and returns that in the body of its response along with any problems that happened whilst routing, encoded using JSON. The client then parses this response and updates the DOM accordingly.

#### Movement of end nodes

The only processing that really goes on client side is the handling of the end nodes which are used to control the positioning of nets, floods and keep outs. To do this I override the onMouseDown Method of each of the squares with my own dragMouseDown.

In order to keep with my grid design for the simplification of routing I rounded the mouse’s value to the nearest grid cell size. This forced everything to work in the grid.

To allow to the user to delete items I check if the control key is pressed when the dragMouseDown event is triggered, if so it executes the deleteCallBack function which was passed to the function when it was initialized, which destroys the object that the dragMouseDown function is assigned to.

So as not to overload the server (routing is an extremely CPU intensive task) the update function is only called when the user stops dragging, however if the routing algorithm was better optimised this could be moved into the elementDrag function to provide real time routing support.

1. **function** dragMouseDown(e) {
2. e.preventDefault()   //Prevents the default method from running
4. // get the mouse cursor position at startup:
5. oldXPos = getRoundedMouseX(e);
6. oldYPos = getRoundedMouseY(e);
8. /\*\*
9. \* If the control key is down then the user wants to delete this item
10. \*/
11. **if** (e.ctrlKey) {
12. deleteCallBack()
13. board.update();
14. } **else** {
15. document.onmouseup = stopDragging;  //Override the onmouseupmethod
16. document.onmousemove = elementDrag;     //Assign the on mousemove elementDrag method to elementDrag
17. }
18. }
20. **function** elementDrag(e) {
21. e.preventDefault();
23. // calculate the new cursor position:
24. newXPos = oldXPos - getRoundedMouseX(e);
25. newYPos = oldYPos - getRoundedMouseY(e);
27. // set the element's new position:
28. elmnt.style.top = newYPos
29. elmnt.style.left = newXPos
31. //Update the old positions
32. oldXPos = getRoundedMouseX(e);
33. oldYPos = getRoundedMouseY(e);
34. }

37. **function** stopDragging() {
38. /\* stop moving when mouse button is released:\*/
39. document.onmouseup = **null**;
40. document.onmousemove = **null**;
42. //find new the cells position in grid terms.
43. let newCellX = Math.ceil(elmnt.offsetLeft/board.cellSize);
44. let newCellY = Math.ceil(elmnt.offsetTop/board.cellSize);
46. cell.x = newCellX;
47. cell.y = newCellY;
49. board.update();
50. }

#### Buttons

The buttons have the JS callback functions linked to them in the HTML.

## D:\Documents\My Documents (Edwin)\Downloads\TP1FQkim48VtSufP5v83X493KcZpIco8Um5X_R155Dd8esnCxbxBIYnQydOpdpxDdzaEeho4YwLUrMVLW2luApoFL42uMwru3sjfrocl_wOlKpfPqJnrGU3_GsOx_Sbqv4NAJxy-v2AnxoTKpvk3gstG-6xDK3gy4GtNfcfL3twTd1DgKQwnYFYeplTo6IzHMgM.emfUML Digramoverview

