Sam's Portfolio

None

Sam Bell

None

Table of contents

1. Sam Bell's Portfolio of Projects

1.1 About Me

I am a 14 year old electronics and programming enthusiast. I love computer science and everything related to digital making, and have since I learnt the basics of Python when I was 8. Since then, it has become my main hobby and the direction I want my career to go in. In 2020, I got my first Raspberry Pi and learnt about electronics. Recently I realised that digital making and robotics combine both of my main interests, and those are what I'm focusing on now. I have made loads (82 that I still have records of!) of projects of varying degrees of difficulty. This portfolio contains the ones that I am most proud of.

1.2 My Favourite Projects

Project	Description	Link
Chess Bot	I created a fully-functional lookahead evaluation chess bot	View Project
CPU	A simple 8-bit CPU that I am currently making	View Project
QR Code Generator	I made a v5 QR code generator in python	View Project
WER 2025	An international robotics competition where I came 1st in the UK in my age category	More information

1.3 My Skills

1.3.1 Technical Skills

- **Programming**: I know how to program in many languages (e.g. C, HTML/CSS, and Go, to name a few), but my strongest are Python and Java.
- **Electronics**: Soldering, PCB Design (KiCad), and simply being able to work out how stuff should work.
- **Tools**: Git, Raspberry Pi, CAD (Autodesk Inventor & OpenSCAD)
- **Systems**: I understand how systems like Linux work due to me constantly changing, fixing and improving it on my own computer.

1.3.2 Soft Skills

- **Problem Solving** - I have come across many challenges over my 5 years of making, and seeing as my family and friends aren't interested in coding or making, I have to solve everything myself using the internet, magazines and books. This has made me pretty good at problem solving.
- **Curiosity** - I have always wanted to know how things work, and I usually find this out by making it (e.g. my QR code generator)
- **Resilience** - I just refuse to give up, even if a problem takes me years to solve! I started many of my projects (such as the Light Fantastic) when I was younger and didn't have enough knowledge, skills or experience, but I kept coming back to them until I managed to finish them.

1.4 Recent Achievements

- **British champion** in an international robotics competition
- **Finalist** in Cyber Switchup 2023 (https://cyberswitchup.net/) (I missed this year's deadline)
- Predicted **Grade 9** in GCSE Computer Science

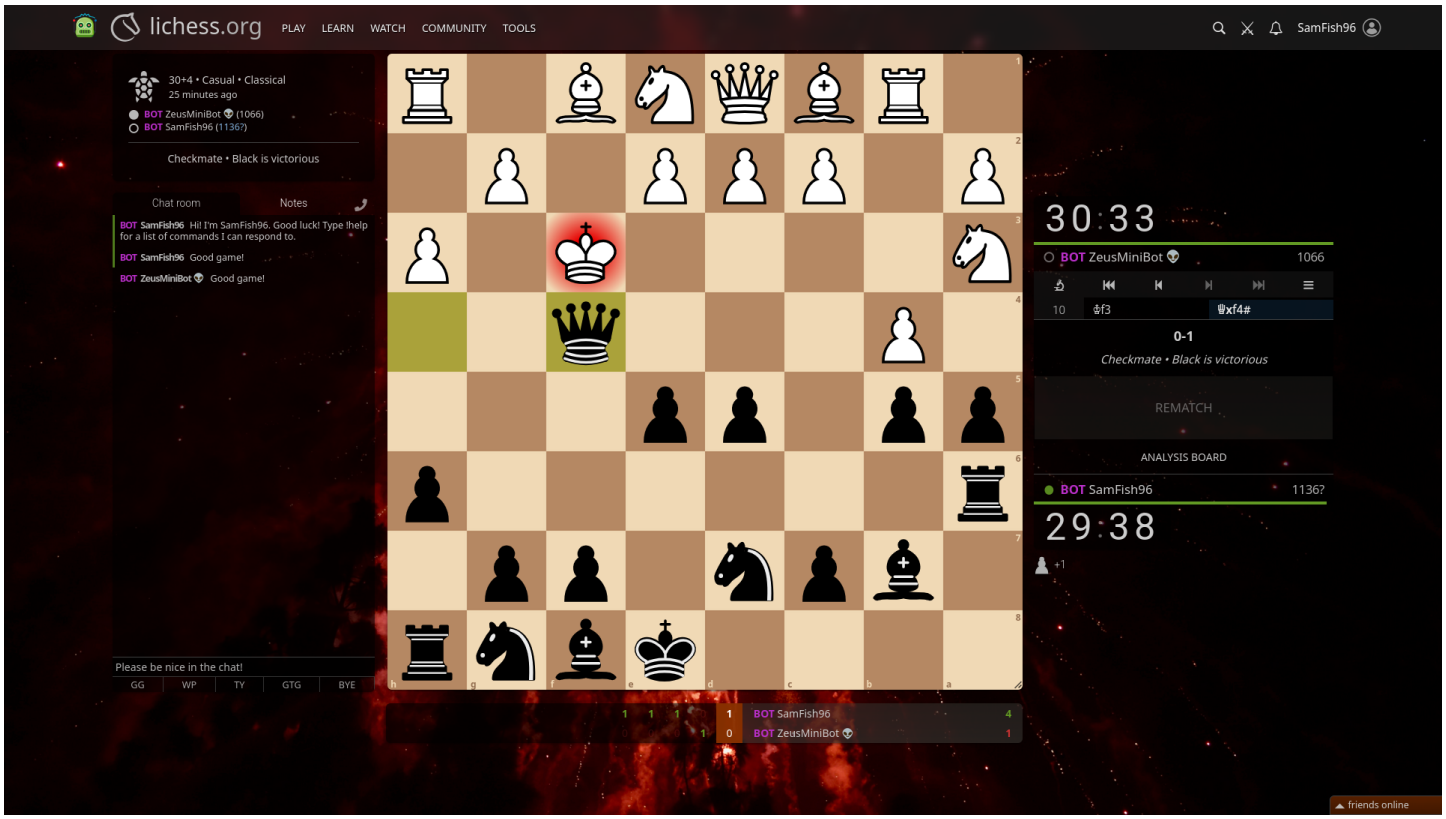
1.5 Get in Touch

- **Email:** samgbell2011@icloud.com

- **Phone:** 07468779147
- **Portfolio:** https://sambell2.github.io (https://sambell2.github.io)

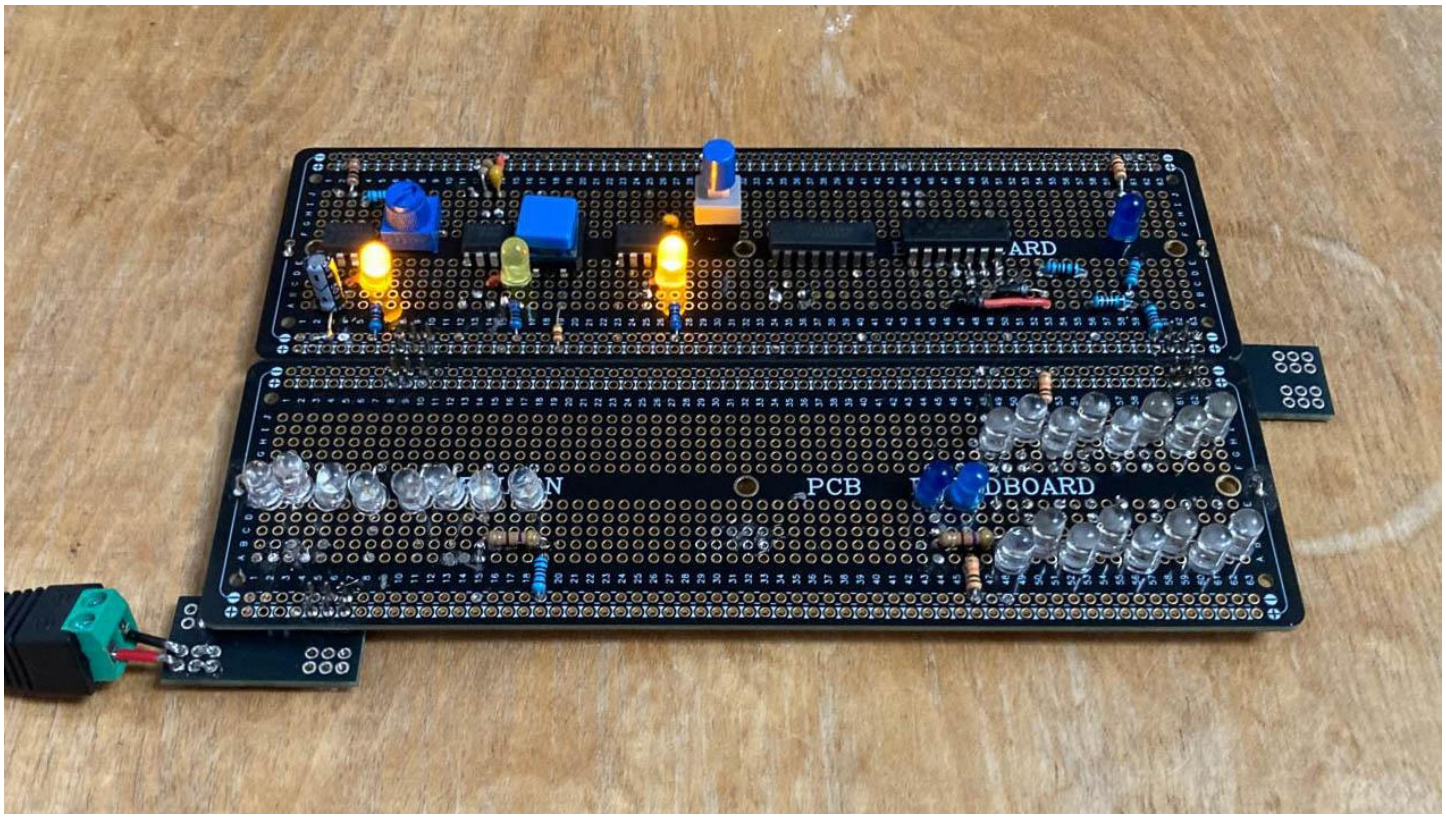
1.5.1 View My Full Portfolio ➜

2. My Projects

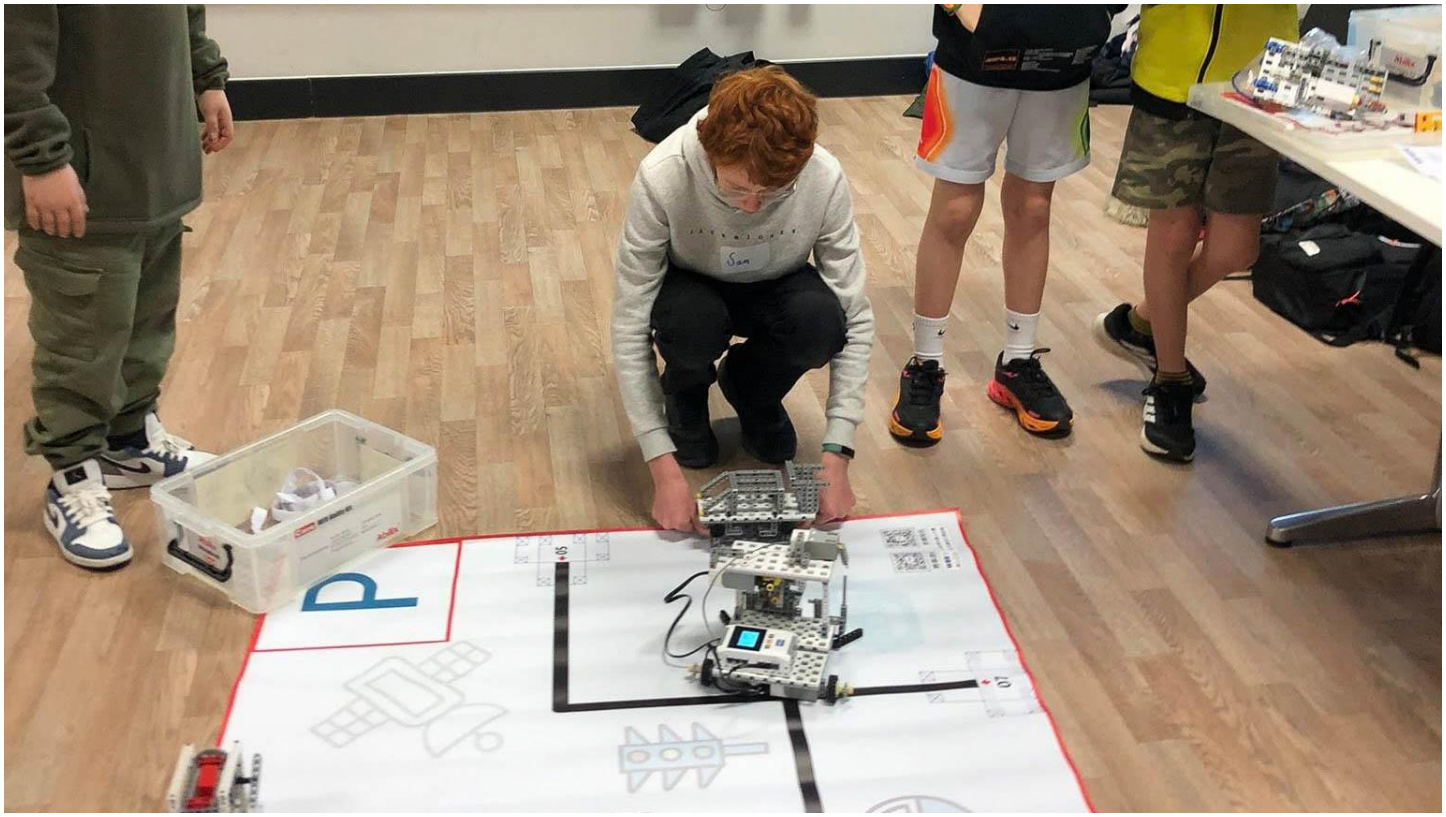


Chess Bot

(2025 - Aged 14)



