GCE Computing

D'Overbroecks College
62315

Michael R. Bell
7894

[BELTRAC]

Model railway automation.

CONTENTS

DEFINITION, INVESTIGATION AND ANALASIS

DEFINITION – NATURE OF THE PROBLEM TO BE INVESTIGATED

THE END USER

The project is being produced for John Peter Thomas, the owner of a small computer company. In his own words, this is a brief description of his company:

> The company is a lifestyle company, i.e. we keep it relatively small, we enjoy our work and we're driven more by the quality of the work we're offered rather than the money. That said, we've recently grown a little to 10 developers who work on a range of bespoke projects for a wide variety of clients. We started out writing code for embedded systems, but have more recently specialised in code for web applications and mobile platforms. We've written primarily for the iPhone, but more recently we've been looking at Android apps. On occasion we are asked to develop full blown web sites, though we have tried to stay away from these as much as possible. We've recently employed a graphics designer for the first time and will most likely take on more web based projects as time moves on.
>
> We've taken on projects for students from D'Overbroecks in the past and without exception have been really pleased with the results. I'm looking forward to seeing a preliminary requirements specification soon.

THE PROBLEM

Mr Thomas has a lobby where people wait before meeting him. As this would be where potential clients would find themselves waiting for meetings with him and his staff it would be good to have something to entertain potential clients while they wait, which at the moment they don't have.

Providing something interesting to entertain has several advantages:

- It prevents potential clients from getting bored and so keeps their mood light for the meeting ahead which can increase the chances of success.
- It can display what the company is capable of or, if not created by the company, can imply an ability to produce similar things or an interest in the particular subject matter. Again, this can increase the chances of a deal being struck as well as providing ideas for alternative products.

RESOURCES PROVIDED

In terms of what Mr Thomas is willing to provide me it would really just be some floor space and a power supply to use, being England this would be 230~240V(RMS) at 50Hz

FURTHER STEPS

Having gathered a bit of information I arranged a meeting with Mr Thomas, in conversation I asked him some questions about the project, I noted down some basic points based on the responses to the questions:

- We decided to call the project Beltrac.
- Beltrac will be on a static platform (no wheels).
- Beltrac will be a loop.
- Beltrac will have a cleaning train which dusts the track when switched on.
- Beltrac will have a standard train which can be controlled.
- Beltrac will have a small display with a simple menu system.
- Beltrac will have automated points that can change to navigate the train to its destination.
- Beltrac will have sensors on the track to detect the location of the train.
- Beltrac will have an integrated circuit breaker to protect the train from surges and from the control board supplying to much current.
- Beltrac will have an easy to use interface that requires no pre instruction (though a manual will be provided.

Of course, this alone is not enough to create a requirement specification but the information discussed is a good start as well as an opportunity to get to know Mr Thomas a bit better.

INVESTIGATION AND ANALYSIS

ASERTAINING THE USER REQUIREMENTS

TALKING TO THE USER

To begin the whole project process I began by emailing Mr Thomas with a brief explanation of what I am looking to do:

> Dear Mr Thomas,
>
> I am an A-level student looking to produce a computing project, I am aware that this may not be the first time you have received a request like this, as my teacher Alan informs me that you often act as a client for projects, despite this I hope you will be willing to review my proposal as I am confident in its merit, particularly to someone with such a vast appreciation of complicated electronic nick-nacks.
>
> Did you ever play with Hornby or some other brand of model railway as a kid? Because I sure did! It was a good, carefree time in my life. My proposal is therefore an attempt to revive the former, stress relieving, glory by attempting to attach the subtle science of railway signalling to something on a smaller scale by creating a railway that would be safe enough to automatically carry passengers from A to B with minimal loss of limbs.
>
> In principal it would be a similar model to that of the Docklands Light Railway, that is, a train controlled by a computer off the train, using data gathered from the tracks about the location of the train and other obstacles, with the advantage of not actually carrying passengers. This eliminates the necessity for a member of railway staff on hand to unjam passengers from the doors (of course the dispatch panel on board the train could be simulated externally for the full experience).
>
> You are probably wondering what this project could possibly do for you but picture this - the world industry is changing, for all you know your company may go into "railway related work" someday and it would certainly be swell to have a fully automated model railway to entertain waiting clients!
>
> Should you have any questions I would be happy to indulge them,
>
> Michael Bell

His 1st reply was:

Hello Michael

Alan did tell me that you had a project that you'd like to do. As Alan probably told you we do like to display interesting projects in our entrance area and I've been very happy to showcase student projects before so your project definitely does interest me. A working train set in the office would definitely draw some interest especially if it has a significant control component.

What sort of control hardware do you have in mind? There's a lot of buzz these days about the Raspberry Pi which is an excellent piece of kit but I suspect that you should consider something more dedicated to control such as the Arduino range of boards. They're programmable in C and have a wide range of analogue and digital inputs and outputs.

Alan has asked me to look at one of his other students' projects this year too - perhaps the two of you could get together and come down to meet us some time in the next week or so.

Cheers

-John

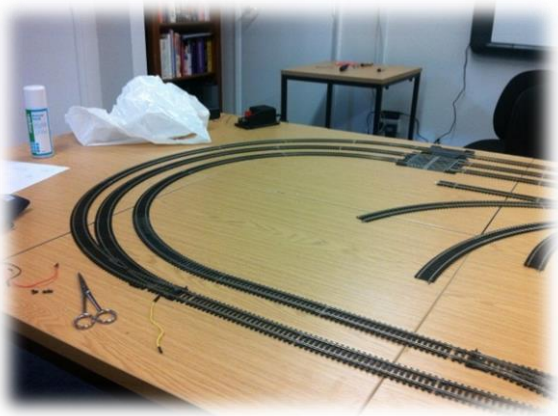
After this we began an exchange in which I sent Mr Thomas a series of questions and he replied with answers to each question, though not the original format, for readability I have separated out each question and the responses given to it. My questions / responses are written in Lilac and Mr Thomas's are written in brown,

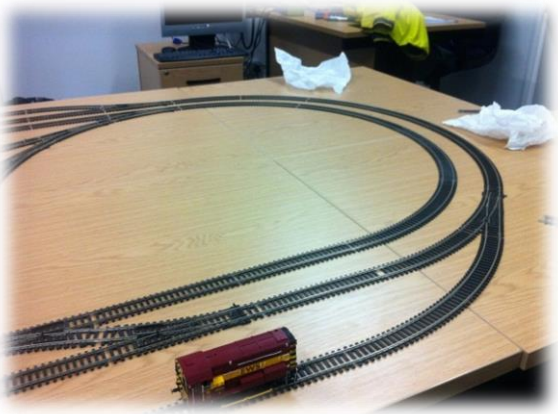
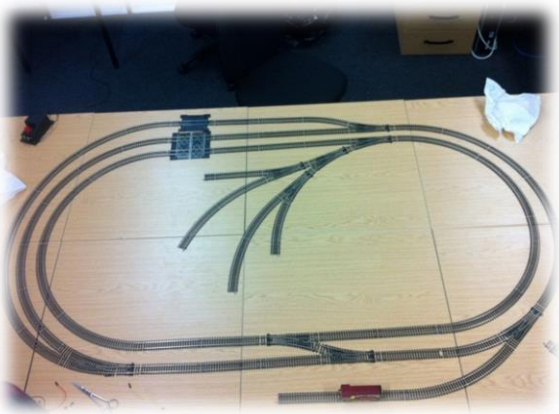
- Would you like an interface to control the train and if so what would be your preference on what it is? LCD, computer program etc.
    - An LCD interface is fine, how many characters? How many lines? Colour or bow?
        - It would have 2 lines probably about 20 chars each
            - That's fine, but you'll need to specify exactly what's going to appear on each of the lines. I suggest for simplicity that you try to adopt a menu structure, but maybe you have other ideas.
                - No, a menu structure sounds good to me
    - A standard PC computer interface would also be great - you might find it sensible in your development to build a PC interface simply to provide yourself with a simple easy to change interface. Programming LCD displays is often trickier. But, clearly we'll need some sort of user interface in the final version and an LCD interface would be fine, assuming of course that it's comprehensive and ideally intuitive. We'd like our visitors to be able to play with it when they're waiting for us and it needs to be self explanatory if that's going to work.
        - The LCD I had in mind is quite easy to program, it is designed specifically for the Arduino board and operating it is as easy as sending it a string of text
            - Sounds good - is it colour?
                - No its just 2 lines of black and white characters

        - Clearly we need to agree on what the user interface is going to allow us to do - that will require quite a bit of discussion.
            - I know you want it to allow selection of a destination, and to initiate the cleaning train, what else would you like it to do?
                -
- The track needs to be cleaned regularly as dust can damage the trains, would you like to clean it yourself or have a second train that cleans the track at regular intervals?
    - A secondary train is an excellent idea. Perhaps it could appear at various intervals? It would be nice to control this from the user interface, i.e. set intervals, initiate cleaning and so on.
        - That's fine though I should say that the abrasive pads on the bottom of the train must be changed by hand when they get clogged up with dust
            - OK - that's something that we can organise on a regular basis.
- Would you like the train system to turn itself on and off at set times? E.g. Running only during office hours
    - An automatic mode is an excellent idea - again a variety of scenarios chosen from the user interface would be good.
- Would you like the primary train to (in automatic mode) run itself to a schedule or pick stations at random
    - Both, set by the user interface.
- How many stops would you like?
    - How many can we have?
        - I would say about 5 would be a good idea
            - OK.
- Would you like the trains to move relatively quickly or slowly?
    - Various speeds - sometimes fast, sometimes slow, again set by the user interface - scripted would be good.
        - Ok I will
- Would you like electronic signals on the track? E.g. Lights, semaphores etc.
    - Definitely.
    - Can I please see some diagrams of how you're planning on laying out the track? Do you have any photographs? It would be good to get a really good idea of how big and complicated the system is.
        - I will have lots of time to do that this weekend
            - Could I see some please?
                - I will send you some pictures ASAP
    - What are your thoughts on user interface functionality?
        - I think that there should be a manual and automatic mode, the modes could be switched by a switch or a key, in automatic mode the screen displays the name of the program on the top line and what the train is currently doing e.g. "Awaiting Right Ahead" or "slowing for stop in block 1"
        - In manual mode the program would have a simple menu structure allowing options like, setting what the train does in automatic mode, picking the next station for the train to reach, initiating cleaning, setting on and off times and checking that the track is clear.
        - When in automatic mode, customers could interface with the train using large colourful illuminated buttons rather than a boring looking LCD display with options like "go" "stop now" "stop at next station" etc.
            - That sounds fine.

- What are you planning to use as your control hardware? I know that Leo is thinking seriously about using an Arduino board - are you going down the same route or do you have some alternatives in mind?
  - Same route

During this conversation Mr Thomas requested some pictures of a basic track layout and the train so I captured these images and sent them to him:




This was his response:

> Thanks for these - I assume that the image of the track is the track you'd like to use. If so, it looks good but I'd like to understand what functionality you are thinking of including. By which I mean things like,

What is the function of the `orphan' pieces in the centre?

How many sensors are you planning on using?

How many points are there in total? Presumably you're intending to control the movement of the train by modifying the position of the points and controlling the speed and the direction of the train - that should do everything you want.

You mentioned a `cleaning' train. Presumably that will sit somewhere off the main track and move out into the central section when required, at which point presumably the train itself will have to move elsewhere - all of which will again presumably be under microprocessor control?

In order to understand exactly what you're proposing, I really would like to see a reasonably complete description of the functionality of the entire operation of the train line. Can you get this to me fairly soon so that we can internally make a final decision on whether the project is interesting enough to take up some of our foyer space.

Thanks

-John

On receiving this I hastened to complete a draft requirement specification as it was clear that it would provide him with a good idea of what I was going for while at the same time nearing us closer to the final signing.

Said specification is shown below.

REQUIREMENTS SPECIFICATION

1.0

USER REQUIREMENTS

1. The user should be able to select a manual or automatic mode.
    a. In manual mode the next destination of the train can be chosen.
    b. In automatic mode the train goes to either random or sequential stations until stopped.
2. The user should be presented with a simple display that can be controlled using directional arrows and a confirm button.

HARDWARE REQUIREMENTS

1. There should be a small train (00 gauge) which is controlled by a microcontroller with The user interface attached.
2. The train should travel at multiple speeds and stop at multiple stations with points used to provide multiple possible routes.
3. Sensors on the track or the train should detect its location and this information should be used to manoeuvre the train.
4. The maximum voltage of the train should never be exceeded.
5. The train should appear to gain or lose speed smoothly.
6. The tracks should be able to isolate a siding so that the train on it can be held, allowing a different train a turn on the tracks.
7. The train should be able to travel forwards and backwards at equal maximum speeds.
8. The layout should be designed that if the train is moved forward that no matter where it is it will always trigger a sensor on its journey.
9. Fail safes such as circuit breakers should be in place to protect the train and the equipment.

SOFTWARE REQUIREMENTS

1. The microcontroller should be able to recognise the location of the train and act accordingly.
2. It should anticipate the possibility that a sensor has failed and should be able to act if the sensors are not called in sequence.
3. The software should be able to plot a route from any station to any station.
4. The software should be able to activate a cleaning train at any time to clean all the tracks and it should be able to manage the activation of both this and the main train by isolating sidings.
5. The software should be able to reascertain the location of the train in case of a power failure by moving it forward until it triggers a sensor.
6. The software must not perform any actions that could damage the train or any other equipment.

After completing this, MR. Thomas added some things and made some changes which left me with a draft of version 1.1, he read this version and gave it his approval so I printed it out and we both signed it, thus completing the requirements of the project.

The final specification is included below. At his request it contains some examples of what the final project will look like and how it will behave.

REQUIREMENT SPECIFICATION

USER REQUIREMENTS

1. The user should be able to select a manual or automatic mode.
    a. In manual mode the next destination of the train can be chosen.
    b. In automatic mode the train goes to either random or sequential stations until stopped.
2. The user should be presented with a simple display that can be controlled using directional arrows and a confirm button.

HARDWARE REQUIREMENTS

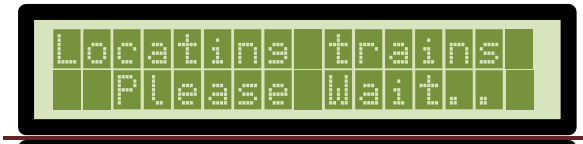
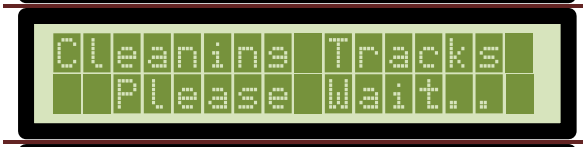
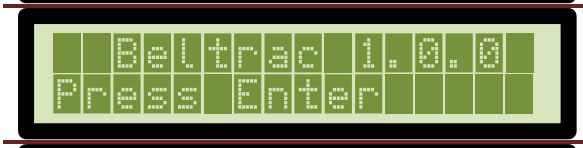
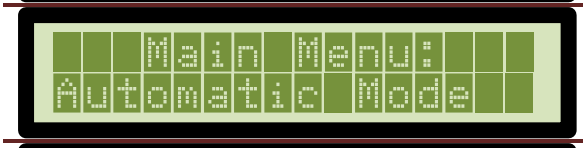
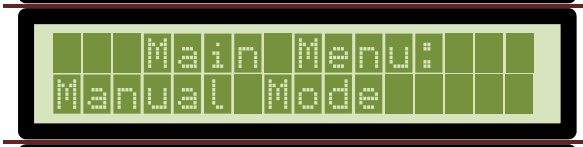
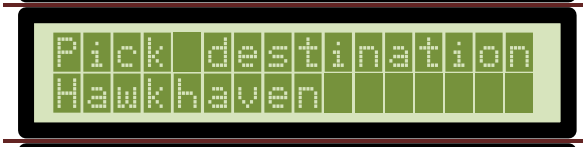
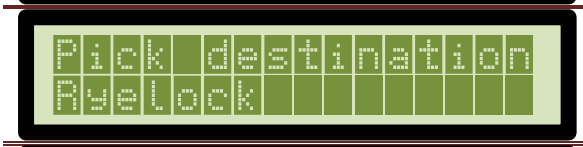
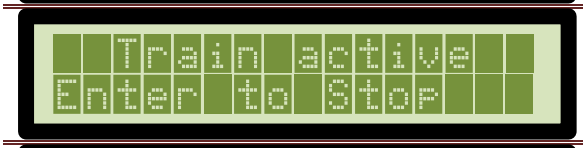
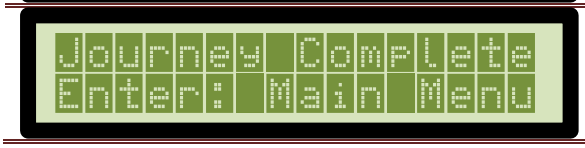
1. There should be a small train (00 gauge) which is controlled by a microcontroller with the user interface attached.
2. The train should travel at multiple speeds and stop at multiple stations with points used to provide multiple possible routes.
3. Sensors on the track or the train should detect its location and this information should be used to manoeuvre the train.
4. The maximum voltage of the train should never be exceeded.
5. The train should appear to gain or lose speed smoothly.
6. The tracks should be able to isolate a siding so that the train on it can be held, allowing a different train a turn on the tracks.
7. The train should be able to travel forwards and backwards at equal maximum speeds.
8. The layout should be designed that if the train is moved forward that no matter where it is it will always trigger a sensor on its journey.
9. Fail safes such as circuit breakers should be in place to protect the train and the equipment.

SOFTWARE REQUIREMENTS

1. The microcontroller should be able to recognise the location of the train and act accordingly.
2. It should anticipate the possibility that a sensor has failed and should be able to act if the sensors are not called in sequence.
3. The software should be able to plot a route from any station to any station.
4. The software should be able to activate a cleaning train at any time to clean all the tracks and it should be able to manage the activation of both this and the main train by isolating sidings.
5. The software should be able to reascertain the location of the train in case of a power failure by moving it forward until it triggers a sensor.
6. The software must not perform any actions that could damage the train or any other equipment.

To help in understanding these requirements a few examples are provided below,

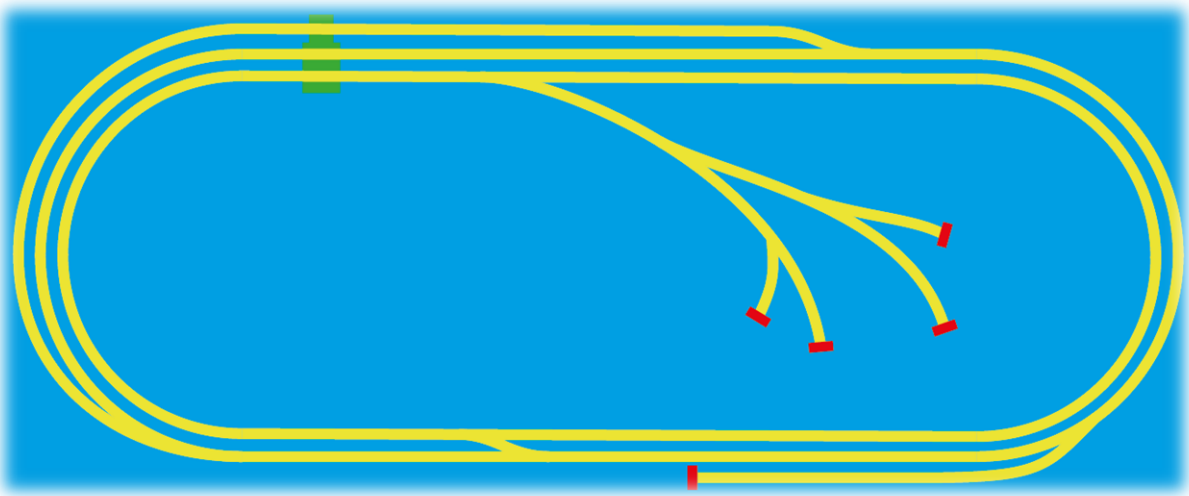
EXAMPLES OF OUTPUTS

	This would be the starting screen, we need to wait while the system moves the active train onto a sensor.
	This would then show while the cleaning train travels the whole track, dusting the tracks, we need to wait a little longer…
	This would be the welcome screen, lets press enter.
	This is the first page of the main menu, lets choose manual mode by pressing down.
	And press enter when we have it.
	Ok so we don't like the idea of going to Hawkhaven so let's press down and choose a different destination.
	Ok so Ryelock seems nice, lets press enter and send the train there.
	(the words 'train active' will flash) let's let the train arrive.
	We did it! The train is safely at Ryelock, if we press enter we can start again and send the train somewhere else.

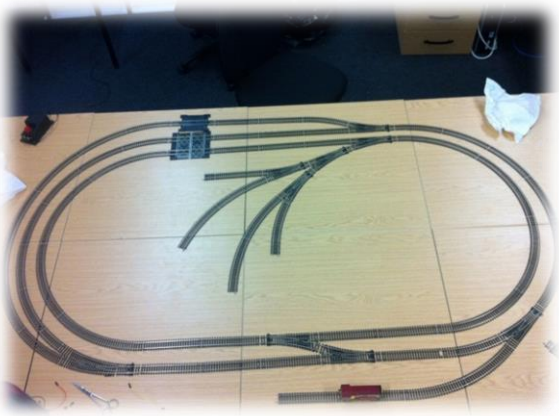
THAUGHTS ABOUT LAYOUT

TRACK LAYOUT

This is my suggestion for the preliminary layout for the track.



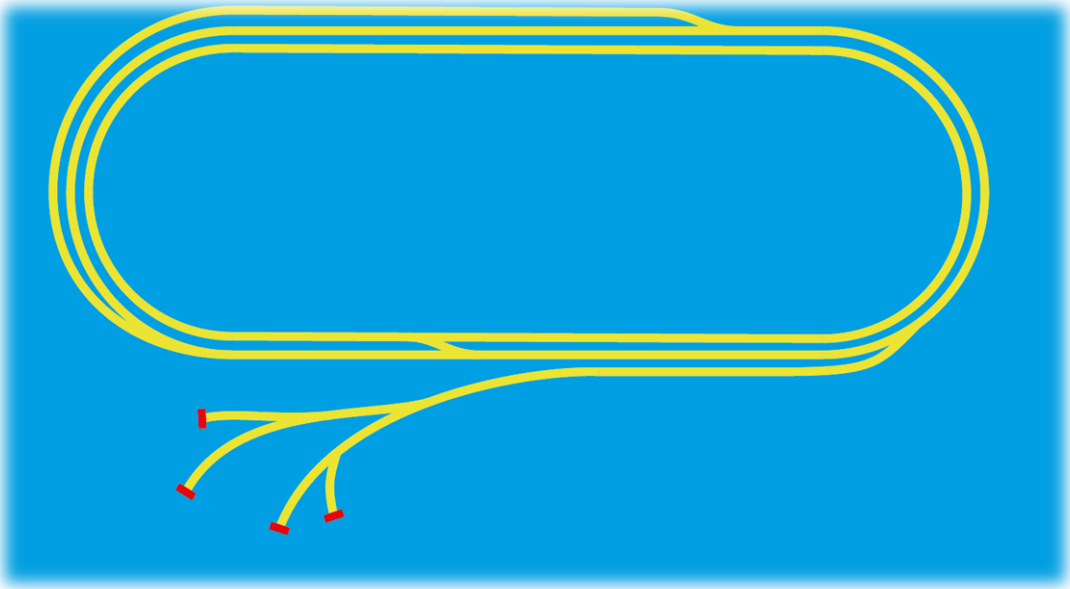
This is the preliminary layout for the track - it looks a lot like this picture although when the picture was taken the central hub was facing the other way, and this had to be changed so that the train would always trip a sensor by moving forwards.

