School of Electronic Engineering and Computer Science

Final Report

Programme of study:

ELECTRICAL AND ELECTRONIC ENGINEERING

Project Title:

THE UNFORGETTABLE CUBE

Supervisor:

DR LORENZO JAMONE

Student Name:

NASMIN UDDIN

Final Year
Undergraduate Project 2021/22

Queen Mary
University of London

Date: 04/05/2022

Abstract

The aim of the project was to create an intelligent sensorized object that can detect and use tactile sensors as input. Over a full academic year, a tactile Rubik's cube was built and evaluated iteratively to improve further past design constraints and produce a more robust system. The cube- also known as *The Unforgettable Cube*- is an interactive object that uses tactile sensors to detect the user's finger presses after a series of tiles (in each face of the cube) light up in a particular order. Multiple design changes were implemented to improve the user experience. Firstly, only one face of the 2x2 Rubik's cube was used, to enable the user to hold the cube without triggering other sensors. Secondly, the PCB has been redesigned to be more efficiently connected and easier to debug, and a display has been added to enable the user to view and track the score. Finally, the plastic tiles (buttons) have been replaced with silicon tiles to increase the tactile sensor's sensitivity.

Contents

Chapter 1: **Introduction**

The *Unforgettable Cube* is a prototype of a 2x2 Rubik’s cube that uses four tactile sensors to detect the user input through the pressure imposed on each tile on one face of the cube. The change in the magnetic field between the magnets and the Hall-effect sensors determines the user input. The user then can receive feedback from the cube through the displays installed on another face of the cube. The game's purpose is to produce a series of patterns by lighting up each tile, and then tracking and waiting for an input response. The score and the high score are only displayed once the player presses the wrong tile.

This technology focuses on using one of the five primary human senses and creating an interactive game to improve memory skills. This report explores how the hardware and software aspects of the system were improved and implemented. The structure follows the background research conducted into sensorized tactile objects, the sensors used and the aims and specifications. It then details the reasoning behind the design and implementation of the software and the hardware aspects. This is followed by the documentation of all testing procedures conducted over the academic year and how these results impacted the final prototype. Finally, the report concludes with an extensive evaluation of the design and any improvements to the tactile cube.

1.1 Background

1.1.1 Previous Work

The unforgettable cube is a revised version of a previously implemented model. This project was previously developed by another student at the Queen Mary University of London. The shape emulated a 2x2 Rubik’s cube with twenty-four tiles in total, each of which was a plastic button with a tactile sensor inside. The cube displayed a pattern and showed the pressure needed to press each tile. Colours emitted by the LED lights specified the pressure sensitivity: red, yellow, and green [1]. Although the original prototype was functional, extensive documentation highlights design issues and areas for improvement.

Hardware aspects:

The circuitry fitted inside the hollow cube ensured the model was aesthetically pleasing. However, due to the limited internal capacity of the cube, components were not assembled neatly. Therefore, during assembly, parts were not secured or appropriately insulated, leading to malfunction in the system. Additionally, the heat emitted from the LEDs caused unreliable fluctuations in the readings for the magnetic field when the system was active for an extended period.

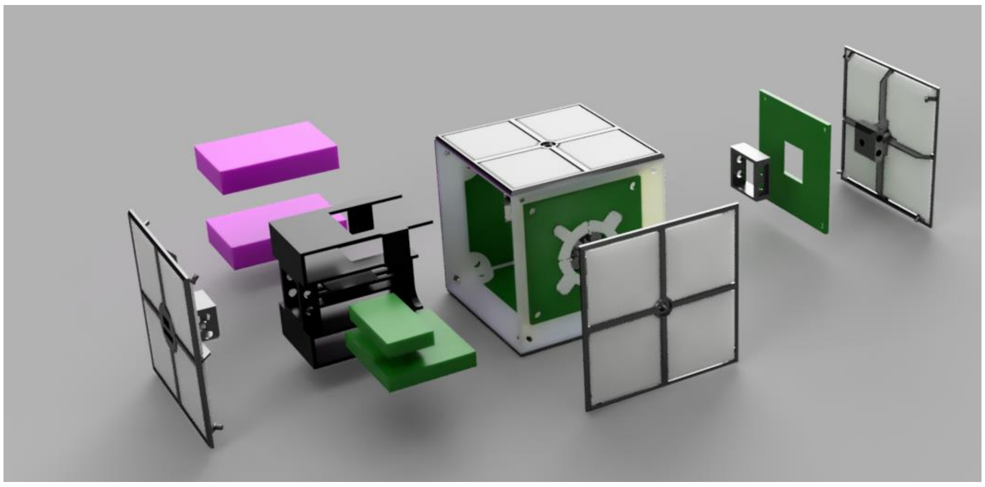


Figure 1 : Basic cube assembly visualisation [1]

Software aspects:

The code could be further refined by adding additional complexity to the game and creating a better user interface.

- Firstly, from a game perspective, adding an incrementing level of difficulty as the game progresses would ensure that the user is more engaged with the game [1]. Additionally, a tracking system for the number of trials would ensure that the user does not get stuck in a sequence forever if the wrong tiles are pressed repeatedly.
- Secondly, from a user interface viewpoint, displaying the score after each attempt would enable the user to keep track of the score. Also, it would be beneficial to allow the user to determine how long each tile should light up, which would enable each user to adapt the game according to their needs.

1.1.2 Tactile Sensors

One of the fundamental aims of this project is to integrate *tactile sensors* with a physical object to create an interactive system. The use of *tactile sensors* is a novel approach prevalent in robotics. Potential future uses of *tactile sensors* are in surgical equipment in the medical field to ease remote surgery, rehabilitation, aiding gadgets for visually impaired people, and haptic interactions for enhanced human-machine interaction [2].

The *tactile sensors* used in this project are composed of three components: a soft body (made out of silicone), a magnet, and a 1- axis Hall-effect sensor. This particular sensor is low cost, requires little maintenance, and is durable since only the silicon shell is directly affected by external factors [3]. A silicon mould encompasses the magnet within it; then, it is placed on top of the Hall-effect sensor. Consequently, as external pressure is exerted on the silicone shell, the distance between the magnet and the Hall-effect sensor changes; this causes a shift in the magnetic field. The Hall-effect sensor detects the magnetic field change, producing an input signal for the system. Although this method produces low cost “sensors that can measure the normal component of the applied force, with high sensitivity, low hysteresis and good repeatability” [3], there are a few factors to be considered:

- The silicone rigidity effects the functionality of the sensors. The stiffer the silicone encasing the more pressure has to be exerted on it.
- The consistency of the Hall-effect sensor measurement is highly dependent on the uniformity of the silicon moulds.
- Other environmental factors may affect the measurement of the magnetic field, such as the temperature, which can either weaken of strengthen the magnet's attractive forces *[1]*.

Therefore, few preventative measures can be taken to minimize these factors such as introducing ventilation ports or adding "an air gap between the silicone shell and the Hall-effect sensor can increase the sensitivity of the system" *[3]*.

1.1.3 Cognitive Training

Dementia, more notably Alzheimer's disease, is increasing rapidly, "the problem is so grave that the social cost for Alzheimer's disease is expected to reach approximately 200 billion USD by the year 2020" *[4].* Thus, causing significant concerns in today's society to explore cognitive training from an early age to reduce the risks. Although there is no clear correlation between cognitive training and the prevention of Dementia, studies have found that cognitive training does carry some benefits in delaying memory decline.

Between March 1998 to October 1999, a group of American researchers carried out a randomised, controlled, single-blind trial for healthy people between the ages of 65 and 94 to determine whether cognitive training does improve memory retention. Through this experiment, they learned that as the subjects continued with the exercises, their recall skills improved. However, as the study only followed up with the subjects for a brief time, no conclusion was able to be drawn for long term effects *[5].* In conjunction, a review carried out in 2016, "*Does Cognitive Training Prevent Cognitive Decline?*" *[6]* has corroborated insufficient data to suggest that cognitive training has a significant long-term impact on people with mild cognitive impairment (MCI). However, the study has found that individuals with normal cognition abilities benefit considerably from recall training with long-lasting effects *[6].*
Therefore, there is a growing demand amongst today's generation to introduce cognitive training games/tools for children to enable them to maintain and further develop their memory recall ability.

1.1.4 Criteria For Game Design

Developing a game is a straightforward process. However, the primary factor that distinguishes a mediocre game from a well-designed one is the ability of the product to keep the users constantly engaged and entertained. In an article published in 2016 -"an empirical study on engagement, flow and immersion in game-based learning"*[7]* some crucial findings were highlighted. In the paper, 174 test subjects played a puzzle-based game (Quantum Spectre), and the results were as follows:

- Making a game more challenging has a positive, and direct effect on the player's engagement level. Additionally, the player becomes more immersed in the game, and their perceived learning increases.
- The player's skill has a positive, and direct effect on their engagement level and game immersion. However, no significant effect is seen on the player's perceived learning.
- Increased engagement level is positively associated with perceived learning.
- Increased immersion in the game shows no significant effect on perceived learning *[7].*

From these results, it is evident that to design a successful game, the game needs to be challenging and engaging. Gradually increasing the game difficulty will allow the user to develop their skills, get used to the layout and improve upon it. Additionally, setting clear goals at each step, with a clean interface, will allow the user to visualise the next milestone and work towards it without overwhelming them.

1.2 Aim

The project aims to create a prototype of a 6-sided cube that simulates the look of a 2x2 Rubik's cube. One face of the cube will be fitted with four silicon tiles that measure the user input through touch. Another face will be used to display the score. An integrated program will allow the cube to light up each tile and display a series of patterns incrementing in difficulty. After each sequence, the player should be able to press each tile to recreate the chain and view the score displayed. Overall, this system aims to create an interactive game used for brain training exercises.

1.3 Objective

1.3.1 Requirements

The system requirements are subdivided into two categories: fundamental requirements and supplementary requirements. Fundamental requirements are the necessary implementations required to ensure the system works at a rudimentary level. Whereas supplementary requirements address additional functionalities that can be added to the model to make it more refined and complex.

Fundamental requirements:

- Detect button presses on tiles to enable the user to receive feedback.
- Light up each tile for at least 1s to enable the user to view the pattern.
- Set an appropriate LED brightness for the user to see through the material of the button layer.
- Allow the system to remember the pattern shown on the cube.
- Allow the system to recognise when the user has pressed the wrong tile.

Supplementary requirements:

- Display the score on a screen once the game ends.
- Increase game difficulty as the user improves, making the game more entertaining and stimulating.
- Change the speed at which each tile flashes as the user progresses through each level.
- Recognise when the user has pressed the wrong key and respond by ending the game and displaying the score on the display.
- Display each score in number format.
- Display a high score.
- Use silicon tiles to enable the cube to identify the pressure imposed on each tile to increase sensitivity.

1.3.2 Constraints

Several constraints have been identified during the various stages of the project plan. The following constraints listed below provide a brief description of how each limitation affected the prototype development and viable solutions.

Design:

The original prototype of the cube had all the sides fitted with tactile sensors; however, for the prototype outlined in this document, it seemed inadvisable to do so. This is because users would not be able to observe two parallel faces of the cube concurrently, making the game impossible to play. Additionally, removing the number of PCBs installed inside the cube would allow more room for other components, favouring the assembly process.

Limited experience:

A primary constraint was the limited experience with component assembly and integration of circuits. Thus, increasing the margin of error, which meant backup funds for components needed to be available. Therefore, this also meant more time was needed to research each component, software and machinery used to construct the various parts of the circuit.

Time management:

Time management was one of the most significant limitations of this project. As a full-time student with different deadlines, it was imperative to realise a feasible schedule to achieve the abovementioned requirements. Components needed to be pre-ordered at least a month in advance to guarantee that each circuitry is completed on time. Allocating a buffer time for each task would ensure that any unexpected malfunction can be fixed in time and a tangible prototype can be built by the end of the academic year.

Budget:

A limited budget allowed little room for error. The number of additional components used for the project needed to be used sparingly to ensure spare components were available in case of any malfunctions. Therefore, a decision was taken only to develop one face of the cube prototype with tactile sensors and carry out more thorough tests to aid future developments for a complete system.

1.3.3 Methodology

To achieve the abovementioned requirements from section **1.3.1** the implementation process has to be subdivided into three phases:
Phase 1 focuses on research and hardware. Hardware components often require a longer time to be delivered. Due to the time constraint, it was essential to start with the most tedious tasks to allow for a buffer time in case of any malfunctions. During this phase, the aims are as follows:

- Research previous projects and how tactile sensors work.
- Order the components required to build the circuit.
- Test PCB.
- Solder components to the PCB.
- 3D-print the physical cube.
- Create silicon moulds for the tiles.
- Test placement and integration of the circuits with the cube.
- Order the display screen.

Phase 2 primarily focuses on the software aspect of the system; by this stage, the basic cube is complete. Additionally, during this stage, more research is conducted on how to integrate additional components such as the 7-segment display with the cube, and any design changes are implemented:

- Implement the software for the memory game and consider/implement supplementary requirements for the game.
- Integrate the display.
- Redesign internal shelves for wiring and components.