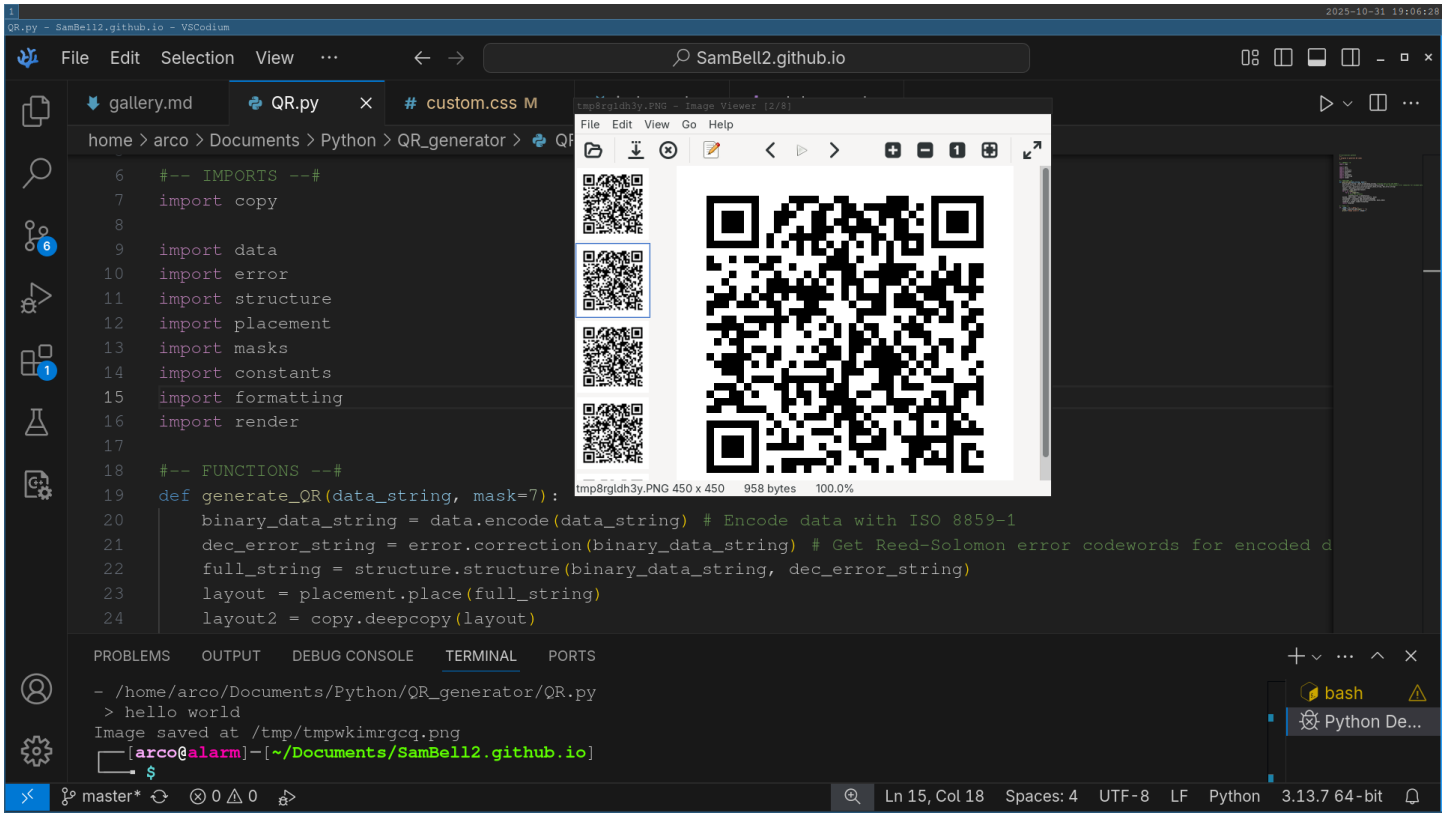
CPU
(2025 - Aged 14)



WER 2025
(2025 - Aged 14)

Light Fantastic

(2023-2025 - Aged 12-14)



QR Code Generator

(2024 - Aged 13)

Laptop

(2023-2025 - Aged 12-14)

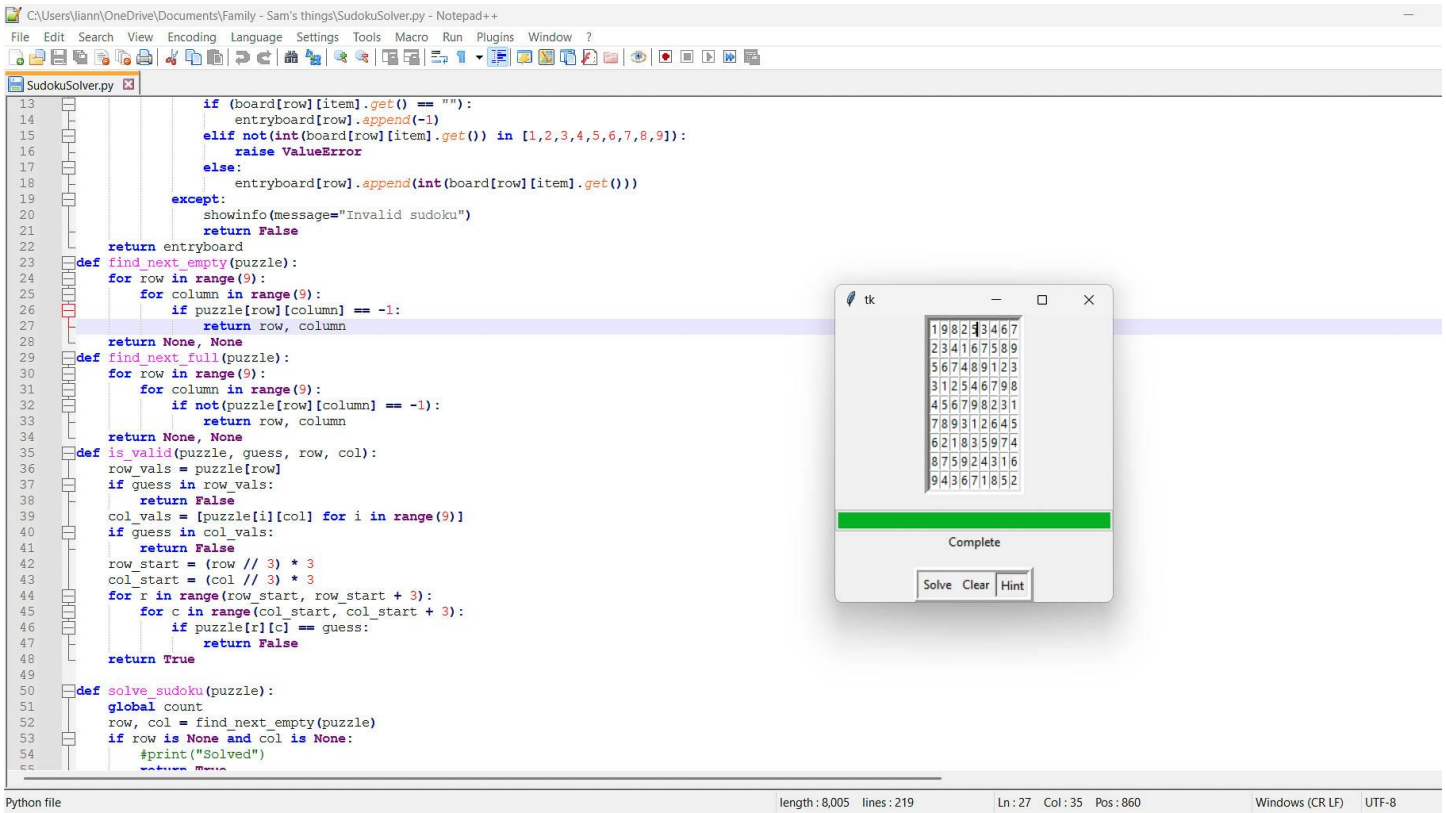
```
arco@alarm:~                                                    2025-10-31 19:10:52
You see an exit to the North.
You see an exit to the South.
You see an exit to the East.
 > move north



Crystal Grotto
Crystals embedded in the walls glow with a soft, ethereal light, creating a mesmerizing display.
There are no items here.
A massive golem made of living crystal, slow but immensely strong.
You see an exit to the North.
You see an exit to the South.
You see an exit to the East.
 > fight golem

A battle begins with the Golem!
You hit the Golem with your Fists. It causes 0.5 damage.
The Golem hits you with its Runic Hammer. It causes 3 damage.
HP: [██████████------] 7/10 HP
Continue?
 > yes
You hit the Golem with your Fists. It causes 0.5 damage.
The Golem hits you with its Runic Hammer. It causes 3 damage.
HP: [████------------] 4/10 HP
Continue?
 > yes
You hit the Golem with your Fists. It causes 0.5 damage.
The Golem hits you with its Runic Hammer. It causes 3 damage.
HP: [█-----------------] 1/10 HP
Continue?
 > 
```

Scripted Journeys

(2024-2025 - Aged 13-14)

Magic Mirror

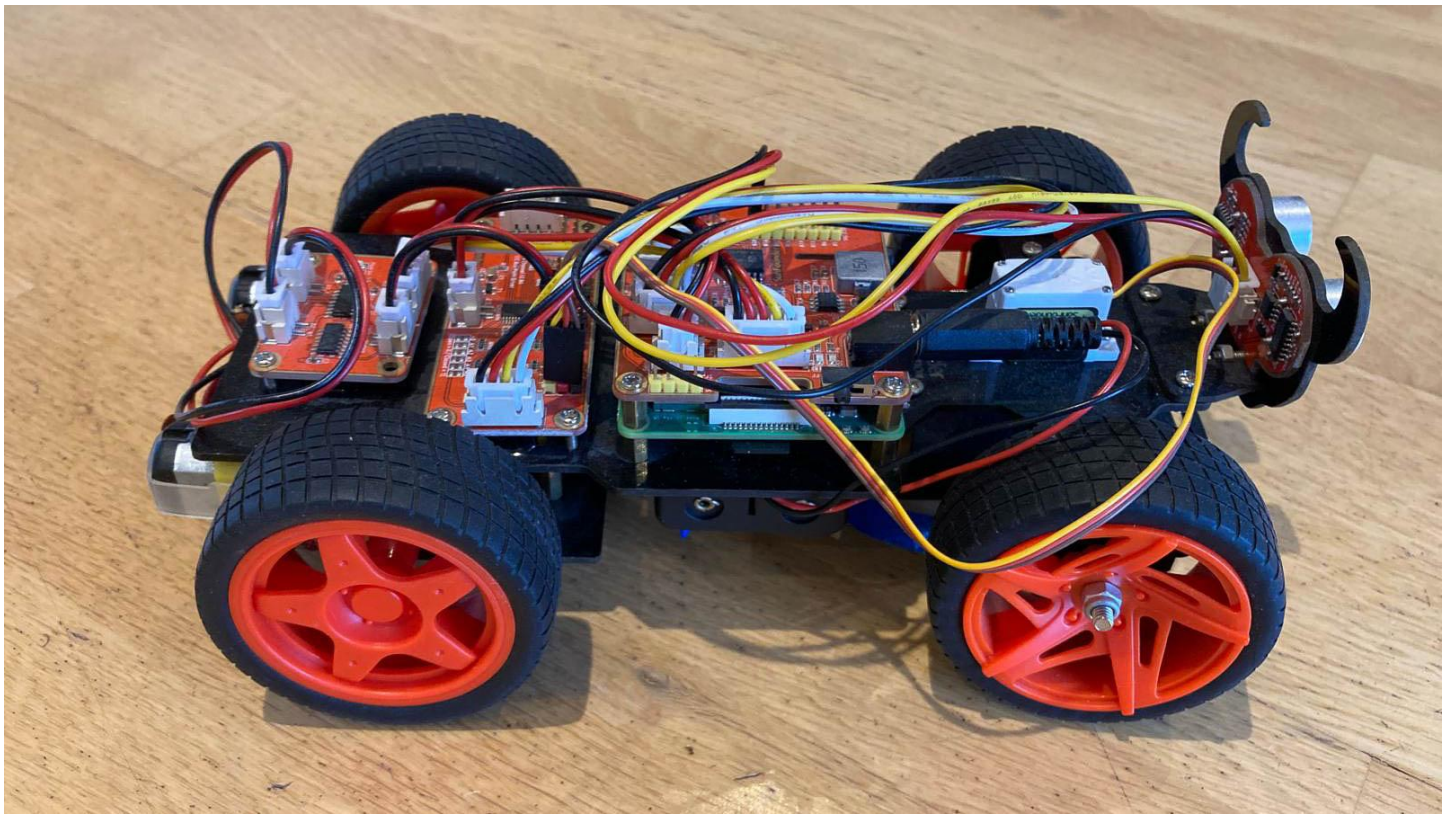
(2023-2025 - Aged 12-14)




```
                if (board[row][item].get() == ""):
                    entryboard[row].append(-1)
                elif not(int(board[row][item].get()) in [1,2,3,4,5,6,7,8,9]):
                    raise ValueError
                else:
                    entryboard[row].append(int(board[row][item].get()))
            except:
                showinfo(message="Invalid sudoku")
                return False
    return entryboard
def find_next_empty(puzzle):
    for row in range(9):
        for column in range(9):
            if puzzle[row][column] == -1:
                return row, column
    return None, None
def find_next_full(puzzle):
    for row in range(9):
        for column in range(9):
            if not(puzzle[row][column] == -1):
                return row, column
    return None, None
def is_valid(puzzle, guess, row, col):
    row_vals = puzzle[row]
    if guess in row_vals:
        return False
    col_vals = [puzzle[i][col] for i in range(9)]
    if guess in col_vals:
        return False
    row_start = (row // 3) * 3
    col_start = (col // 3) * 3
    for r in range(row_start, row_start + 3):
        for c in range(col_start, col_start + 3):
            if puzzle[r][c] == guess:
                return False
    return True

def solve_sudoku(puzzle):
    global count
    row, col = find_next_empty(puzzle)
    if row is None and col is None:
        #print("Solved")
```