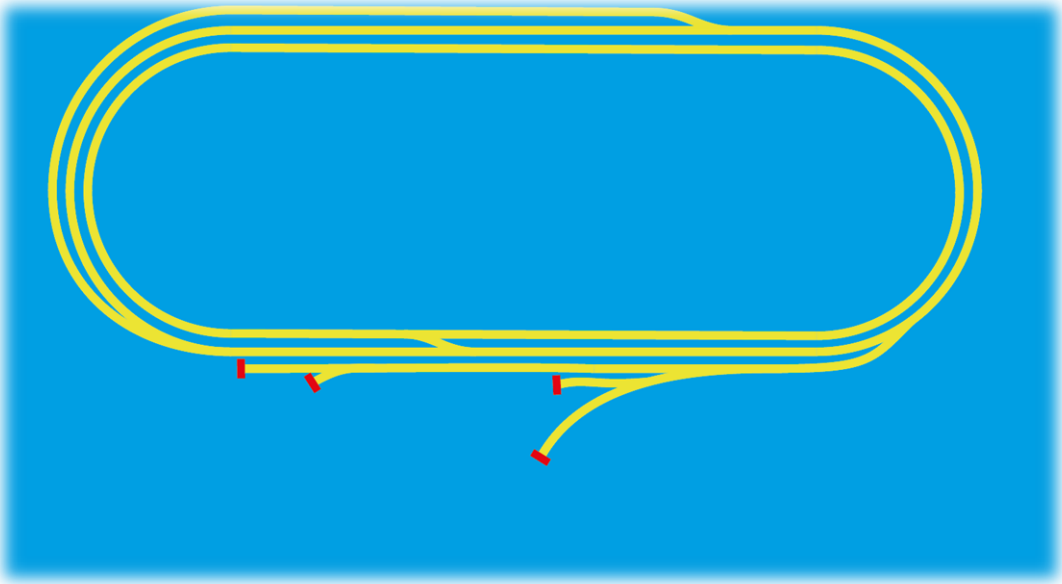


The red bars are buffers and the green boxes are decorative level crossings.

All the points are automated, there will be sensors at strategic locations although they aren't in place yet.

Though the top one would be my recommendation the folowing are possible alternatives:





X John P. Thomas
John P. Thomas
Client

X M
Michael R. Bell
Designer

DESIGN

NATURE OF THE SOLUTION

DESIGN OBJECTIVES

The basic objectives for the design are:

- The device should be simple and easy to use.
- It should control the train according to instructions given by the user.
- It should drive the train without needing any interference from the user.
- It should provide a user interface consisting of a display and some controls.
- The code should be adaptable to different situations with little modification.
- The user should not have to manually control the train.

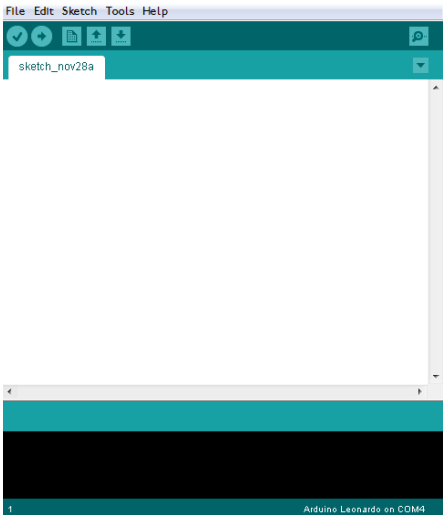
HARDWARE

PROTOTYPING BOARD

The design process began with choosing a prototyping board to use, I chose the Arduino Leonardo, depicted here.

This particular model (the Leonardo) is the first Arduino board to integrate the main processing and the USB processing onto one chip. It is just as effective as the other boards but a bit cheaper.

The idea behind getting a prototyping board is that it can easily be programmed to control the train (with the addition of the motor shield) which I will cover later.



It also has digital inputs which can be used to receive information about the location of the train and send information to relays controling the points.

Arduino uses an easy to grasp language and writing environment (depicted here) based on processing which was a beginners' language created at MIT.

MOTOR CONTROL BOARD

As well as the Arduino board an additional piece of hardware is required to output to the motor as the Arduino board has no analogue outputs and can only go up to a potential difference of 9V which is not enough to power the train at its maximum speed (12V).

There was only one clear choice for this, the Arduino motor shield (depicted here)

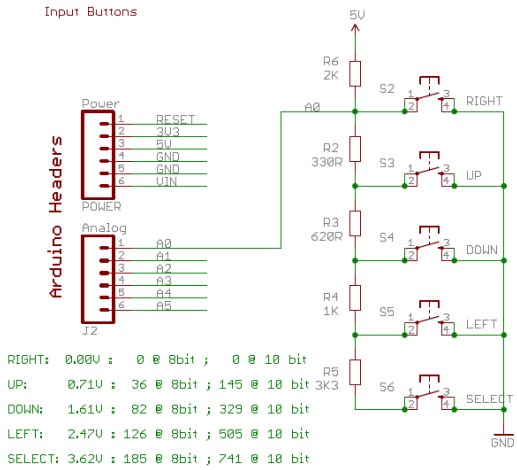
What it consists of is an input voltage terminal and two analogue motor outputs that can control the output voltage, direction and apply eddy current braking by opening a switch between the terminals.

OUTPUT DISPLAY

The final piece of hardware I needed to decide on was some sort of output. In the end I settled on the Fretronics LCD Shield. There were many LCD systems out there but this one stood out for several reasons:

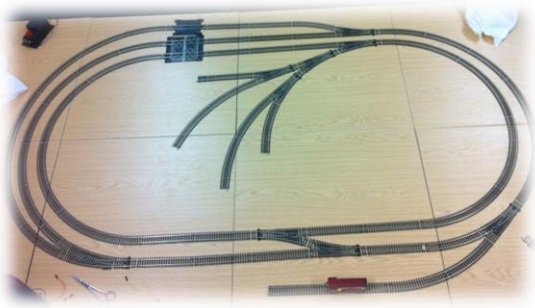
First off, it easily slots into the top of the motor shield, when the motor shield is slotted into the Arduino board and interfacing it is very easy as it uses a library built into Arduino called “Liquid Crystal” and writing to it is as simple as setting the cursor in a defined location and then writing text to it in a line from that location.

Secondly, it has a backlight in a nice shade of blue which makes the display easy to read (the brightness of the backlight can even be controlled by the Arduino board).



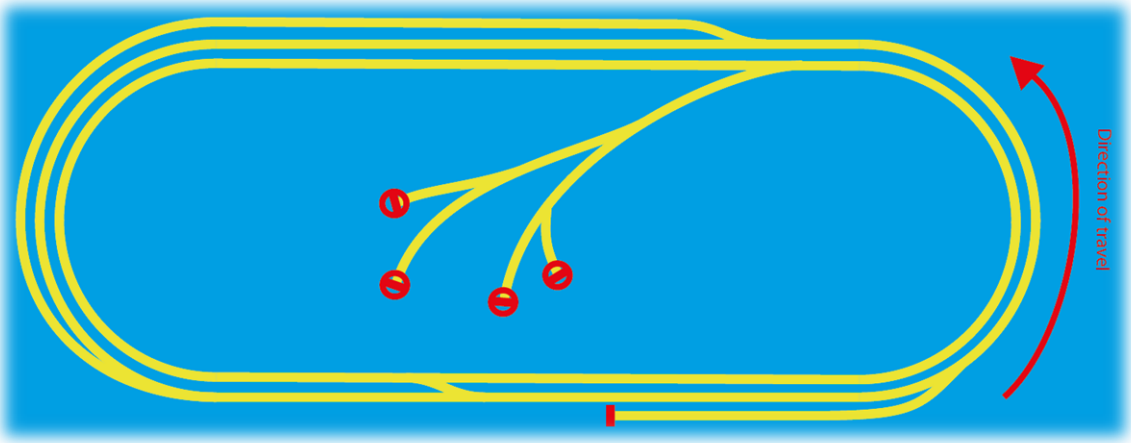
Thirdly the LCD shield has 5 buttons on it (up, down, left, right and select) which can be used to tab through a menu system to highlight and select items. In this case it also uses a clever resistor network (shown here) which means that rather than taking up 5 digital pins the buttons only take up one analogue pin by outputting a different voltage depending on which button is pressed.

TRACK

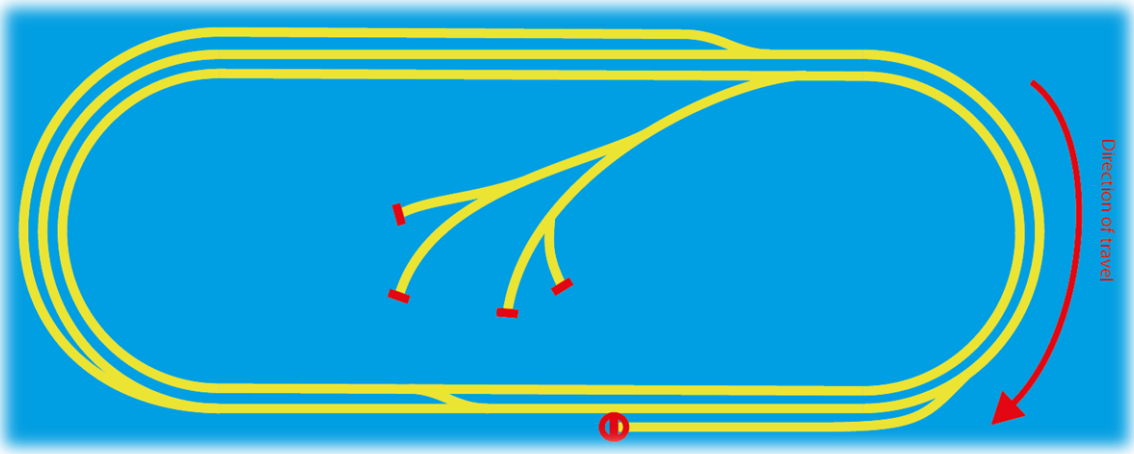


As I already own the track to be used in this project there was no decision making involved in choosing what type of track to use. This is not a great loss as there is very little choice available out there and with the options that are available there is little difference between them.

Shown here is the basic layout that Hornby suggests using when constructing the track, however, the requirement specification dictates that *"the layout should be designed that if the train is moved forward that no mater where it is it will always trigger a sensor on its journey"* this is there for a good reason as I have illustrated below:



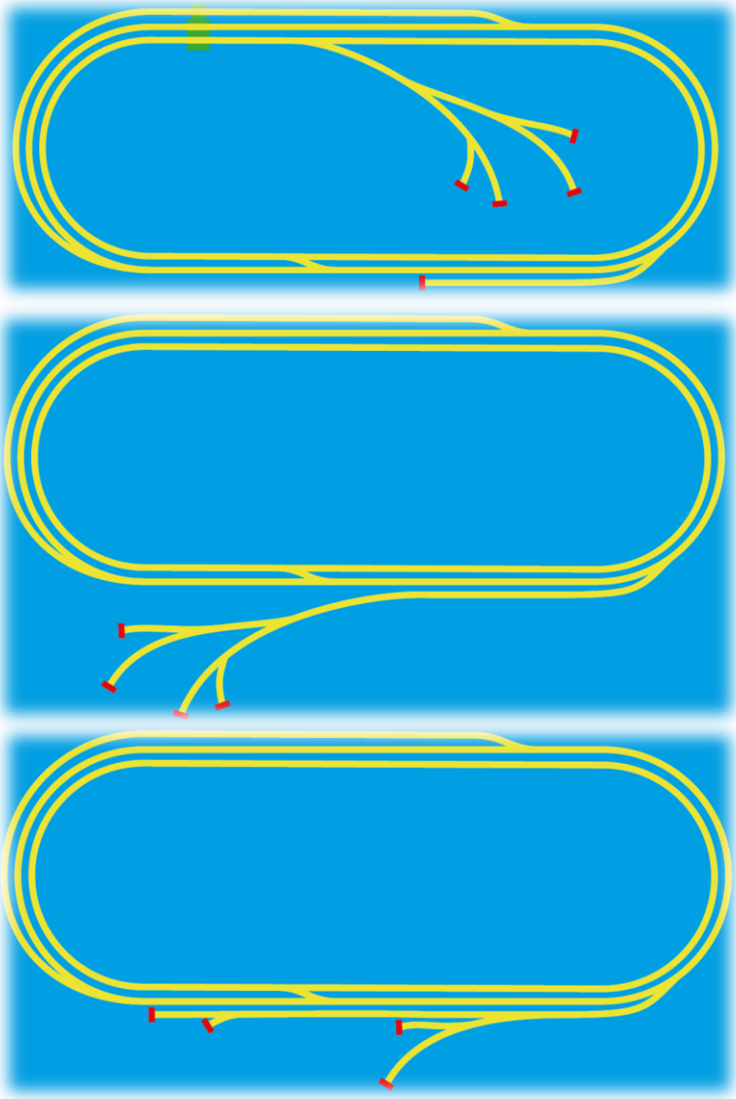
The Train is travelling clockwise around the track and, if the points are set right, it could arrive at one of the circled areas. If it does then the train will run up against a buffer and stop but the board could find itself unaware of the train's location. This is a problem if one of the sensors fails because the control board would lose the position of the train and be unaware that the train has hit a buffer, and this would then mean the train will keep running and grind its wheels into oblivion. If the train is travelling clockwise instead then this would happen:



The train can still end up in a red circle and therefore find itself in a similar situation if the board fails to realise where the train is. So we need to do something to prevent this from happening. We have several options:

1. Design a track layout where the train cannot run up against a buffer in a given direction and only oppose the direction when absolutely necessary, e.g. backing into a siding.
2. Program the board to recognise when a train has skipped a sensor by noticing that the sensor after the failed sensor has been tripped and then
    a. Compensate by skipping to the instruction set on the next sensor
    b. Stop the train if it should have stopped at the previous sensor
    c. Display an error message telling the user to replace the sensor
3. Program a failsafe into the board in which if the train has been travelling for a given time without tripping a sensor it stops and displays an error message.

These are all good options so I will try to implement all 3, the combination should prevent this problem from ever happening, here are a few designs I have chosen that conform to point 1



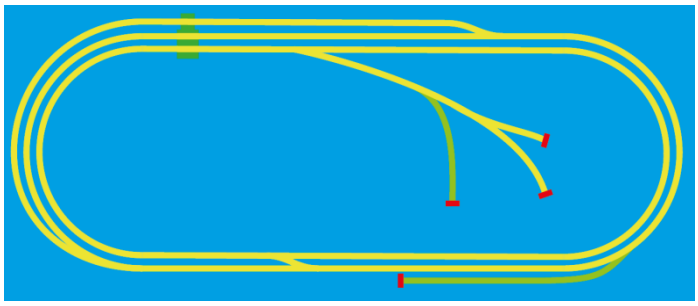
In all 3 the track is designed for the train to travel anticlockwise normally but all 3 could be reflected for the train to travel the other way

The issue with the lower two is that they both take up quite a bit of space so I chose to go with the top design.

ISOLATABLE SIDINGS

Another important element of the design is controlling which of the two trains (the main train and the cleaning train) are active and which is waiting in a siding.

My solution to this problem is to use relays. A relay is a device which can be switched on or off by sending a current to it. The idea behind using these is to move the train onto a separate piece of track that can then be used, if the design incorporates two of these then the control board can isolate one of the two trains on one of these pieces of track while the other runs.



What could happen is that the two sidings highlighted in green are separated from the other track by a tiny gap and so do not share current with them, the control board supplies them with power separately which can be switched on and off at will. When the control board wants to switch trains it backs the train to go offline onto the siding on which it belongs, then isolates that piece of track, it then sets the points for the next train and then de-isolates the piece of track that that bit is on and drives the train off it.

LOCATION DETECTION

An important part of the design is deciding how to detect the location of the train on the tracks without interfering with, or risk derailing the train. I had many different options on how to do this which I will go through.

The best place, in my opinion, to draw inspiration from is full scale railways themselves. The simplest and oldest viable method is using something called a treadle (shown here).

What happens is, the train passes over the treadle and the rim of the wheel presses the metal bar (highlighted). The bar is basically an electrical switch and when pressed down it sends a signal to a signal box. In the days before computers, these where widely used in combination with relays to control the signals and tell the signalman the position of the train.

The issue with using this on a much smaller scale is how fiddly and easily bent the metal bar that the rim touches would be if it was scaled down to 00 gauge (which is the track size), so this option isn't really feasible.

Another option is to use something called track circuits, these are created by running electric current through tracks which is spilled into other tracks by the train wheels, while this is a cheap and effective solution on real railways, they would be very difficult to use on a Hornby set because the track must be a complete circuit for the train to move.

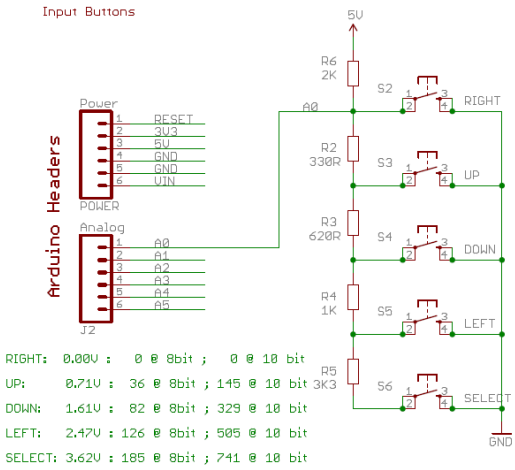
A third option is to use some kind of magnetic detection. These too are used on a grand scale. Imaginatively called a magnet. It can detect when a train passes over it using magnetism as well as measure the speed of the train and tell the cab computer what the signal ahead is showing.

Realistically of course, we don't need it to do anything more than detect the location of the train. But the idea of using magnetism appealed to me a lot so I did some research and discovered an easy way of using it to detect the location of the train.

This is a Hall Effect switch. When the south pole of a magnet passes over the upturned face it allows current to flow between the middle and right hand terminals (the left is used to power it). One of these isn't much use but if many of them where combined in a network throughout the track they could send an electrical pulse to the prototyping board every time a train passed over them, allowing the board to keep track of which block (a given section of track) the train is in without ever touching the train or interfering with the current through the train. Not only that but they are tiny and so easily fit in the small gap between the track and the train. For these reasons I

chose to use Hall Effect switches to locate the train.

The next decision to make then was how to provide the magnetic field needed to activate the switch. It needed to be strong enough to breach the gap between the train and the tracks but the magnet needed to be small enough to not touch the switch or anything else. In physics when one thinks “powerful permanent magnets” the next thought is always “Neodymium” a rare earth metal which when used in an alloy with iron and boron produces extremely powerful magnets. These magnets are usually quite expensive but as I needed them to be small they wouldn’t cost that much, show in an image of the magnets I have chosen, they are 3mm thick, perfect for attaching to the train (there is a gap of about 7mm between it and the track).

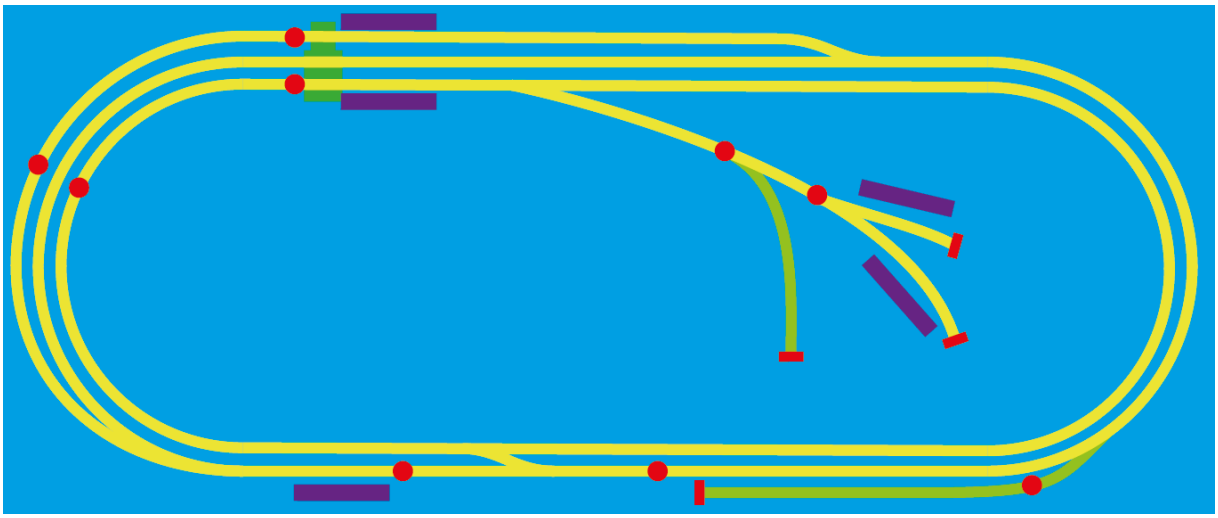


After deciding to use the Hall Effect switches I then had to decide how to receive information from them with the prototyping board This was important because there wouldn’t be enough inputs on the board to receive data from each sensor individually, luckily though, I could draw inspiration from the open source design of the LCD display, which took the inputs from 5 different buttons but only used one analogue input plug to do it. Here is the section of the schematic showing how it worked.

The idea is that it outputs a different voltage depending on which button is pressed. All I had to do was replace the buttons with the Hall Effect switches and read the analogue pin to find out which switch is being pressed.

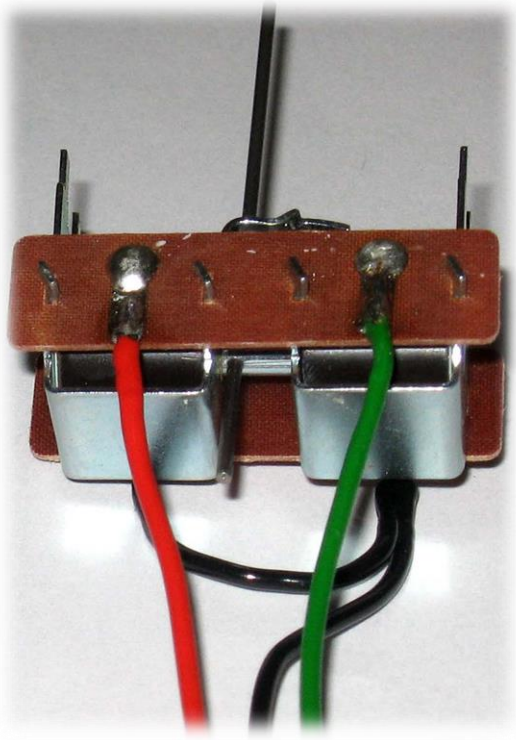
This can easily be adapted to be used in Beltrak.

The last phase in the design was to decide where to place the sensors on the track -



In this image the purple bars represent stations, the green track is an isolatable siding and the red dots are sensors.

POINT CONTROL



In order to control where on the track the train goes it is not only necessary to control the speed and direction of the train, but it is also necessary to control the points.

Hornby suplies point motors to be used to move points on manualy operated train sets, it is possible to control these using the arduino board as well.

My original idea for controling the points was to use the motor shield to suply the correct voltage to move the motor and to use a specialy created relay network to select the correct set of points to move.