Finally, in *phase 3*, no other functionalities are added to the game. During this stage, only design changes to the physical appearance of the cube are made. The process of hardware and software components integration occurs to create a cohesive system. This phase focuses on refining the system produced from phases 1 and 2:

- Integrate the software and the hardware aspects of the system.
- Improve the code.
- Improve/alter the design of the cube if required.

1.3.4 Project Schedule

The structure of the project schedule breakdown is shown below, it is used as guideline to ensure that the system can be fully built by the end of the academic year.

Phase 1:

TASK	DEADLINE	DEADLINE MET?	AMMENDED DEADLINE
Research	24/10/2021	✓	
Order/collect components	28/10/2021	✓	
Test PCB	29/10/2021	✓	
Solder components to PCB	19/11/2021	✗	24/11/2021
3D-print physical cube	19/12/2021	✓	
Create silicon moulds	06/12/2021	✗	03/03/2022
Evaluate the fit of the circuit inside the cube	23/12/2021	✓	
Order/collect display	24/01/2021	✗	07/04/2022

Phase 2:

TASK	DEADLINE	DEADLINE MET?	AMMENDED DEADLINE
Implement software requirements	10/03/2022	✓	
Integrate display	14/02/2022	✗	10/04/2022
Design internal shelves for wiring and components	25/02/2022	n/a	n/a

Phase 3:

TASK	DEADLINE	DEADLINE MET?	AMMENDED DEADLINE
Integrate software and hardware	12/03/2022	✓	
Refine code	24/03/2022	✗	10/04/2022
Make any design changes if required	15/04/2022	✓	

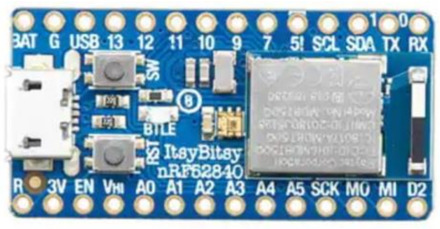
Chapter 2: **Implementation**

2.1 Hardware

In this section a detailed description of the hardware design, assembly and configuration of the cube is provided.

2.1.1 Component List

Figure 2.1 : Adafruit ItsyBitsy 32u4 - 3V 8MHz


BAT G USB 13 12 11 10 9 7 5! SCL SDA TX RX
R 3V EN VHI A0 A1 A2 A3 A4 A5 SCK MO MI D2
ItsyBitsy nRF52840

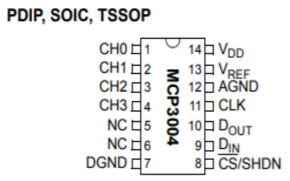
[8]

Description: 3.6cm long by 1.8cm wide Arduino compatible microcontroller with six power pins, six analog & digital pins and seventeen digital pins.

Figure 2.2 : 2.7V 4-channel 10-Bit A/D converter

[9]

Figure 2.3 : Pin configuration



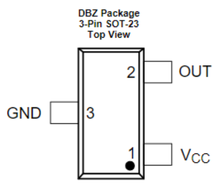
[10]

Description: An A/D converts an analogue signal into digital signal.

Figure 2.4 : DRV5053 Analog-Bipolar Hall Effect Sensor

Figure 2.5 : Pin configuration:

[11]



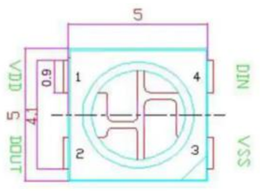
[12]

Description: A hall effect sensor detects the change in magnetic field.

Figure 2.6 : RGB SMD LED

[13]

Figure 2.7 : Pin configuration:



[14]

Description: RGB LED can produce any colour of different brightness with red, green, and blue (additive colours).

Figure 2.8 : Multilayer Ceramic Capacitors 0.01uF

[15]

Figure 2.9 : Multilayer Ceramic Capacitors 2.2uF

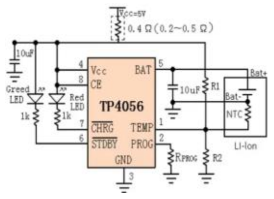
[16]

Description: A capacitor stores electrical energy.

Figure 2.10 : TP4056 5V 1A Micro-USB 18650 Lithium Battery Charging Board Module

[17]

Figure 2.11 : Pin connection:



[18]

Description: The TP4056 is a linear charger for single cell lithium-ion batteries [16].

Figure 2.12 : 3P SPDT Panel Mount Micro Slide Switch

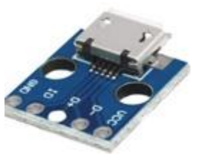
[19]

Description:

The switch allows the control of the current flow within a circuit

(e.g., system ON/OFF)

Figure 2.13 : Breakout Converter Module



[20]

Description:

Breakout Converter Module converts voltage into selectable voltages.

Figure 2.14 : Neodymium magnets

[21]

Description:

Size: 2mm x 1mm.

Figure 2.15 : SMD Chip Resistor 536 ohm

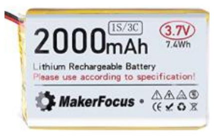
[22]

Description:

The resistor creates resistance to the flow of current.

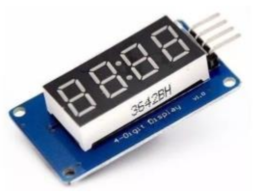
± 1%, 100 mW, 0603 [1608 Metric]

Figure 2.16 : 3.7V 2000mAh Lithium Rechargeable Batteries


2000mAh
1S/3C
3.7V
7.4Wh
Lithium Rechargeable Battery
Please use according to specification!
MakerFocus

[23]

Figure 2.17 : TM1637 4-digit 7-segment display



[24]

Figure 2.18 : Micro-USB charging cable

[25]

2.1.2 PCB Schematic

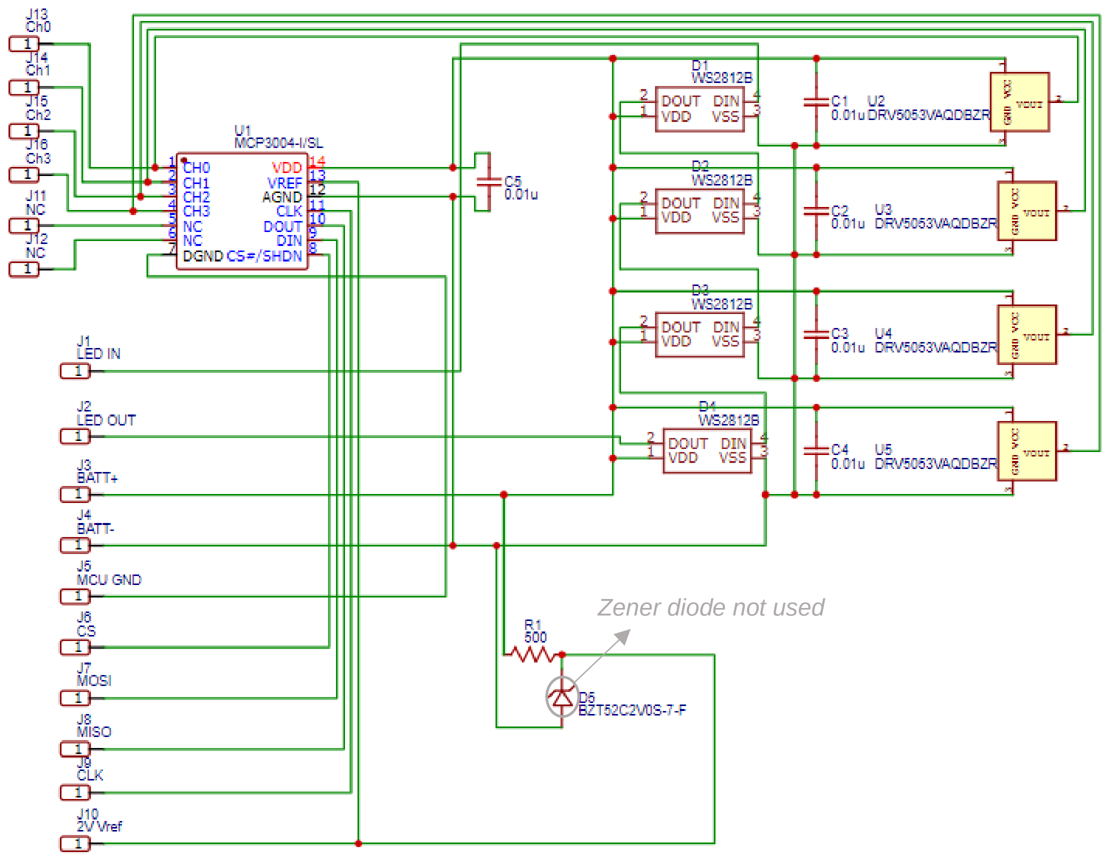


Figure 2.19 : PCB schematic [1]

The diagram above shows the schematic and the interconnection between all the components in the PCB:

- C1-C5 are 0.01uF Multilayer Ceramic Capacitors.
- R1 is 536-ohm resistor.
- D1-D4 are RGB SMD LEDs.
- J1-J16 are pin spacing connectors.
- U1 is a 4-channel 10-Bit A/D converter.
- U2-U5 are single-axis Hall-effect sensors.

2.1.3 PCB board and Connections

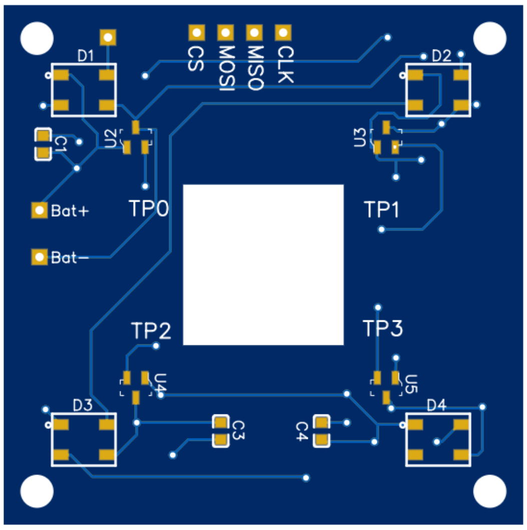


Figure 2.20 : 2D simulated view of the top layer (EasyEDA)

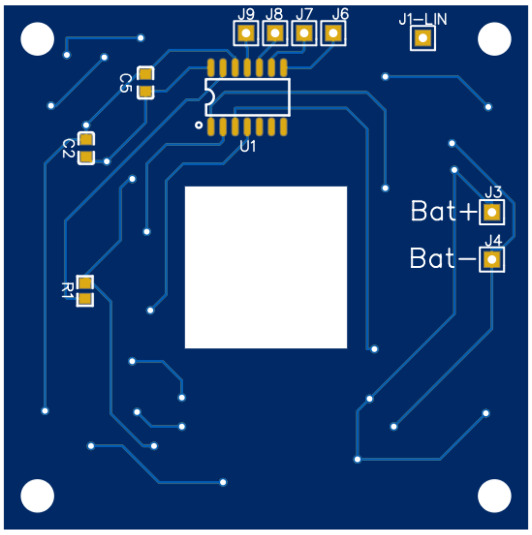


Figure 2.21 : 2D simulated view of the bottom layer (EasyEDA)

The size of the PCB is about 5.5cm x 5.5cm, and most components are less than 5mm in size. Therefore, a traditional soldering iron cannot be used - as it is simply too big; for the circuit assembly, a *nano soldering* station with *micro tweezers* is used, with a *digital microscope* monitor for vision. Alternatively, most PCB manufacturing companies often offer the option to have components soldered on one face of the prototype, as a result, the design above ensures that the least amount of manual soldering is needed. However, for this project, all the components have been soldered by hand.

Soldering Process:

- Place soldering paste on each conductive pads of the PCB.
- Use micro tweezers to heat up the component’s legs.
- Carefully place the component on top of the pads, aligning each leg to the corresponding pad (check orientation of the component).
- Allow for the solder to harden.
- Double check through a digital microscope monitor that each leg of the component is connected with the PCB track.