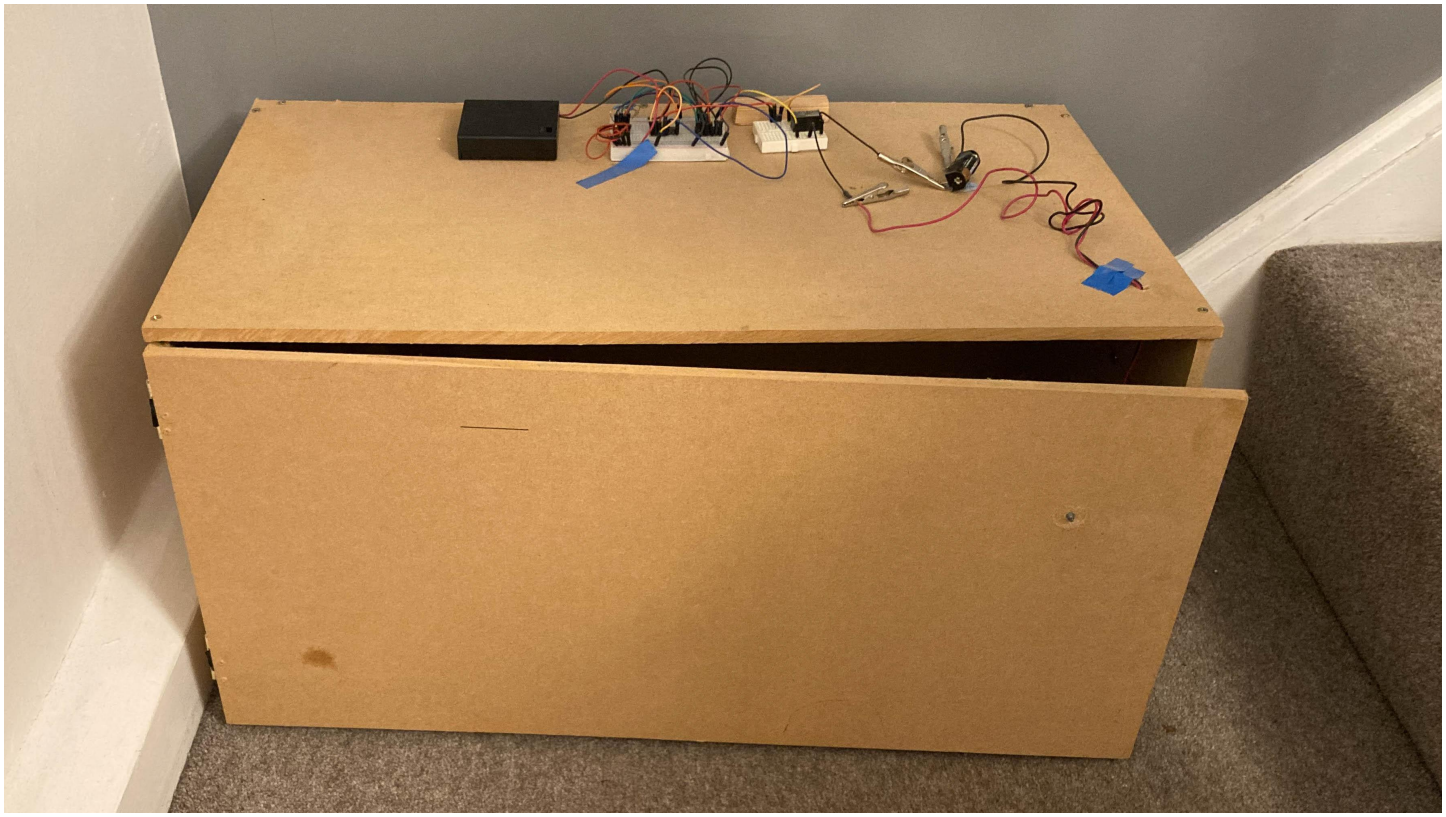
Sudoku Solver

(2023 - Aged 12)



Picar S

(2023-2025 - Aged 12-14)

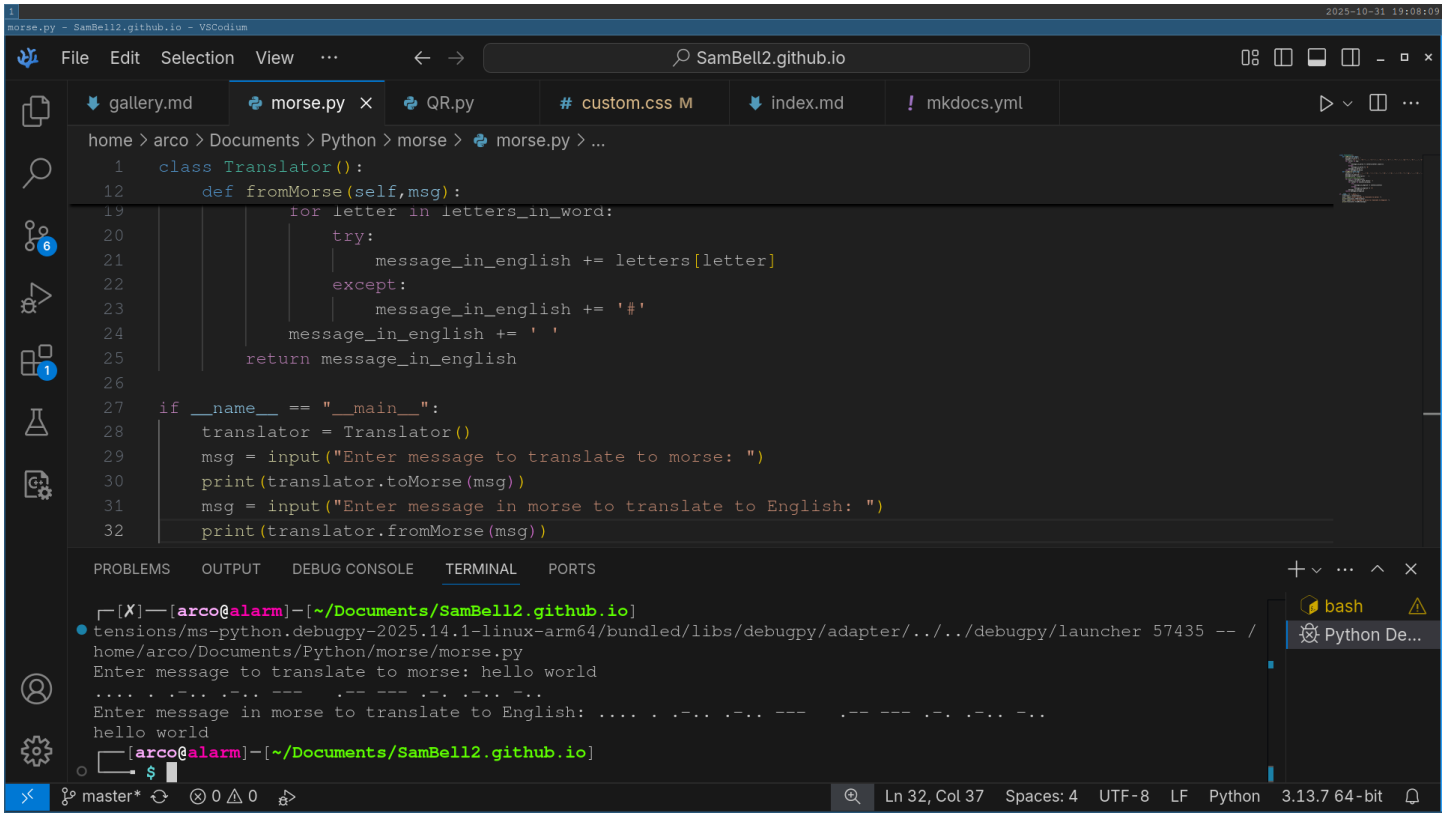


Safe

(2021 - Aged 10)

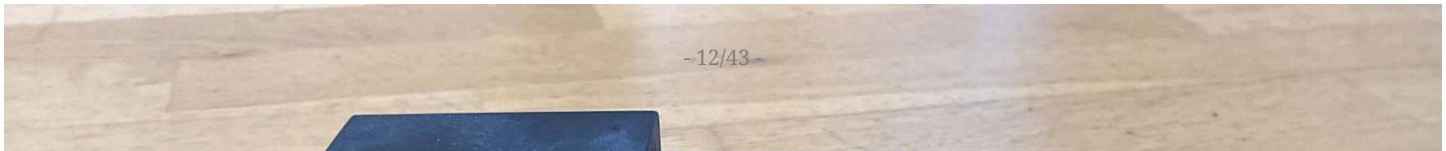
Wildlife Camera

(2022 - Aged 11)



Morse Code Translator

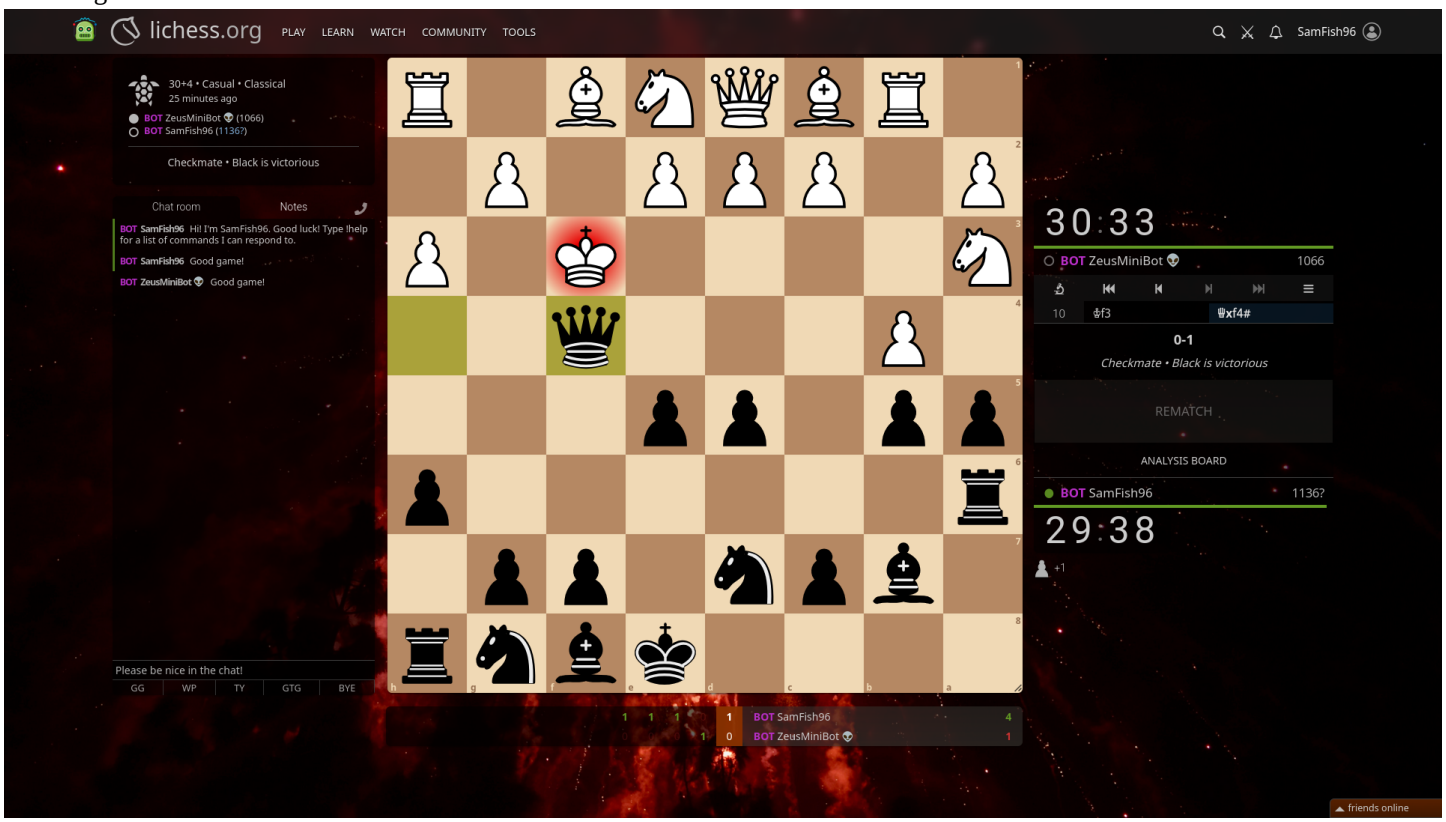
(2021 - Aged 10)



3. Programming

3.1 Chess Bot

2025 - Aged 14



A Lichess.org (https://lichess.org/h98ou7COBKNf) game where my bot won

3.1.1 Overview

I created a Java program that can play chess. It generates all legal moves and evaluates them, giving them points. It then plays the highest-ranking move. I have given it a UCI (Universal Chess Interface) frontend so it can interact with other chess apps. For example, I have given it a Lichess account (SamFish96 (https://lichess.org/@/SamFish96)) and occasionally it is online.You can see the source code on GitHub (https://github.com/SamBell2/SamFish/tree/master/src/chess).

3.1.2 Inspiration

I've been interested in complex decision making programs for a while, and a chess bot was a suitably complex but not overwhelming introduction to them.

3.1.3 How it works

It uses an 8x8 array of pieces or empty squares to represent the board. Whenever the chess app tells it that a move has been made, it simulates the move on its own board. When it needs to move one of its pieces, it looks at all of the possible moves that it can make, discards any that put it in check, then evaluates all of them. If it has time, it looks at the next turn or two as well. It then picks the move it thinks is best and plays it.

3.1.4 Key Features

- Generates almost all possible moves.
- Uses syzygy tablebases (https://syzygy-tables.info/) for the endgame to ensure perfect moves.
- Detects checks, checkmates, stalemates and threefold repetition.
- Automatic detailed logging to see the bot's logic.
- Interacts with the Lichess bot API to play online.