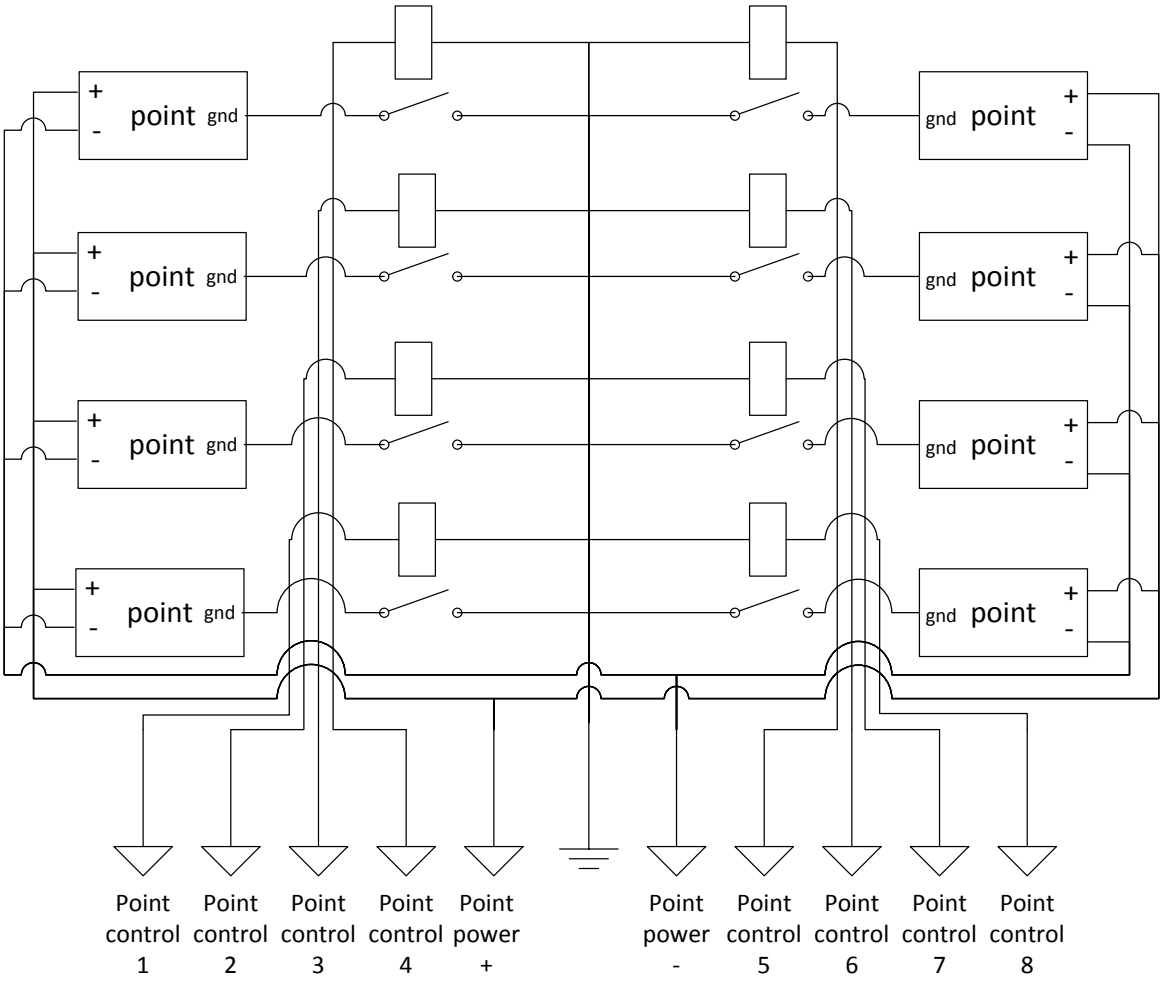
On the back of the boxes for the point motors it claims that the minimum potential diference for the point motor to move is 12V DC which is conveniently the suply voltage for the train, supplied by the motor shield.

Knowing this, I set about producing a system whereby multiple sets of points can be controled independently using the same output from the motor shield (as there are only two and the first is used to control the train).

My idea was to use a set of relays[1] which were switched using the digital outputs from the main Arduino board to control which points receive the power suplied by the motor shield.

[1] Devices which are switch on or off a current based on the supply of a smaller current.

This was my original design.



The point motor (as shown on the previous page) has 3 wires comeing off it, the black wire goes to ground and the red and green wires are atached to a supply. Putting a voltage over the red wire pulls the point toward it as does the green. In the design, the relays are used to switch on or off the ground wire (if it is switched off the point won't budge) and the board then puts a voltage over either all the green or all the red wires and the points that are switched on will move.

In the diagram the arrows at the bottom represent the pins on the arduino board and motor shield. When pulled HIGH[2] the point control pins supply 5V and the point power pins suply 12V.

The rectangular objects are the coils of relays. When they are supplied with the 5V from the board they close the switch adjacent to them, grounding the point they are attached to, then a pulse of power is sent down either the + or – pins, theoreticaly flipping the points.

Preliminary testing however showed that this system would not do.

While the point motors would move correctly they did not move with enough force to move the points. This was most likley a combination of two factors;

[2] Switched on by the board

1. The board actualy supplies about 11.9V which is under the minimum rating for the points, the maximum is 16V and so the points where moving at minimum capacity and

2. The points in my train set are old and slightly rusty and require more force to move than a brand new set.

My solution to this problem was to suply a higher voltage to the points, unfortunatly this threw up an additional problem as no part of the arduino system will accept a voltage over 12V[3] without risking serious, irreparable damage to the system.

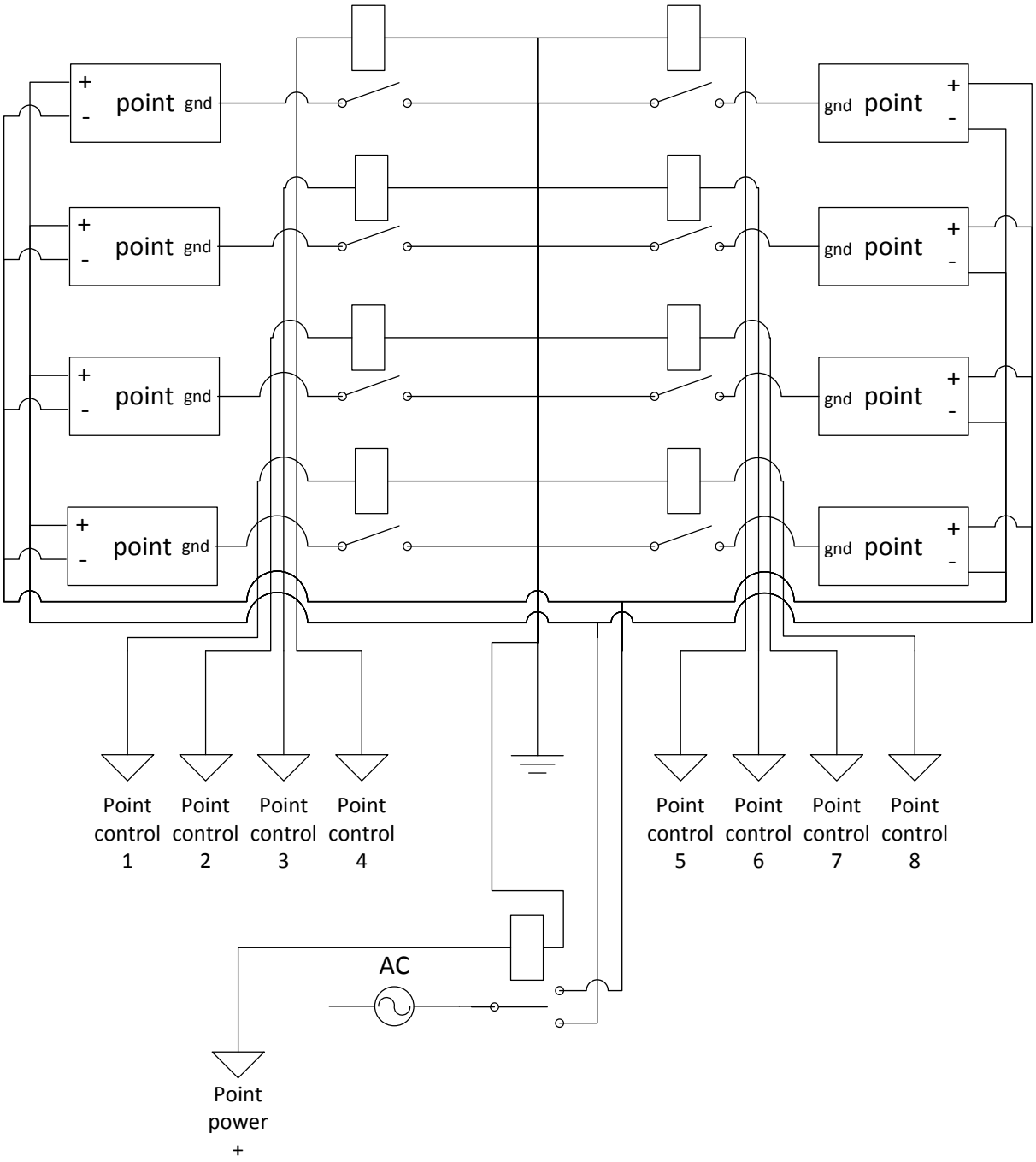
This problem was overcome by adding an additional, larger relay to the system which was connected to the terminals which where previously used to supply the points with their original 12V.

This kind of relay was called a “single pole, double throw” which can chose between two different current outputs.[4] Using this meant that either the point power + supply could be on or the point power - supply could be on, meaning that the position of this relay could be used to select a direction to move the points.

My chosen relay could switch curents up to 240V which was plenty given that it only needed to switch 14(ish)V.

[3] 12V for the motor shield and 9V for all other parts.
[4]This differs from a “double pole, double throw” where two switches or ‘throws’ can be moved independently of each other.

With these factors taken into acount my new design took this form.



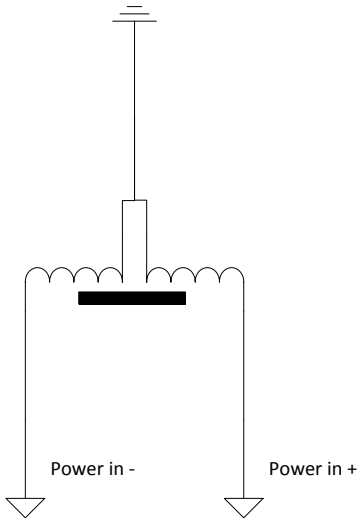
The new relay has been added to the design at the bottom as well as the AC supply from the original point motor supply. This was something I was originaly not planning on using but as it also output the 12V DC for the train to run on it turned out to be quite convenient.

Unfortunatly, this design also had a big flaw in it.

I encountered the problem during the assembly of the track, after installing only one point motor the system worked perfectly, but, after installing an additional motor, and then another, the system stopped working.

Instead of moving the points the selected motor buzzed and moved slightly but not enough and the other point motors jolted a tiny bit and moved a little.

This lead me to believe that the point motors where all grounded instead of just the selected one but after testing the grounds with a multimeter this was disproved. It took many hours of pondering, but I eventually discovered the problem which was not so much in my design but more in the way the point motors worked and to understand this problem we will have to look inside a motor.



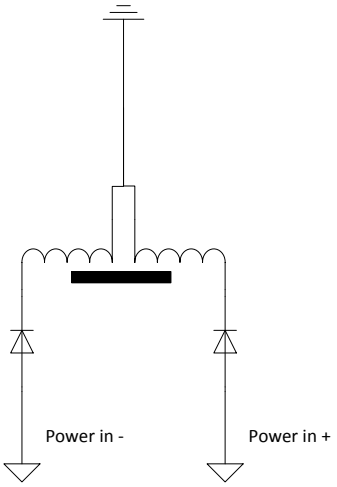
This is the standard motor design.

When power is suplied into either power in + or power in – the coil nearest it produces magnetic flux, pulling the point bar towards it and moving the points.

In a stand alone system this works because the only route the curent can take is from the power pin too the ground.

When multiple motors are attached to my design however, the current can take a different route. If point 1 is grounded and power suplied through pin + some current takes the standard route but some current flows through both coils of point 2 (which isnt grounded) and out of its – pin, because all the + pins and the – pins are connected. It can then flow into the – pin of point 1, activating the wrong coil and resisting the motion of the points, stopping motion in point 1 and making point 2 jiggle.

Once the problem was identified it was easy enough to find a solution.



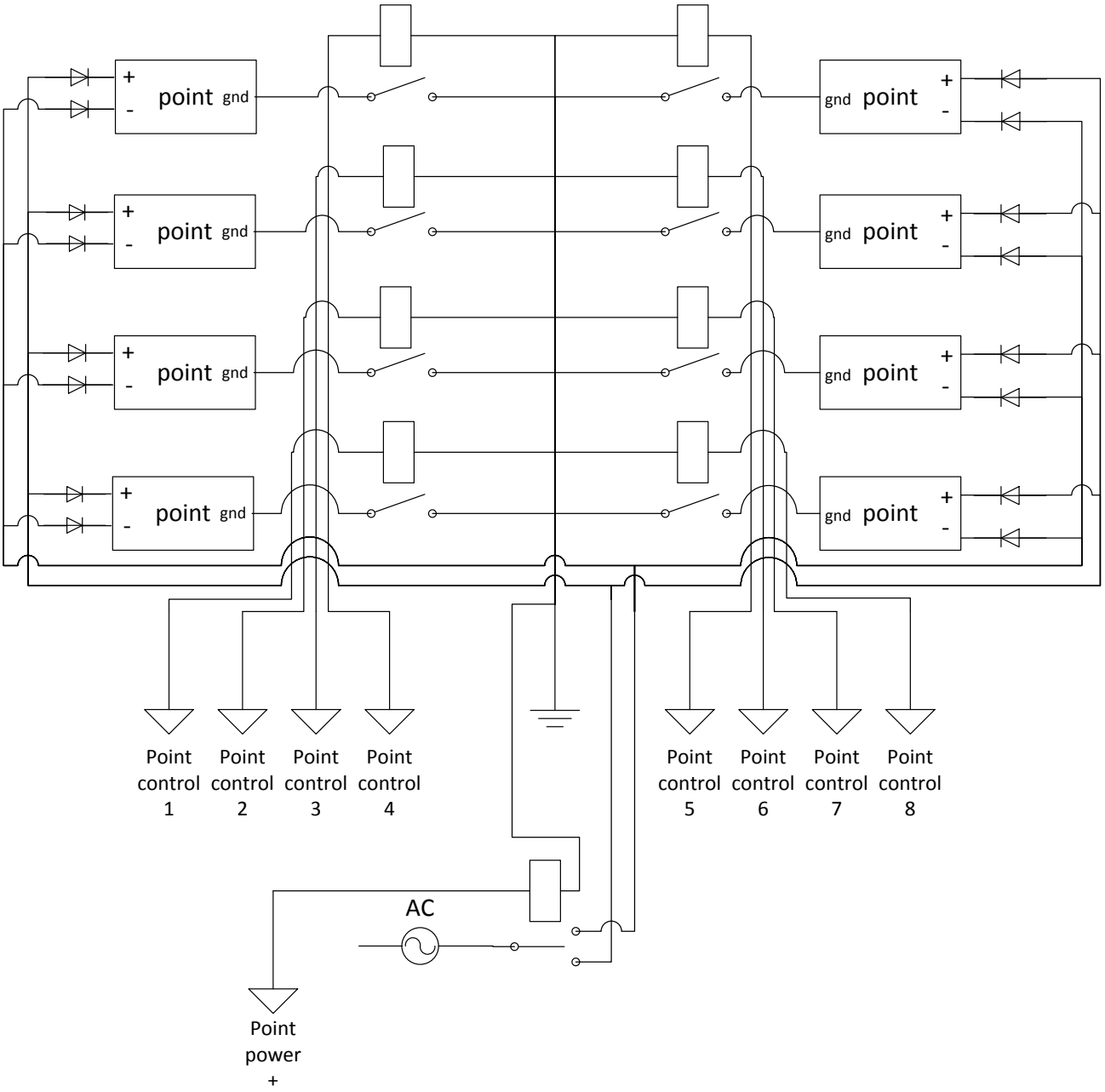
This is my modified motor design.

The problem is solved by using two diodes that face each other over the two coils.

Diodes only allow current to flow in one direction. When an AC supply (which is what we are dealing with here) is attached to a diode, it takes out one of the two directions which the AC supply alternates between (as it normaly goes in both directions).

This doesn't effect operation but if the current which has been stripped of one direction then flows over another diode facing the other way then it is also stripped of the other direction, stopping flow. This forces current flowing in on either pin to go to ground as it can only pass one of the two diodes.

With the addition of these diodes, this is the final design of the point motor circuits.



I am pleased to say that this new design works!

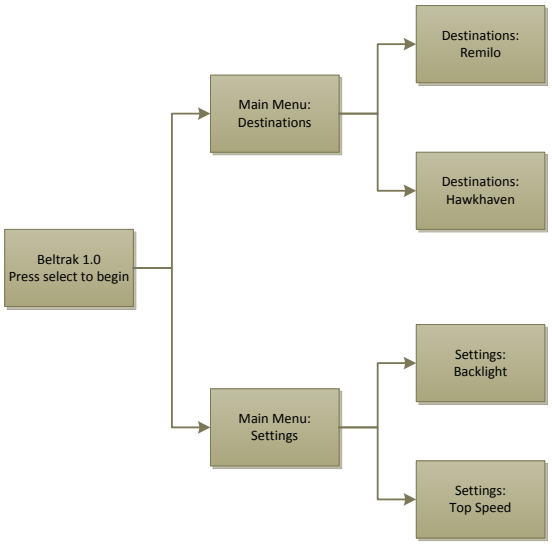
DESIGN OBJECTIVES

After much deliberation I decided to program the Arduino board to run as a finite state machine.

This means that the machine will be in a fixed state, for example, speeding up, and is programmed to move into another state when a certain triggering condition is met. For example, reached maximum speed" at which point it would switch into a different state e.g. "travelling at constant speed".

This has several important advantages, most of all the fact that the requirements for changing state can be programmed into an array. Then the array can be edited to allow for changes in sensor positions or track arrangements without having to change the base code. This means that the software could potentially work for any track arrangement simply by reprogramming the array. It also saves writing out the code repeatedly for different conditions as the array makes the code adaptable

I have also decided on using a simple menu structure for the LCD display that resembles a tree with branches:



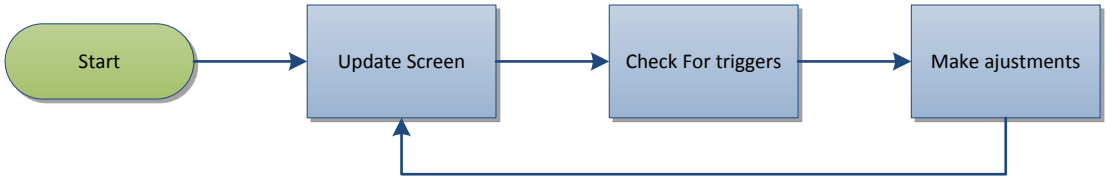
Navigation through this system should be easy enough, the arrow keys navigate the paths and the select key chooses the final option.

This method was outlined in the requirement specification and I still believe it to be the best way of providing a menu within the constraints provided, that is to say, a 16 by 2 LCD display with 4 arrow keys and a select button.

There will also be certain other screens displayed, for example when it's starting up a loading screen might be displayed and when the train is running a progress screen might be displayed.

PROCESSES AND MODULES

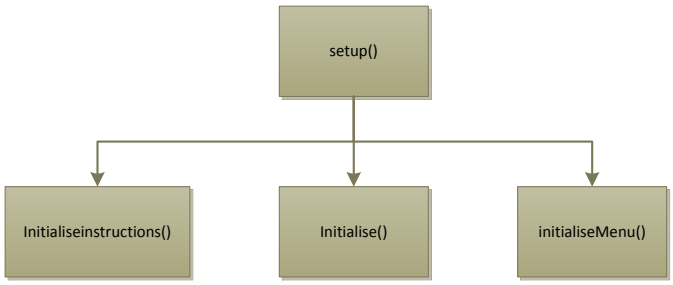
OVERALL PROCESS



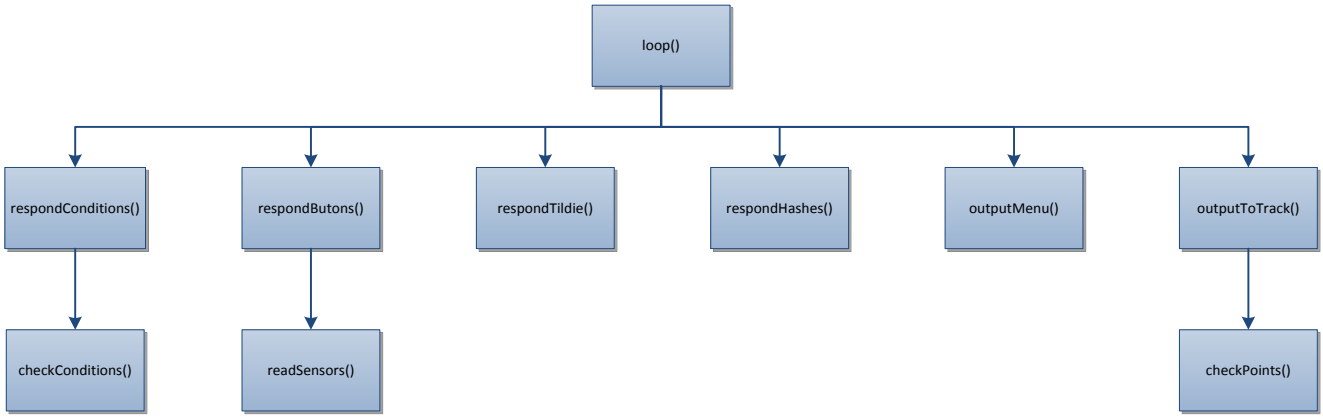
This is the full process for the Beltrac software, the process is looped over and over again, endlessly from power on to power off. The code (and therefore the design) can be divided into these 3 logical sections.

In the Arduino environment you start with a main file containing two functions, setup() and loop(). Setup is run once when the system starts and loop is run repeatedly until power off.

Before dividing up these sections I had to create something to fill setup. This took the form of 3 functions purely for initialising variables. They are shown below.



After running these the board then runs loop() which consists of the following functions.



All of these functions combined make up the program.

DATA STRUCTURES

MENU STRUCTURE ARRAY

The menu structure array is a 3 dimensional array in which the text for the display and the sectional instructions are stored based on their position in the structure, as shown here:

Position	[0][]	[1][]	[2][]
[][0]	Welcome to Beltrac	Destinations	Hawkhaven
[][1]	~	#	Remilo
[][2]	~	#	~
[][3]	~	Settings	Backlight
[][4]	~	~	Top Speed

This table shows the contents of [][][0] (the first line of text) and [][][1] (the second line of text).

To understand this system imagine starting on the welcome to Beltrac cell, if you press an arrow key and you don't go over the edge of the table you move in that direction. If you reach a tilde you move back one in the direction you came. If you reach a hash you move forward one in the direction you entered the cell. When you reach a piece of text that is displayed on the screen.

This is a rough outline of the array as the full version would be much more complicated.

SWITCHING CONDITIONS

The conditions for switching between different states will be defined by a three dimensional array containing encoded conditions.

The idea behind this is that when the board initiates a journey it looks up the conditions in the array where they are stored for a particular destination or action.

In the array the 1st dimension is the number for a destination or instruction for example [1][x][x] might contain directions to get to Hawkhaven, [30][x][x] might send out the cleaning train.

The second dimension contains the position in the instruction set for example [x][0][x] is the first instruction, [x][4][x] is the fifth instruction.