Figure 2.22 : JBC NASE-2C Nano Rework Station (Temperature Selection 90-450 ºC) [26]

2.1.4 Cube schematic

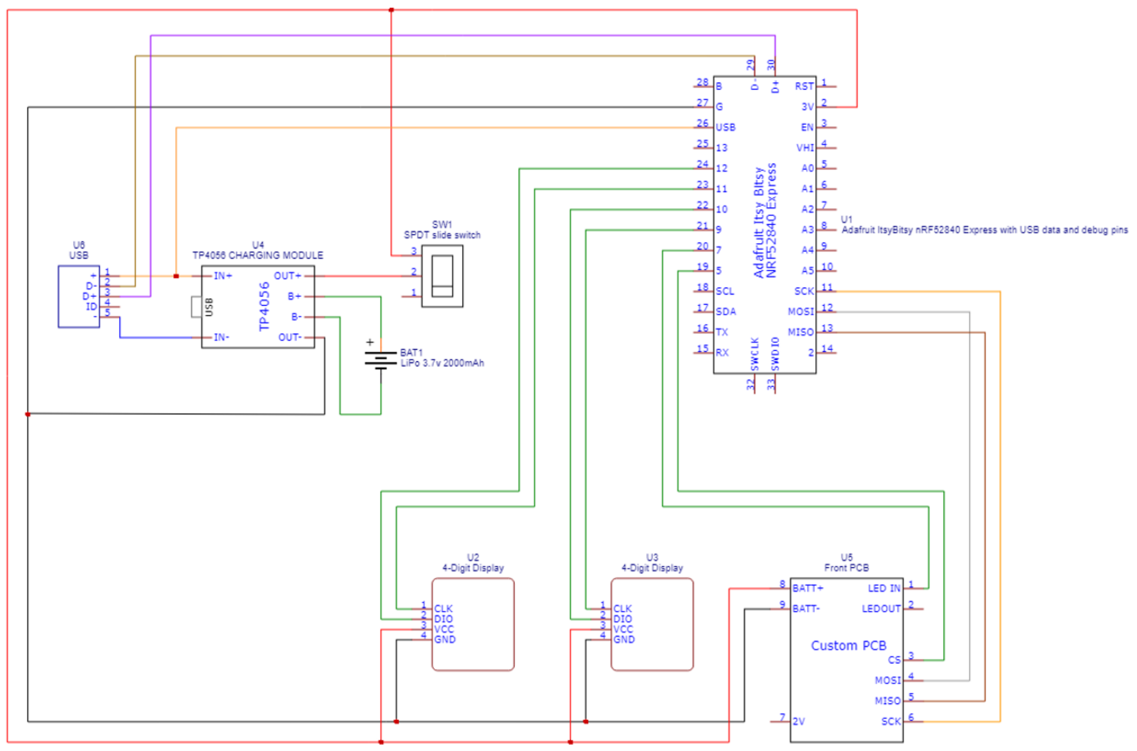


Figure 2.23 : Complete cube schematic

The diagram above shows the schematic of the cube assembly. The cube consists of one PCB and two 7-segment displays that connect to the Itsy-Bitsy microcontroller. Additionally, the schematic shows the connections for a rechargeable battery unit, however, due to time constraints and connection issues, this feature has not been implemented, instead, a micro-USB cable is used to power the MCU.

Pin connections of each component:

Pins on the PCB	Pins on the MCU
BAT+	3V
BAT-	G
SCK	SCK
MOSI	MOSI
MISO	MISO
CS	5

Pins on the displays	Pins on the MCU
VCC	3V
GND	G
CLK	9,11
DIO	10,12

Pins on TP4056	Pins on micro-USB	Pins on battery	Pins on slide switch	Pins on MCU
OUT+			2	
B+		POSITIVE		
B-		NEGATIVE		
OUT-				G
IN+	+			USB
IN-	-			
	D-			D-
	D+			D+
			3	3V

2.1.5 Silicon Moulds

Silicon moulds are optimal for this project due to their flexibility, enabling the tactile sensors to have a versatile sensitivity threshold. Therefore, the programmer is able to adjust the response reactiveness based on the appropriate requirement. For example, decreasing the sensitivity threshold ensures minimal pressure is required to trigger the system, therefore reducing the time needed to press each tile.

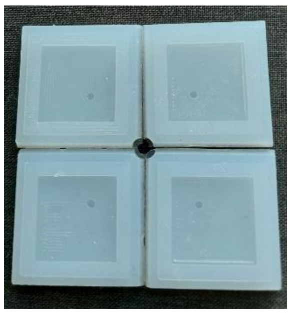


Figure 2.24 : Silicon buttons with designated magnet holders

The silicon moulds consist of two components: the lid and the base. The purpose of the lid is to apply pressure to the silicon while curing and minimise air bubbles by supplying additional silicon through small holes - in the lid - to fill the base.

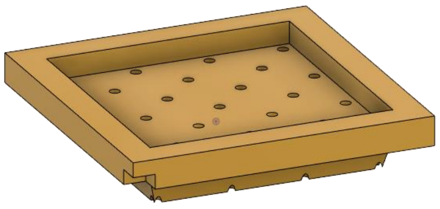


Figure 2.25 : Lid with heightened side walls for the reservoir to hold additional silicon

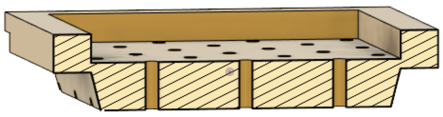


Figure 2.26: Lid with holes to allow excess silicon from the lid to flow into the base

The base holds the liquid silicon in place and allows it to mould into shape once cured. For this project, the button design includes a square-shaped indentation in the middle to replicate a controlled air gap between the magnet and the Hall-Effect sensor. This feature makes the buttons more sensitive while ensuring a bounce effect from each press, making the difference in the magnetic field more prominent. Additionally, the walls are slightly angled to create a locking mechanism between the cube skeleton and buttons - as the buttons are 1mm bigger - and require little to no adhesive.

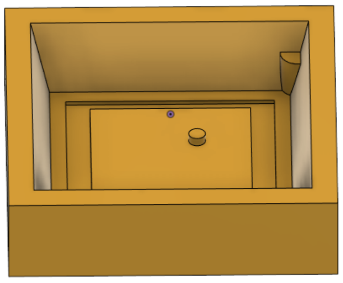


Figure 2.27: Base with squared shaped airgap and slanted inner walls

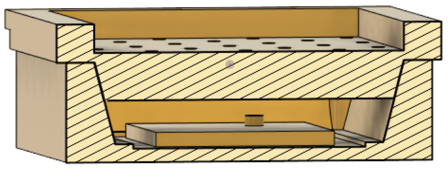


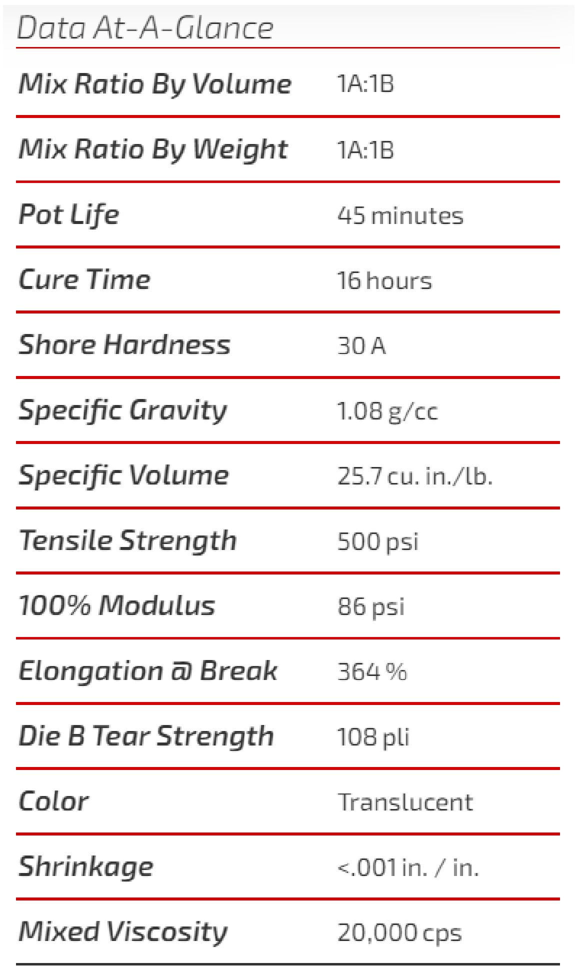
Figure 2.28 : Interior layout of the mould

Moulding process:

- The material used to create the silicon buttons is called Dragon Skin^{TM} 30 *[27]*, which comes with two solutions named Part A and Part B.
- Before coming into contact with the solutions, gloves should be worn to minimise contamination risk. “Wear vinyl gloves only. Latex gloves will inhibit the cure of the rubber” *[27]*.
- Pour equal ratios of Part A and Part B (1A:1B) into a container making sure that the solutions in Part A and Part B only come in contact in the mixing container. For four moulds, 40g of each liquid has been used.
- Add 8% of a thinner solution to make the mixture more viscous.
- Thoroughly mix for about three minutes.
- Place the mixture in a vacuum chamber for about 10 mins or until no bubbles are visible to get rid of any air bubbles.

- Take the mixture and carefully pour it onto the base of the mould, ensuring to pop any air bubbles with a sharp object.
- Place the lid on top. Some of the solutions will outpour to the sides and exit through the holes in the lid into the reservoir.
- Pour the remainder of the mixture on top of the lid in the reservoir, making sure to evenly fill it, as the excess silicon on top will be used for the display interface.
- Leave the moulds to cure in a dust free area for 16 hours or longer.

A detailed explanation of the moulding procedure and the handling of the product is available here.



Data At-A-Glance

Mix Ratio By Volume	1A:1B
Mix Ratio By Weight	1A:1B
Pot Life	45 minutes
Cure Time	16 hours
Shore Hardness	30 A
Specific Gravity	1.08 g/cc
Specific Volume	25.7 cu. in./lb.
Tensile Strength	500 psi
100% Modulus	86 psi
Elongation @ Break	364 %
Die B Tear Strength	108 pli
Color	Translucent
Shrinkage	<.001 in. / in.
Mixed Viscosity	20,000 cps

Figure 2.29 : table showing the specifications of Dragon Skin^{TM} 30 [27]

2.2 Software

In this section a detailed description of the software design and implementation of game logic is provided.

2.2.1 Software and MCU setup

The Adafruit ItsyBitsy nRF52840 Express MCU is Arduino compatible, therefore the Arduino IDE has been used to program it:

- To use the MCU, version 1.8 or higher of the Arduino IDE is required.
- Once the IDE is installed go to *File* ➔ *Preferences* ➔ *Settings.*
- Under the settings page in the *Additional Boards Manager URLs* box copy the following link :

https://adafruit.github.io/arduino-board-index/package_adafruit_index.json

- Press *OK* to save the preferences.

For more detailed instructions click here.

2.2.2 Game logic flow chart

2.2.2.1 Overall game:

The flowchart below illustrates the overall game implementation logic (the complete code is available in **Appendix 1**). Majority of the program implementation occurs in the *main loop*. In the first line, an array named *NumTab* is populated with the respective values of the digits 0 to 9 for displaying scores on the 7-segment display:

```
int8_t NumTab[] = {0,1,2,3,4,5,6,7,8,9}; // 0-9 config
```

In the following line, a counter variable *c* is decremented (`if (c>0) c--`); this counter ensures appropriate time allocation for each pattern. The variable *c* varies as the sequence length increases (*len_i*):

$$c = 1000 + 3000(level)$$

Following these two lines of code, a switch statement is implemented. The switch statement consists of two cases, case 0 and case 1.

In case 0:

- The score and the high score are displayed.
- The speed at which each pattern is shown, is set (*delayVal*).
- The number of fake flashes is determined (*fake_flashes=2*).
- The populate function is called to fill the sequence array.
- The respective RGB LEDs are lit up based on the sequence array values through the pattern function.

In case 1:

- The change is magnetic field is constantly read by the Hall-effect sensors. The fluctuation indicates a button press.
- The registered press is compared with the values held in the sequence array.
- If any of the presses do not match the respective value in the *sequence* array, the game resets (LEVEL-1) and returns to *case 0*. The score and the high score are displayed.
- If the press matches the element in the *sequence*, the next value is compared with the subsequent button press. This iterative loop continues for all the values in the array - or until the counter *c* is equal to zero.
- If the allocated time runs out (*c*), a different sequence of the same length is produced, and the abovementioned procedure is repeated.
- If all the correct tiles are pressed in the correct order within the time limit, a new longer sequence is generated, the *c* counter is set accordingly, and the *delayVal* is decreased. The pattern length will keep increasing as long as a correct succession of inputs is detected from the user.