3.1.5 Improvements

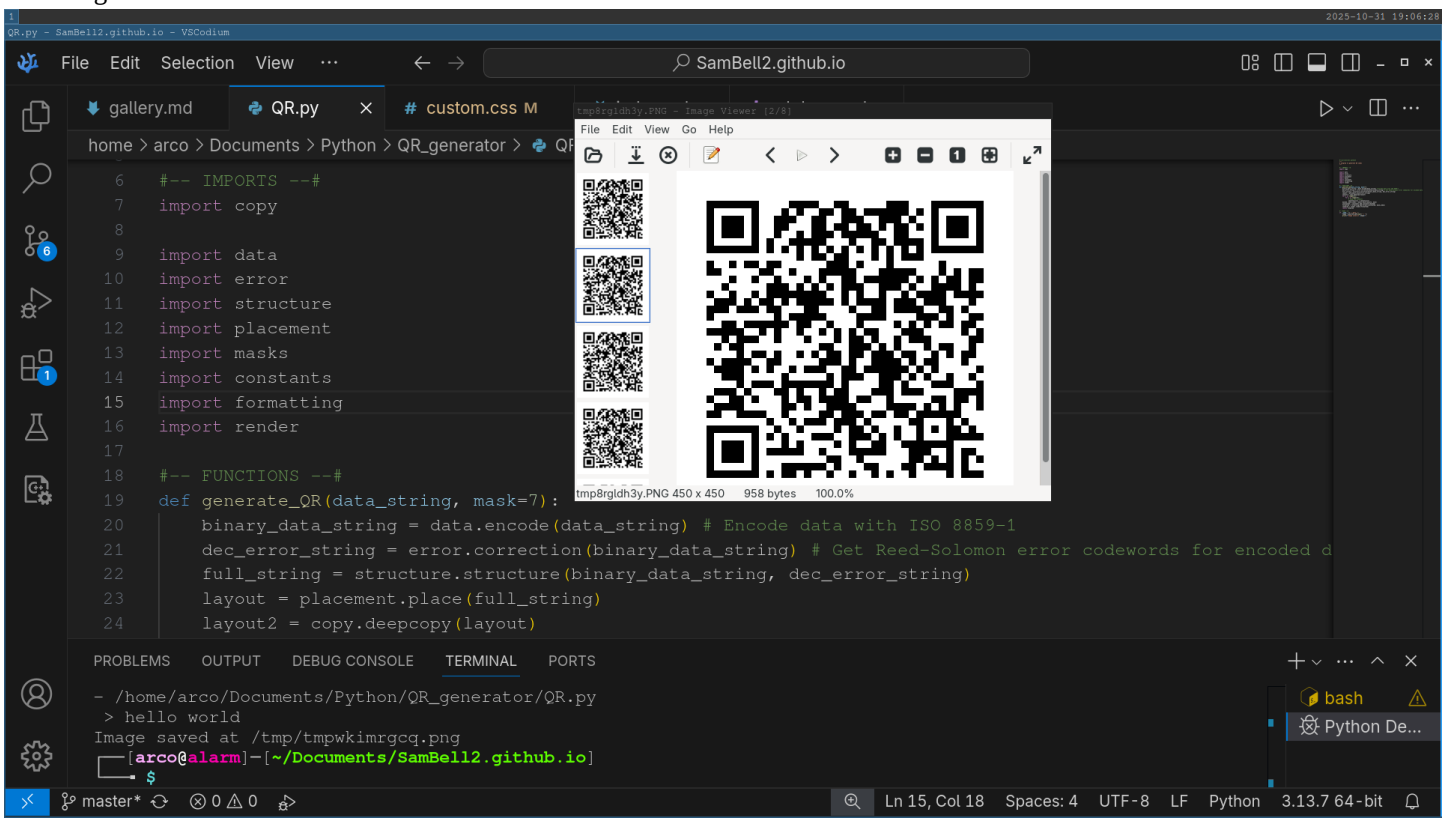
- Add castling and en passant capabilities
- Increase speed

3.1.6 Challenges

I learnt a lot about Java, as before I had mainly programmed in Python. In particular, I use interfaces quite a bit. Looking for checks, checkmates, stalemates and other draws was far harder than I had anticipated, but I did eventually manage. Interacting with syzygy tables was a bit too complex for me to do (there are no Java libraries to read them) so I used an existing library called Fathom (https://github.com/jdart1/fathom/) to do it.

3.2 QR Code Generator

2024 - Aged 13



Part of the code and the ouput of the QR code generator

3.2.1 Overview

I got the inspiration for this project because QR codes are everywhere but very few people actually think about how they work. I then found an amazing guide (https://www.thonky.com/qr-code-tutorial/) that showed every step. QR codes are far more complicated than you would think, mostly due to the error-correction. You can imagine that if you scan a QR code in not ideal conditions, then the camera may misread one or two pixels. This means we need error correction, which is really complex! The code is on my Github (https://github.com/SamBell2/QR_generator).

3.2.2 What It Does

It takes a line of text as input from the user, then generates the QR code, saves it as a temporary file and opens it in your default image viewer. From there you can save it somewhere else or share it.

3.2.3 Technical Summary

It is made of almost 700 lines of Python code split over 9 files. I have used no external modules relating to QR codes, only `pillow` for graphics, `copy` to duplicate it and `tempfile` to save the image. It generates a v5 error level Q alphanumeric QR code, which can hold 87 characters.

3.2.4 Steps

There are multiple steps in making a QR code:

1. Encode the text in a particular text encoding (ISO 8859-1)
2. Generate error correction codewords
3. Interleave the data and error codewords to make a final binary string
4. Place the individual bits of the binary string in the correct places
5. Apply the mask that makes the QR code easier to read
6. Add constant elements (e.g. the corners)
7. Render

3.2.5 Encoding

This is one of the simplest steps, as all you have to do is connect the mode string (0100 for text) to the length of the string in binary. You then have to convert each character in the text to the ISO 8859-1 standard. This is easier than I thought it was going to be as it is the standard that Python's `chr()` function uses. After that, you just need to fill out the rest of the space with a repeating pattern.

3.2.6 Error Correction

This is by far the hardest part of the QR code. It has quite a lot of maths involved, including polynomial long division in a Galois Field 256. You have to convert the data string to a message polynomial and get a generator polynomial. You then have to divide the message polynomial by the generator polynomial, and the coefficients of the remainder polynomial are the error correction codewords.

3.2.7 Structuring

You have to break the data codewords into blocks, then interleave the first codeword from the first block, followed by the first from the second block and so on. Once you have done the first codeword from each block, you then move onto the second from each block. Once you have done all the data, you then do the same with the error correction codewords. You then need to add a certain number of zeroes to the end so the QR code will be full.

3.2.8 Placement

This is a relatively simple part, although still more complex than you would imagine as there are a number of pixels that you have to avoid. You just start at the bottom-right of the QR code and work your way up in a zig-zag pattern, avoiding all of the reserved areas.

3.2.9 Masks

Occasionally, when you make a QR code, there will be areas of data that look like special symbols such as the corner patterns. This would really confuse the scanner, so you have to apply a mask. These are just lists of which pixels to invert. You have to apply all of them and evaluate which is the best (i.e. has the fewest confusing patterns) then apply that one.

3.2.10 Constants

This is the simplest bit as most of it is pre-set. You have to add the corners, borders, alignment patterns, timing patterns, dark module and the format information areas. The format information needs to be generated but isn't hard. The format string is mostly the same for my codes as I have only one type of QR code. The only bit that changes is the mask pattern used. You then have to error-correct it and add it.

3.2.11 Render

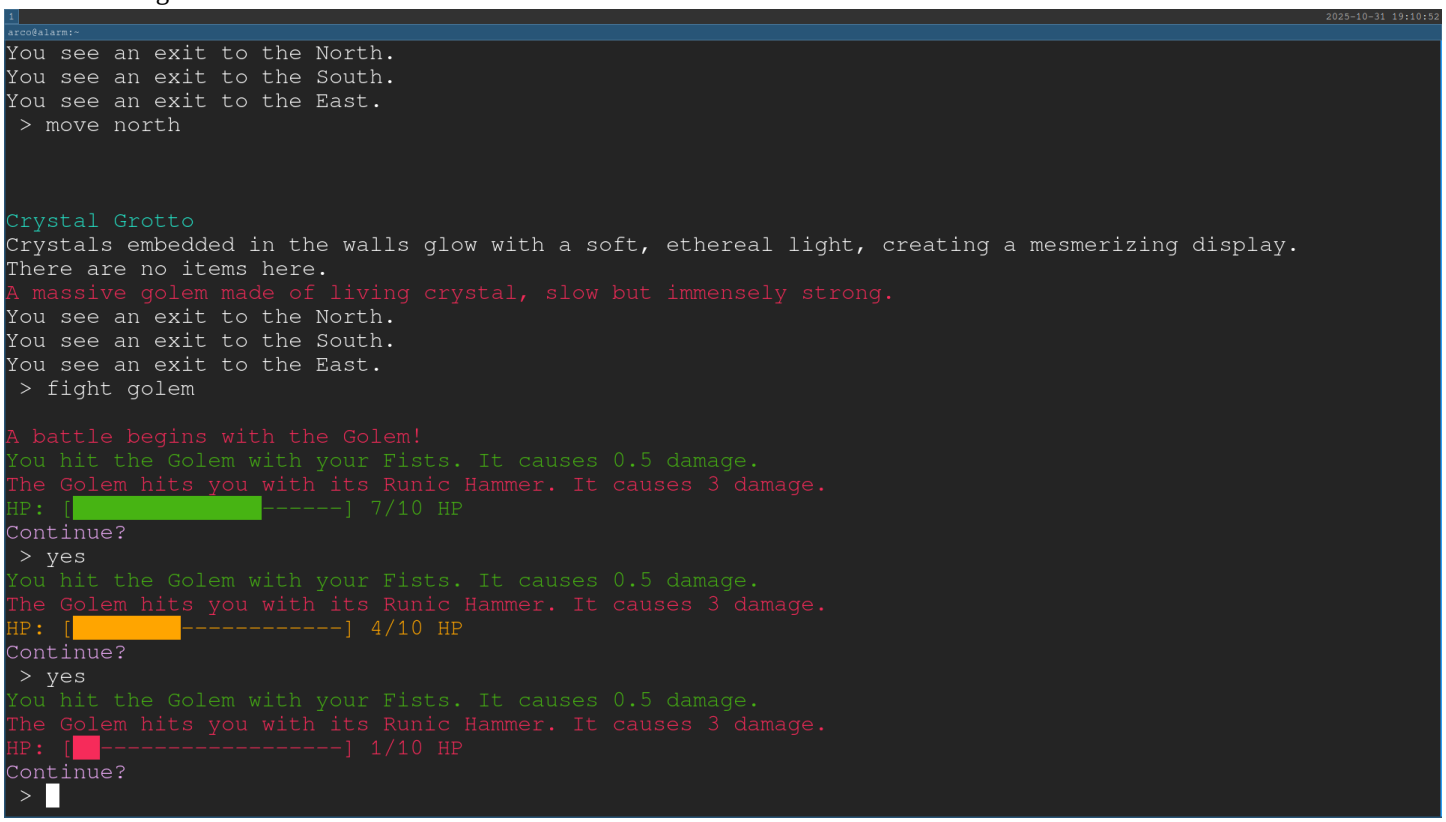
For this, I use Python's `pillow` library to work with images. Each pixel of the QR code is 10 pixels wide in the image so when I open it in an image viewer, it doesn't blur the edges between pixels. I just loop over each pixel in the QR code and place a 10x10 square of whatever colour it is in the image.

3.2.12 Challenges

The hardest bit of this was the error correction, as at the time I didn't even know what a polynomial was or how binary worked. I had to write code to do maths with strings in weird number systems. It took me about a week to make 160 lines of code, and another week to test and debug it!