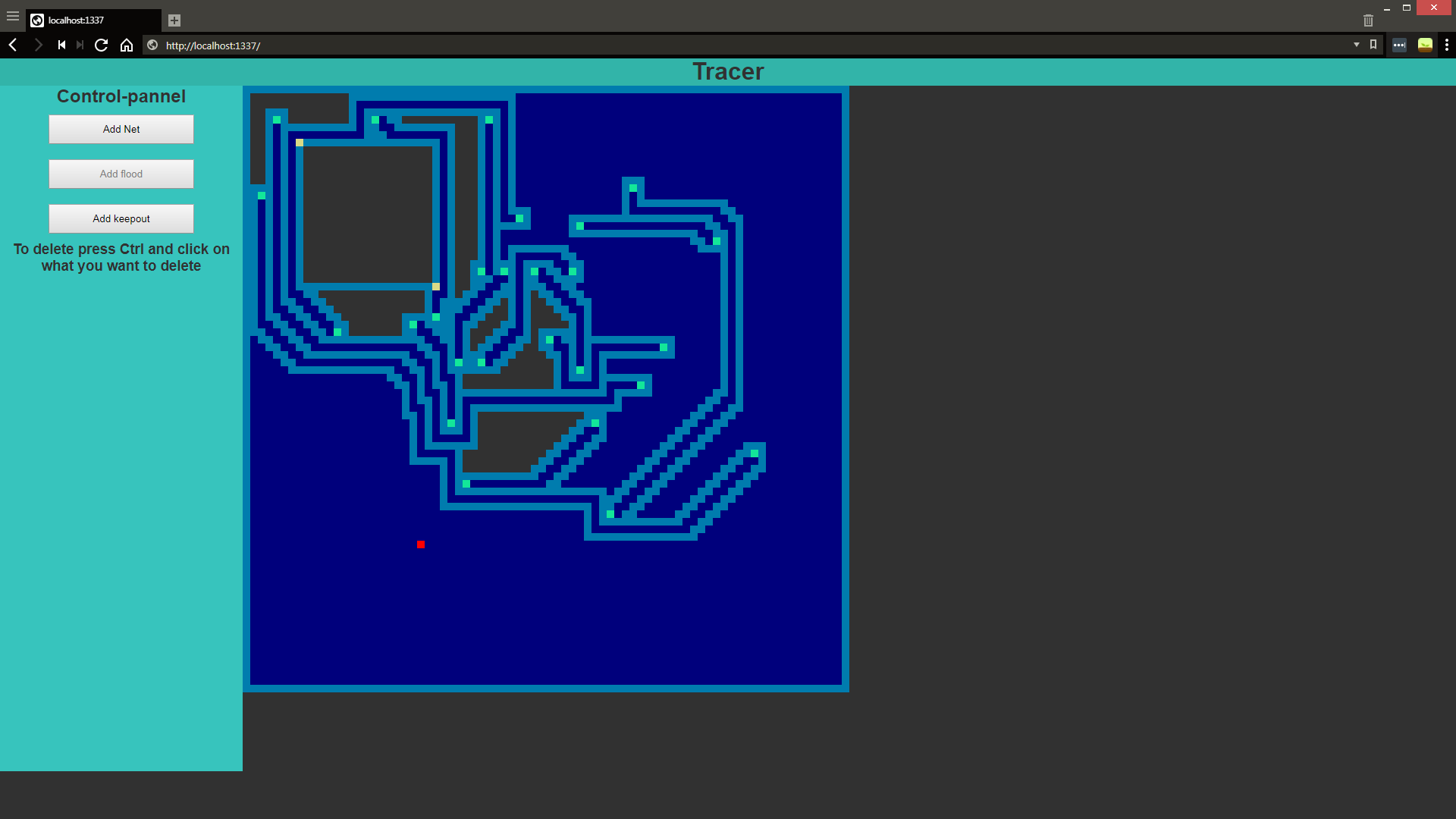
#### Board

The board is represented by

## Data structures

|  |  |  |
| --- | --- | --- |
| FileName/Name | **Type** | **Purpose** |
| index.js/fileTypes | Dictionary | To allow for the mimetype of a file to be looked up from its file extension |
| main/board.js/Grid | 2D Array of Cell objects representing a graph | A 2D array of cell objects contained within the Board object that represents everything that is happening on the board.  The router traverses this array as if it were a graph by interpreting the neighbours of each of the squares adjacent nodes. |
| index.js/responseContent | JSON String | A JSON string which is transferred between client and server. It contains a complete representation of all of the client side objects. |
| main/netRouter.js/toCheck | Heap | Hold the list of items which have are on the edge of where has been checked and poping them gives the next one to check |
| main/boardRouter.js/errors | Stack | All the non-crashing error messages that need to be displayed to the user. |
| main/svg.js/DOM | DOM | Holds the Document Object Model to be displayed to the user that has been generated using the SVG class. |
| main/boardRouter.js/netsToRipup | Set | Used to store each of the nets that need to be ripped up, due to the way that these are selected the algorithm may detect them on multiple instances should only store them once as they can only be ripped up once. |
| main/boardRouter.js/netlist | List | Stores the nets and the order in which to route them. |
| indexScript.js/deleteCallBack | Function | Passed to the makeInteractable function and |
|  |  |  |

## Screen Shots



Sources for route finding:

Need to look at heuristics and preprocessing alogrithims

<http://aigamedev.com/open/tutorial/symmetry-in-pathfinding/#1>"

<https://qiao.github.io/PathFinding.js/visual/>

<http://theory.stanford.edu/~amitp/GameProgramming/Heuristics.html>

<https://www.microsoft.com/en-us/research/wp-content/uploads/2004/07/tr-2004-24.pdf>

<http://aigamedev.com/open/review/near-optimal-hierarchical-pathfinding/>

# Bibliography

Altium LLC. (2018). *BEST AUTOROUTER*. Retrieved November 18, 2018, from Altium: https://www.altium.com/solution/best-autorouter

LEE W. RITCHEY, S. E. (1999). *PCB ROUTERS AND ROUTING METHODS.* Retrieved from Speeding Edge: http://www.speedingedge.com/PDF-Files/pcbrouters.pdf

Techie Delight. (2018). *Shortest path in a maze | Lee algorithm*. Retrieved from Techie Delight: https://www.techiedelight.com/lee-algorithm-shortest-path-in-a-maze/

Wikimedia Foundation. (2018, October 22). *Routing (electronic design automation)*. Retrieved November 18, 2018, from Wikipedia: https://en.wikipedia.org/wiki/Routing\_(electronic\_design\_automation)

Zhou, P. H. (2000). *EECS 357 Introduction to VLSI CAD.* Retrieved from eecs.northwestern.edu: http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf

1. A *Net* is a collection of wires that connects all of the points or pins in a single circuit [↑](#footnote-ref-2)
2. I have chosen to export in SVG form as it is easy to describe using a programing language which will make the export stage easier. [↑](#footnote-ref-3)
3. A Thermal Sink is an area of the PCB which is entirely copper and is used to sink heat from high current devices [↑](#footnote-ref-4)
4. Isometric routing is where multiple routes are routed to be approximately the same length to reduce skew in parallel data transition. [↑](#footnote-ref-5)
5. EDA - Electronic Design Application, CAD software for modelling PCB’s [↑](#footnote-ref-6)
6. KiCAD was chosen as the target platform as it is open source and so there is better documentation on the file structure. [↑](#footnote-ref-7)
7. Most breadth first algorithms have a complexity similar to [↑](#footnote-ref-8)