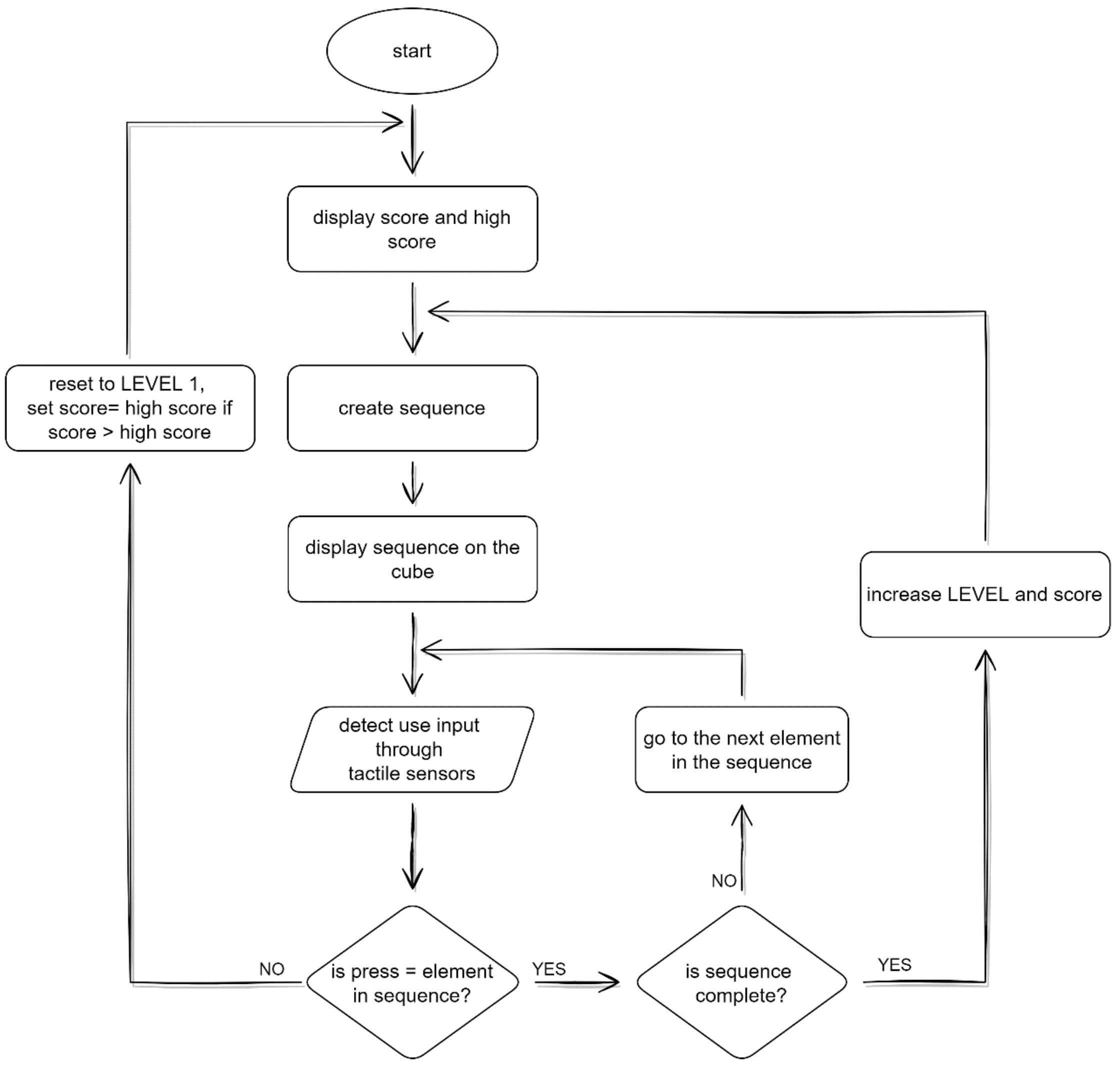

start
display score and high score
reset to LEVEL 1, set score= high score if score > high score
create sequence
display sequence on the cube
increase LEVEL and score
detect use input through tactile sensors
go to the next element in the sequence
NO
NO
is press = element in sequence?
YES
is sequence complete?
YES

Figure 2.30 : Flowchart illustrating the overall game logic

2.2.2.2 Populate function:

This *`void`* function generates random numbers between 0 and 3 and fills the elements in the *`sequence`* array, which hold the values that determine the tile order. The array can hold up to 50 integer values. However, the *`sequence`* is populated only up to the *`len_i`* position. The *`len_i`* variable is determined in the *`main loop`*. Additionally, through this function, the *`sequence`* array is printed to the serial monitor (for debugging purposes).

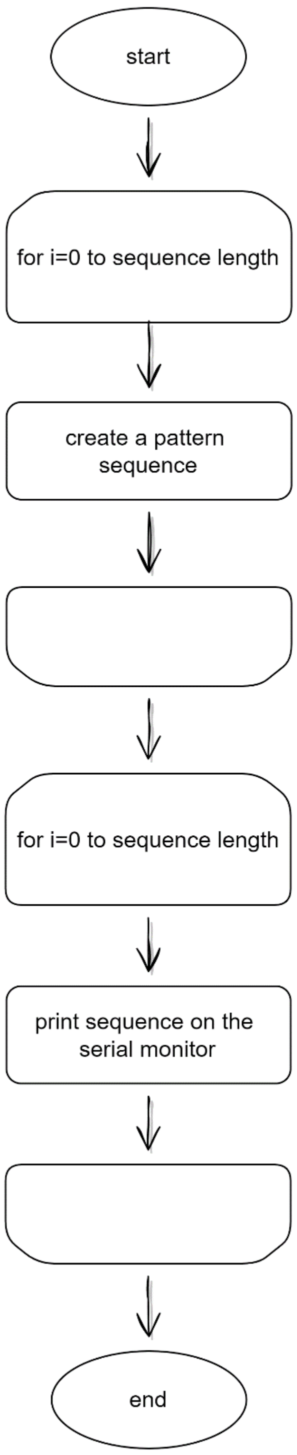

start
for i=0 to sequence length
create a pattern sequence
for i=0 to sequence length
print sequence on the serial monitor
end

Figure 2.31 : Flowchart illustrating the `void populate()` function

2.2.2.3 Pattern function:

This `void` function turns on the corresponding RGB LED based on the elements in the `sequence` array. This is to enable the player to view the pattern generated. The correct succession of tiles is shown by turning each LED the colour `blue`. The fake pattern is highlighted by lighting the LEDs `orange`. Each tile flashes for a designated time - determined by the `delayVal` variable. The `toss` variable randomises the position of the fake flash in the sequence. For this implementation, the probability of experiencing a fake flash in the pattern sequence is `1/3`. The total number of fake flashes per sequence is determined in the `main loop`. Once the pattern is shown, all the LEDs turn `magenta` for 200ms to prompt the user to provide input feedback to the system by pressing each button.

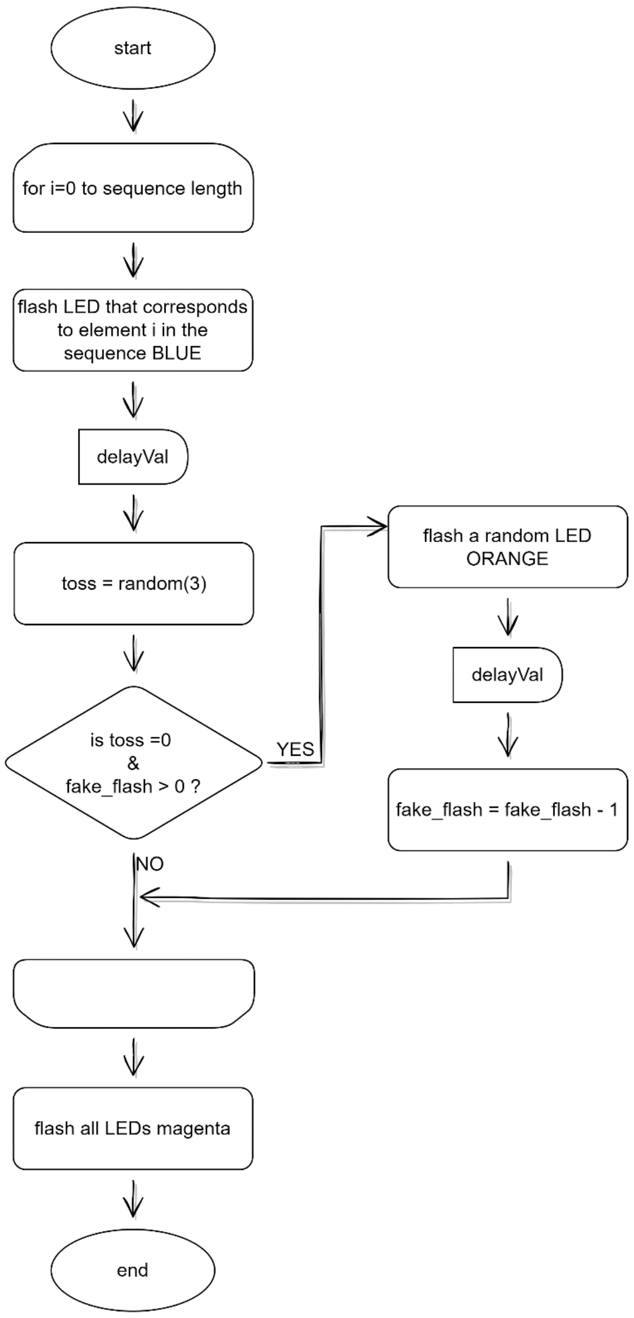

start
for i=0 to sequence length
flash LED that corresponds to element i in the sequence BLUE
delayVal
toss = random(3)
is toss =0 & fake_flash > 0 ?
YES
NO
flash a random LED ORANGE
delayVal
fake_flash = fake_flash - 1
flash all LEDs magenta
end

Figure 2.32 : Flowchart illustrating the `void pattern()` function

2.2.3 Progressively Increasing game difficulty

2.2.3.1 delayVal:

`delayVal` is a variable used to determine the speed at which each pattern is shown on the cube. This is achieved by determining the time required to toggle each LED. The following equation exponentially increases the speed at which each pattern is displayed:

$$delayVal = \frac{900}{2^{speed\_factor\,(len\_i-2)}} + 100$$

The game always starts with a pattern of two tiles (*len_i=2*); therefore, by substituting this value in the equation, it is evident that the first delay will be *1000ms*. This is the slowest delay factor at which the tiles will flash. The *speed factor* indicates the rate at which the exponential decay will occur - for this case, *speed_factor=0.5*. Increasing the *speed factor* means the graph converges to its asymptote - *100* - faster. As the length of the sequence increases, the delay (`delayVal`) decreases due to the growing denominator value of the equation, thus increasing the speed of the flashes. For example, this game reaches the maximum speed by approximately LEVEL-20 (*len_i=20*).

Pattern Length vs Delay graph

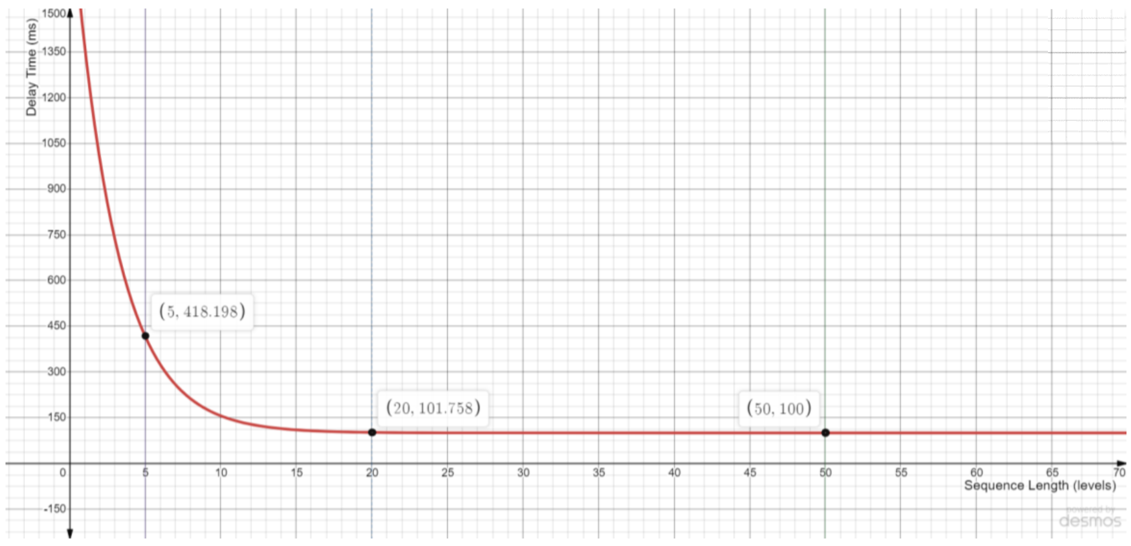


Figure 2.33 : The graph above demonstrates that as the pattern length increases the `delayVal` exponentially decreases

2.2.3.2 Fake flash:

The process of generating pseudo-flashes occurs in the `pattern` function. As stated in section **2.2.2.3,** the probability of experiencing a fake flash is 1/3 of the time; this is achieved by storying a random integer value between 0 and 2 in the `toss` variable whenever the program calls the `pattern` function. The `fake_flash` variable determines the maximum number of fake flashes; for this instance, the highest number of pseudo-tiles per sequence is 2.

If the `toss` value is zero and the number of `fake_flashes` is not exceeded, then a random RGB LED will light up the colour orange to indicate to the user that a pseudo-pattern has been introduced in the sequence.