3.3 Scripted Journeys

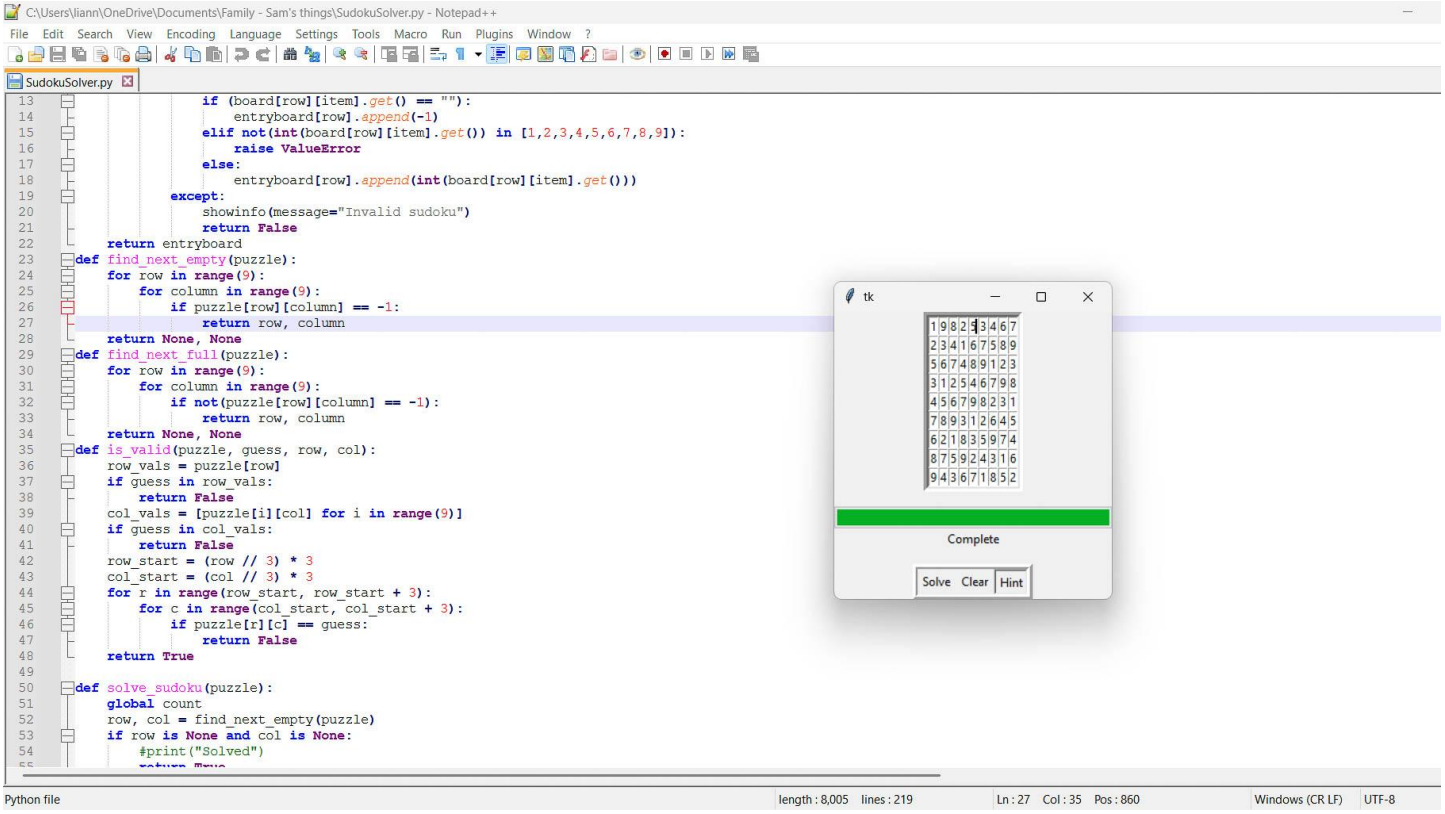
2024-2025 - Aged 13-14



3.3.1 Details coming soon!

3.4 Sudoku Solver

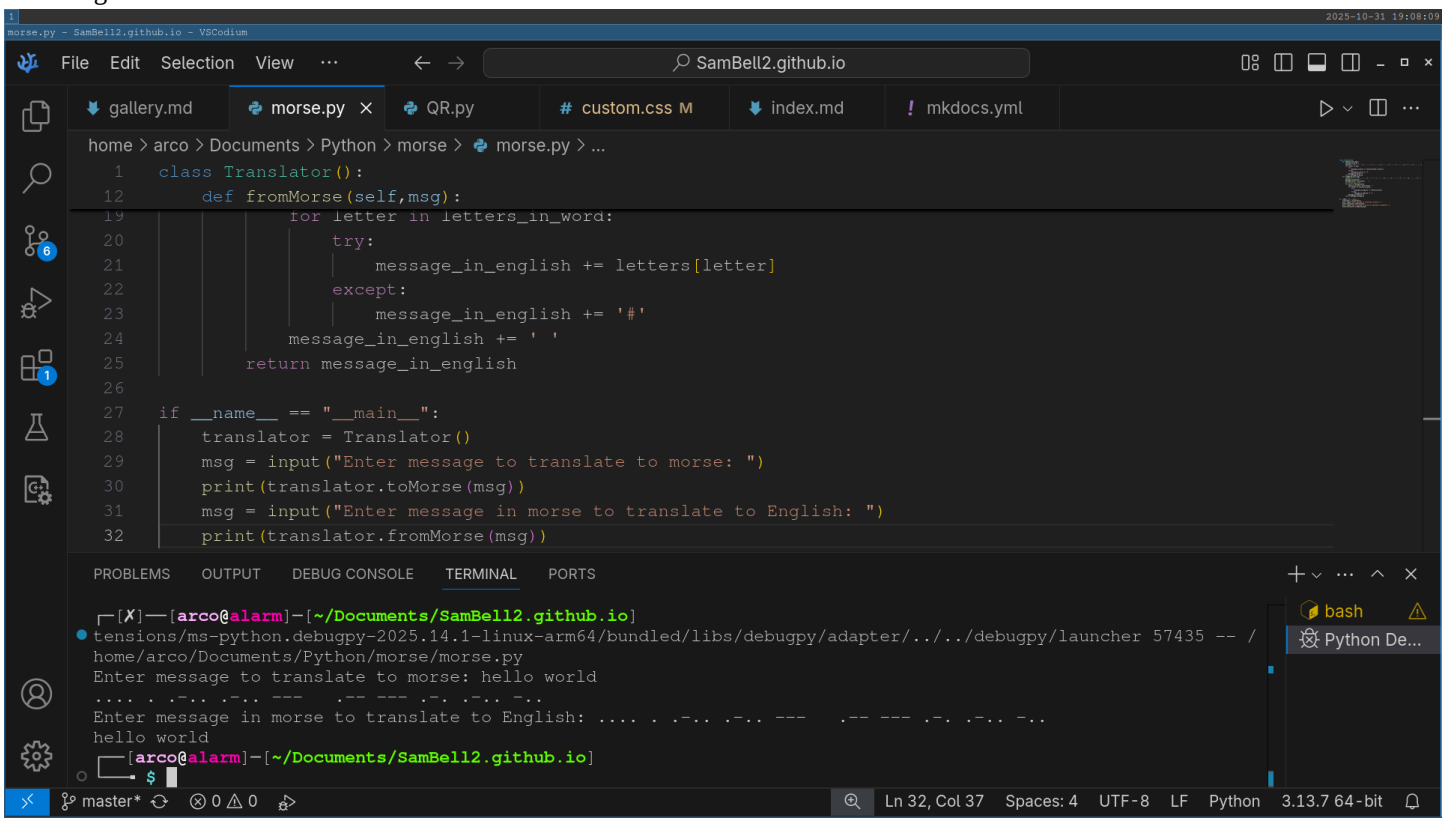
2023 - Aged 12



3.4.1 Details coming soon!

3.5 Morse Code Translator

2021 - Aged 10



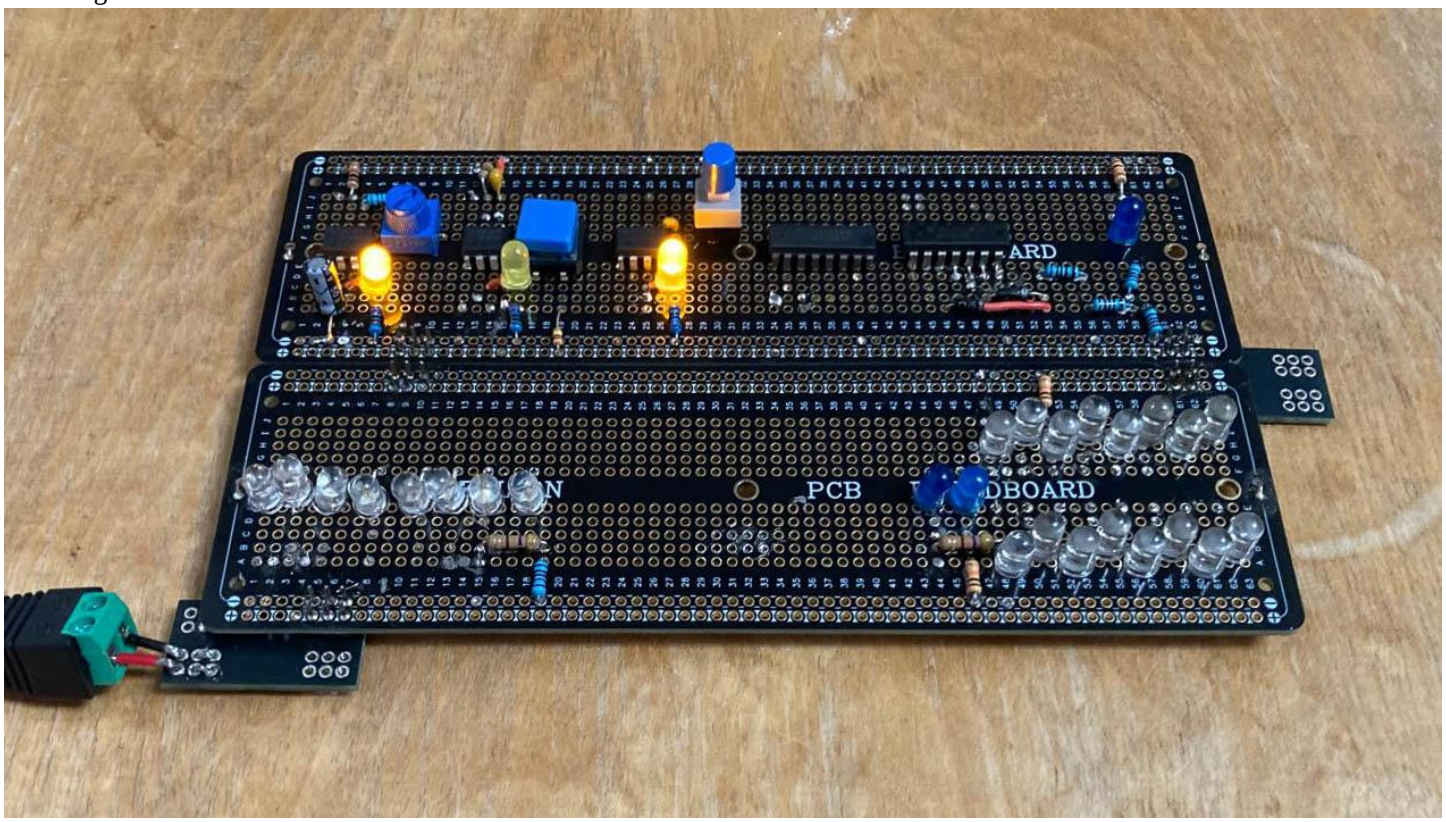
3.5.1 Details coming soon!

4. Electronics

4.1 CPU

4.1.1 CPU

2025 - Aged 14



An image of my CPU so far

Overview

This is my current project, it is an 8-bit CPU built out of individual ICs (Integrated Circuits). I got the idea in my Computer Science lessons at school, where I learnt how a CPU works and realised that it isn't actually as complex as I thought. I then started to make parts of it to test in Minecraft, before finding Ben Eater's similar project (https://www.youtube.com/playlist?list=PLowKtXNTBypGqImE405J2565dvjafglHU) and using that as a guide to making the basic parts of the computer.

Specs

- 6 - 48 Hz variable clock speed
- 1 core
- 64kB RAM
- 16 registers; r0 is a zero register
- 8-bit maths

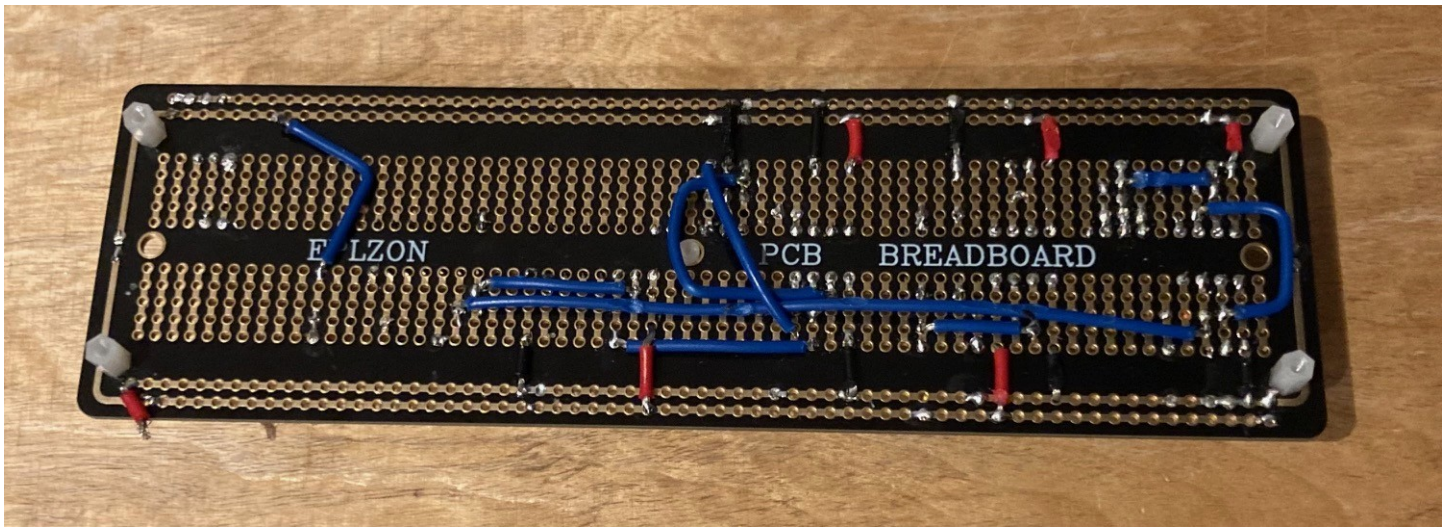
- Von Neumann architecture

My Aim

I want to make an 8-bit CPU with 16 instructions. It will be fully Turing-complete. It will be able to do maths, store data, jump and branch to other parts of a program, and input and output to other devices. The instruction set I want to implement are adding, subtracting, NORing, right shifting, loading from RAM, storing to RAM, loading from a port, storing to a port, loading immediate values, adding immediate values, jumping around the program, branching to other parts of a program depending on a condition, and halting.

How I'm Building It

I am making it module by module, first breadboarding it to test it and make sure it works, then soldering it down to a solderable breadboard (https://www.amazon.co.uk/Prototype-Solderable-Breadboard-Electronics-Gold-Plated-2-Matte-Black/dp/B082KY5Y5Z?th=1). So far, I have completed the clock module, and mostly done the ALU. I am soldering the components to the top of the board so you can see them easily and the wires to the bottom of the board, just because I think it looks neater.



The back of my CPU clock with all the wires

What I'm Learning

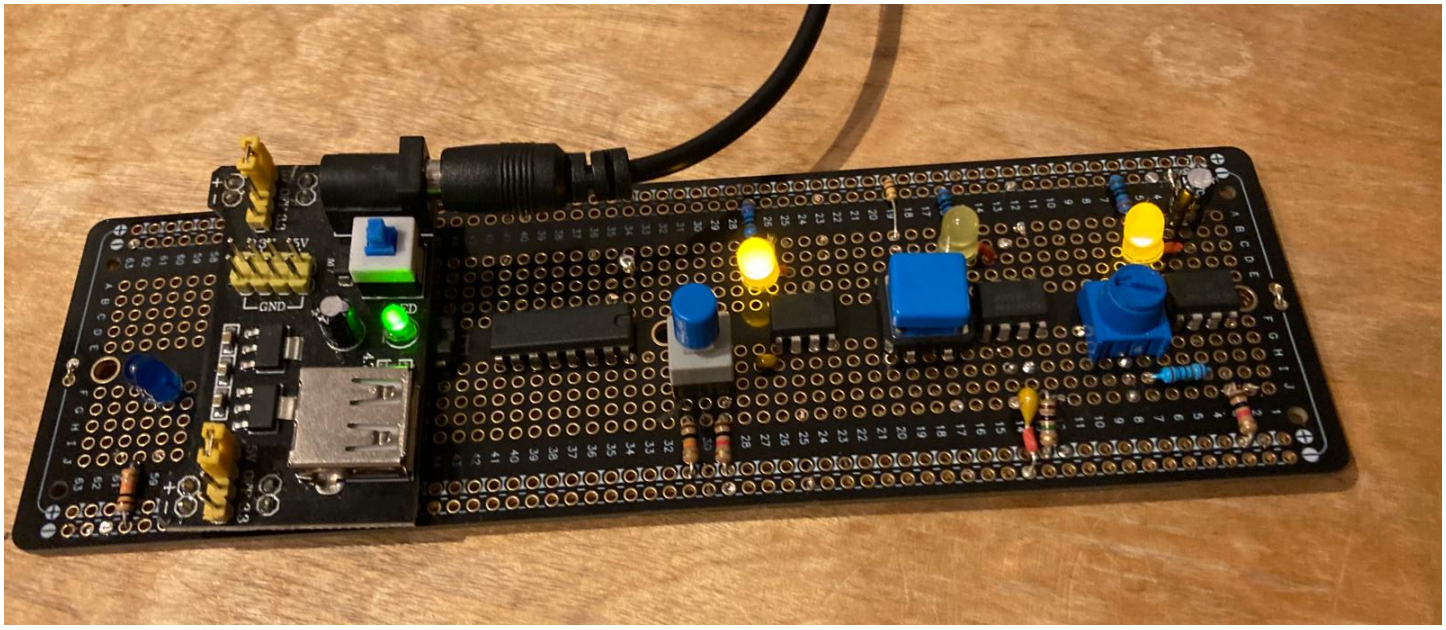
I am learning LOADS about the internal mechanisms of how computers actually work at the most detailed level, as well as soldering skills, PCB design, and electronic principles such as why pull-up/down resistors are actually necessary.