In the third dimension, position [0] is the condition, [1] is the value of that condition, [2] is the state to change to when hitting that condition, [3] is the value of that state, for example:

[0]	[1]	[2]	[3]
B	4	S	1
In block	Four	Set speed to	Setting one

Conditions:

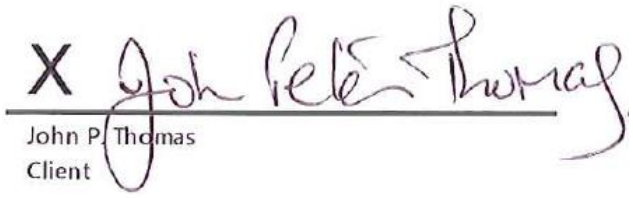
Code	Condition
B	In block x trigger this condition, where a block is when the train is over a sensor and x is the number of that sensor
W	Start a timer for X hundred milliseconds, trigger when the timer ends

States:

<table>
<tr><th>Code</th><th>Action</th></tr>
<tr><td>S</td><td>Set the Proportional Potential Difference and therefore the speed to x where x means:

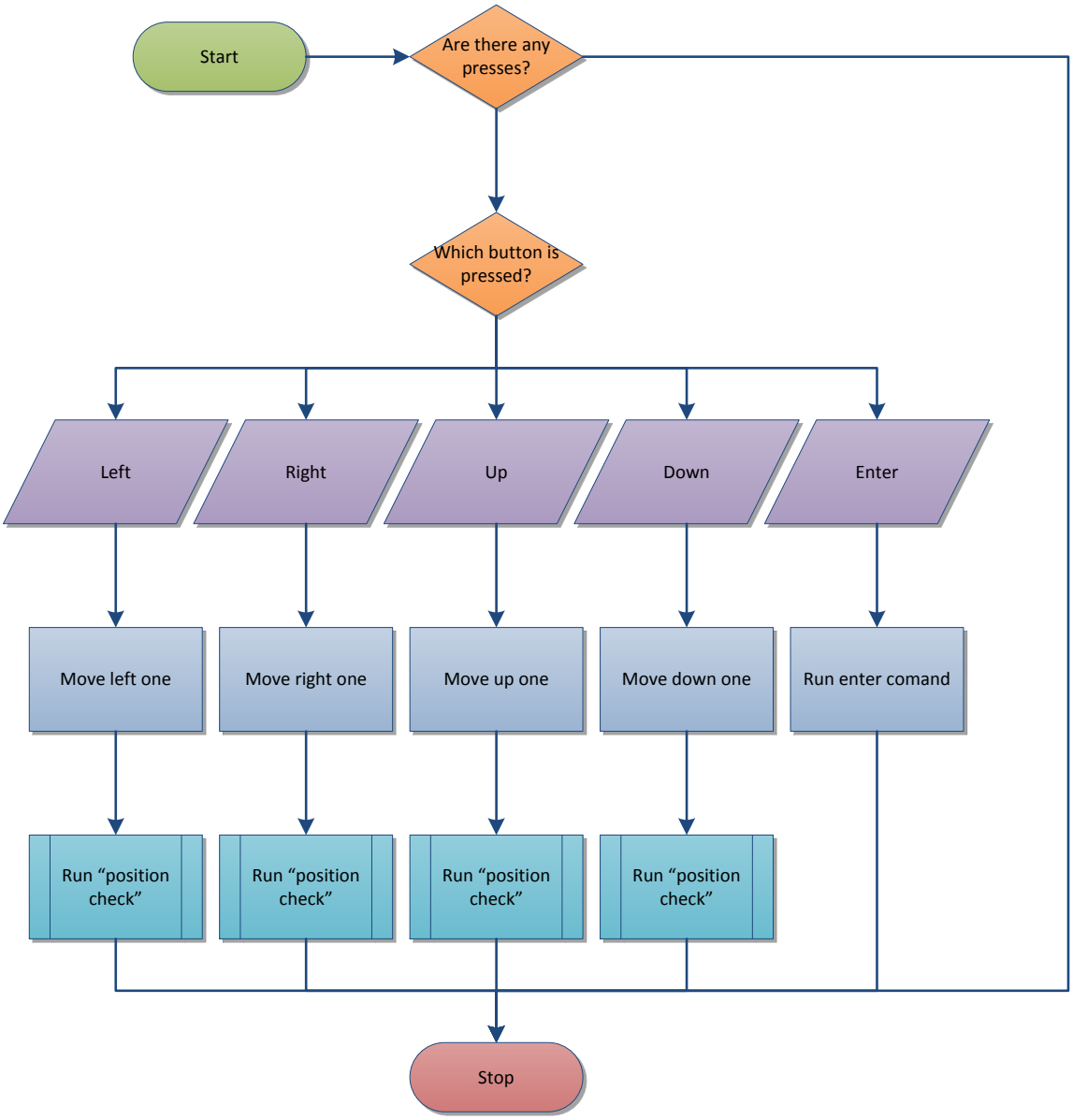
<table><tr><td>[X]</td><td>PPD</td></tr><tr><td>0</td><td>0%</td></tr><tr><td>1</td><td>50%</td></tr><tr><td>2</td><td>100%</td></tr><tr><td>3</td><td>-50%</td></tr><tr><td>4</td><td>-100%</td></tr></table></td></tr>
<tr><td>C</td><td>Set point x to converge</td></tr>
<tr><td>D</td><td>Set point X to diverge</td></tr>
<tr><td>X</td><td>Program complete, wait for input – when this is called the train is stopped, then the interface is unlocked and waits for input</td></tr>
</table>

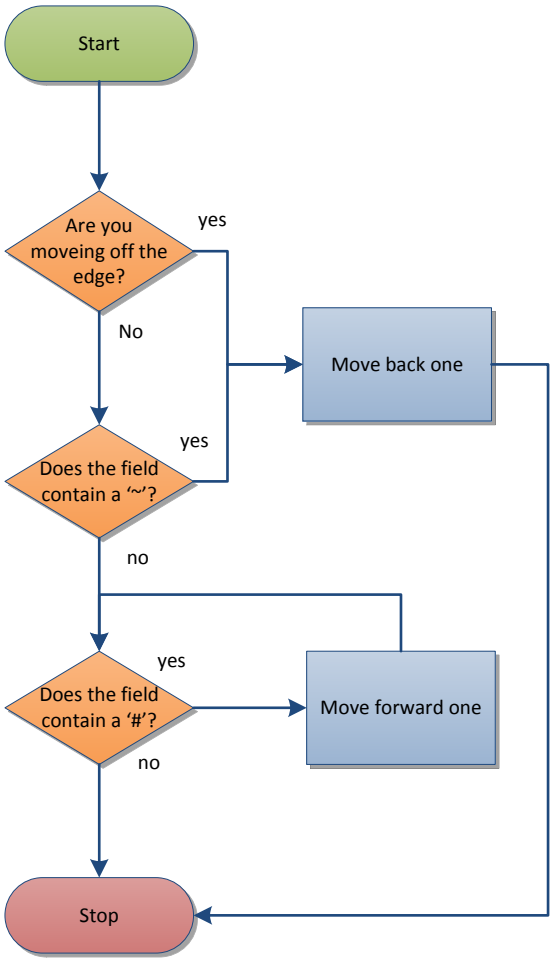
These designs where showed to Mr Thomas who signed below to confirm that they were satisfying.



UPDATE SCREEN

Shown below is the algorithm to check for and respond to a button being pressed.



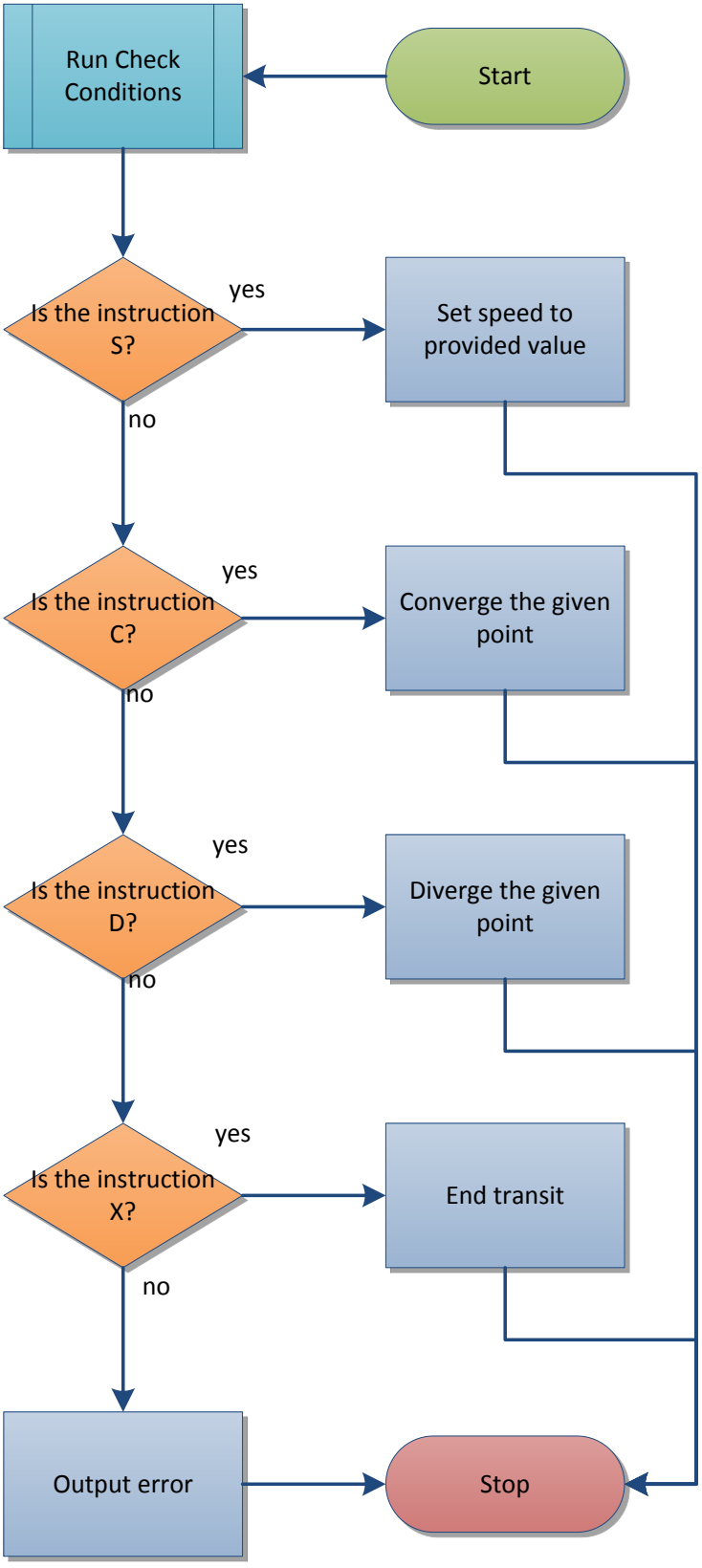


Shown here is the "run position check" mentioned in the previous algorithm.

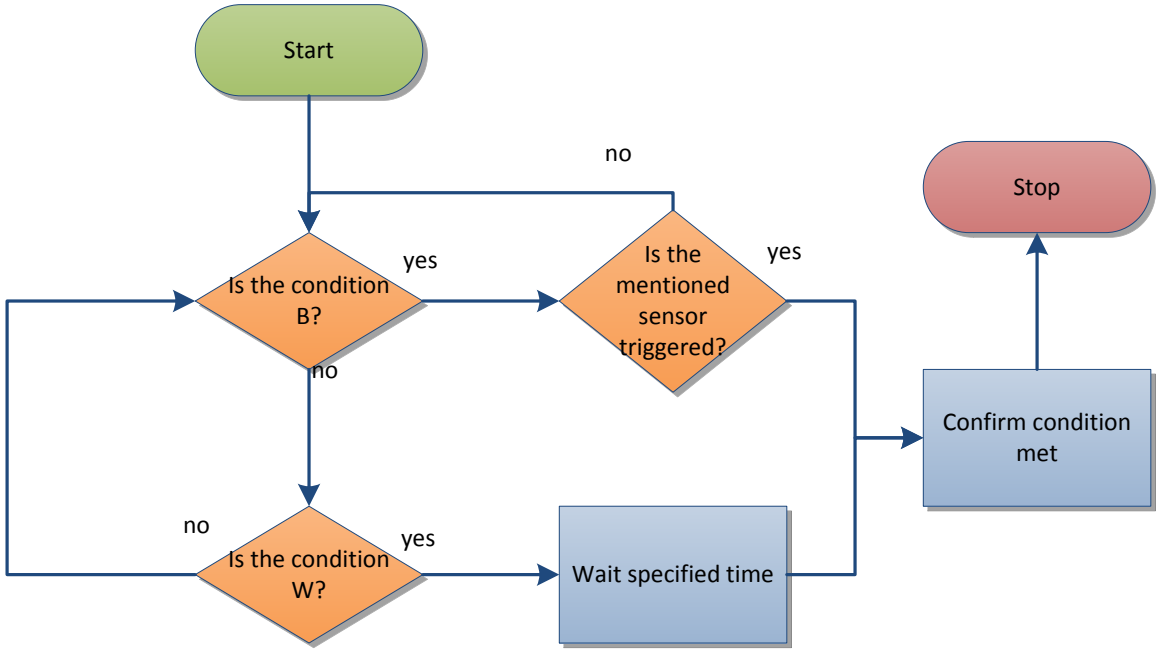
As outlined in the hardware design, the hardware for the buttons and the sensors is roughly the same, it is a network of resistors that outputs a different voltage depending on which button is pressed or which sensor is triggered.

RESPOND CONDITIONS

When driving the train the system moves through a number of different instructions. This is the algorithm for reading the instructions.



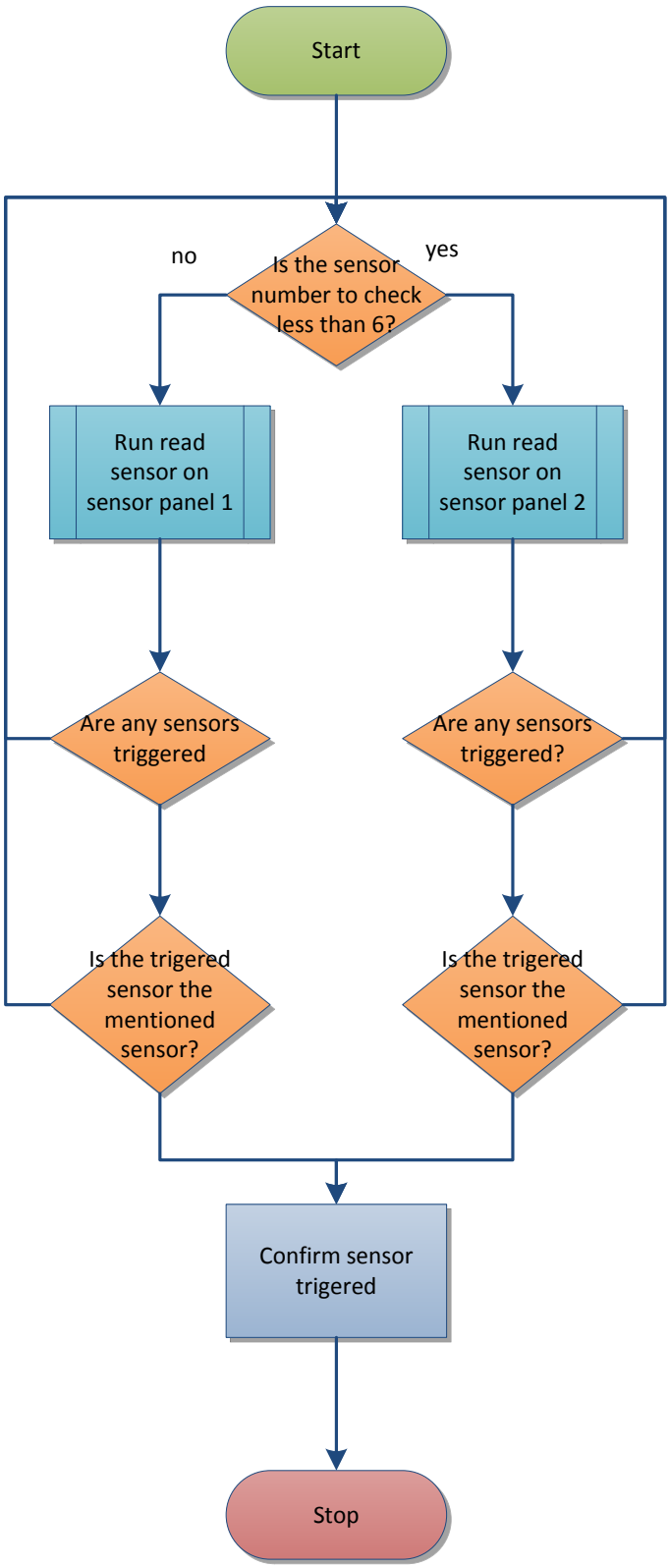
This is the mentioned Check Conditions, which is looped until the condition is met, at which point the algorithm above continues.



This full set is looped while the train is in transit.

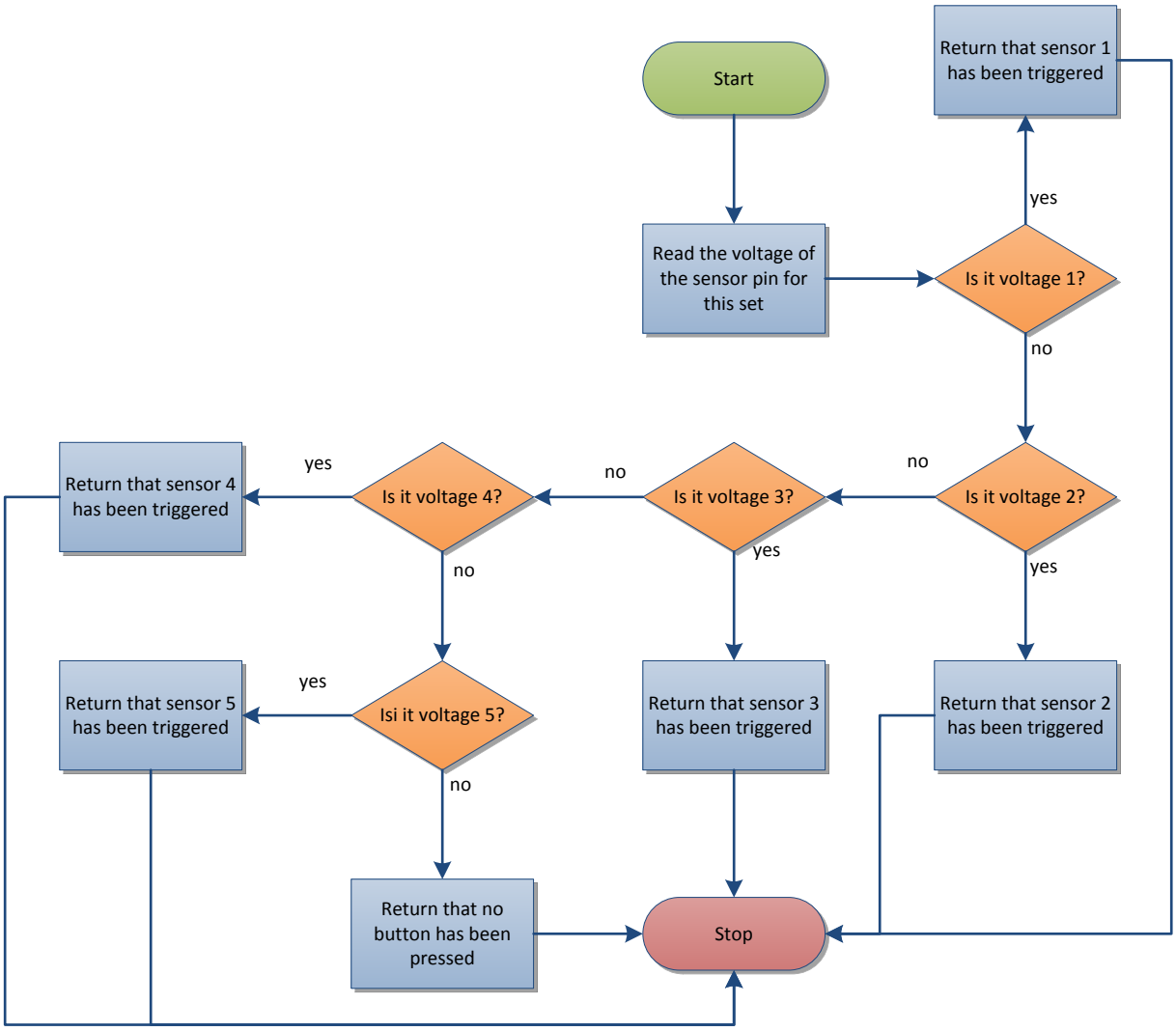
CHECK SENSOR

This algorithm is run whenever the board wants to know if a sensor has been triggered.



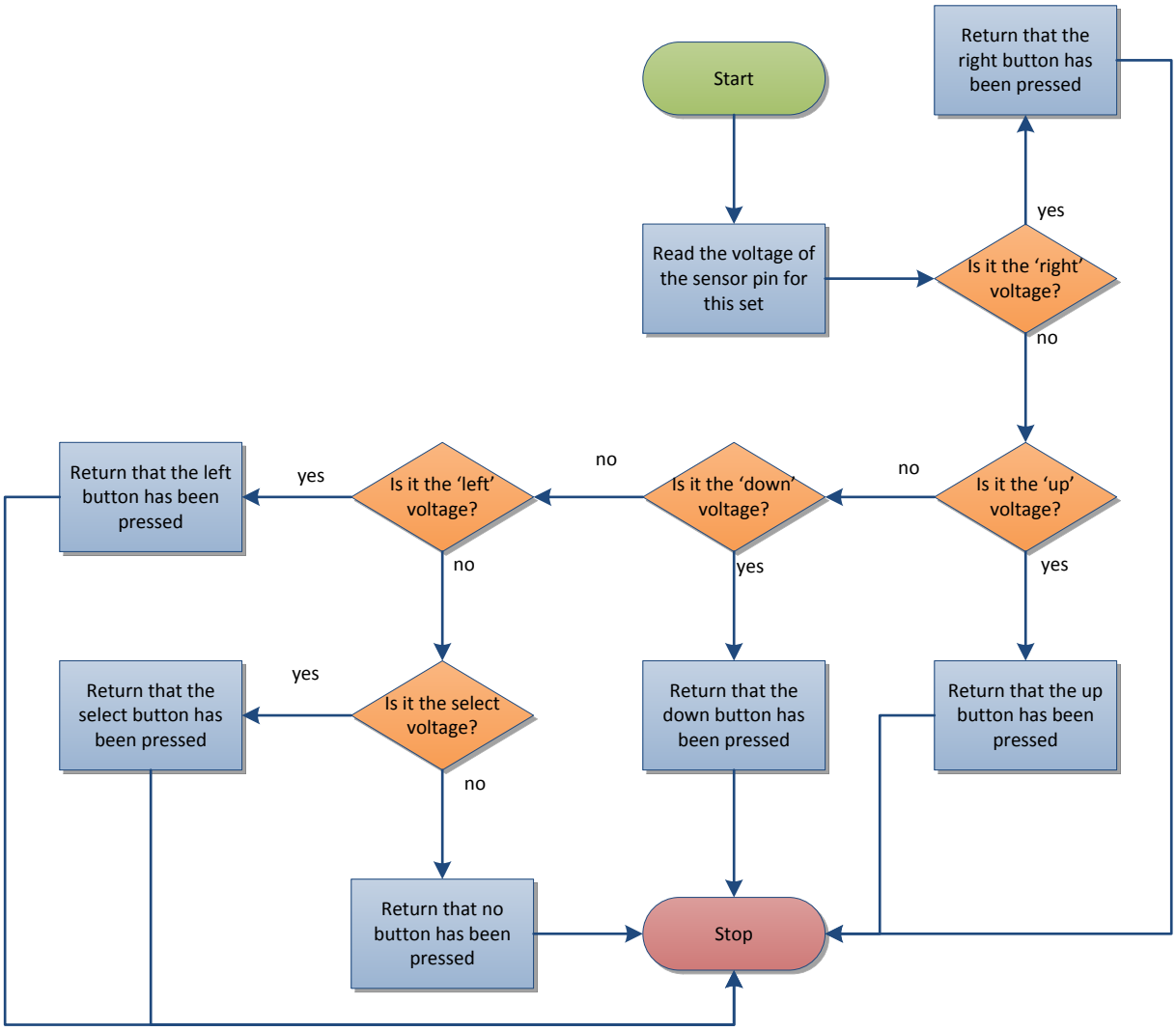
READ SENSOR

This is the algorithm mentioned above which reads the sensors



READ BUTTONS

This algorithm is similar to the one above but it reads the buttons instead.