Code to generate fake patterns within a sequence:

```
/* function for displaying the sequence on the cube and generating
fake flashes */
void pattern(){
  for(int i=0; i<len_i; i++) {
      .
      .
    //if fake flash, then turn on a random LED orange
    if(toss==0 && fake_flash>0){
                        // random tile              // Orange
      pixels.setPixelColor(random(4),pixels.Color(70, 30, 0));
      pixels.show();
      //determines the speed of the flash
      delay(delayVal); //pause before next pass-through loop
      pixels.clear();  //turn all LEDs off
      pixels.show();   //send the updated colour to the hardware
      //keep track of the number of fake flashes
      fake_flash=fake_flash-1;
    }
  }
      .
      .
}
```

2.2.3.3 Increase in pattern length:

For this prototype, the length of the pattern generated increases only if the previous sequence of tiles is pressed correctly. The base pattern consists of two tiles (LEVEL-1); once a wrong tile is pressed, the game restarts from LEVEL-1. Assuming the player continuously presses the correct tiles, the length of the pattern (len_i) will increment by one with each right sequence until LEVEL-48 is reached, with fifty tiles being lit consecutively in a single succession. The maximum number of tiles illuminated in a single series is fifty, as the array sequence can only hold a maximum of fifty integer elements.

2.2.4 Displaying scores

The system provides feedback to the user through two 7-segment displays, which disclose the score and the high score; the gradual incrementation of the score variable occurs every time a correct sequence of tiles is pressed. The high score is calculated by comparing the current score with the previous high score; if the current score is greater than the high score, the high score updates with the score. The system will only display the score if the player presses the wrong tile, thus ending the succession of correct answers. In `case 0`, once the score is displayed, it is reset to zero as the game restarts. In order to show the correct score format on the 7-segment display, a set of procedures have been applied. For example, if the player scores 135, the program will conduct the following calculations to display the score in the right unit columns:

Starting from right to left (`score =135`):

Shift the number to the correct position		Keep the integer part of the number		Isolate the correct digit of the number	position on 7-segment display (right to left)
				135%10=5	1st digit
135/10=13.5	➔	floor(13.5)=13	➔	13%10=3	2nd digit
135/100=1.35	➔	floor(1.35)=1	➔	1%10=1	3rd digit
135/1000=0.135	➔	floor(0.135)=0	➔	0%10=0	4th digit

The same procedure is repeated for the high score.

Code to show the score and the high score on the 7-segment display:

```
void loop() {
   int8_t NumTab[] = {0,1,2,3,4,5,6,7,8,9}; // 0-9 config
   if (c>0) c--; //decrement counter
   switch(state){
      case 0:
         high_score= max(score, high_score);
         //if game reset to level 1 show score and high score
         if(len_i==2){
            //print the units correctly on the displays
            dis1.display(3,NumTab[int((score))%10]);
            dis1.display(2,NumTab[int(floor(score/10))%10]);
            dis1.display(1,NumTab[int(floor(score/100))%10]);
            dis1.display(0,NumTab[int(floor(score/1000))%10]);
            dis2.display(3,NumTab[int((high_score))%10]);
            dis2.display(2,NumTab[int(floor(high_score/10))%10]);
            dis2.display(1,NumTab[int(floor(high_score/100))%10]);
            dis2.display(0,NumTab[int(floor(high_score/1000))%10]);
           …
```

2.3 Integrated System

2.3.1 Assembly:

The cube consists of two sides made of silicon: the top with the silicon buttons and the right side with the display.

The cube assembly process required the following steps:

- Use the plastic buttons to adorn the remaining four sides of the cube as they are purely decorative.
- On one side of the cube, ensure the cube skeleton with a gap in the middle is attached - to create a path for the USB cable.
- The silicon moulds serve a dual purpose, as described in section **2.1.5**. Use the additional silicon covers formed on top of the lid (in the reservoir) to create a silicon interface for the display; this will provide a clean and smooth interface that only reveals the scores while hiding the body of the 7-segment components.
- Align the 7-segment displays parallel to each other while ensuring to leave equal space on either side to make the display interface symmetric and aesthetically pleasing.
- Connect all the wires according to section **2.1.4** whilst covering the wires with a layer of electrical tape; electrical tape secures all the jumper cables to stay rigidly connected and provides a layer of insulation between components, preventing any short circuit.
- Screw the PCB in place with four M3x10mm screws and four M3 nuts.

A simple diagram of the placement of the components is shown below:

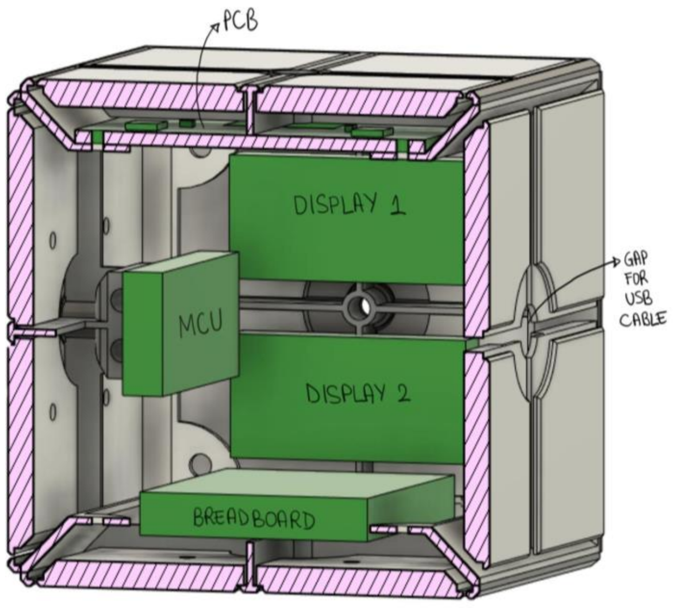


Figure 2.34 : Image illustrating the placement of the displays, breadboard and the MCU inside the cube skeleton.

2.3.2 Game rules

As described in the abovementioned sections, the sensorized cube uses a program - in the form of a game - to enable user interaction with the system. These are the instructions needed to understand and play the game:

1. Use the micro-USB cable to supply power to the cube.
2. Hold the cube facing the side with the silicon buttons.
3. Once power is on, the tiles will immediately light up blue. The first pattern consists of a two-tile sequence.
4. If a tile lights up orange, it indicates a fake tile that is not part of the sequence has been introduced. Therefore, avoid pressing it.
5. After the sequence is complete, all the tiles will flash in the colour magenta to indicate a user input is required.
6. The player will have limited time to press the correct sequence. If the timer expires, a new pattern will be generated.
7. After pressing each button, if the tile turns green, it indicates the tile is part of the sequence shown. However, if the tile turns red, it signifies the tile is not part of the sequence.
8. If the user input is wrong, the score and high score will be displayed, and the game will reset.
9. If the user input is correct, the game will continue, and a new pattern with a higher sequence length will be displayed. Steps 3 to 9 will be repeated while incrementing the score with each level. Additionally, with each progression of levels, the game will get increasingly more challenging, as the sequence will be shown for a shorter period.
10. The game only displays a maximum of 50 tiles per sequence.
11. The game continues forever unless the user presses the wrong tile.
12. The game will reset if the power supply is removed.

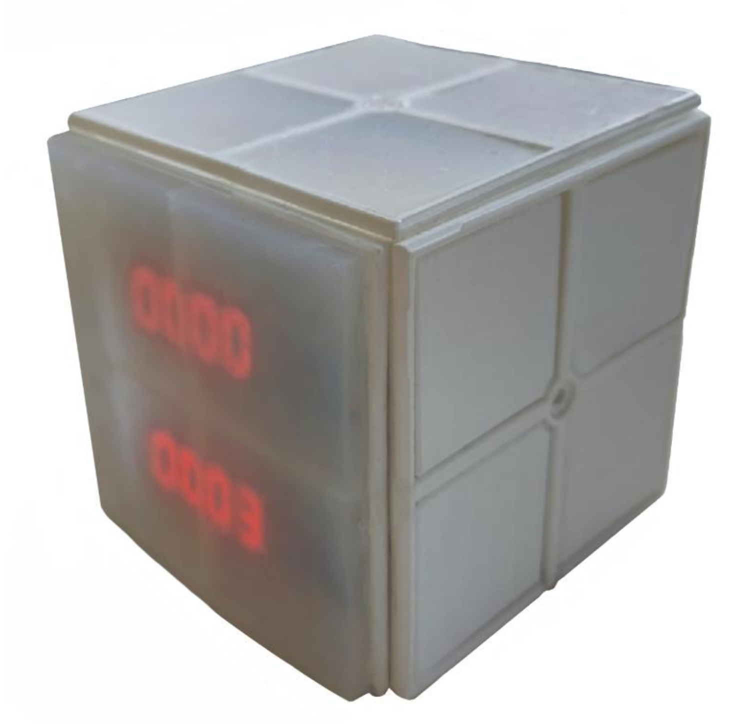


Figure 2.35 : Image of the assembled "Unforgettable Cube" with tactile sensors with silicon buttons on the top face and display on the left face

Chapter 3: **Design and Testing**

This section provides detailed documentation of all the testing procedures and malfunctions that occurred during the implementation process.

3.1 Hardware

3.1.1 PCB

The PCB used for this project is a prototype and not industrially produced; therefore, copper track residue was present (as shown in *Figure 3.1*). Extensive testing of the tracks was necessary to ensure the proper connection of paths without any bridging. The procedure was very vital to prevent short-circuiting the PCB. Initially, a visual inspection of the PCB can help to detect any obvious bridging. However, it is crucial to do a continuity test with a *digital multimeter* to determine all bridging causes.

Testing procedure:

- Use a multimeter with milliohm sensitivity.
- Set the multimeter to the continuity test mode with the buzzer (noise) function.
- Insert black lead into the COM jack.
- Insert red lead into the VΩ jack.
- Connect the test leads across each path and check for any bridging.
- The digital multimeter will produce noise if the two paths are connected, as the resistance will be low, which indicates bridging.

Results:

After completing the test, several path overlaps were highlighted. The overlapped areas were targeted with the help of a scalpel by making an incision to separate the connecting pieces. Additionally, a broken track was detected (*Figure 3.2*), which meant only one LED worked. Initially, the issue was resolved by soldering a small wire across the broken paths to connect them. However, a better solution was achieved in section **3.1.1.2**

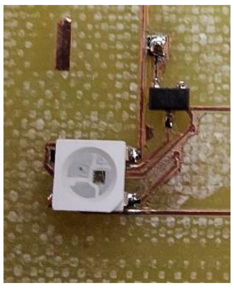


Figure 3.1: Image of copper residues across the tracks

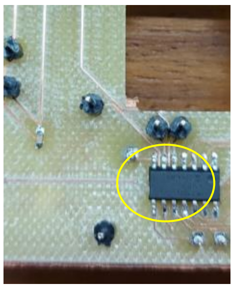


Figure 3.2: Yellow circle highlighting a broken track

3.1.1.1 Unexpected Malfunction

A sudden malfunction occurred on 19/11/2021. One LED component was not soldered securely enough, which led the component to shift slightly off the PCB connector pads and caused interference between two parallel paths. Therefore, the LED module had to be removed from the PCB board.
Components on PCBs cannot be desoldered with the traditional method. This is due to their size and the thickness of the track. Hence, a *hot air rework* station was used to melt the LED off the board and replace it with a new one.

Procedure:

- Isolate the LED from the rest of the components by a heat resistant funnel.
- Apply heat to the LED until the solder underneath melts (around 350°C).

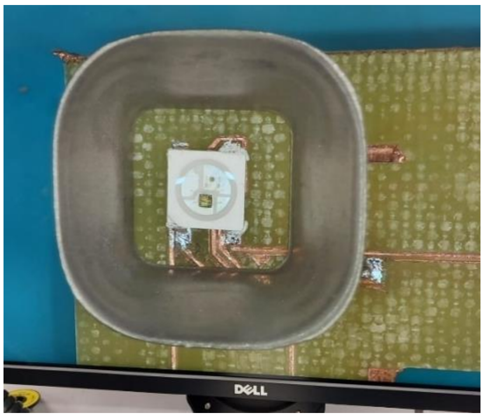


Figure 3.3 : Heat resistant funnel isolating the LED component

3.1.1.2 PCB redesign:

The PCB redesign was prompted by the numerous copper track problems, which included short circuits and partially broken tracks, causing the PCB to malfunction, and output unreliable results. The cause of all the issues was the inefficient design of the PCB. Tracks were remarkably close together, increasing the likelihood of short circuits and connections between the copper layers being too small (via size=0.3mm) meant the presence of disrupted connections between the front and the back of the PCB. These factors made the debugging process of the PCB time-consuming and tedious. Therefore, a decision was made to redesign the entire system layout. In the images below, all the issues with the old PCB are highlighted with a yellow ring around them. Figure 3.4 shows that every track connected to or surrounding the Hall-effect sensors was compromised, indicating a high margin for error during the assembly and manufacturing process. Additionally, in figure 3.4, it can be observed that copper tracks are positioned underneath the LED pads, therefore, making the debugging process after assembly inefficient and costly - because the LEDs would need to be desoldered, (as discussed in section **3.1.1.1)**. Figure 3.5 highlights the numerous broken/short-circuited track issues that arose while fracturing the prototype.