More Details

- Clock Module
- ALU

4.1.2 CPU Clock

2025 - Aged 14



An image of my CPU clock

Overview

In a CPU, it needs something to tell it when to do things. This is the clock. Most computers run at a speed of around 3*GHz* (3 *billion* cycles per second). Mine will run at up to 48Hz. The clock is made of 4 parts: the automatic clock, the manual clock, the mode selector and the clock logic. The automatic clock just turns on and off again at a variable speed, set by a potentiometer (a dial). The manual clock pulses whenever you push a button, mainly for debugging and testing. The mode selector lets you pick which mode to use and the clock logic actually does the switching.

The Automatic Clock

This was the simplest bit, as it was just a 555 timer in astable mode. I used a potentiometer between pins 6 & 7 so I could change the speed. The rest of the connections were quite straightforward.

The Manual Clock

This was far more difficult than I had anticipated, as buttons often bounce, triggering multiple pulses when you only wanted one. I had to use a 555 timer in monostable mode, so when you push the button, it turns on and 0.3 seconds after you release the button, it turns off. This means that if the button does bounce, the extra pulses will be caught in the 0.3 seconds before it turns off.

The Mode Selector

Both this and the clock logic could be replaced by a simple 3-pin switch, but again you get the problem of button bounce. Instead, I had to use a 555 timer in bistable mode, where you can push a button once and turn it on, and push the button again and turn it off. This was the hardest part, and I took a while to get it working. After a while, I found that you should add a 100nF capacitor between pin 5 & ground, so I did and it worked.

The Clock Logic

Now that I have 3 signals (automatic clock, manual clock and mode), I had to make a circuit to switch between auto and manual modes. This wasn't too hard. First, I had to invert the mode so I had a mode signal and a not mode signal. I then ANDed the auto clock signal with the not mode signal, and the manual clock with the mode signal. I then ORed the two outputs together with a homemade diode OR gate. This was my first problem, as it either gave a high output or a floating output, which was registered as high by the next gate. I fixed it by adding some pull-down resistors. Finally, the output of that gets ANDed with the inverted halt signal to get the output.

Challenges

When testing the clock module, the circuit that should have made the bistable 555 timer didn't work, so I had to experiment with a lot of different methods. I eventually got it working by tying a 0.1µF capacitor across pin 5 and ground, but weirdly when I soldered it down, it worked without the capacitor. I still added it in just in case. In addition, I had a problem with floating outputs of the diode OR gate, but I fixed it with a pull-down resistor. However, sometimes now when it is on the automatic clock, it just stays off. I think I just need to reduce the strength of the resistors, but I haven't done this yet.

4.1.3 ALU

2025 - Aged 14 An image of my ALU *An image of my ALU*

Overview

Doing maths is one of the main purposes of a CPU, and the ALU (Arithmetic and Logic Unit) is the bit that actually does it. As with the rest of my CPU, this is less powerful than most modern ones. It has 4 modes, add, subtract, NOR, and right shift (divide by two, ignore any remainder). I am making it on 2 breadboards: the top one to show the inputs, output, and setting, as well as switching between modes, and the bottom one to actually do the maths. The top breadboard has 26 LEDs on it, and will have 2 8-bit 2:1 multiplexers. The bottom one has 2 quad-XOR gates, 2 4-bit adders and 2 quad-NOR gates. The right shift is simply connecting the inputs to the output in a different place.

The LEDs

This was a simple part. All I had to do was solder down a whole load of green and blue LEDs, some resistors and some wires. Despite this, I still came across a few issues when prototyping. The current-limiting resistors I used were 470R resistors, which pulled too much current and so the chips to actually use the data sometimes didn't have enough current to know if it was a 1 or a 0. This meant the built-in pull-up resistors took over and the actual data was ignored. I fixed this by using 10kR resistors instead.

Adding and Subtracting

I used 4 chips for this, 2 adders and 2 XOR chips. Adding 2 numbers is simple: just feed them into the adders (I do know how the adders actually work, but making my own would be too expensive). Subtracting is harder, as you have to negate the second number and then add them together. This is essentially just doing 5 + -3 instead of 5 - 3. In binary, negative numbers are expressed in two's complement, which you convert into by inverting all of the bits and adding 1. I accomplished this by using a single subtract signal (this is actually the least significant bit of the setting as the setting for add is 00, so it won't subtract, subtract is 01 so it will subtract, and the other two don't matter as they won't use the adders anyway). The subtract signal is XORed with all of the data bits for the second number, as any number XOR 0 is itself, and any number XOR 1 is itself inverted. The subtract signal then also goes to the carry in of the first adder, so it will add one. This setup means that the second number will always go through the XOR gates, simplifying the wiring even if it does make the logic more complex.

NOR Operation

This was a simple bit, as I just connected the inputs of the ALU to the inputs of the NOR gates. I came across no problems, and it was straightforward.

Right Shift

I haven't done this yet, as it will be very simple. I don't have the multiplexers yet so I have to wait to connect the inputs to one.

Mode Selection

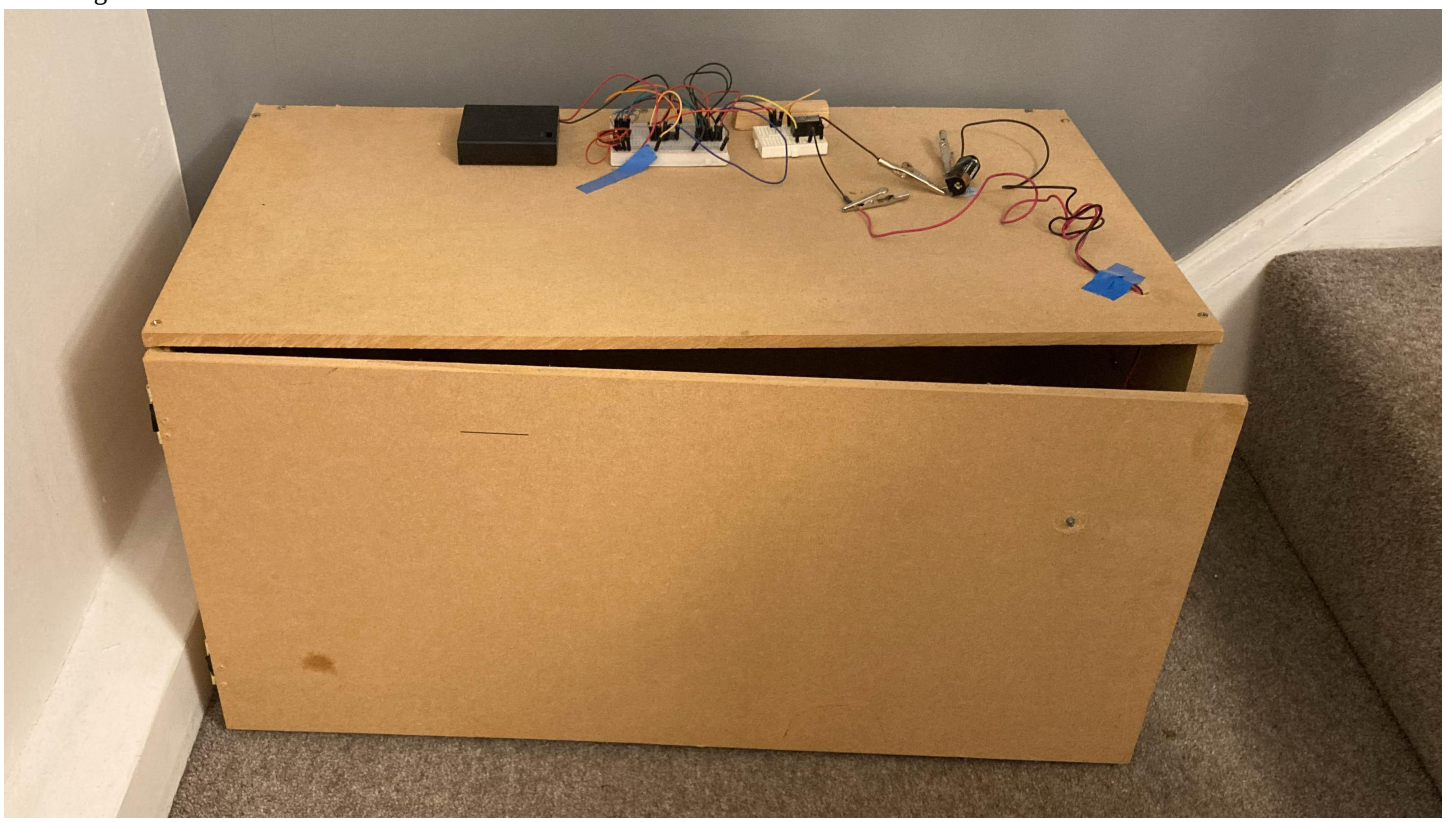
I haven't actually made this yet, but I have the logic all worked out. It needs 2 8-bit 2:1 multiplexers. The first one chooses between the NOR and right shift operation, and the control bit is the least significant bit of the setting. If it is 0 (in NOR or ADD), then the multiplexers will choose the NOR output and the adders will add. If it is 1 (in RSH or SUB), they will choose the right shift output and the adders will subtract. We now have 2 8-bit numbers, one from the NOR/RSH and one from the adders. The most significant bit of the setting decides this one. If it is 0 (in ADD or SUB), the multiplexers will choose the adders. If it is 1 (in NOR or RSH), they will choose the output from the first multiplexers. The output of these multiplexers will go straight to the output LEDs.

Challenges

I didn't come across many challenges (yet) when making this. The main one was the weak resistors pulling all of the current so the chips registered a high input, but this was easily fixed. I still have the multiplexers to add though, so I may come across more problems.

4.2 Safe

2021 - Aged 10

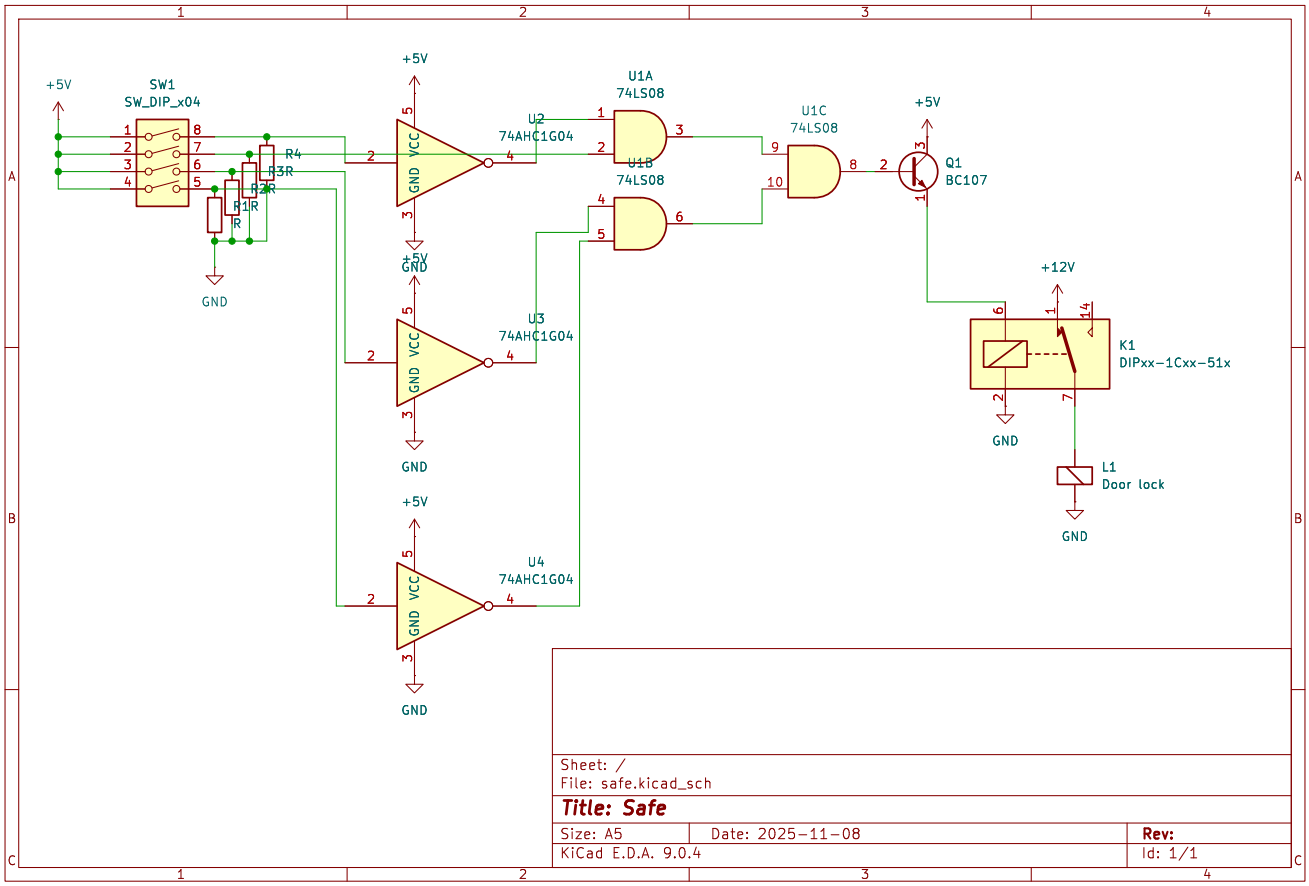


My finished safe

4.2.1 Overview

This is my first large electronics project. It isn't very complex by the standards of my other projects but it is on here because I had basically never done this sort of thing before. It is a homemade safe with a 4-bit code to unlock the electromagnet. I sawed the MDF sheets and screwed them together, before adding an electromagnet inside and some metal washers on the door. On top are the electronics, which consist of a 4.5V battery pack, a DIP (Dual Inline Package) switch, 4 AND gates, 6 NOT

gates, a transistor, a relay and a 12V battery.