TEST STRATEGY

Beltrac has two interfaces that need to be tested, one of them is the interface and the other is the train on the track.

INTERFACE TESTING

The interface must be tested to ensure that it can be correctly navigated by the end user and that each option it presents, when selected, does as it says.

NAVIGATION THROUGH OPTIONS TESTING

This testing is to ensure that the user can navigate through all possible options on the menu without getting stuck or revealing any item that is not intended to be displayed, for example a hash or tilde. When performing these tests the tester must move from the start of the menu to the item displayed and then move back to the beginning again by continuously pressing the back button, this ensures that not only can the object be reached but that all other objects can be reached as well.

Object	Can be reached	Can return to menu
Destination: Hawkhaven		
Destination: Remilo		
Destination: Allantown		
Destination: Gregville		
Destination: Leovetticutte		
Destination: Regantra		
Destination: Vancoville		
Setting: Top Speed		
Setting: Backlight		

SELECTION OF OPTIONS TESTING

This testing ensures that when the enter button is pressed the selected option is activated if it is intended to and nothing happens if it is not. If it was intended to perform a function then it must perform the correct function to pass.

Object	Expected Result	Result Met
Welcome Page	-	
Destination	-	
Destination: Hawkhaven	Select Instruction Set 1	
Destination: Remilo	Select Instruction Set 2	
Destination: Allantown	Select Instruction Set 3	
Destination: Gregville	Select Instruction Set 4	
Destination: Leovetticutte	Select Instruction Set 5	
Destination: Regantra	Select Instruction Set 6	
Destination: Vancoville	Select Instruction Set 7	
Settings	-	
Setting: Top Speed	Open Speed Selector	
Setting: Backlight	Open Backlight Setter	

NAVIGATION BOUNDRY TESTING

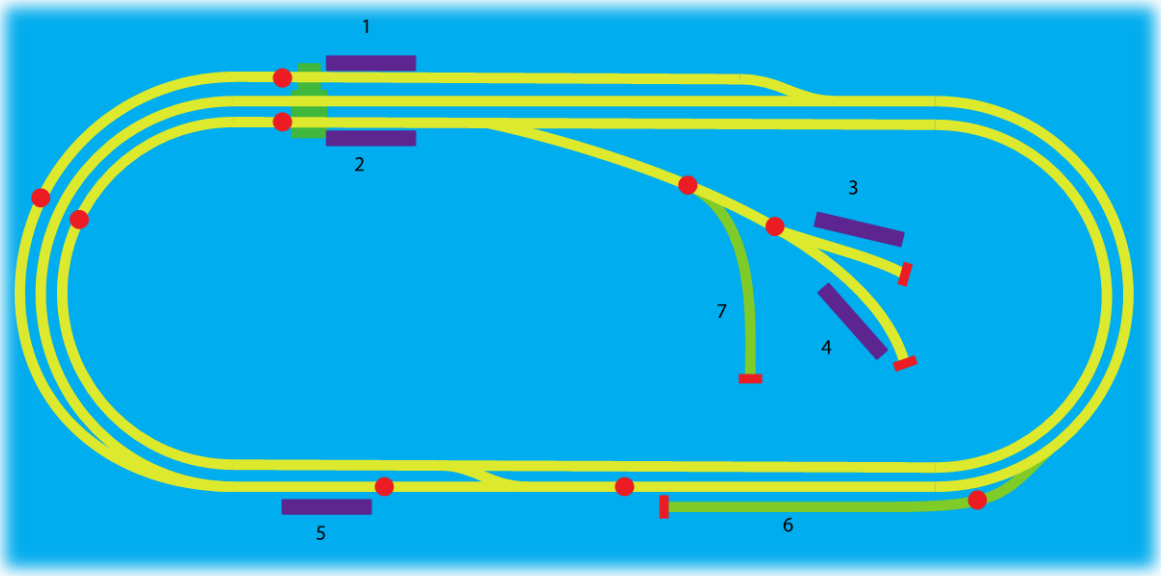
This testing is to ensure that the selector never goes out of bounds at any point. It lists an item to be selected before the test begins and then an arrow key to press, along with what is expected to happen the test is passed if the expected action happens.

Object	Direction	Expected Result	Result Met
Welcome page	Up	-	
	Down	-	
	Left	-	
	Right	Destination	
Destinations	Up	-	
	Down	Settings	
	Left	Welcome Page	
	Right	Hawkhaven	
Settings	Up	Destinations	
	Down	-	
	Left	Welcome Page	
	Right	Top Speed	
Hawkhaven	Up	-	
	Down	Remilo	
	Left	Destinations	
	Right	-	
Remilo	Up	Hawkhaven	
	Down	Allantown	
	Left	Destinations	
	Right	-	
Allantown	Up	Remilo	
	Down	Gregville	
	Left	Destinations	
	Right	-	
Gregville	Up	Allantown	
	Down	Leovetticutte	
	Left	Destinations	
	Right	-	
Leovetticutte	Up	Gregville	
	Down	Regantra	
	Left	Destinations	
	Right	-	
Regantra	Up	Leovetticutte	
	Down	Vancoville	
	Left	Destinations	
	Right	-	
Vancoville	Up	Regantra	
	Down	-	
	Left	Destinations	
	Right	-	
Top Speed	Up	-	
	Down	Backlight	
	Left	Settings	
	Right	-	
Backlight	Up	Top Speed	
	Down	-	
	Left	Settings	

	Right	-	

TRAIN ON THE TRACK TESTING

The train on the track must be tested to ensure that it can navigate to all required areas of the track without hitches from many different positions on the track. For example, rather than simply going from A to B the train must be able to get to B from anywhere on the track where it would normally stop.



This diagram shows the positions of the stations and sidings where the train would be required to stop. Testing this is easy but long winded. This grid shows all the tests that must be performed:

		Start						
		1	2	3	4	5	6	7
Destination	1							
	2							
	3							
	4							
	5							
	6							
	7							

The tests are performed by filling in the white squares. When the train successfully gets from the start number to the finish number and the finish number corresponds to the one selected (if it arrives at the wrong finish number it doesn't count.).

USER ACEPTANCE TESTING

The user acceptance testing is performed in questionnaire style where the person performing the questionnaire circles a number from one to five showing how strongly they agree to the statement

They are then asked to answer some general questions about how they rate the product.

RATEING THE END PRODUCT

Statement	Strongly disagree	Disagree	Unsure	Agree	Strongly agree
Beltrac performs to my expectations.	1	2	3	4	5
I am happy to use Beltrac in the setting that I originally intended it.	1	2	3	4	5
Beltrac is aesthetically pleasing.	1	2	3	4	5
Beltrac meets the design specification	1	2	3	4	5
Beltrac meets the requirement specification	1	2	3	4	5
Having used Beltrac, I am happy with its interface	1	2	3	4	5
I found Beltrac easy to use	1	2	3	4	5
I was amused by Beltrac	1	2	3	4	5

RATEING THE PERFORMANCE OF THE DESIGNER

Statement	Strongly disagree	Disagree	Unsure	Agree	Strongly agree
Mr Bell performed to my expectations	1	2	3	4	5
I would consider commissioning Mr Bell again	1	2	3	4	5
Mr Bell's workmanship is of an acceptable quality	1	2	3	4	5
Mr Bell's designs are of an acceptable quality	1	2	3	4	5

GENERAL QUESTIONS

What pleases you about Beltrac?

What displeases you about Beltrac?

Did Beltrac fail to meet any of your expectations and if so, which ones?

Do you have anything else to add?

SOFTWARE DEVELOPMENT AND TESTING

BELTRAK.INO

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

//declarations of librarys
#include <LiquidCrystal.h>

//initialise librarys
LiquidCrystal lcd(8, 9, 4, 5, 6, 7);

//declare global variables

//train control variables
float PPD; //Percentage Potential Difference -- what the board multiplies
the maximum voltage by, this controls the speed of the train with 100 as
the maximum and 0 for stop (PPD does not control direction)
boolean reverser; //controls the direction of the train, when TRUE the
train is reversed

//track control variables
//boolean pointState[10]; //this says the curent state of the points with
FALSE for converge and TRUE for diverge
//boolean pointSwitch[10]; //this holds the desired state of the points
with FALSE for converge and TRUE for diverge

//pin number constants
#define pinPD 3 //the pin number for the Potential Diference output
#define pinDIR 12 //the pin number of the direction pin
#define pinButtons A0 //the button input pin
#define pinLowSensors A1 //the first 5 sensors
#define pinHighSensors A2 //the last 5 sensors

#define point1 0 //the pins for the point relays
#define point2 1
#define backlightPin 2

int backlight;


//          #define pointPower 11
#define pointDir 11

//the instruction array
```

```
char inst[8][7][4]; //array containing the switching instructions

//the menu array
String menu[3][10][2];

//position in the menu
int menuPosX;
int menuPosY;

//the position in the instruction array
int instSet; //the set of instructions to be folowed, eg. go to hawkhaven,
this is set by the user through the menus
int instPos; //the position in the instructions, when a condition is met
and a state is changed this incriments, when instSet changesm, this becomes
0


//timing variable
/*when condition W is in force this is incrimented every iteration of loop
untill it meets the given value at which point it
 is reset and met is set to true*/
int timer;

//virtual sensors
/*these are variables that are used to test the program before sensors are
intoduced true means HIGH false means LOW, !remember
 to change these in the code when introducing sensors!*/
//boolean VS[5];
//boolean sensor[10];

//tranition boolean
/*when this is false the train is stationary and the menu is displayed,
when it is true, the program executes the given
 instructions and the UI is locked*/
boolean inTransit;

//button voltages
/*these are the ADC readings taken on A0 and the button presses they
represent*/
#define rightADC 0
#define upADC 145
#define downADC 329
#define leftADC 505
#define selectADC 741

//button voltage sensitivity
#define ADCsensitivity 10

//button output numbers
/*to make the output from the buttons easear to understand these are used
in place of the numbers representing the button output*/
#define noneOut 0
#define rightOut 1
#define upOut 2
#define downOut 3
#define leftOut 4
#define selectOut 5

//sensor output numbers
#define sensorNone 0
#define sensorOne 1
```

```
#define sensorTwo 2
#define sensorThree 3
#define sensorFour 4
#define sensorFive 5

#define sensorSix 6
#define sensorSeven 7
#define sensorEight 10
#define sensorNine 9
#define sensorTen 10

//anti-multipress boolean
/*to prevent the board from reading a button as pressed multiple times the
board sets this to true when it responds to a press
 it will then not respond again untill this is set to false by the board
detecting that no button is pressed*/
boolean butonCaptured;

//last menu move
  int lastMenuMove;


void setup()
{
  initialise();
  initialiseInstructions();
  initialiseMenu();
}

void loop()
{

  /* the "loop" is devided into 3 logical sections,

   section 1: the menu position is ascertained and text is output to the
screen, variables are ajusted to provide the other sections with user input

   section 2: the instruction array is queried and the sensors are checked,
variables are ajusted to set the curent instructions for the train

   section 3: variables from section 2 are checked and the points are
ajusted if necesary, then the speed of the train is set

   these three are repeated endlessly untill power off */

  //section 1: read user input and output display
    //print a welcome message
    if(inTransit)
    {
      lcd.setCursor(0,0);
      //delay(5000);
      lcd.print( "In Transit     " );
      lcd.setCursor(0,1);
      lcd.print("                ");
      //Serial.println("in transit");
    }
    else
    {
      /*if the train is not in transit then we execute three consecutive
steps, in step one the program reads A0 to see if any butons
```

```
       are pressed, and sets the value of buttonOut to match. in step two
this value is used to query the menu structure array
       and that is then displayed on the screen, finaly in step 3 we check
to see if the enter button has been pressed, if it has we
       execute the instructions apropriate to the currently selected menu
option*/
       respondButons();
       respondHashes();
       respondTildie();
       outputMenu();
       //Serial.println("not in transit");
    }

  //section 2: check position and instructions
    /* section 2 is devided into two parts, part one checks if the given
condition is met, part two carries out an instruction and moves
     to the next condition if it is, when a condition is met, the variable
'met' is set to true, section two is only run if met is true
     it sets met to false once it has been run*/
    if(inTransit)
    {
      respondConditions();
    }
  //section 3: output to track
    if(inTransit)
    {
      outputToTrack();
    }

//  if(menuPosX == 0 && menuPosY == 0 && !inTransit)
//  {
//    lcd.setCursor(0,0);
//    lcd.print("Welcome to      ");
//  }

  setBacklight();

  Serial.print(menuPosX); //the position in the menu, used for testing
  Serial.print(",");
  Serial.println(menuPosY);
  delay(1); //protective delay to prevent over running the serial buffer
and used to time iterations
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function takes in the number for a sensor and reads the apropriate
sensor set then outputs a boolean
to indicate weather or not the sensor is currently triggered*/


boolean checkSensor(int no) //no is the number of the sensor that we are
reading
{
  if (no < 6) //if no is less than 6 then it belongs to the first set of
sensors
  {
    int highSensor = readSensors(pinLowSensors); //we find out what the
high sensor in this set is

    if (highSensor == sensorNone) //if there isnt one we return false
      return false;

    else if (highSensor == no) //as this is the first set the number in the
set and the overall number are the same
      return true; //if they match we return true

    else
      return false;  //in all other cases we return false

  }

  else if (no < 11) //if no does not belong to the first set and is less
than 11 then it belongs to the second set
  {
    int highSensor = readSensors(pinHighSensors); //we find out what the
high sensor in this set is

    if (highSensor == sensorNone) //if there isnt one we return false
      return false;

    else if (highSensor == (no - 5)) //in the second set the sensor number
is equal to no - 5
      return true; //if it maches then we return true

    else
      return false; //in all other cases we return false
  }
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function initialises the complete instruction set for the board,
these instructions
are followed by the train when it travels to a given destination.

the first number is the instruction set, this is generaly a set of
instructions to get to a
destination but can be something like "clean the track" or "return to
siding"

the second number is the position in the instruction set

the third number is the part of the instruction 0 is a condition, 1 is the
value of the condition
2 is the instruction to execute if the condition is met and 3 is the value
of that instruction


in part 0 W means wait with part 1 being the time, B means when sensor is
triggered with part 1 being
the sensor number.

in part 2 C and D stand for converge and diverge with part 3 being the
point number to set to
converge or diverge, S sets the speed with part 3 being the speed setting
and X stops the train and
ends the instruction set, all sets end in X, no value is required


the speed settings are: 0 stops the train, 1 travels at roughly half speed
forwards, 2 is full
speed forwards, 3 is roughly half speed backwards and 4 is full speed
backwards*/



void initialiseInstructions()
{
  //Hawkhaven
  inst[0][0][0] = 'W';  inst[0][0][1] = '1';  inst[0][0][2] = 'S';
inst[0][0][3] = '2';
  inst[0][1][0] = 'B';  inst[0][1][1] = '2';  inst[0][1][2] = 'S';
inst[0][1][3] = '4';
```

```
  inst[0][2][0] = 'B';  inst[0][2][1] = '7';  inst[0][2][2] = 'S';
inst[0][2][3] = '0';
  inst[0][3][0] = 'W';  inst[0][3][1] = '2';  inst[0][3][2] = 'X';

  //Remilo
  inst[1][0][0] = 'W';  inst[1][0][1] = '1';  inst[1][0][2] = 'S';
inst[1][0][3] = '2';
  inst[1][1][0] = 'B';  inst[1][1][1] = '6';  inst[1][1][2] = 'S';
inst[1][1][3] = '0';
  inst[1][2][0] = 'W';  inst[1][2][1] = '1';  inst[1][2][2] = 'X';

  //Allantown
  inst[2][0][0] = 'W';  inst[2][0][1] = '1';  inst[2][0][2] = 'S';
inst[2][0][3] = '2';
  inst[2][1][0] = 'B';  inst[2][1][1] = '2';  inst[2][1][2] = 'S';
inst[2][1][3] = '4';
  inst[2][2][0] = 'B';  inst[2][2][1] = '7';  inst[2][2][2] = 'D';
inst[2][2][3] = '2';
  inst[2][3][0] = 'B';  inst[2][3][1] = '8';  inst[2][3][2] = 'D';
inst[2][3][3] = '2';
  inst[2][4][0] = 'W';  inst[2][4][1] = '1';  inst[2][4][2] = 'S';
inst[2][4][3] = '3';
  inst[2][5][0] = 'W';  inst[2][5][1] = '1';  inst[2][5][2] = 'S';
inst[2][5][3] = '0';
  inst[2][6][0] = 'W';  inst[2][6][1] = '1';  inst[2][6][2] = 'X';

  //Gregville
  inst[3][0][0] = 'W';  inst[3][0][1] = '1';  inst[3][0][2] = 'S';
inst[3][0][3] = '2';
  inst[3][1][0] = 'B';  inst[3][1][1] = '2';  inst[3][1][2] = 'S';
inst[3][1][3] = '4';
  inst[3][2][0] = 'B';  inst[3][2][1] = '7';  inst[3][2][2] = 'C';
inst[3][2][3] = '2';
  inst[3][3][0] = 'B';  inst[3][3][1] = '8';  inst[3][3][2] = 'C';
inst[3][3][3] = '1';
  inst[3][4][0] = 'B';  inst[3][4][1] = '9';  inst[3][4][2] = 'S';
inst[3][4][3] = '3';
  inst[3][5][0] = 'W';  inst[3][5][1] = '3';  inst[3][5][2] = 'S';
inst[3][5][3] = '0';
  inst[3][6][0] = 'W';  inst[3][6][1] = '1';  inst[3][6][2] = 'X';

  //Leovetticutte
  inst[4][0][0] = 'W';  inst[4][0][1] = '1';  inst[4][0][2] = 'S';
inst[4][0][3] = '2';
  inst[4][1][0] = 'B';  inst[4][1][1] = '2';  inst[4][1][2] = 'S';
inst[4][1][3] = '4';
  inst[4][2][0] = 'B';  inst[4][2][1] = '7';  inst[4][2][2] = 'C';
inst[4][2][3] = '2';
  inst[4][3][0] = 'B';  inst[4][3][1] = '8';  inst[4][3][2] = 'D';
inst[4][3][3] = '1';
  inst[4][4][0] = 'B';  inst[4][4][1] = '9';  inst[4][4][2] = 'S';
inst[4][4][3] = '3';
  inst[4][5][0] = 'W';  inst[4][5][1] = '3';  inst[4][5][2] = 'S';
inst[4][5][3] = '0';
  inst[4][6][0] = 'W';  inst[4][6][1] = '1';  inst[4][6][2] = 'X';

  //Regantra
  inst[5][0][0] = 'W';  inst[5][0][1] = '1';  inst[5][0][2] = 'S';
inst[5][0][3] = '2';
  inst[5][1][0] = 'B';  inst[5][1][1] = '2';  inst[5][1][2] = 'S';
inst[5][1][3] = '2';
```

```
  inst[5][2][0] = 'W';  inst[5][2][1] = '3';  inst[5][2][2] = 'S';
inst[5][2][3] = '4';
  inst[5][3][0] = 'B';  inst[5][3][1] = '3';  inst[5][3][2] = 'S';
inst[5][3][3] = '3';
  inst[5][4][0] = 'W';  inst[5][4][1] = '2';  inst[5][4][2] = 'S';
inst[5][4][3] = '0';
  inst[5][5][0] = 'W';  inst[5][5][1] = '1';  inst[5][5][2] = 'X';

  //Vancoville
  inst[6][0][0] = 'W';  inst[6][0][1] = '1';  inst[6][0][2] = 'S';
inst[6][0][3] = '2';
  inst[6][1][0] = 'B';  inst[6][1][1] = '1';  inst[6][1][2] = 'S';
inst[6][1][3] = '3';
  inst[6][2][0] = 'W';  inst[6][2][1] = '1';  inst[6][2][2] = 'S';
inst[6][2][3] = '0';
  inst[6][3][0] = 'W';  inst[6][3][1] = '1';  inst[6][3][2] = 'X';

  inst[7][0][0] = 'W';  inst[7][0][1] = '1';  inst[7][0][2] = 'X';
inst[7][0][3] = '1';
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function initialises the menu display array, the first number is the
X position, the second number is the
Y position and the third number is the line number, 0 is line 1 and 1 is
line 2

~ are used to fill the bottom of columbs as well as blocking the cursor
from moveing between options which are on
different sub menus

# are used to bridge the gap between menu options that should apear to be
next to eachoter on the Y axis but actualy
arent*/

void initialiseMenu()
{
  menu[0][0][0] = "Welcome to      ";        menu[1][0][0] = "Destinations
";  menu[2][0][0] = "Hawkhaven       ";
  menu[0][0][1] = "Beltrak 1.0     ";        menu[1][0][1] = "
";  menu[2][0][1] = "                ";

  menu[0][1][0] = "~               ";        menu[1][1][0] = "#
";  menu[2][1][0] = "Remilo          ";
  menu[0][1][1] = "                ";        menu[1][1][1] = "
";  menu[2][1][1] = "                ";

  menu[0][2][0] = "~               ";        menu[1][2][0] = "#
";  menu[2][2][0] = "Allantown       ";
  menu[0][2][1] = "                ";        menu[1][2][1] = "
";  menu[2][2][1] = "                ";

  menu[0][3][0] = "~               ";        menu[1][3][0] = "#
";  menu[2][3][0] = "Gregville       ";
  menu[0][3][1] = "                ";        menu[1][3][1] = "
";  menu[2][3][1] = "                ";

  menu[0][4][0] = "~               ";        menu[1][4][0] = "#
";  menu[2][4][0] = "Leovetticutte   ";
  menu[0][4][1] = "                ";        menu[1][4][1] = "
";  menu[2][4][1] = "                ";

  menu[0][5][0] = "~               ";        menu[1][5][0] = "#
";  menu[2][5][0] = "Regantra        ";
```

```
  menu[0][5][1] = "                ";        menu[1][5][1] = "
";  menu[2][5][1] = "                ";

  menu[0][6][0] = "~               ";        menu[1][6][0] = "#
";  menu[2][6][0] = "Vancoville      ";
  menu[0][6][1] = "                ";        menu[1][6][1] = "
";  menu[2][6][1] = "                ";

  menu[0][7][0] = "~               ";        menu[1][7][0] = "#
";  menu[2][7][0] = "~               ";
  menu[0][7][1] = "                ";        menu[1][7][1] = "
";  menu[2][7][1] = "                ";

  menu[0][8][0] = "~               ";        menu[1][8][0] = "Settings
";  menu[2][8][0] = "Backlight       ";
  menu[0][8][1] = "                ";        menu[1][8][1] = "
";  menu[2][8][1] = "                ";