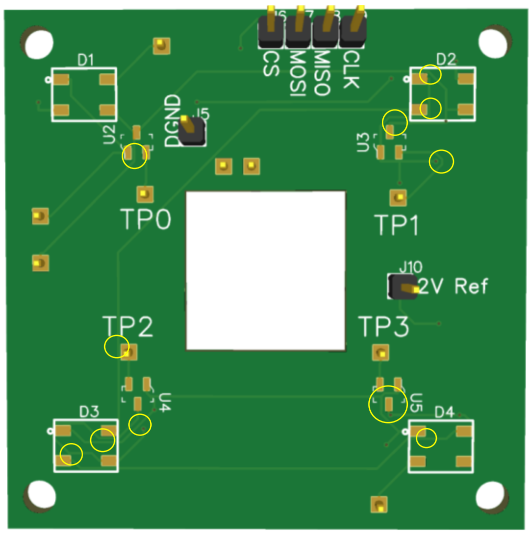


Figure 3.4 : 3D simulated front view of the earlier design (EasyEDA)

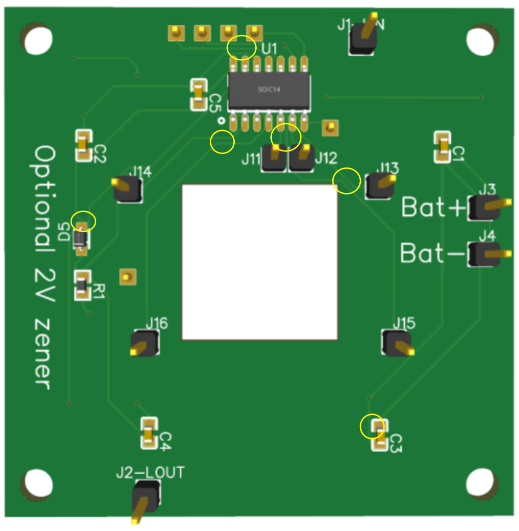


Figure 3.5 : 3D simulated back view of the earlier design (EasyEDA)

In the new PCB design, the via holes to connect the copper layers were increased from 0.3mm to 0.6mm, and line widths were set to 0.254mm to minimise the risks of having broken tracks. Rewiring all the connections ensured sufficient space between modules to avoid short circuits. Additionally, unnecessary pin connections were removed to create more room for the copper tracks. Components such as the capacitors have been moved to the top layer of the PCB to make the soldering process more manageable and convenient. As PCB manufacturing companies often offer to solder one side of the PCB face. The software used to create the new PCB is called EasyEDA [28].

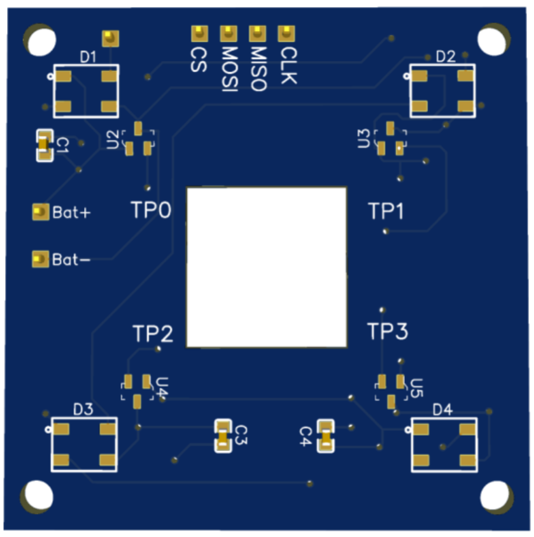


Figure 3.6 : 3D simulated top view of the new design (EasyEDA)

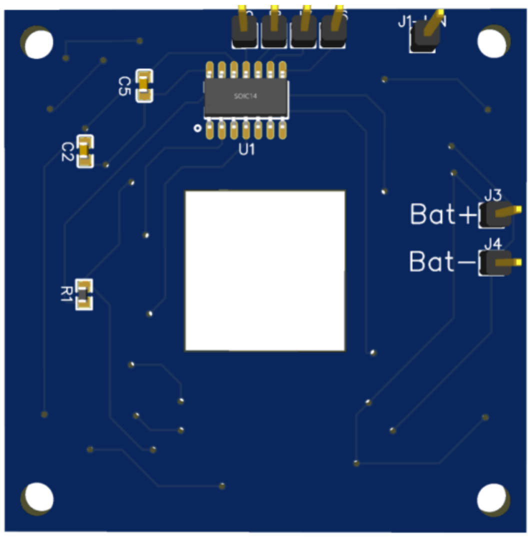


Figure 3.7 : 3D simulated bottom view of the new design (EasyEDA)

3.1.2 Silicon button design:

The moulds for the silicon buttons were designed with Fusion 360 [29], a cloud-based 3D modelling software. The initial mould model closely emulated the plastic buttons. However, this turned out to be an impractical design due to the small size of the silicon buttons, which sunk inside the cube skeleton and touched the hall effect sensors, making the change in the magnetic field hard to detect. In order to counteract the highlighted issues, a new design with a 1.3 times larger and thicker button compared to the original prototype was developed. Additionally, introducing an air gap ensured fluctuations in the magnetic field were more pronounced. The air gap between the magnet and the Hall-effect sensor allows the magnet in the silicon button to bounce when pressure is imposed on it and removed, therefore creating a more significant fluctuation in the Hall-effect sensor's readings, increasing the sensitivity of the tactile sensor. The two images below showcase the difference between the initial design and the final button used for the cube. (An in-depth explanation of the final mould design can be found in section **2.1.5**)

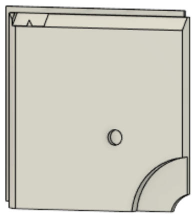


Figure 3.8 : Initial button design

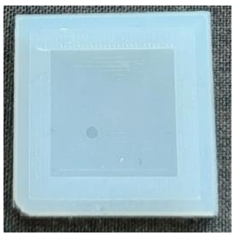


Figure 3.9 : Refined button design with air gap and thicker walls

3.1.3 Wire testing

To connect the components in the system jumper wires were used. These types of cables are very versatile and easy to use. However, due to mass production, they are not as dependable. Therefore, performing the following tests guaranteed wires used in the system were robust: Firstly, through visual inspection, wires with damaged insulation sleeves were detected and discarded, as exposed copper might interfere with other components. Secondly, a continuity test was conducted on the wires to check for damaged cables (an in-depth description of the continuity test is available in section **3.1.1**). Finally, assessing the grip strength of each wire prevented loose wires from being incorporated into the system. Additionally, a further layer of electrical tape was wrapped around the wires once connected to the components as a precautionary measure.

3.1.4 Discarded component – MPU6050 module

Initially, after completing the rudimentary requirements of the system, an attempt was made to integrate the MPU6050 component into the cube design. The MPU6050 module is a 6-axis motion tracking device that uses a 3-axis accelerometer and a 3-axis gyroscope. However, the design was discarded due to the lack of ideas on how to implement this feature into the system to enhance the user experience. The initial idea was to implement the motion detection element to allow the user to change the colour scheme of the game layout.

3.2 Software

3.2.1 RGB LED testing

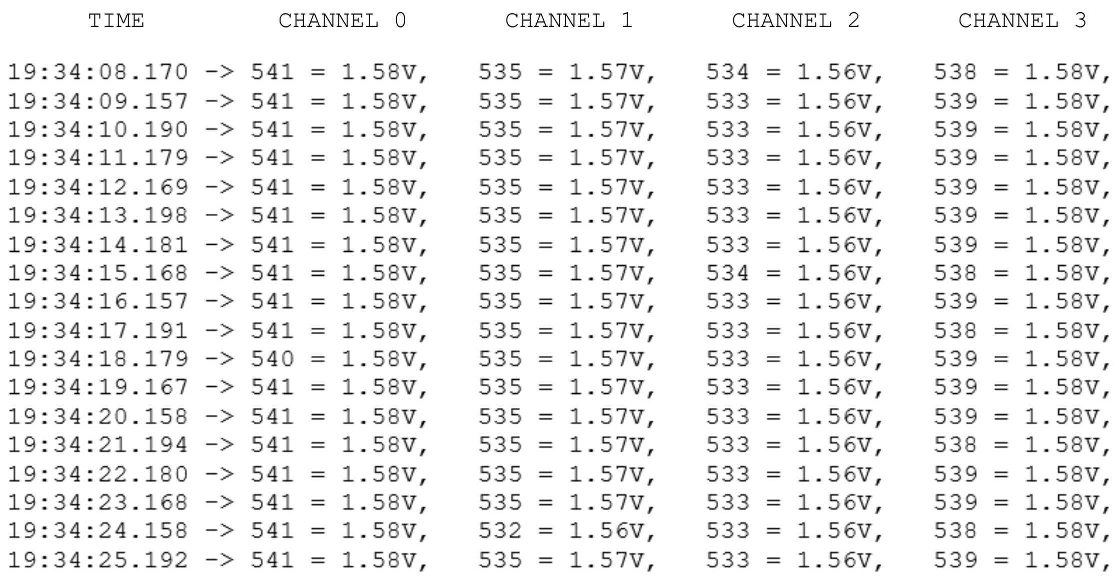
To test the LEDs an example code from the *`Adafruit_NeoPixel.h`* library has been used. This specific code ensures that all the LEDs connected to each channel of the 4-channel A/D converter light up and can be controlled through the converter module. The testing code can be found in section **8.1**. Once the testing code is uploaded, all the LEDs should sequentially turn on, highlighting any wiring or component issue.

3.2.2 Hall-effect sensor testing

The code to assess the Hall-effect sensors is available under section **8.2**. To communicate with the Hall-effect sensors the *`Adafruit_MCP3008.h`* library is needed. In order to check each Hall-effect sensor, the following line of code is used:

```
//adc is an object of Adafruit_MCP3008 and i is the channel number (0 to 3)
adc_Reading = adc.readADC(i);
```

In this test code, each Hall-effect sensor is constantly read and outputted in the serial monitor to determine the base value for each sensor. By conducting a continuity test on each Hall-effect module, it is observed that around *1.6V* passes through each channel (which is approximately half of the *3.3V* supplied by the MCU), therefore, the expected digital base value read in each channel should be around half of *1024*. In the table below, the readings produced by each Hall-effect sensor is displayed:



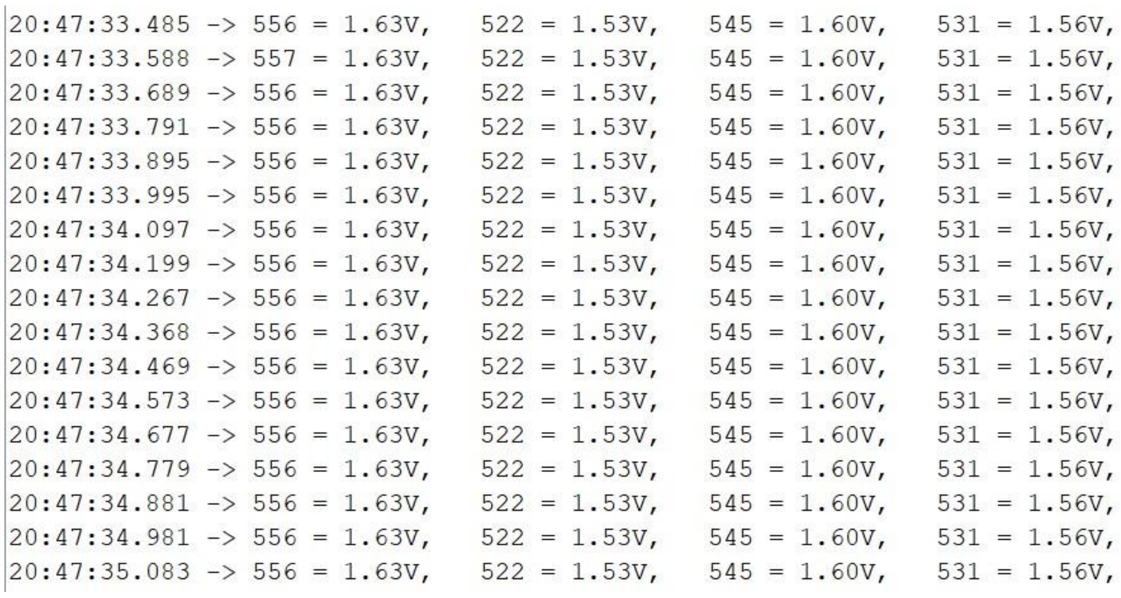
```
     TIME          CHANNEL 0        CHANNEL 1        CHANNEL 2        CHANNEL 3

19:34:08.170 -> 541 = 1.58V,    535 = 1.57V,    534 = 1.56V,    538 = 1.58V,
19:34:09.157 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:10.190 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:11.179 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:12.169 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:13.198 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:14.181 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:15.168 -> 541 = 1.58V,    535 = 1.57V,    534 = 1.56V,    538 = 1.58V,
19:34:16.157 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:17.191 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    538 = 1.58V,
19:34:18.179 -> 540 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:19.167 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:20.158 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:21.194 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    538 = 1.58V,
19:34:22.180 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:23.168 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
19:34:24.158 -> 541 = 1.58V,    532 = 1.56V,    533 = 1.56V,    538 = 1.58V,
19:34:25.192 -> 541 = 1.58V,    535 = 1.57V,    533 = 1.56V,    539 = 1.58V,
```

Figure 3.10 : Hall-effect sensor base readings without magnetic buttons

Consequently, the same code can be used to determine the base value of each sensor once the buttons have been placed on top of the sensors to determine the sensitivity threshold (upper bound threshold = base value + 25, lower bound threshold = base value - 25):

```
       TIME            CHANNEL 0         CHANNEL 1         CHANNEL 2         CHANNEL 3
```



```
20:47:33.485 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:33.588 -> 557 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:33.689 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:33.791 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:33.895 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:33.995 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.097 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.199 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.267 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.368 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.469 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.573 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.677 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.779 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.881 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:34.981 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
20:47:35.083 -> 556 = 1.63V,    522 = 1.53V,    545 = 1.60V,    531 = 1.56V,
```

Figure 3.11 : Hall-effect sensor base readings with magnetic buttons

3.2.3 Screen testing

In order to assess the 7-segment display the `TM1637.h` library is required. Through this library each digit of the 4-digit display can be accessed. The simple test code can be found under section **8.3**. the code counts from *0* to *9999* and displays the increments. This test is to understand the configuration of each 7-segment digit and check if any segment is damaged. This display type uses 7 individual segments that turn ON and OFF in different orders to portray numbers and letters. However, due to the *TM1637.h* library, these values don't have to be individually initialised by setting each segment separately.