A schematic diagram of how the electronics work

4.2.2 My Aim

I wanted to make a safe with a door that locked, and you could only open it if you know the code. I mostly achieved that goal, as there is some resistance to the door when it is locked, but none when unlocked. However, it isn't a very safe safe as the electromagnet isn't very strong so you can just pull the door open, or you could just unplug the batteries.

4.2.3 How it works

1. There are 4 switches, each of which is on or off,
2. The 1st, 3rd and 4th switches are inverted,
3. The outputs of the inverters and the 2nd switch are ANDed together.
   This means that the output of this is only on if the 1st input is off, the 2nd input is on, and the 3rd and 4th inputs are off.
4. The output of this switches a transistor to increase current,
5. The output of the transistor switches a relay to increase voltage,
6. The relay switches the electromagnet,
7. The electromagnet pulls of washers glued to the inside of the door to close it

4.2.4 Challenges

It was my first large DIY project so I had to learn to use a saw and a drill safely, as well as the electronics. The main problem I had with the electronics was that I had forgotten pull-down resistors, so the inputs were floating (no definite value) when the switches were off. I eventually Googled the problem and put the resistors in and it worked, but I didn't realise why until I started making my CPU.

4.2.5 What I would change

If I built it again now, I would change a few things:

- I would use a solenoid pushing into a hole instead of an electromagnet.
- I would use a PCB to stop people unplugging wires.
- I would place all of the electronics inside the safe, with only the switches on top.
- I would use more switches, e.g. a 10-bit code

5. Digital Making

5.1 Laptop

2023-2025 - Aged 12-14

5.1.1 Overview

I made this fully working miniature laptop from scratch. It uses a Raspberry Pi 4, a PiTFT touchscreen display, a miniature keyboard and a 3D printed case. It also has a rechargeable battery, but I haven't added an on/off switch yet so it needs to be plugged in. So far, it only boots to CLI but I am confident I can get it working to desktop.

5.1.2 Electronics

The battery is wrapped in duct tape to stop any possible shorts, then put in between the Pi and the screen. The screen is mounted directly on the Pi's GPIO pins, and has a small breakout of some of the pins. However, I had to bend these so the battery fit in. I am using a PowerBoost 1000C to connect the battery to the computer. The battery's power and ground lines are soldered onto the powerboost. Another pair of wires goes from the powerboost to some of the pins on the PiTFT to power it and the Pi. I also have a small speaker soldered to an amp, but don't yet have the amp's power or data lines connected and I need to add the power switch so I can turn it on and off. The keyboard is just a Rii mini wireless keyboard (https://thepihut.com/products/miniature-wireless-usb-keyboard-with-touchpad-1). All I had to do was plug the dongle into the Pi and it worked.

5.1.3 Software

The Raspberry Pi has the standard Raspberry Pi OS installed. I have tried to install the display drivers and they half work. It does display to the screen and is the right way up, but it only shows the console. When I use framebuffer tools such as `fbi` and `pygame`, it displays to the screen, so I can make keyboard-based Pygame games. However, as there is no X server running the mouse doesn't work. When I have another display hooked up to the HDMI ports, then it and the PiTFT have the exact same output until I run `startx`, then the PiTFT freezes, and only the HDMI display works until I end the X session.

5.1.4 Case

The case is 3D printed, and I found all the parts online here (https://www.thingiverse.com/thing:864547). I didn't have to do any CAD. However, because of the resolution of my printer, the screw threads didn't print properly, so I am holding the back on with tape. Also, the Pi can't be screwed into the holes so I have to design and print something to keep it from falling out.

5.1.5 Challenges

There were a number of challenges in making this project, some of which I haven't solved yet. The first one was that I used a PowerBoost 1000 instead of a PowerBoost 1000C, so I had no Micro USB charging port. I didn't even realise this until quite a long way into the project, when I tried to charge the battery and found I couldn't. I had to buy a new one, desolder/cut off wires from the old one and resolder the new one. The next (current) challenge is figuring out how to mirror the HDMI output to `/dev/fb1` which is the screen's framebuffer.

5.2 Light Fantastic

2023-2025 - Aged 12-14

5.2.1 Overview

The Light Fantastic is a project from the book Raspberry Pi Projects for Dummies (https://www.dummies.com/book/technology/computers/hardware/raspberry-pi/raspberry-pi-projects-for-dummies-281598/). It is a 4x4 RGB illuminated button matrix, so 16 individually addressable RGB lights under 16 buttons. In the book, they run a ribbon cable to a raspberry pi outside the box, but as the Pico has now been released, I am using that inside the box. It is now mostly working, but there are still 2 buttons that don't register presses.

5.2.2 How Was It Built?

1. I took a pre-made PCB and altered it
2. I soldered a bunch of LEDs and diodes
3. I attached wires around the different ports
4. I attached the wires to the Raspberry Pi Pico
5. I programmed it
6. I put it in a case

5.2.3 PCB Altering

First of all, I took the Sparkfun button pad pcb (https://www.sparkfun.com/button-pad-4x4-breakout-pcb.html) (target="_blank" rel="noopener"). This was designed for this sort of project. However, I was going to use Neopixels, not standard RGB LEDs, so it needed some modification. I cut a whole load of traces on the board to prevent short-circuits. I also accidentally cut a trace that I wasn't supposed to so I had to solder a wire across it.

5.2.4 LEDs and Diodes

This project needed 16 WS2812B LEDs for the illumination and 16 diodes to prevent crashing and possible damage when 2 or more buttons are pressed at once. I soldered the diodes first, but not very well as it was one of my first times soldering. I then did the LEDs, which was a bit better because I had got more experience in my break for a few months. They weren't much better, though, as the pins were much closer together so were much harder to do.

5.2.5 Wiring it Up

The first wires I did were the Din wire, through a resistor, to the first LED. I then did the Dout of the last LED of each row to the Din of the first LED on the next row. Next, I did the power and ground rails for the LEDs. I was then able to test them. I found that 2 of the Neopixels were just broken so we had to get some more, but after that it worked! I then had to do the 8 wires for the button matrix which went quite well.

5.2.6 Pico Connection

I didn't really feel confident soldering wires straight to the Pico, and I also might want to remove it and use it for something else later. My solution to this was to buy some 2.54mm pitch screw terminals and solder only the pins I needed. This worked really well and when I attached the wires to them they just worked.

5.2.7 Programming

I am programming in CircuitPython as I don't want to download a new firmware file for every update to the code. I made a main library that handles the actual interfacing with the board. It reveals functions to set LEDs to values based on either their index or xy coordinates, clear the LEDs, write changes, get buttons pressed, wait for a button to be pressed and get multiple button presses. I can then use that to make games for it. So far, I have only made a Lights Out game, where you have to turn all of the LEDs off. When you push a button, it inverts it and the 4 buttons it's touching. There are multiple levels, each one requiring more and more presses to win.

5.2.8 Case

It was quite fragile outside of a case, so I designed a case in TinkerCAD and 3D printed it. I made it in two parts so I could put the electronics inside, but I didn't think how to close it. Currently, the two parts are held together by tape.

5.2.9 Progression

This was one of the main defining projects for me, as it taught me a lot. When I started, I didn't really know how to solder or how electronics worked, and I could only program in simple Python. Now, I can solder quite well, have a deep understanding of how electric devices work, and I can program microcontrollers to do what I want.

5.2.10 Challenges

I did come across multiple challenges when making this. The first main one was when I cut the wrong PCB trace and had to solder a wire across it. Then the pins of the LEDs were too close for me to solder at that skill level, so I had to do other things for a long time before I was able to do it. I then had trouble working out how to attach the wires to the Pico, which I only answered in early 2025. Finally, in addition to games, I wanted it to be used as a sort of macro keyboard. This is why I used CircuitPython. When it turned on, it would try to connect to the computer to register itself as a keyboard. However, this mean it would crash when it isn't connected to a computer, for example if I just wanted to play a game on it. I couldn't debug this as it only crashed when it is *not* connected to a serial port. Currently, I am just adding games to it as this is the main aim of it anyway, but I will come back to this problem eventually. I reckon I will have to connect it to my Raspberry Pi via UART instead of USB to debug it.

5.3 Magic Mirror

2023-2025 - Aged 12-14

5.3.1 Details coming soon!

5.4 Wildlife Camera

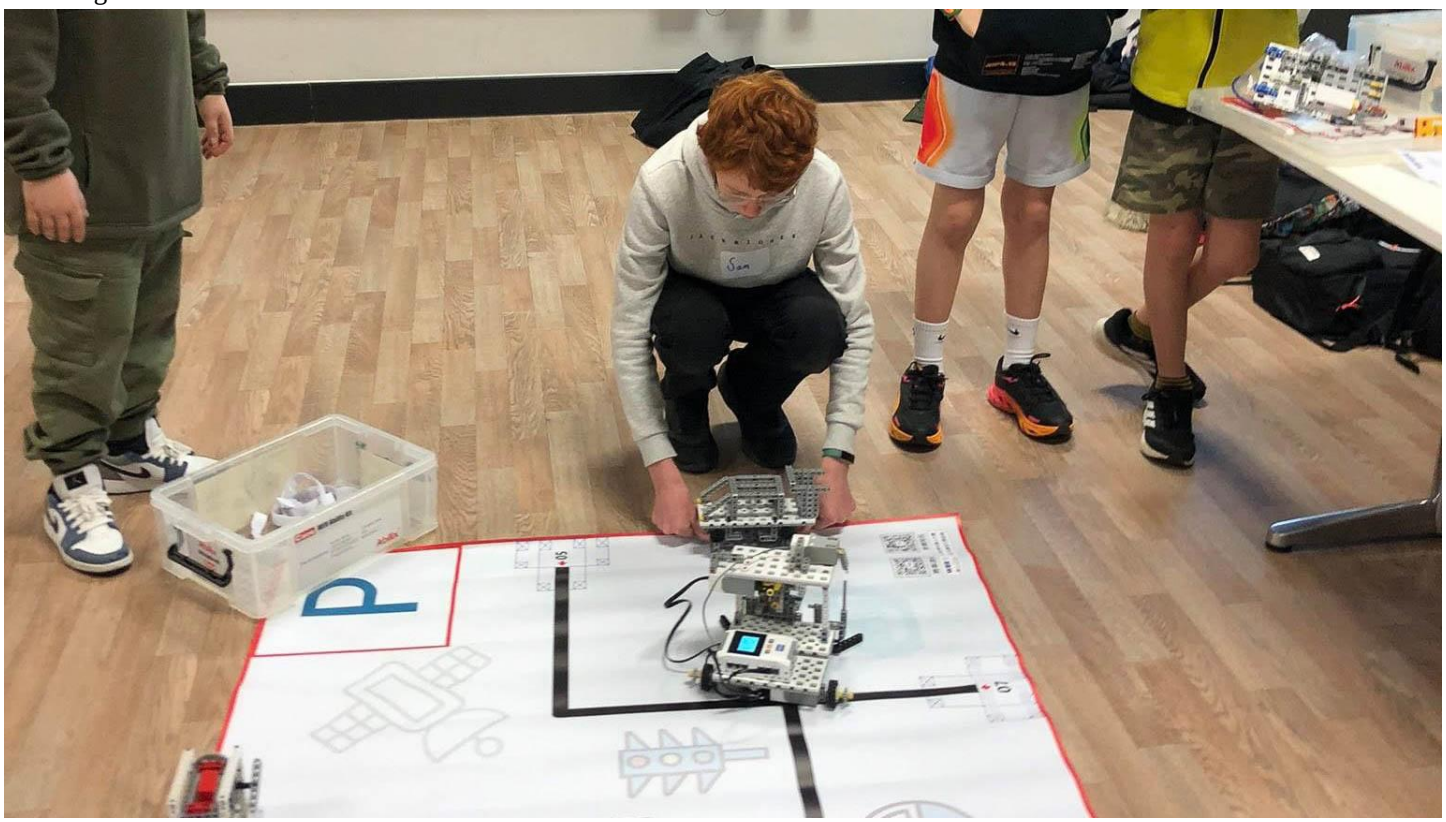
2022 - Aged 11

5.4.1 Details coming soon!

6. Robotics

6.1 World Educational Robotics 2025

2025 - Aged 14



Me testing my robot on one of the tasks

6.1.1 Overview

WER (http://wergame.org/) is an international educational robotics curriculum. There are many countries that host WER extra-curricular clubs with millions of participants, and some countries host robotics competitions. The most regular competitions take place in Britain every July and the winners from that competition qualify for the world championships in December in Shanghai. It is a large event, with over 6000 contestants in the championships (at least before Covid - there are fewer now). I joined in late December last year, and I went to the competition in England. I came 4th in my age category, but the top 3 all came from China to compete, so I qualified as the top British contestant and went to the world championships in Shanghai. The championships were on the 13th of December. It was a very stressful day, but in the end I got 3rd place!