  //this line originaly contained the top speed setting but it was removed
when testing showed that changeing the
  //top speed caused the train to miss stations.
  menu[0][9][0] = "~               ";        menu[1][9][0] = "~
";  menu[2][9][0] = "~               ";
  menu[0][9][1] = "                ";        menu[1][9][1] = "
";  menu[2][9][1] = "                ";
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

void respondButons()
{
 /*in this function the program checks the position of the menu to make
sure that when the requested move is made it wont move
 off the edge of the array and that it wont move onto a ~ which is not
alowed, it also checks that a buton is not being held
 down by setting button captured to true when a buton is down and setting
it to false when a buton is up, it will only respond
 to a press if button captured is false which means that holding down a
button does not register multiple presses,

 once that has been confirmed it then moves the position in the menu and
records the move that was just made for use in the
 hashRespond() function later in the program*/
  switch(readSensors(pinButtons))
    {
    case rightOut:
      {
        if(menuPosX < 2 && butonCaptured == false) //check conditions
        {
          menuPosX++; //move menu position
          butonCaptured = true; //say that a button is being pressed
          lastMenuMove = rightOut; //record the last move
        }
        break;
      }

    case upOut:
      {
        if(menuPosY > 0 && butonCaptured == false)
        {
          menuPosY--;
          butonCaptured = true;
          lastMenuMove = upOut;
        }
        break;
      }

    case downOut:
      {
        if(menuPosY < 9 && butonCaptured == false)
        {
          menuPosY++;
```

```
          butonCaptured = true;
          lastMenuMove = downOut;
        }
        break;
      }

    case leftOut:
      {
        if(menuPosX > 0 && butonCaptured == false)
        {
          menuPosX--;
          butonCaptured = true;
          lastMenuMove = leftOut;
        }
        break;
      }

    case selectOut:
      {
        if(!butonCaptured)
        {
          respondEnter(menuPosX, menuPosY); //calls the function for
responding to enter being pressed
          butonCaptured = true;
        }
        break;
      }

    case noneOut:
      {
        //lcd.print("Beltrak 1.0     ");
        butonCaptured = false;
        break;
      }
    }
}
```

CHECKCONDITIONS.INO

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

boolean checkConditions()
{
  switch(inst[instSet][instPos][0]) //this reads position 0 of an
instruction set, to see what the condition is
    {
    case 'B': //met when sensor x goes HIGH
      {
        //Serial.println((inst[instSet][instPos][1]) - 48); //this tells us
that the board has read state B

        if(checkSensor((inst[instSet][instPos][1]) - 48) == true) //this
reads the virtual sensor dictated by position 1
        {
          Serial.println("sensor high"); //this is run if the sensor is
high or the VS is true
          return true; //the condition is met so this goes true
        }

        else
        {
          return false;
        }


      }
    case 'W': //wait for x miliseconds then meet
      {
        //Serial.println("state is W!"); //this tells us that the board has
read state W
        if (timer > ((inst[instSet][instPos][1]) - 48) * 100) //checks if
the timer has exceded the stated time in multiples of 100
        {
          return true; //the condition has been met
        }
        else
        {
          timer++; //incriment timer
          return false;
```

```
            }

        }
    }
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

void initialise()
{

  //initialise the lcd screen with rows and columns
  lcd.begin( 16, 2 );

  //print the loading screen
  lcd.setCursor( 0, 0 );
  lcd.print( "Loading..." );

  //initialise global variables
  PPD = 0; //full stop
  reverser = false; //the train will travel forward

  //initialise pins
  pinMode(pinPD, OUTPUT); //instruct the board to output to the PD pin
  pinMode(pinDIR, OUTPUT); //instruct the board to output to the direction
pin
  pinMode(pinButtons, INPUT ); //instruct the board to read from the button
pin
  digitalWrite(pinButtons, LOW); //make sure that the board doesent pass
voltage to the button pin

  pinMode(point1, OUTPUT); //the point pins
  pinMode(point2, OUTPUT);

  pinMode(backlightPin, OUTPUT);

  backlight = 2;

//  pinMode(pointPower, OUTPUT);
  pinMode(pointDir, OUTPUT);


  //initialise array positions
  instSet = 7;
  instPos = 0;

  //initialise the timer
  timer = 0;

  //initialise points
  for(int i=0; i<10; i++)
```

```
  {
//    /*these two values are set to differ so that the board is forced to
move all the points on the first run, this makes sure
//     that the board knows their positions and reveals any malfunctioning
points*/
//    pointState[i] = true;
//    pointSwitch[i] = false;
  }

  //initialise transition boolean
  inTransit = false;

  //initialises the anti-multipress boolean
  /*this is set to true so that if a button is stuck down when the board
turns on it is not registered*/
  butonCaptured = true;

  //menu positions
  menuPosX = 0;
  menuPosY = 0;
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

void outputMenu()
{
  /*this function sets the cursor to the correct position and outputs the
required text to the lcd*/
  lcd.setCursor(0,0); //position the cursor
  lcd.print(menu[menuPosX][menuPosY][0]); //output text
  Serial.println(menu[menuPosX][menuPosY][0]); //this prints what the
screen is showing (used for testing)
  lcd.setCursor(0,1);
  lcd.print(menu[menuPosX][menuPosY][1]);
  Serial.println(menu[menuPosX][menuPosY][1]);
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

void outputToTrack()
{

  //set the points
  //checkPoints();

  //apply the PPD to the train
  analogWrite(pinPD, int((PPD / 100) * 255));

  //apply the reverser to the train
  if (reverser)
  {
    digitalWrite(pinDIR, HIGH);//if this is high the train travels
backwards
  }
  else
  {
    digitalWrite(pinDIR, LOW);
  }


}
```

```
  /*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function returns an intager value coresponding the the current
button (or lack therof) being
pressed, it only works when either 1 or 0 butons are being pressed, the
intager values output are
#defined at the begining of the program as rightOut upOut etc.

the sensor board outputs a diferent voltage depending on which buton is
pressed, these voltages are
#defined at the begining of the program and are called upADC, downADC etc.

the sensitivity (or hysteresis) is how far out the voltage can be on either
side of the defined value,
this allows for the tolerence of the resistors used producing the voltages

the voltage used is stored in buttonVoltage*/

int readSensors(int sensorNumber)
{
  unsigned int buttonVoltage = analogRead(sensorNumber); //this reads the
voltage and stores it
  //Serial.println(buttonVoltage);
  if(buttonVoltage < (rightADC + ADCsensitivity)) //the rightADC is 0 so we
check that it is beneath 0 +the sensitivity
  {
    return rightOut; //this returns the button pressed
  }

  else if(buttonVoltage >= (upADC - ADCsensitivity) && buttonVoltage <=
(upADC + ADCsensitivity)) //this checks the voltage is in range
  {
    return upOut;
  }

  else if(buttonVoltage >= (downADC - ADCsensitivity) && buttonVoltage <=
(downADC + ADCsensitivity))
  {
    return downOut;
  }

  else if(buttonVoltage >= (leftADC - ADCsensitivity) && buttonVoltage <=
(leftADC + ADCsensitivity))
  {
```

```
    return leftOut;
  }

  else if(buttonVoltage >= (selectADC - ADCsensitivity) && buttonVoltage <=
(selectADC + ADCsensitivity))
  {
    return selectOut;
  }

  else
  {
    return noneOut; //if the voltage matches none of these then we return
that no button is being pressed
  }
  }
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

void respondConditions()
{
  if(checkConditions())
    {
      switch(inst[instSet][instPos][2]) //reads position 2 in the
instruction set
      {
      case 'S': //set the PPD to a set value
        {
          switch(inst[instSet][instPos][3]) //reads position 3 which
contains the PPD setting
          {
          case '0':
            {
              PPD = 0; //sets the ppd to 0%
              break;
            }

          case '1':
            {
              PPD = 75; //sets the PPD to 50%
              reverser = false; // sets the direction forward
              break;
            }

          case '2':
            {
              PPD = 100; //sets the PPD to 100%
              reverser = false; // sets the direction forward
              break;
            }

          case '3':
            {
              PPD = 75; //sets the PPD to 50%
              reverser = true; // sets the direction backwards
              break;
            }

          case '4':
            {
```

```
                PPD = 100; //sets the PPD to 100%
                reverser = true; // sets the direction backwards
                break;
              }
            }
            instPos++;
            break;
          }

        case 'C':
          {
            setPoint(false, (inst[instSet][instPos][3] - 48)); //instructs
the board to converge the given points
            Serial.println("converge");
            //delay(1000);
            instPos++; //moves us to the next instruction
            break;

          }

        case 'D':
          {
            setPoint(true, (inst[instSet][instPos][3] - 48)); //instructs the
board to diverge the given points
            Serial.println("diverge");
            //delay(1000);
            instPos++; //moves us to the next instruction
            break;
          }

        case 'X':
          {
            PPD = 0; //stop the train
            inTransit = false; //stop following instructions
            break;
          }
        }
        timer = 0; //resets the timer to be used by a diferent call of W
         //this moves to the next instruction set
        //          for(int i = 0; i < 10; i++)
        //          {
        //            Serial.print(i);
        //            Serial.print(":");
        //            Serial.println(pointSwitch[i]);
        //          }
        //
        //           delay(10000);
            }

}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

 /*if the program moves onto a hash it moves forward untill it reaches a
menu option except when it moves left onto the hash
 in which case it goes up untill it reaches an option*/

void respondHashes()
{
  while (menu[menuPosX][menuPosY][0] == "#                ")
  {
    if (lastMenuMove == downOut)
      menuPosY++;

    else if (lastMenuMove == upOut)
      menuPosY--;

    else if (lastMenuMove == leftOut)
      menuPosY--;

    else
      {
        menuPosX = 0; //this is here in case none of these conditions are
met, it resets the menu position to break the loop
        menuPosY = 0;
      }
  }
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*if the program reaches a tildie this is run, when the program moves onto
a ~ it moves back one step except when it moved left
onto a tildie in which case it goes up untill it reaches a menu option*/

void respondTildie()
{
  while (menu[menuPosX][menuPosY][0] == "~                ")
  {
    if (lastMenuMove == downOut)
      menuPosY--;

    else if (lastMenuMove == upOut)
      menuPosY++;

    else if (lastMenuMove == leftOut)
      menuPosY--;

    else if (lastMenuMove == rightOut)
      menuPosX--;

    else
      {
        menuPosX = 0; //this is here in case none of these conditions are
met, it resets the menu position to break the loop
        menuPosY = 0;
      }
  }
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function takes in a point number and a point state, looks up the pin
for that point's relay then
sets the point to the given state.*/

 //C false converge A | D true Diverge B

 void setPoint(boolean CorD, int point)
 {
   Serial.println("pointSet called"); //used for debuging
   int pointPin;

   //These are the two points we use
   if (point == 1)
     pointPin = point1;
   if (point == 2)
     pointPin = point2;

   if (CorD)
   {
     digitalWrite(pointDir, HIGH); //if the points diverge we switch on the
direction relay
     Serial.println("director high"); //used for debuging
     //delay(500);
   }

   //analogWrite(pointPower, 255);

   digitalWrite(pointPin, HIGH); //we ground the points causing them to
move
   Serial.println("ground point"); //used for debuging

   delay(50); //we give them time to move
   //analogWrite(pointPower, 0);
   digitalWrite(pointPin, LOW); //we unground them so they wont heat up
   Serial.println("unground point"); //used for debuging
   if (CorD)
   {
     delay(50);
     digitalWrite(pointDir, LOW); //if the points diverge we switch off the
direction relay
     Serial.println("director low"); //used for debuging
   }}
```

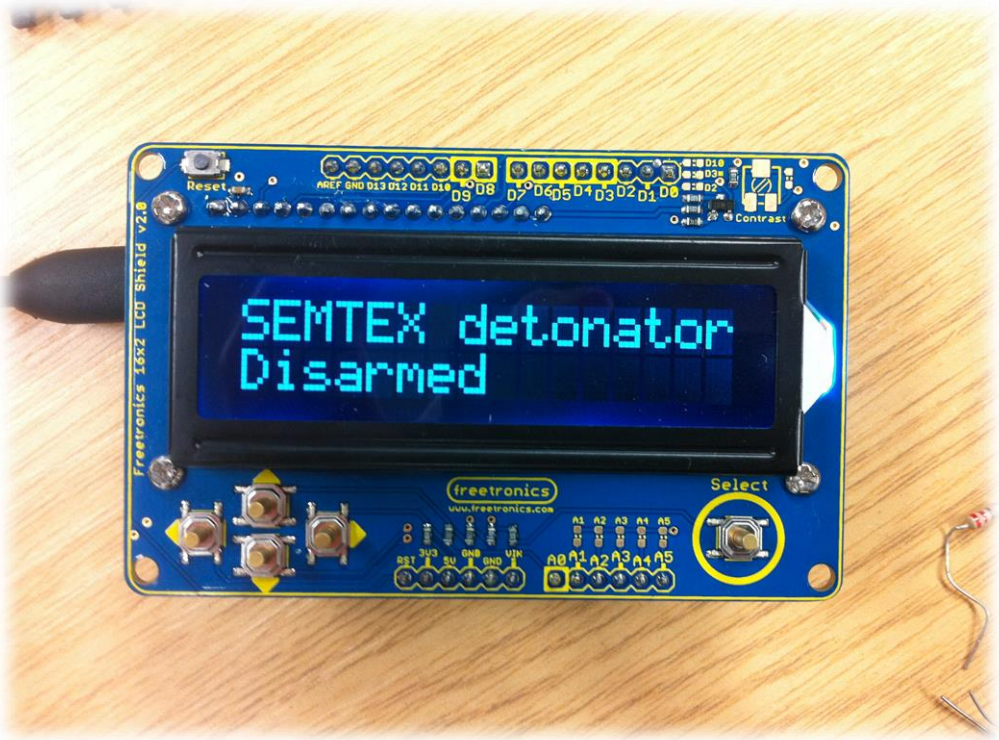
```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

/*this function was originaly used to check the position of the points but
it turned out
to cause multiple errors and so was removed and replaced with code that
simply switched
the points without remembering their position*/


void checkPoints()
{
//  for (int i = 1; i < 11; i++)  //goes through all the points
//  {
//    if (pointState[i] != pointSwitch[i]) //if the state doesent match
what it should be
//    {
//      setPoint(pointSwitch[i], i); //move the points
//      pointState[i] = pointSwitch[i]; //change the state
//    }
//  }
}
```

```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

 /*this function is called when enter is pressed it either sets the
instruction set to the apropriate station or
 it runs the function for backlight or top speed*/

void respondEnter(int menuX, int menuY) //the function takes in the current
menu position
{
  if(menuX == 2 && menuY < 7 && !inTransit) //only options in the third
colum do things so we check if the selected option is there
  //this section is for when a station is selected, the first 7 options are
stations so we check weather that is in bounds
  //if the train is already in transit then the enter button should not do
anything so we check that as well
  {
    instSet = menuY; //the y position of the cursor also corresponds to the
relivent instruction set, this is selected when enter is pressed
    instPos = 0; //we then go to the start of the instructions

    menuPosX = 0; //the cursor is reset for when transit is complete
    menuPosY = 0;

    inTransit = true; //we set the train in transit which starts the
instructions and locks the screen
  }
  else
  {
    if(menuPosX < 2 && butonCaptured == false) //check conditions
        {
          menuPosX++; //move menu position
          butonCaptured = true; //say that a button is being pressed
          lastMenuMove = rightOut; //record the last move
        }
  }

  if(menuX == 2 && menuY == 8) //this is the backlight position in the menu
  {
    if(backlight == 2)
      backlight = 1; //1 switches the backlight off
    else
      backlight = 2; //2 switches it on
  }
}
```

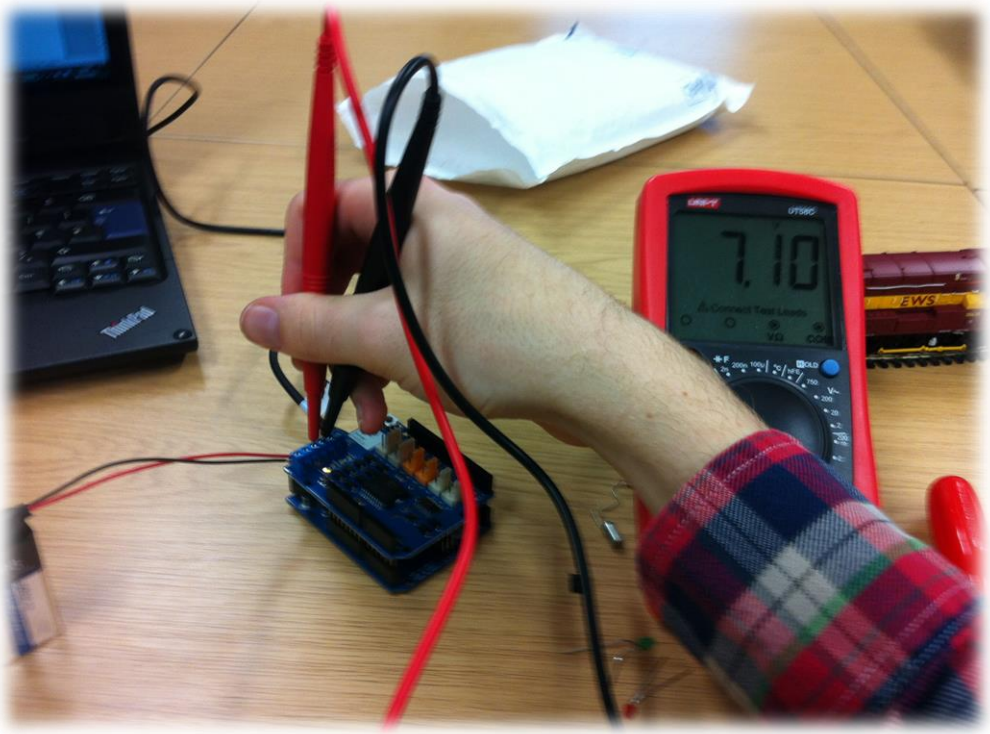
```
/*

 BELTRAK

 V1.0

 Hornby trainset automation

 By Michael Bell

 Programing started: 02/02/2013 at 14:08

 Programing completed: 06/05/2013 at 17:45

 */

 void setBacklight()
 {
   if (backlight == 1)
     digitalWrite (backlightPin, 0); //this switches the backlight off

   if (backlight == 2)
     digitalWrite (backlightPin, 255); //this switches it on
 }
```