Figure 3.12 : Image illustrating a 4-digit 7-segmet display

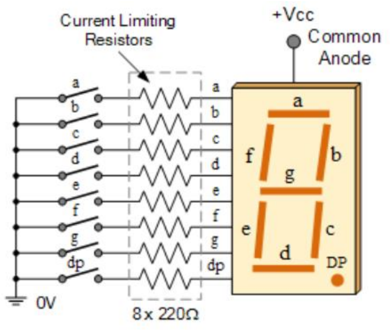


Figure 3.13 : Image of a 7-segment display schematic [30]

3.2.4 Pseudo- random values

An integral part of the game is the random reproduction of patterns. However, true randomness is still part of a philosophical debate, where some mathematicians and scientists believe it is achievable in the quantum state. Therefore, a suitable alternative to make pseudorandom numbers appear random to the human eye is by using a seed function to manipulate the pseudorandom number generator. Before the application of the seed function, whenever the cube was turned on the same sequence of tiles would be displayed. A seed function uses the noise detected by unconnected pins from their surrounding environment to feed the pseudorandom number generators and produce varying starting values each time the generator functions reset. Ensuring varying -yet limited in the long term- sequences of values are produced. This enables the player to perceive as though truly random patterns of tiles are being generated, preventing the system from displaying identical sequences noticeable to the user.

For this project, the unconnected `pin 13` was used in the seed function:

```
void setup() {
        .
        .
    /* initialize the pseudo-random number generator by reading
      voltage fluctuations from an unconnected pin (pin 13) */
    randomSeed(analogRead(13));
        .
        .
}
```

WITHOUT **SEED FUNCTION, TILES GENERATED IN LEVEL-1 (SEQUENCE = 2)**		
TRIAL N.	**1ST ELEMENT**	**2ND ELEMENT**
1	1	3
2	1	3
3	1	3
WITH **SEED FUNCTION, TILES GENERATED IN LEVEL-1 (SEQUENCE = 2)**		
TRIAL N.	**1ST ELEMENT**	**2nd ELEMENT**
1	1	0
2	2	3
3	0	3

Figure 3.14 : table illustrating the effects of the `seed()` function in the randomisation of the pattern

3.3 Integrated Design

3.3.1 Magnet polarity test - Hall effect sensor and RGB LED

The magnet polarity affects the readings produced by the Hall-effect sensor. From the base value pressing the magnet can generate two outcomes, an increase, or a decrease in voltage. Therefore, to test the polarity of the magnet the code from section **8.4** has been used. If pressing the button causes the voltage to decrease, the LED turns green, otherwise, if pressing the magnet causes the voltage to increase, the LED turns red. By checking the polarity of each button, the programmer can choose how to set up the buttons, to best fit the requirements.

3.3.2 Speed test - game feasibility

The speed test is conducted by changing the `speed factor` variable of the equation:

$$delayVal = \frac{900}{2^{speed\_factor\ (len\_i-2)}} + 100$$

The table below shows how increasing and decreasing the `speed factor` changes the delay value and affects how quickly the time becomes asymptotic (`100ms`). It is evident from the table that increasing the `speed factor` means a higher speed at which each pattern is shown, as `delayVal` decreases faster. The table below shows the relationship between the `speed_factor` and the first level at which the asymptotic time is reached:

speed_factor	Delay time per tile at level 50 (ms)	First level at max at asymptotic delay
0.01	132	100
0.25	100	41
0.5	100	21
1	100	11
2	100	6

Figure 3.15 : table illustrating the relationship between the speed factor and the delay time

Results:

From the table above, it is evident that incrementing the speed factor by a small weight has little effect on the speed at which the complete fifty tile sequence is shown; however, it significantly impacts how quickly the pattern converges to the minimum delay time per tile. Making the speed factor greater than 0.5 makes the program pace up too fast, therefore, leaving the user overwhelmed by the drastic change. However, making the speed factor less than 0.5 has the opposite effect, as the speed increment is not noticeable unless the user reaches considerably higher levels in the game.

Chapter 4: **Evaluation**

4.1 Hardware Design

The cube skeleton was sourced from a previous prototype developed by the student Bozser, G. G. last year. Alterations such as: removing the internal shelving unit, replacing plastic buttons with silicon ones, and removing small fragments of the skeleton to fit components were required for this project. The removal of the shelving unit was prompted by the lack of space inside the cube for the components. Additionally, replacing the plastic buttons with silicon buttons ensured flexibility when setting the sensitivity of the buttons and improving the user experience.

The cube assembly required superglue as the interlocking mechanism did not work due to the pressure imposed from the wire inside the cube. Small incisions have been made on one side of the skeleton to allow space for the displays. Furthermore - throughout the assembly process- securing jumper wires to components with the aid of electrical tape allowed the system to be more robust and less prone to conductivity issues between components. As an added measure, each module has been secured in place with the help of plastic wires, which are not conductive. However, as the rechargeable battery was not implemented, the cube is confined by the reach of the micro-USB cable. Finally, an internal shelving unit for the cube was not developed due to time constraints.

4.2 Tactile Sensors with silicon buttons

In this prototype, silicon buttons replaced the plastic buttons. This enabled the Hall-effect sensor to read highly varying voltages as the encased magnet was able to oscillate substantially. The fluctuation produced by the button pressed translated to the change in the magnetic field, thus increasing the sensitivity level of the tiles. To further exacerbate the tactile sensors' sensitivity an air gap was introduced between the magnet and the Hall-effect sensor. Additionally, due to the large surface area of the tiles, Dragon Skin^{TM} 30 *[27]* was used to produce sturdier models, however, its curing time required at least 16 hours, making the process time-consuming.

4.3 Memory Game Implementation

Overall, from a programming perspective, the game developed for this prototype met all the requirements set. The game has three factors that ensure that the user is engaged and immersed in the experience.
Increasing pattern length as the user improves their cognitive recall allows the player to keep pushing their recall abilities, making the game more enticing. In addition, progressively increasing the sequence displaying speed provokes the player to focus more, making the game exciting. Finally, displaying fake tiles in conjunction with the correct pattern adds to the complexity, making it harder for the user to reach the final level and therefore preventing them from feeling

underwhelmed by the experience. The amalgamations of all these factors ensure that the user is engaged and feels challenged and stimulated throughout the game and therefore keeps playing for extended periods. Additionally, the simple interface design allows the user to get adapted to the features quickly:

- A pattern with blue tiles indicates the correct sequence.
- Fake elements in the series are revealed by orange tiles.
- All tiles turning magenta prompt the user to start playing.
- A green tile after pressing a button signifies a correct press.
- A red tile after pressing a button implies a wrong press.

In conjunction, once the user loses (presses the wrong tile), the score and the high score are displayed on one side of the cube as an incentive to continue playing.

4.4 Future Improvements

Hardware-wise, the improvements to make would be as follows:

- Design a shelving unit to keep all the wires in place without taking too much space, allowing room to incorporate more components.
- Solder wires directly to the components to ensure only the necessary cables are used to avoid short-circuiting or having loose connections, making the system more robust.
- Implement a rechargeable battery unit to render a fully portable cube, increasing the likelihood of user interaction with the object.
- Use a sturdier material for the cube skeleton to minimise fractures due to pressure.

Software-wise, the improvements to make would be as follows:

- Implement different functionalities that can be accessed through a button press to make the cube more versatile.
- Use the accelerometer to track the cube movements to make the current game increase in difficulty or make a variation, therefore reaching a greater audience.
- Develop an application that allows the user to store their score and remotely control the system.
- Use EEPROM (electrically erasable programmable read-only) memory to store the high score value, this will allow the cube to access the high score even when the power supply is disconnected and connected back; however, limiting the lifespan of the product as EEPROM memory is single-use.

Chapter 5: **Conclusion**

Over the academic year, a complete system -integrated with both, hardware and software- was successfully designed and implemented. The *Unforgettable Cube* aimed to explore the uses of tactile sensors in devices designed for entertainment purposes. Additionally, this project sought to tackle and resolve flaws highlighted in the previous prototype of a cube developed in 2021. Although the system is fully functional, certain aspects such as the internal shelving unit, and rechargeable battery integration were not incorporated in the final product due to time constraints. The iterative testing process highlighted and explained key concepts of the programming aspects and the component functionalities. The report also provides a detailed description of the system assembly and execution to allow future prototypes to use this documentation to improve upon it and add more features.

The overall system developed resembles the characteristics of a 2x2 Rubik's cube, with only one side of the cube face adapted with the game interface to allow room for further exploration of the software element of the system and have more time to refine the overall product. Additionally, this decision guaranteed a more efficient schedule to complete the project on time and deliver a fully functional system by the end of the academic year. From a software perspective implementing the following features enhanced the game complexity and made the game entertaining and engaging for the user: varying sequence length, varying speed, fake flashes, and score display. Increasing sequence length as the game progresses matched the improving recall abilities of the players. Speeding up the sequence pace ensured long sequences did not take too long to display while growing the program's complexity. Adding a fake series of tiles within the pattern made the user alert. Furthermore, incorporating the 7-segment displays allowed the user to view and track their score for each session.

Finally, silicon buttons helped improve user experience by increasing tile sensitivity which enabled the user to apply minimal pressure to trigger a response from the system. Thus, successfully representing the integration of tactile sensors in entertainment gadgets and their possible uses.

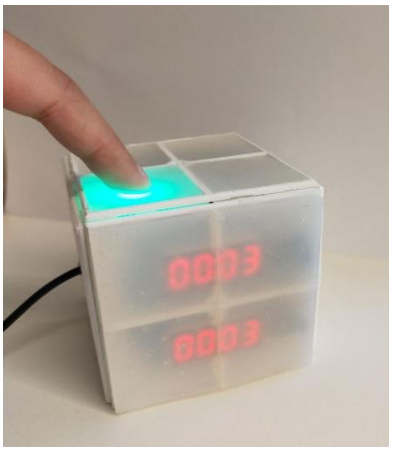


Figure 4: Final cube prototype displaying scores and a correct button press

Chapter 6: **References**

[1] Bozser, G. G. (2020). *Tactile Cube Project – Documentation*, Available at: https://drive.google.com/file/d/1_JvwRSQC_bM64DNmyev7FzSqalDCMJQ1/view?usp=sharing [Accessed 20 Nov. 2021].

[2] Ozioko, O. and Dahiya, R. (2021). Smart Tactile Gloves for Haptic Interaction, Communication, and Rehabilitation. *Advanced Intelligent Systems*, p.2100091.

[3] L. Jamone, L. Natale, G. Metta and G. Sandini, "Highly Sensitive Soft Tactile Sensors for an Anthropomorphic Robotic Hand," in IEEE Sensors Journal, vol. 15, no. 8, pp. 4226-4233, Aug. 2015, doi: 10.1109/JSEN.2015.2417759.

[4] Kwon, G.H., Kim, L. and Park, S. (2013). Development of a cognitive assessment tool and training systems for elderly cognitive impairment. [online] IEEE Xplore. Available at: https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=6563988 [Accessed 20 Nov. 2021].

[5] Ball K, Berch DB, Helmers KF, et al. Effects of Cognitive Training Interventions With Older Adults: A Randomized Controlled Trial. *JAMA.* 2002;288(18):2271–2281. doi:10.1001/jama.288.18.2271

[6] Butler, M., McCreedy, E., Nelson, V.A., Desai, P., Ratner, E., Fink, H.A., Hemmy, L.S., McCarten, J.R., Barclay, T.R., Brasure, M., Davila, H. and Kane, R.L. (2017). Does Cognitive Training Prevent Cognitive Decline? *Annals of Internal Medicine*, 168(1), p.63.