6.1.2 About the competition

Each year, the competition has a different theme and map. This year was autonomous driving, and had pre-set tasks relating to the theme, e.g. Replacing the batteries in a model electric car and picking up containers to transport. You must compete in teams of 1 - 3. You get given a basic robot (plastic plate, 2 motors with wheels, 5 line sensors and a controller)

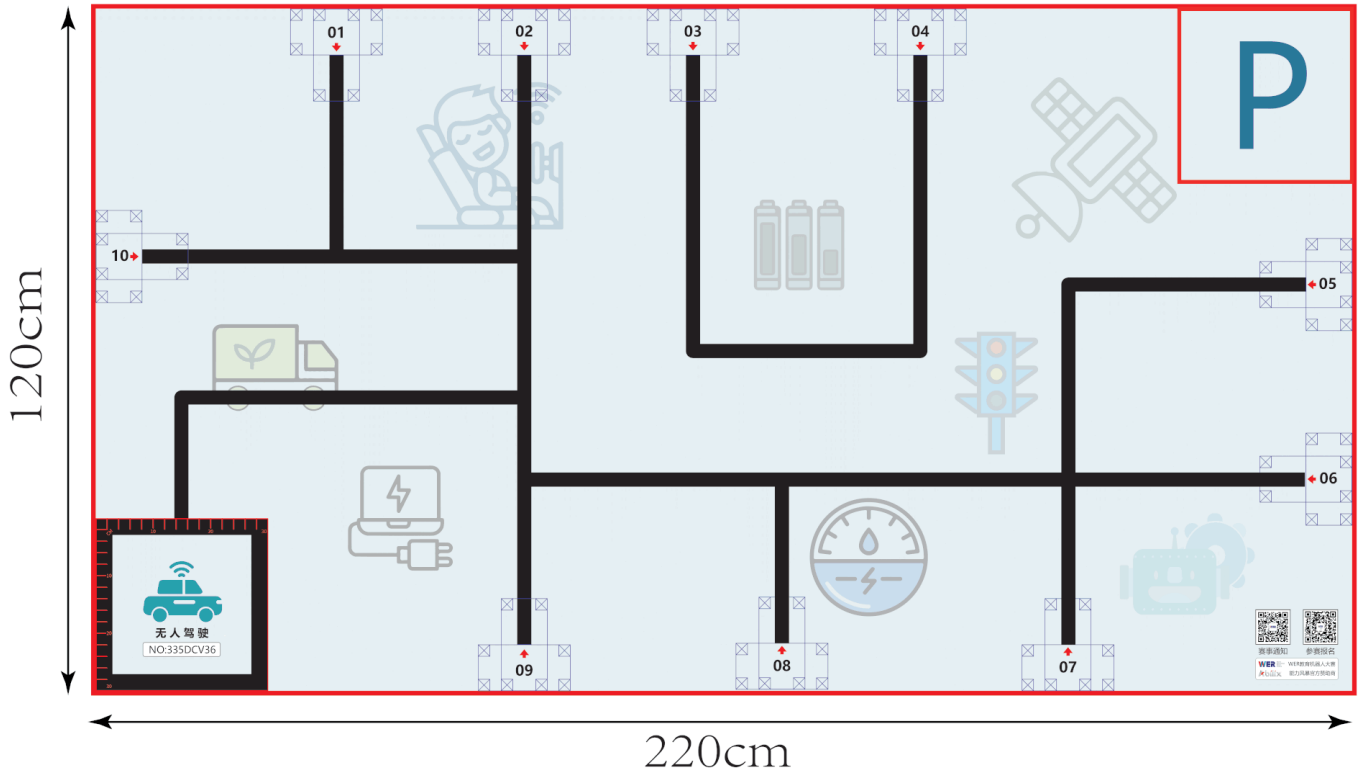
and must build additional things to complete the required tasks. It uses Abilix robots which have a meccano-style construction. You must also program the robot using either a version of Scratch or Python to move around the map and complete the required tasks.

6.1.3 This Year's Setup

The map was 120cm x 220cm, and it had a start box, 10 different positions and a parking space. The different ways of getting points were as follows:

- **Departure** *20pts*: Leave the start box at least once.
- **Task 1** *50pts*: Spin a bar around a certain number of times.
- **Task 2** *90pts*:
    1. *10pts x 3*: Remove 3 different coloured plastic blocks from a stand.
    2. *10pts x 3*: Take all 3 blocks back to the start box.
    3. *30pts*: Keep the blocks in the same order as they were on the stand.
- **Task 3** *60pts*:
    1. *30pts*: Spin a bar around to drop a red block on the ground.
    2. *30pts*: Catch the red block and take it back to the start box.
- **Task 4** *160pts*: This is a model of an electric car. It has a model of a dead battery in the bottom and a model of a charged battery in the top.
    1. *30pts*: Remove the dead battery from the car.
    2. *30pts*: Remove the charged battery from the car.
    3. *30pts*: Take the dead battery back to the start box.
    4. *30pts*: Take the charged battery back to the start box.
    5. *40pts*: Put the charged battery back where the dead battery was.
- **Parking** *60pts*:
    1. *30pts*: End the run with at least 1 wheel in the parking space.
    2. *30pts*: Have the red block from **Task 3** in the parking space at the end of your run.
- **Bonus points** *Up to 40pts*: You get 40 bonus points. Each time you have to pick the robot up outside the start box, you lose 10 of these bonus points, until you have none left.
- **Field tasks** *100pts x 3*: There are 3 unseen tasks that first appear at the competition. Each one is worth 100pts.

Each of the tasks could be in any position. In case you're interested, the full rules are here and this was the map:



6.1.4 Saturday Classes

I go to a club every Saturday for an hour and a half. This is where I do most of my progress, testing, and learning. Here I get to use one of the club's robots. Before I went to the competition, about half of these sessions were learning things such as calibration curves and how to line up neatly with the tasks, and the other half were just preparing for the competition. Since the British competition, I have needed all the time I can get to train and practise, but now that the championships are over, I can relax again.

6.1.5 British Competition

When I got to the competiton, I saw where the tasks were on the map and what the 2 new tasks were. I had 2 hours to get the robot to do all of the pre-set tasks in their locations and engineer solutions to the new tasks. Then we had a 3-minute judging session, lunch, and in the afternoon all of the tasks had been moved around. We had another 2 hours to prepare the

robots and code for the new positions, then a 3 minute judging and the awards ceremony.

I am being presented my award

6.1.6 Home programing

As well as preparing my code at the Saturday classes, I also do some at home. The main bit that I did was a few months ago where I wrote all of the code to get to all of the positions and back in a single week, then on Saturday I tested it and most of it worked or almost worked first time! It only took me about 2 hours of debugging to get all of it working. I have also done other programming and preparing at home, sych as thinking of a few different methods for task 4. Ultimately, I didn't actually use any of those methods, as I saw some other really good designs at the competition which I have taken inspiration from.

6.1.7 World Championship

Shortly after we arrived at the venue, we saw the 3 new tasks:

- You had to lift a bar entirely off a stand and drop it on the floor.
- I cant remember
- You had to lift a small plastic box up and place it on a higher platform.

In the opening ceremony, I had to read out the competitors oath (essentially saying that we won't cheat) in English. However, while I was waiting to go up on the stage, most other teams started writing code, testing and debugging early, so I lost about 40 minutes of preparation time compared to the other teams. I then managed to test everything, but I realised

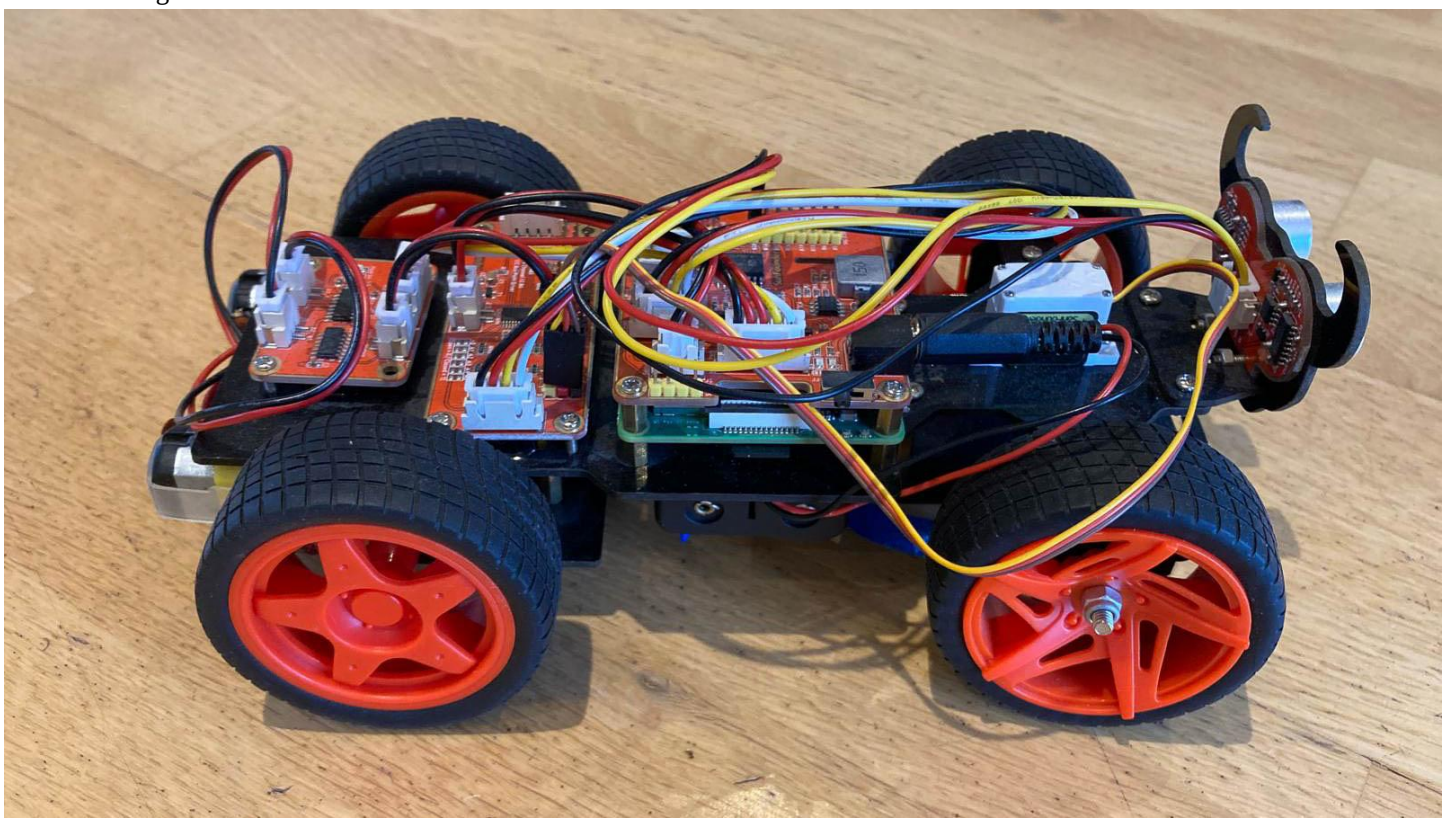
that I had built my solution to task 4 the wrong way round! My code to go to the parking space also stopped working. In the end, I managed to get 350 points in the morning, which was the highest of the entire British division. In the afternoon, I managed to do a bit more. I solved the first of the additional tasks, and got the red block to the parking space (although neither of my wheels were actually in the box). I think I got 440 points. It was then the awards and closing ceremony. I came 3rd! I was aiming for top 50, so I was shocked when they read my name out!

6.1.8 Challenges

Doing this had a lot of challenges, from my first sessions building the robot, starting to program it, learning how to use C++ with the robot, etc. One of the biggest (unexpected) challenges was when they completely changed the syntax of the function calls *3 weeks* before the competition! I had to spend a lot of time at home converting my 300 lines of code. Shortly after the competition, they changed from the old C202 robots to the new SK201 robots. This meant that I had to change all of my code from C++ to Python. However, the functions were very similar so it didn't take too long. The most challenging task model was task 4, because the dead battery was too low for a motor to reach, so I had to use a crank linkage with a claw on the end to move it. Also, you had to put the battery back in the low position so the claw had to be able to grab and release.

6.2 Picar S

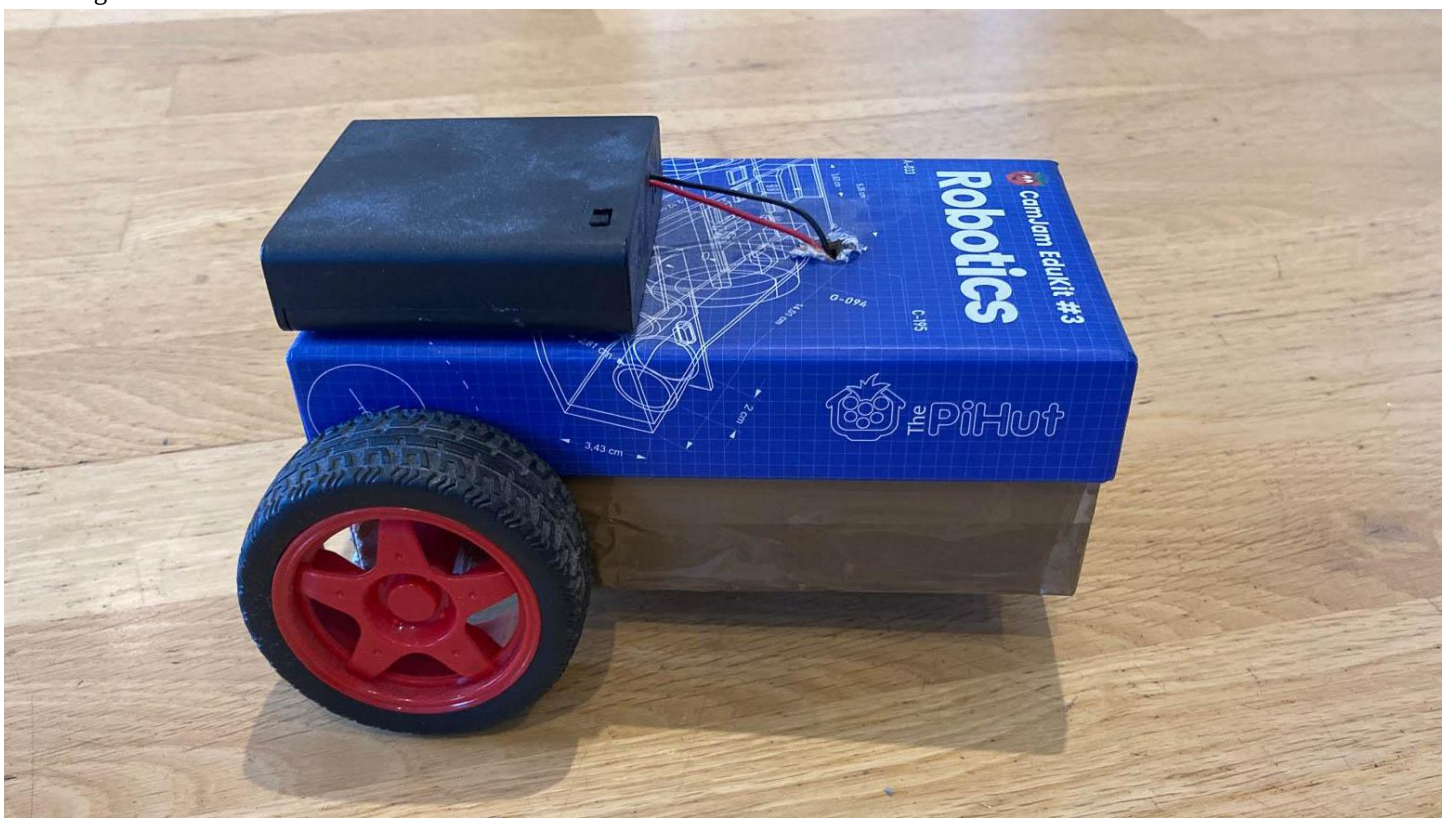
2023-2025 - Aged 12-14



6.2.1 Details coming soon!

6.3 My First Robot

2022 - Aged 11



6.3.1 Details coming soon!