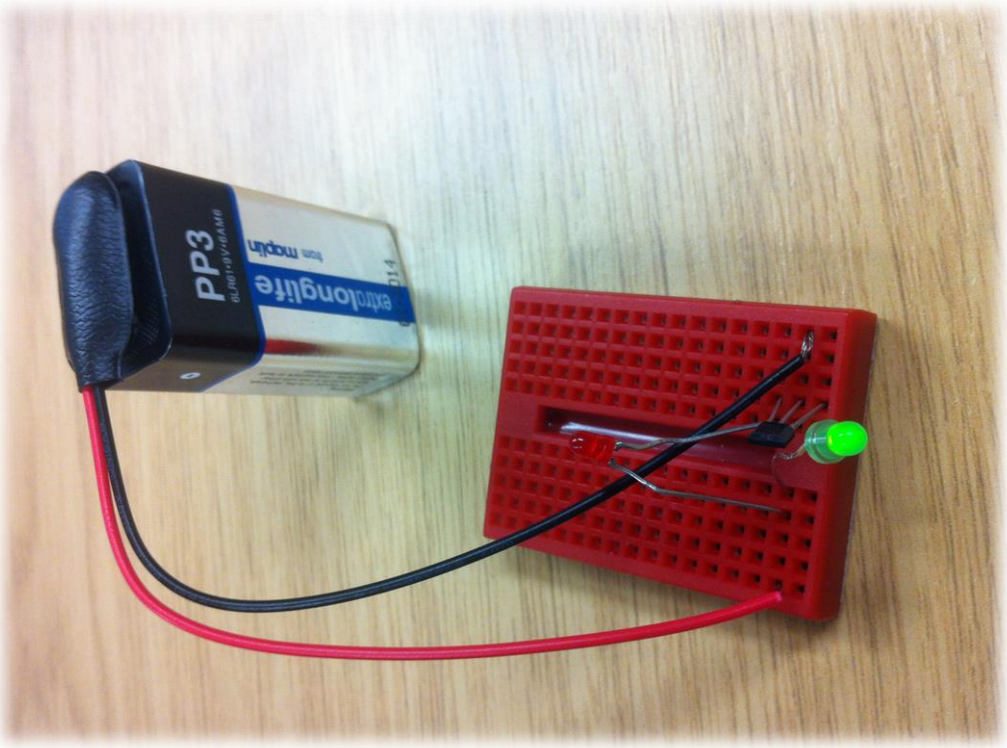
PICTURES

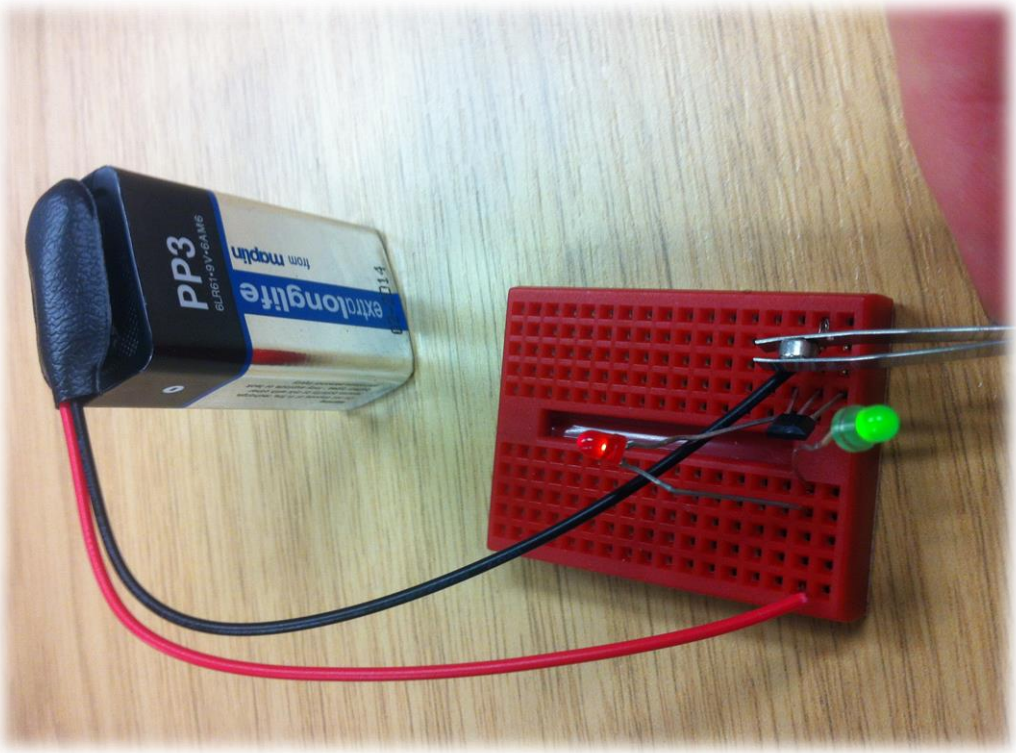
TESTING THE LCD

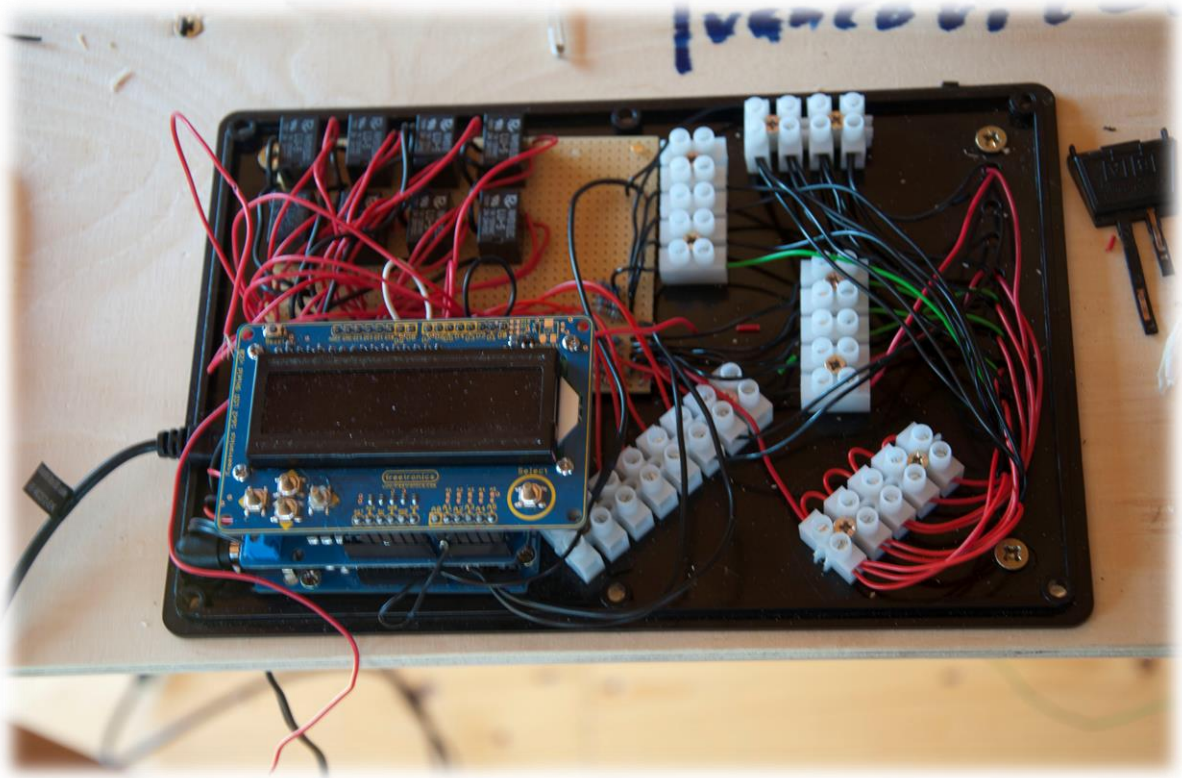


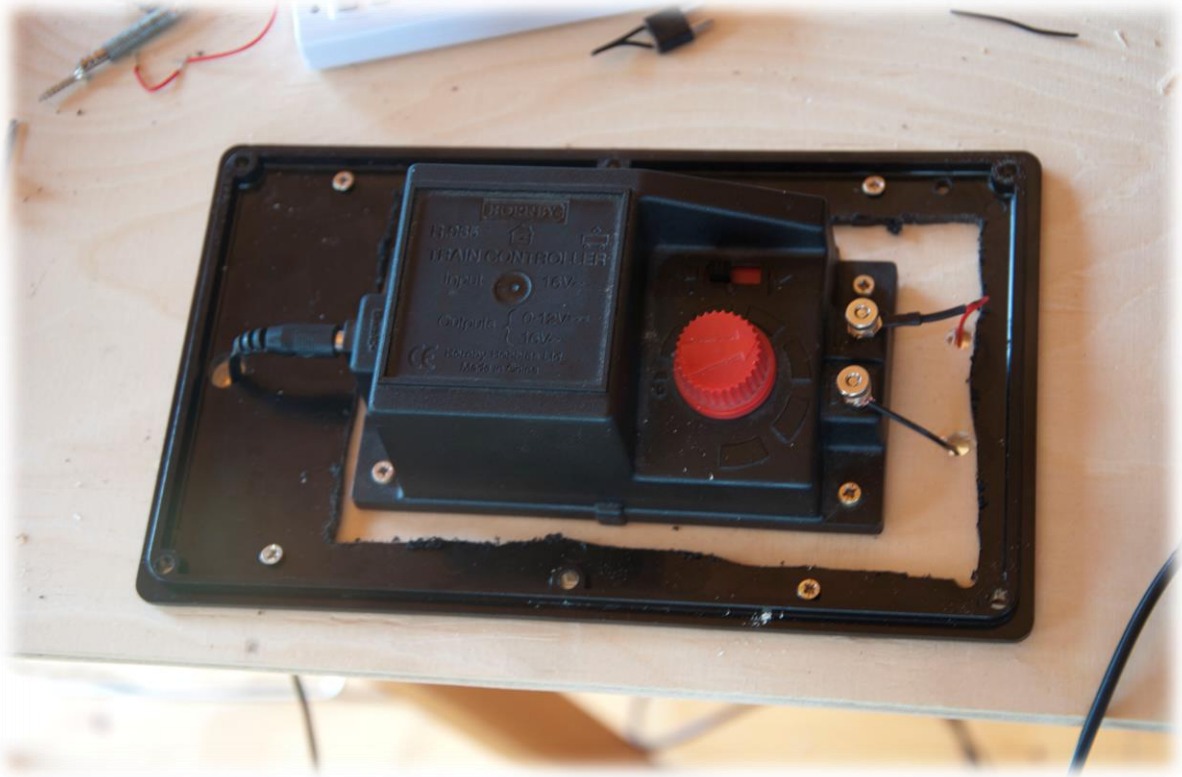
TESTING THE VOLTAGE OUTPUT



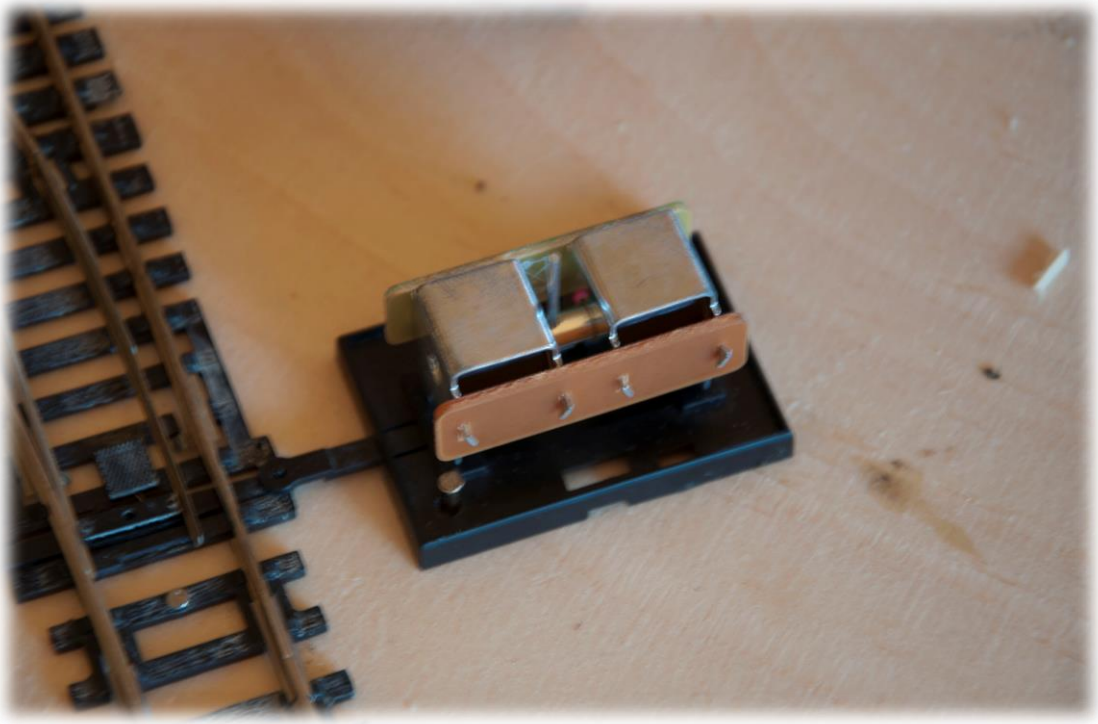

PP3
extralonglife
from maplin


PP3
extralonglife
from maplin

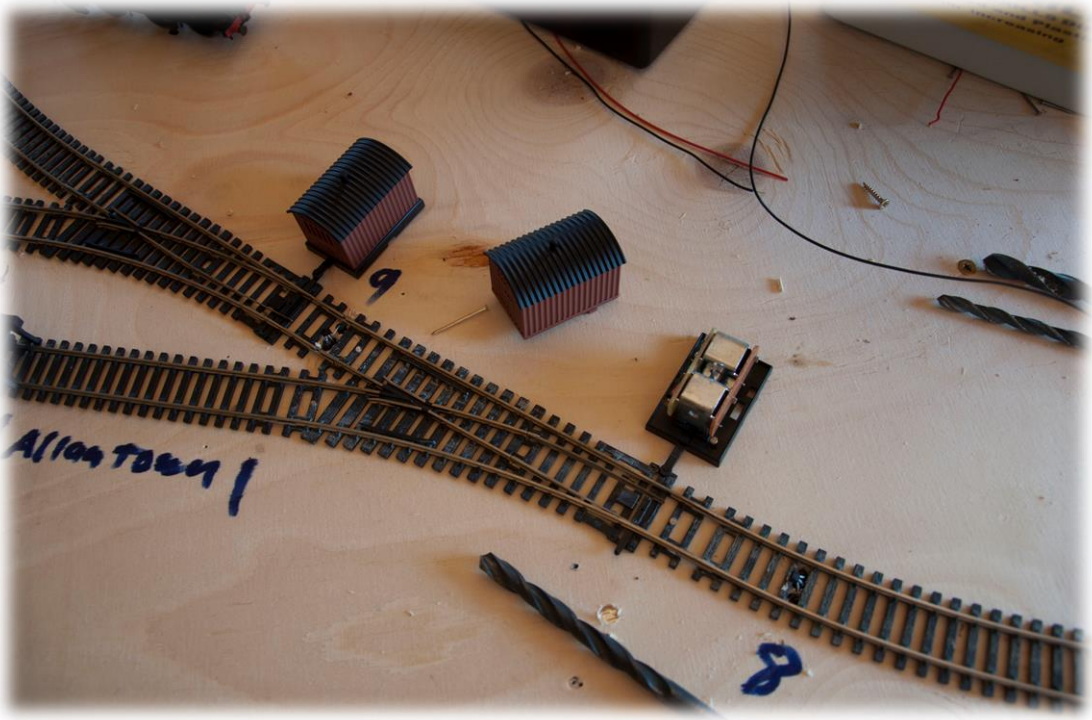


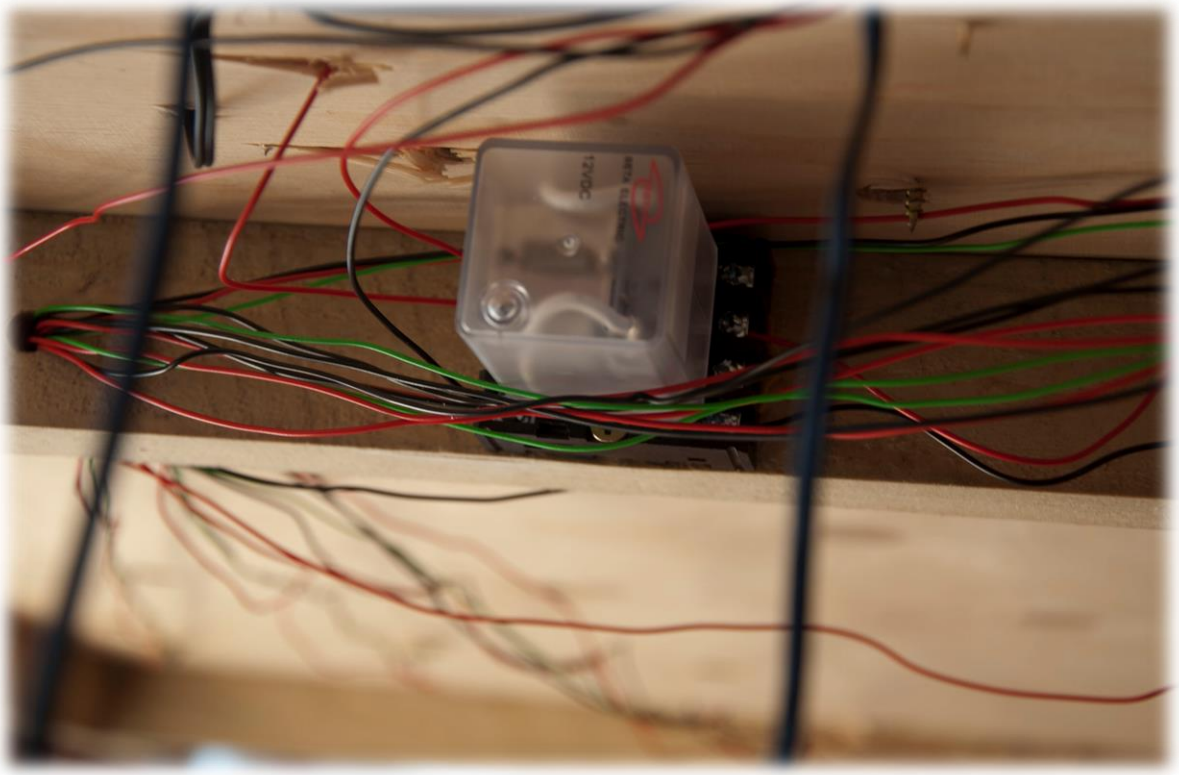

HORNBY
TRAIN CONTROLLER

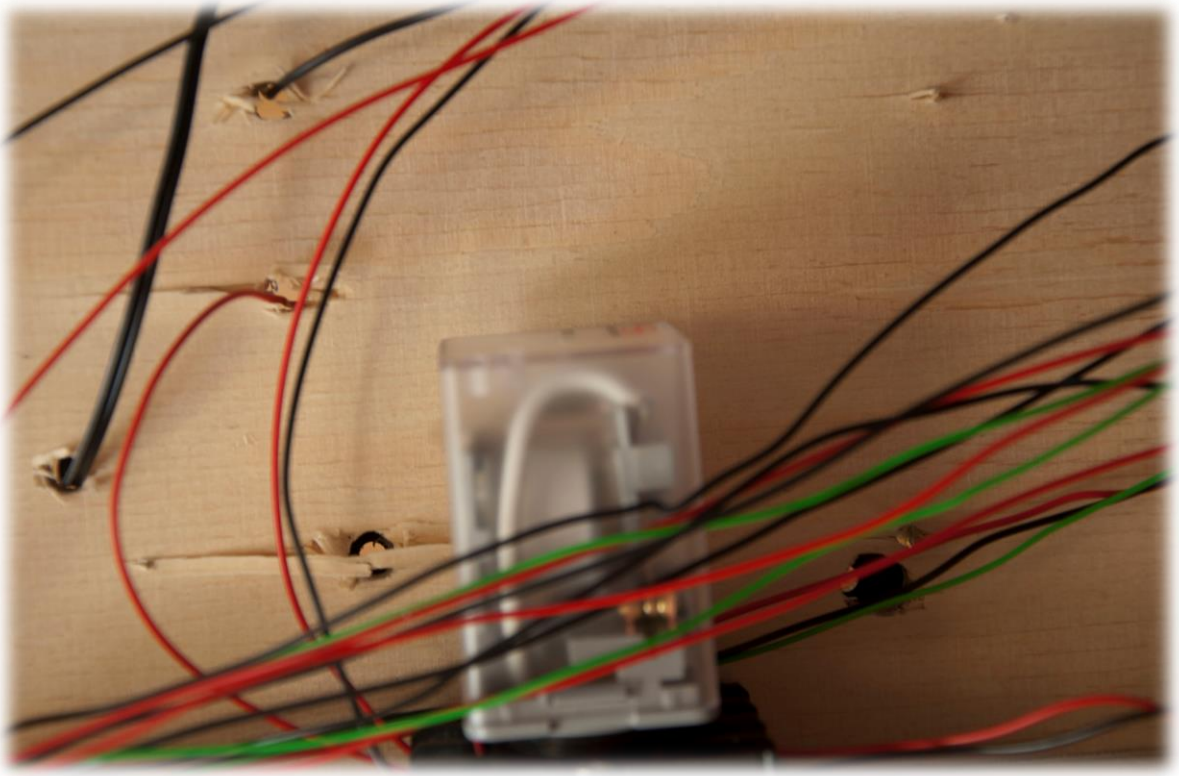
AN INSTALLED POINT MOTOR



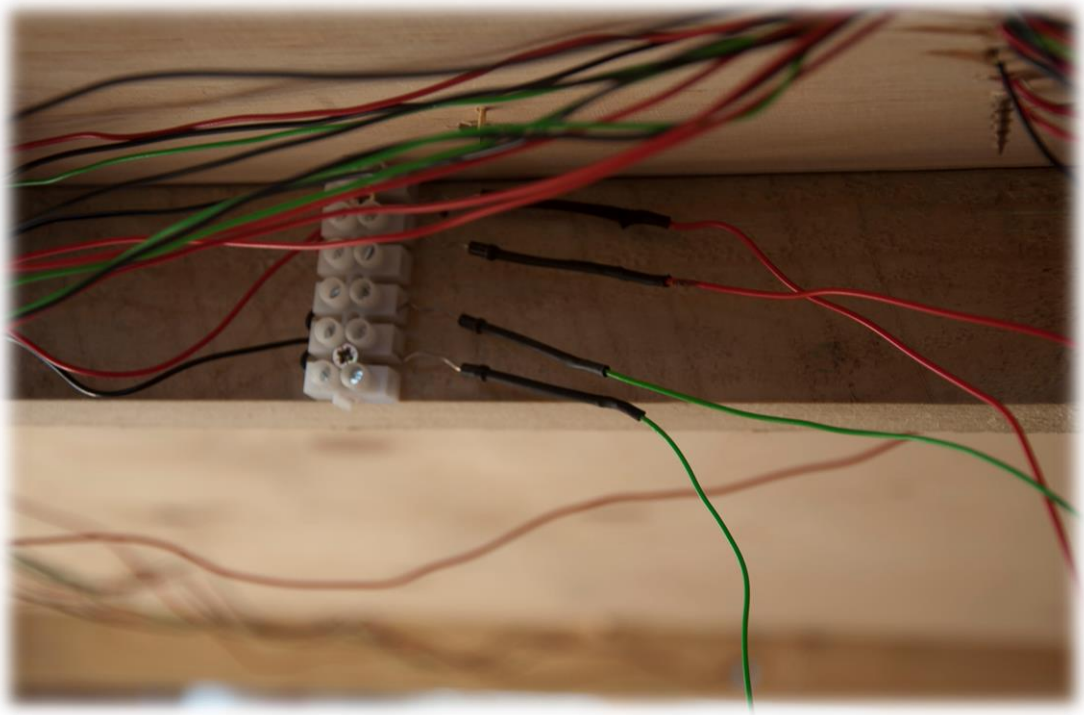
POINT MOTORS WITH AND WITHOUT COVERS





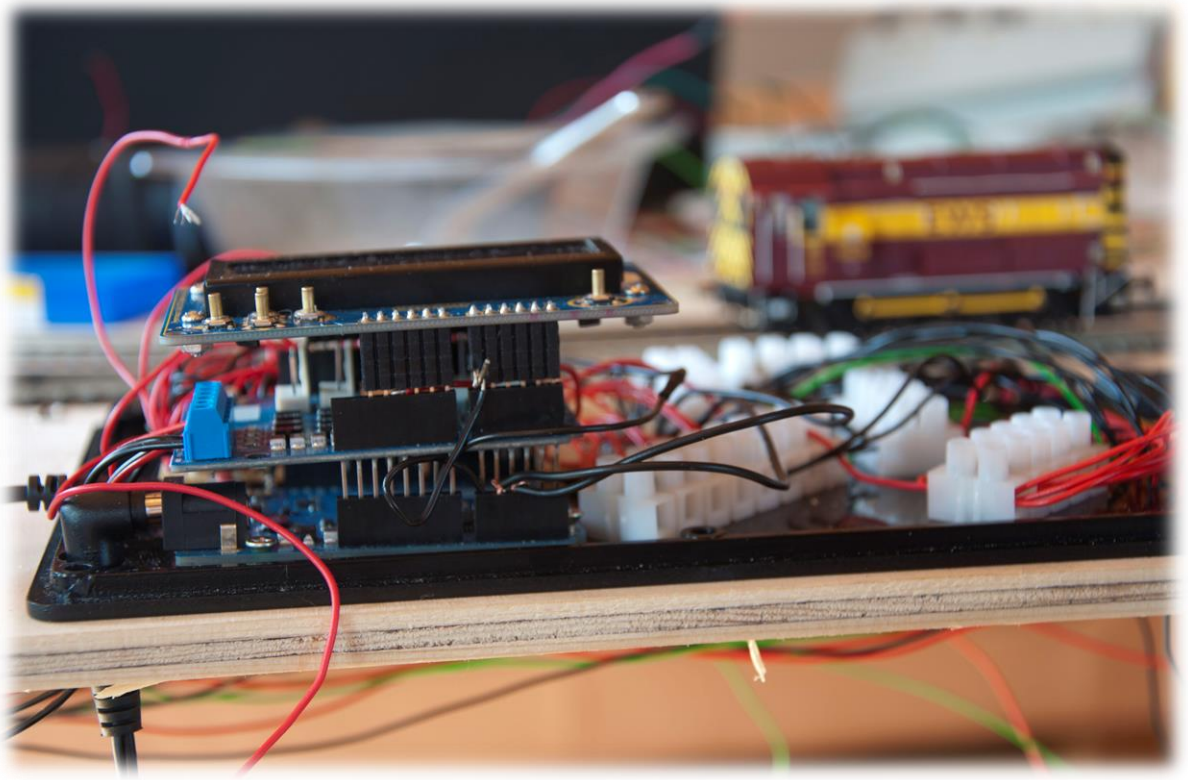


THE FABLED DIODES

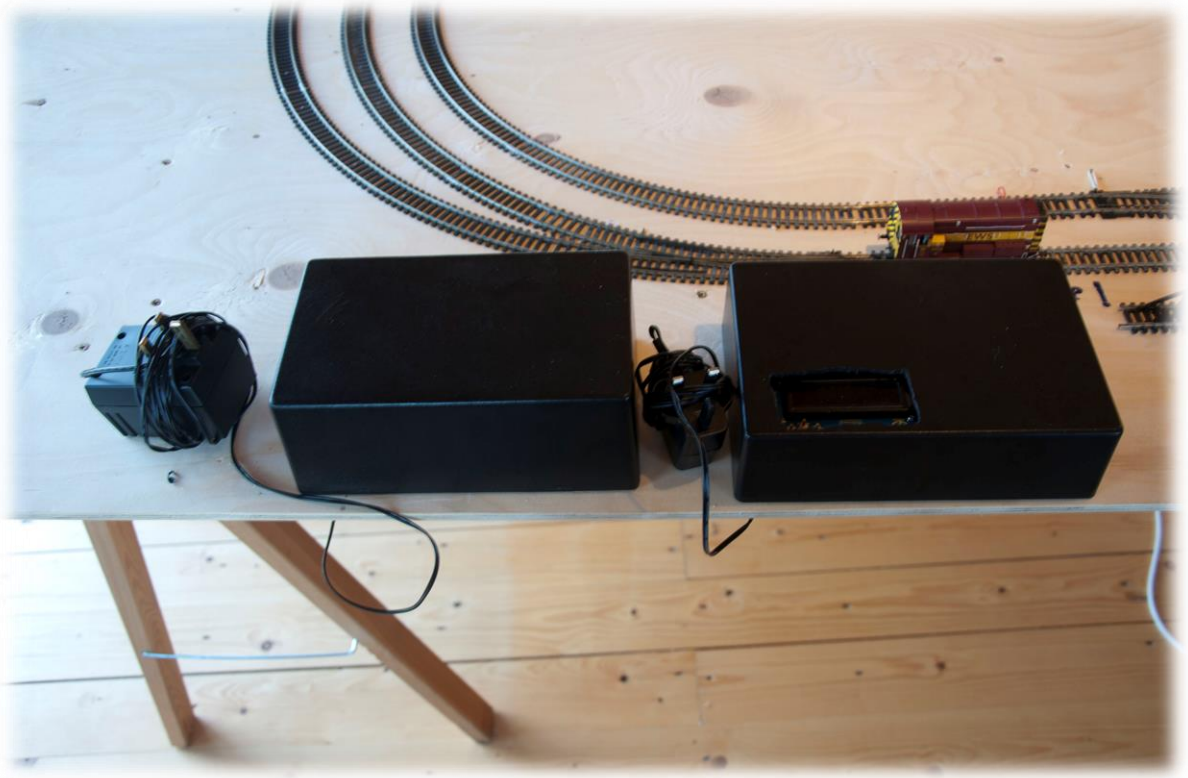


THE FABLED MESS

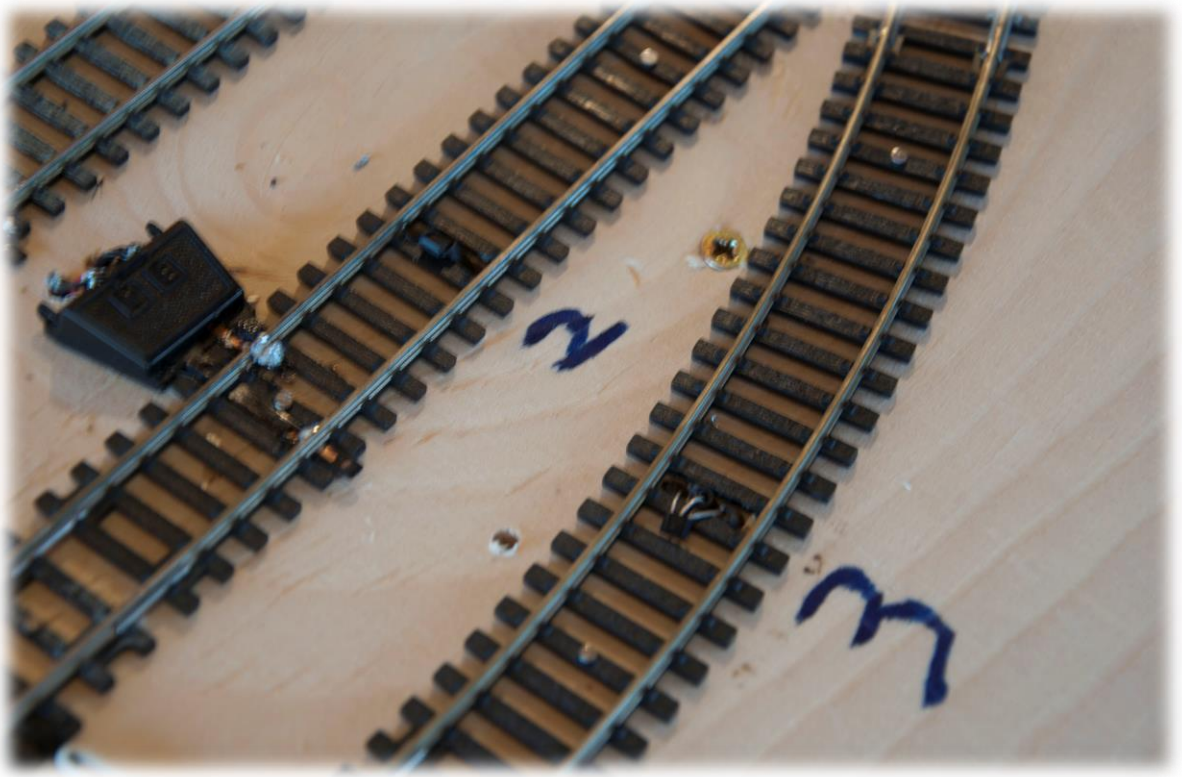
BELTRAC IN PROFILE



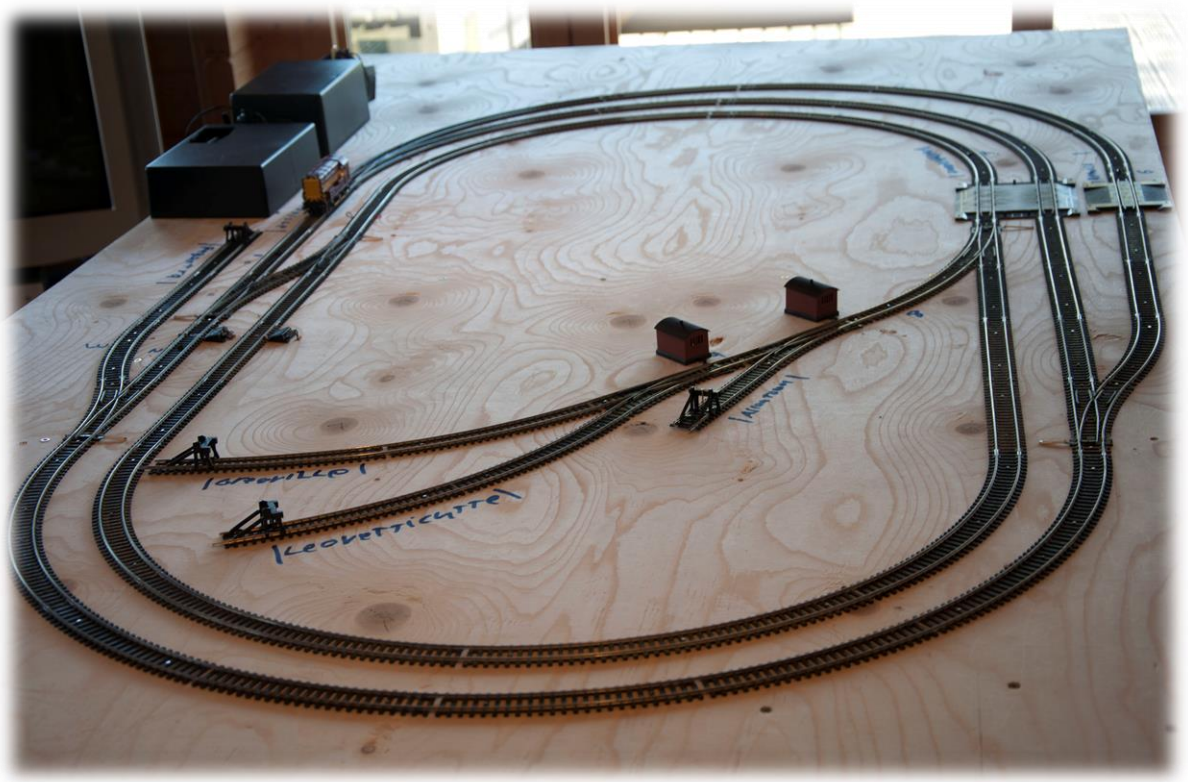
THE COVERS IN PLACE



THE SENSORS



THE MENU HOLE


Leovettichette

TESTING

ALPHA TESTING

Development Milestone	Test	Result	Action taken
Set out placeholders	Compile	Fail	Discovered that files can't be called main, changed to Beltrac (didn't notice incorrect spelling, never mind!)
		Pass	
Created speed output function	Run and read pins with voltmeter	Pass	
Commented	Compile	Pass	
Created instruction data structure	Compile with test values	Pass	
Created sensor reader and speed setter	Using an array of Booleans (VS[]) masquerading as sensors tested response to different sensors	Pass	
Added loading screen and welcome message	Run on LCD	Pass	
Added Wait function	Gave a time and instructed to apply voltage after said time, read with volt meter	Pass	
Created point switching variables	Compiled	Pass	
Created point switching instruction set	Instructed to switch certain points in the point array	Pass	
Added the end of instruction set instruction (X)	Ran test instructions	Pass	
Plugged in train for the first time	Run all speed settings	Fail	The half speed settings where too slow, set them to be ¾ speed instead
		Pass	
Created button reader	Pressed buttons and had serial out name them	Pass	
Created the Menu array	Compiled	Pass	

Added ability to move through menu	Moved through menu	Pass	
Hooked up 3 point relays	Tried to switch them	Fail	Had soldered wires in wrong places, corrected
		Pass	
Added hash responder	Navigated menu	Pass	
Added tilde responder	Navigated menu	Fail	Realised my current algorithm meant that you had to go all the way back up to the top to return to the entry page for the menu, rewrote tilde algorithm so that hitting one from the right caused the cursor to leap up until it left the tidies
		Pass	
Created function to set a point	Connected point	Pass	
Added some comments	Compiled	Pass	
Added compatibility with large point direction relay	Switched in code	Pass	
Adapted button reading code to read sensors as well (they use the same resistor network)	Hooked up some sensors	Pass	
Completed building	Tried to drive train	Fail	Train repeatedly ground to a halt mid journey, suspected transformer issue, tested voltage supply from transformer while train was running, supply dropped, tested current draw of transformer and subtracted current draw of train discovered that 0.1A was going missing, tested other components, discovered that the large point relay was being left switched

			on, adapted code
	Tried to drive train	Fail	Train could not visit all areas as some point motors would not switch when they were expected to, realised it was being caused by the point array system which thought that a point had been switched when it hadn't, replaced system with a function that always moved the point regardless of where the board thought the point was
		Pass	
	Tried to navigate lcd	Fail	LCD produced random bars when visiting some items rather than the expected words, isolated different parts of the code and discovered that when the instructions weren't initialised the lcd worked, isolated instruction sets in turn discovered that if the last few where turned off the lcd worked, read numbers of them and deduced that if only the last ones where interfering then the instruction array must be overrunning into the menu array, considered possible causes and eventually realised that the instruction array was larger than had been declared, updated declaration to a larger size
		Pass	

PERFORMED BY ME

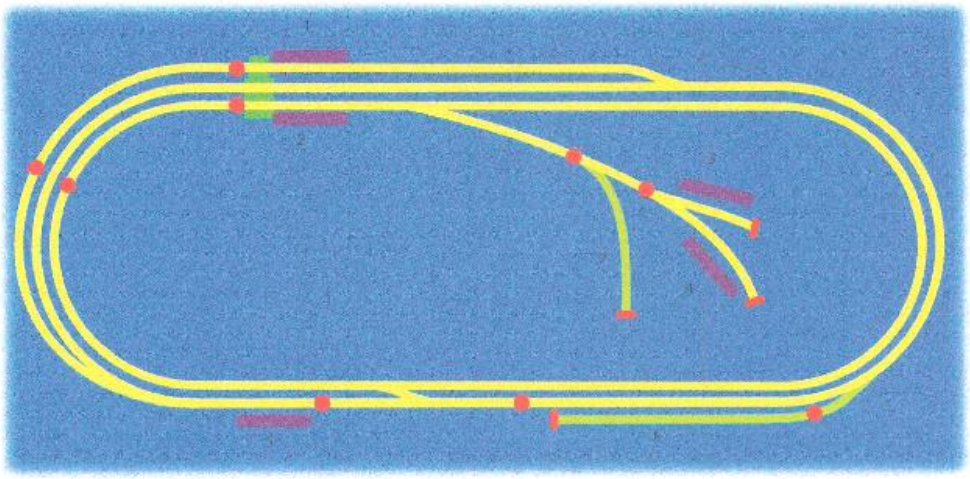
INTERFACE TESTING

NAVIGATION THROUGH OPTIONS TESTING

Object	Can be reached	Can return to menu
Destination: Hawkhaven	✓	✓
Destination: Remilo	✓	✓
Destination: Allantown	✓	✓
Destination: Gregville	✓	✓
Destination: Leovetticutte	✓	✓
Destination: Regantra	✓	✓
Destination: Vancoville	✓	✓
~~Setting: Top Speed~~ CUT	~~✓~~	~~✓~~
Setting: Backlight	✓	✓

SELECTION OF OPTIONS TESTING

Object	Expected Result	Result Met
Welcome Page	-	✓
Destination	-	✓
Destination: Hawkhaven	Select Instruction Set 1	✓
Destination: Remilo	Select Instruction Set 2	✓
Destination: Allantown	Select Instruction Set 3	✓
Destination: Gregville	Select Instruction Set 4	✓
Destination: Leovetticutte	Select Instruction Set 5	✓
Destination: Regantra	Select Instruction Set 6	✓
Destination: Vancoville	Select Instruction Set 7	✓
Settings	-	✓
~~Setting: Top Speed~~ CUT	~~Open Speed Selector~~	~~✓~~
Setting: Backlight	Open Backlight Setter	✓



		Start						
		1	2	3	4	5	6	7
Destination	1		✓	✓	✓	✓	✓	✓
	2	✓		✓	✓	✓	✓	✓
	3	✓	✓		✓	✓	✓	✓
	4	✓	✓	✓		✓	✓	✓
	5	✓	✓	✓	✓		✓	✓
	6	✓	✓	✓	✓	✓		✓
	7	✓	—	✓	✓	✓	✓	

NAVIGATION BOUNDRY TESTING

Object	Direction	Expected Result	Result Met
Welcome page	Up	-	
	Down	-	
	Left	-	
	Right	Destination	
Destinations	Up	-	
	Down	Settings	
	Left	Welcome Page	
	Right	Hawkhaven	
Settings	Up	Destinations	
	Down	-	
	Left	Welcome Page	
	Right	~~Top Speed~~ Backlight	
Hawkhaven	Up	-	
	Down	Remilo	
	Left	Destinations	
	Right	-	
Remilo	Up	Hawkhaven	
	Down	Allantown	
	Left	Destinations	
	Right	-	
Allantown	Up	Remilo	
	Down	Gregville	
	Left	Destinations	
	Right	-	
Gregville	Up	Allantown	
	Down	Leovetticutte	
	Left	Destinations	
	Right	-	
Leovetticutte	Up	Gregville	
	Down	Regantra	
	Left	Destinations	
	Right	-	
Regantra	Up	Leovetticutte	
	Down	Vancoville	
	Left	Destinations	
	Right	-	
Vancoville	Up	Regantra	
	Down	-	
	Left	Destinations	
	Right	-	
~~Top Speed~~	~~Up~~	-	
	~~Down~~	~~Backlight~~	
	~~Left~~	~~Settings~~	
	~~Right~~	-	
Backlight	Up	~~Top Speed~~	
	Down	-	
	Left	Settings	
	Right	-	

PERFORMED BY A THIRD PARTY

INTERFACE TESTING

NAVIGATION THROUGH OPTIONS TESTING

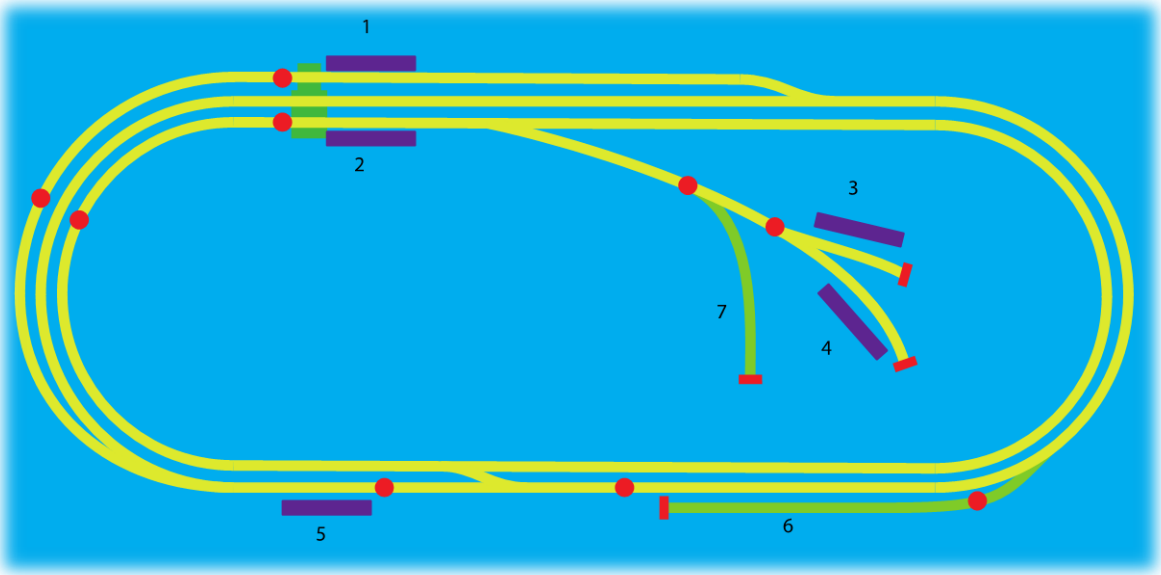
Object	Can be reached	Can return to menu
Destination: Hawkhaven		
Destination: Remilo		
Destination: Allantown		
Destination: Gregville		
Destination: Leovetticutte		
Destination: Regantra		
Destination: Vancoville		
Setting: Top Speed		
Setting: Backlight		

SELECTION OF OPTIONS TESTING

Object	Expected Result	Result Met
Welcome Page	-	
Destination	-	
Destination: Hawkhaven	Select Instruction Set 1	
Destination: Remilo	Select Instruction Set 2	
Destination: Allantown	Select Instruction Set 3	
Destination: Gregville	Select Instruction Set 4	
Destination: Leovetticutte	Select Instruction Set 5	
Destination: Regantra	Select Instruction Set 6	
Destination: Vancoville	Select Instruction Set 7	
Settings	-	
Setting: Top Speed	Open Speed Selector	
Setting: Backlight	Open Backlight Setter	

NAVIGATION BOUNDRY TESTING

Object	Direction	Expected Result	Result Met
Welcome page	Up	-	
	Down	-	
	Left	-	
	Right	Destination	
Destinations	Up	-	
	Down	Settings	
	Left	Welcome Page	
	Right	Hawkhaven	
Settings	Up	Destinations	
	Down	-	
	Left	Welcome Page	
	Right	Top Speed	
Hawkhaven	Up	-	
	Down	Remilo	
	Left	Destinations	
	Right	-	
Remilo	Up	Hawkhaven	
	Down	Allantown	
	Left	Destinations	
	Right	-	
Allantown	Up	Remilo	
	Down	Gregville	
	Left	Destinations	
	Right	-	
Gregville	Up	Allantown	
	Down	Leovetticutte	
	Left	Destinations	
	Right	-	
Leovetticutte	Up	Gregville	
	Down	Regantra	
	Left	Destinations	
	Right	-	
Regantra	Up	Leovetticutte	
	Down	Vancoville	
	Left	Destinations	
	Right	-	
Vancoville	Up	Regantra	
	Down	-	
	Left	Destinations	
	Right	-	
Top Speed	Up	-	
	Down	Backlight	
	Left	Settings	
	Right	-	
Backlight	Up	Top Speed	
	Down	-	
	Left	Settings	
	Right	-	



		Start						
		1	2	3	4	5	6	7
Destination	1							
	2							
	3							
	4							
	5							
	6							
	7							

Tested By: _____

X _____

Signed:

PERFORMED BY CLIENT

RATEING THE END PRODUCT

Statement	Strongly disagree	Disagree	Unsure	Agree	Strongly agree
Beltrac performs to my expectations.	1	2	3	4	5
I am happy to use Beltrac in the setting that I originally intended it.	1	2	3	4	5
Beltrac is aesthetically pleasing.	1	2	3	4	5
Beltrac meets the design specification	1	2	3	4	5
Beltrac meets the requirement specification	1	2	3	4	5
Having used Beltrac, I am happy with its interface	1	2	3	4	5
I found Beltrac easy to use	1	2	3	4	5
I was amused by Beltrac	1	2	3	4	5

RATEING THE PERFORMANCE OF THE DESIGNER

Statement	Strongly disagree	Disagree	Unsure	Agree	Strongly agree
Mr Bell performed to my expectations	1	2	3	4	5
I would consider commissioning Mr Bell again	1	2	3	4	5
Mr Bell's workmanship is of an acceptable quality	1	2	3	4	5
Mr Bell's designs are of an acceptable quality	1	2	3	4	5

GENERAL QUESTIONS

What pleases you about Beltrac?

What displeases you about Beltrac?

Did Beltrac fail to meet any of your expectations and if so, which ones?

Do you have anything else to add?

X

John Peter Thomas

EVALUATION

DISCUSSION OF THE DEGREE OF SUCCESS IN MEETING THE ORIGINAL OBJECTIVES

Beltrac works, which is a good result no matter the circumstances. There were a few things however that were not up to the standard that I would have liked.

For a start, the designs allowed for 8 point motors which could be implemented to control the train but in the end only 3 of them ever worked to standard despite constant debugging and alteration to the design. In the end, most of the point motors where replaced by elastic bands which sprung the points back to a certain position after the train passes over them. Though seemingly inconvenient, the train could still reach all the destinations like this by carefully writing the instructions to direct the train over these elasticated points such that it would go in the correct direction. However, this makes the routes more long and complicated than they need to be which interferes with the next problem:

The transformer from Hornby which supplies the power to Beltrac has a nasty habit of overheating, and its response to this is to back off on the power supplied to Beltrac which stops the train the only way to stop this problem was to turn it off and allow it to cool. This would screw up any long term use of the device and so an additional timer had to be installed to switch the transformer off and allow it to cool periodically - a serious inconvenience when running the train in a display environment.

To cover the full objective list here are my original objectives:

- The device should be simple and easy to use.
    - Very successful – Beltrac's menu has only a few options and they are clear and easy to understand.
- It should control the train according to instructions given by the user.
    - Very successful – Beltrac's destinations are always chosen by the user
- It should drive the train without needing any interference from the user.
    - Very successful – after repairing the transformer issue and the point motors, the train was able to travel with no human intervention
- It should provide a user interface consisting of a display and some controls.
    - Very successful – as shown in the pictures, the system has both of these
- The code should be adaptable to different situations with little modification.
    - Very successful – the code need only have a few variables adjusted and it is easy to understand how they work
- The user should not have to manually control the train.
    - Very successful – while this is an option that can be achieved through removal of a cover and turning of the speed control, it is not necessary to do.

All in all, I personally would consider the project a great success!

EVALUATION OF THE USER'S RESPONSE TO THE SYSTEM

DESIRABLE EXTENSIONS

Though infeasible at the time of creation, one of the most desirable extensions would be adding the multiple train functionality, either as outlined in the designs in this write-up or a full functionality where 2 trains could run on the same track at the same time and perform different tasks.

It occurred to me that Beltrac's casing is not particularly good looking, particularly its cut away window for the LCD display, in the future I would like to put more time into improving its appearances.

Another issue with the final product was that it could not handle 'or' commands, for example, a lot of instructions required sensor 2 to be triggered but sensor 2 was quite far away from the rest of the track and it needed to be passed by the train before it could travel to the next step even if the train was very close to (or on) its destination, making the routes taken by the train very inefficient. This could be achieved with more complex code.

Besides this however, all other aspects of the system work, and it does an effective job of what is intended!

Here's to the hardest and most time consuming project of my life so far, I don't know what I will do with myself now!