[7] Hamari, J., Shernoff, D.J., Rowe, E., Coller, B., Asbell-Clarke, J. and Edwards, T. (2016). Challenging games help students learn: An empirical study on engagement, flow, and immersion in game-based learning. *Computers in Human Behavior*, [online] 54, pp.170–179. Available at: https://www.sciencedirect.com/science/article/pii/S074756321530056X.

[8] Industries, A. (n.d.). *Adafruit ItsyBitsy 32u4 - 3V 8MHz*. [online] www.adafruit.com. Available at: https://www.adafruit.com/product/3675 [Accessed 21 Feb. 2022].

[9] MCP3004-I/SL Microchip Technology | Mouser (2012). *Microchip MCP3004-I/SL*. [online] Mouser Electronics. Available at: https://www.mouser.co.uk/ProductDetail/Microchip/MCP3004-I-SL?qs=sGAEpiMZZMsxiS4eJwGuBkL3m%2Fna1azx [Accessed 22 Nov. 2021].

[10] Microchip Technology Inc. (2008). *MCP3004/3008*. [online] Available at: http://ww1.microchip.com/downloads/en/DeviceDoc/21295d.pdf.

[11] Farnell.com. (2021). Hall Effect Sensor, Bipolar, 2.3 mA, SOT-23, 3 Pins, 2.5 V, 38 V. [online] Available at: https://uk.farnell.com/texas-instruments/drv5053vaqdbzr/hall-effect-sensor-bipolar-sot/dp/3008993 [Accessed 22 Nov. 2021].

[12] B (mT). (n.d.). [online] Available at: https://www.ti.com/lit/ds/symlink/drv5053.pdf?ts=1638046748538&ref_url=https%253A%252F%252Fwww.google.com%252F [Accessed 22 Nov. 2021].

[13] Mixtrónica. (n.d.). *WS2812B-MINI - Led RGB, SMD, PLCC4, 5.5x1.6mm*. [online] Available at: https://mixtronica.com/smd-leds/25030-ws2812b-mini-led-rgb-smd-plcc4-5-5x1-6mm.html [Accessed 22 Nov. 2021].

[14] www.digikey.co.uk. (n.d.). *WS2812B Datasheet - Parallax Inc. | DigiKey*. [online] Available at: https://www.digikey.co.uk/en/datasheets/parallaxinc/parallax-inc-28085-ws2812b-rgb-led-datasheet [Accessed 22 Nov. 2021].

[15] 06035C103JAT2A Kyocera AVX | Mouser (2021). *Kyocera AVX 06035C103JAT2A*. [online] Mouser Electronics. Available at: https://eu.mouser.com/ProductDetail/KyoceraAVX/06035C103JAT2A?qs=sGAEpiMZZMsSCEehGlfQHYkbjz8Fl3BE [Accessed 22 Nov. 2021].

[16] www.rapidonline.com. (n.d.). *Suntan TS170R2A223KSBBB0R 0.022uF 10% 100V X7R 5.08mm Radial Ceramic Capacitor*. [online] Available at: https://www.rapidonline.com/suntan-ts170r2a223ksbbb0r-0-022uf-10-100v-x7r-5-08mm-radial-ceramic-capacitor-11-3422 [Accessed 22 Nov. 2021].

[17] www.amazon.co.uk. (n.d.). *AZDelivery TP4056 Micro USB 5V 1A 18650 Lithium 3.7 V Li-Ion Battery Charging Board Module Including E-Book! : Amazon.co.uk: Electronics & Photo*. [online] Available at: https://www.amazon.co.uk/AZDelivery-TP4056-Battery-Charger-Module/dp/B082M9NTFG [Accessed 22 Nov. 2021].

[18] Richards, E. (2020). *TP4056 Lithium Battery Charging Module Pinout, Examples, Applications*. [online] Microcontrollers Lab. Available at: https://microcontrollerslab.com/tp4056-linear-lithium-ion-battery-charging-module/.

[19] www.amazon.co.uk. (n.d.). *sourcingmap 10Pcs 2 Position 3P SPDT Panel Mount Micro Slide Switch Latching Toggle Switch : Amazon.co.uk: DIY & Tools*. [online] Available at: https://www.amazon.co.uk/sourcingmap-Position-Switch-Latching-Toggle/dp/B01N367QLZ/ref=asc_df_B01N367QLZ?tag=bingshoppinga-21&linkCode=df0&hvadid=80676723059417&hvnetw=o&hvqmt=e&hvbmt=be&hvdev=c&hvlocint=&hvlocphy=&hvtargid=pla-4584276300071567&psc=1 [Accessed 22 Nov. 2021].

[20] www.amazon.co.uk. (n.d.). WayinTop 10pcs Micro USB to DIP 5Pin Pinboard 2.54mm Power Adapter Board 5V Breakout Converter Module for DIY USB Power Supply Breadboard Design: Amazon.co.uk: Computers & Accessories. [online] Available at: https://www.amazon.co.uk/WayinTop-Pinboard-Breakout-Converter-Breadboard/dp/B07W13X3TD [Accessed 22 Nov. 2021].

[21] www.amazon.co.uk. (n.d.). *Magnet Expert® 2mm dia x 1mm thick N42 Neodymium Magnet - 0.11kg Pull (Pack of 50) : Amazon.co.uk: Business, Industry & Science*. [online] Available at: https://www.amazon.co.uk/gp/product/B007JTKHR6/ref=ppx_yo_dt_b_asin_title_o00_s00?ie=UTF8&psc=1 [Accessed 22 Nov. 2021].

[22] Farnell.com. (2021). *SMD Chip Resistor, 536 ohm, ± 1%, 62.5 mW, 0402 [1005 Metric], Thick Film, General Purpose*. [online] Available at: https://uk.farnell.com/multicomp/mc00625w04021536r/res-536r-1-0-0625w-0402-thick/dp/1803007 [Accessed 22 Nov. 2021].

[23] www.amazon.co.uk. (n.d.). *Seamuing 3.7V 2000mAh Lithium Rechargeable Batteries 1S 3C Lipo Batteries Battery with Protection Board, Insulated Tape and Micro JST 1.25 Plug for Arduino Nodemcu ESP32 Development Board (4 Pack) : Amazon.co.uk: Toys & Games*. [online] Available at: https://www.amazon.co.uk/gp/product/B08TQSC5G9/ref=ppx_yo_dt_b_asin_title_o04_s00?ie=UTF8&psc=1.

[24] www.amazon.co.uk. (n.d.). *Amazon.co.uk*. [online] Available at: https://www.amazon.co.uk/AZDelivery-Display-Digits-Segment-Digital/dp/B0813R898L [Accessed 12 Jan. 2022].

[25] Argos. (2022). *Buy 2m Micro USB Cable - Black* | *Mobile phone chargers* | *Argos*. [online] Available at: https://www.argos.co.uk/product/8356976 [Accessed 12 Jan. 2022].

[26] kaisertech.co.uk. (n.d.). *JBC NASE-2C Nano Rework Station*. [online] Available at: https://kaisertech.co.uk/jbc-nase-2c-nano-rework-station?gclid=CjwKCAiAqIKNBhAIEiwAu_ZLDvA1w1tBv7L172RjdO6o8oF3ncaoiYb8lQHwU-XuWTjpu3wKJHpogxoCZloQAvD_BwE [Accessed 20 Nov. 2021].

[27] Smooth-On, Inc. (n.d.). *Dragon Skin^{TM} 30 Product Information*. [online] Available at: https://www.smooth-on.com/products/dragon-skin-30/ [Accessed 12 Jan. 2022].

[28] Easyeda.com. (2019). *EasyEDA - Online PCB design & circuit simulator*. [online] Available at: https://easyeda.com/.

[29] Autodesk.com. (2021). *Fusion 360* | *3D CAD, CAM, CAE & PCB Cloud-Based Software* | *Autodesk*. [online] Available at: https://www.autodesk.com/products/fusion-360/overview?term=1-YEAR&tab=subscription.

[30] Basic Electronics Tutorials. (2018). *7-segment Display and Driving a 7-segment Display*. [online] Available at: https://www.electronics-tutorials.ws/blog/7-segment-display-tutorial.html.

Chapter 7: **Appendix 1 – Full program**

```
//LIBRARIES
#include <Adafruit_MCP3008.h>   //for ADC
#include <Adafruit_NeoPixel.h>  //for RGB LEDs
#include "TM1637.h"             //for 7-segment display

/*--------------------------------------------------------------------*/
//MACROS
#define PIN        7 //LED-IN pin (PCB) connected to the NeoPixels
#define NUMPIXELS  4 //number of NeoPixels
#define NHALL      4 //number of hall effect sensors
#define CS         5 //ADC CS pin
//pins definitions for TM1637 display 1
#define CLK1       9
#define DIO1       10
//pins definitions for TM1637 display 2
#define CLK2       11
#define DIO2       12
//hall effect sensors base value
#define HAL_0      556
#define HAL_1      522
#define HAL_2      545
#define HAL_3      531

//GLOBAL VARIABLES
float delayVal=1000;     //time (in ms) to pause between pixels
float speed_factor= 0.5; //speed factor to determine delayVal
//initialise score and high score
int high_score=0, score=0;
int sequence[50]; //initialise sequence of 50 elements
//counter to keep track of the elements in the sequence
int step_S= 0;
int state=0, j=0;
int c=10000;      //delay counter to run each pattern sequence
int len_i=2;      //game level (level 1 = 2 tiles)
//determines the range for base values of each hall effect sensor
int stride=25;
int fake_flash=2; //max number of fake flashes per pattern
int toss=1;

//set TM1637 objects as dis1 and dis2
TM1637 dis1(CLK1,DIO1);
TM1637 dis2(CLK2,DIO2);
//set Adafruit_NeoPixel object as pixels
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
//set Adafruit_MCP3008 object as adc
Adafruit_MCP3008 adc;
```

```
//pixels.Color() takes RGB values, from 0,0,0 up to 255,255,255
uint32_t magenta = pixels.Color(50, 0, 50);

/*---------------------------------------------------------------------*/
void setup() {
    //put your setup code here, to run once:
    pixels.begin(); //initialise NeoPixel strip object
    Serial.begin(9600);
    //custom pins to use a software SPI interface
    adc.begin(SCK, MOSI, MISO, CS);
    //initialise TM1637 objects
    //BRIGHT_TYPICAL = 2,BRIGHT_DARKEST = 0,BRIGHTEST = 7;
    dis1.init();
    dis1.set(BRIGHT_TYPICAL);
    dis2.init();
    dis2.set(BRIGHT_TYPICAL);

    /* initialize the pseudo-random number generator by reading voltage
       fluctuations from an unconnected pin (pin 13) */
    randomSeed(analogRead(13));
}

/*---------------------------------------------------------------------*/
void loop() {
    int8_t NumTab[] = {0,1,2,3,4,5,6,7,8,9}; // 0-9 config
    if (c>0) c--; //decrement counter
    switch(state){
        case 0:
           //compare score and high score
            high_score= max(score, high_score);
            //if game reset to level 1 show score and high score
            if(len_i==2){
                //print the units correctly on the displays
                dis1.display(3,NumTab[int((score))%10]);
                dis1.display(2,NumTab[int(floor(score/10))%10]);
                dis1.display(1,NumTab[int(floor(score/100))%10]);
                dis1.display(0,NumTab[int(floor(score/1000))%10]);
                dis2.display(3,NumTab[int((high_score))%10]);
                dis2.display(2,NumTab[int(floor(high_score/10))%10]);
                dis2.display(1,NumTab[int(floor(high_score/100))%10]);
                dis2.display(0,NumTab[int(floor(high_score/1000))%10]);
            }

            score=0;        //reset score
            pixels.clear(); //turn all the pixels off
            //send the updated pixel colours to the hardware
            pixels.show();
            //change speed based on level
            delayVal= 900/pow(2,speed_factor*(len_i-2))+100;
            fake_flash=2;   //reset number of fake flashes
            //call function to generate and show the sequence
            if(len_i<51){
                populate();
            }
            else{
                len_i=50;
                populate();
            }
            pattern();
            j=0;
            state=1; //move to the next state
            step_S = sequence[j];

        case 1:
            //read and store the change in magnetic field from each hall
            //effect sensor
            int a0=adc.readADC(0);
            int a1=adc.readADC(1);
            int a2=adc.readADC(2);
            int a3=adc.readADC(3);

            if(a0>HAL_0+stride ||a0<HAL_0-stride){
            //if the value in the sequence = button press turns LED green
                if(step_S == 0){
                    len_i=len_i+(j==len_i-1); // increment level
                    //LED 0 = green
                    pixels.setPixelColor(0,pixels.Color(0, 70, 0));
                    pixels.show();
                    //next value in the array
                    j=j+1;
                    step_S = sequence[j];
                    delay(500);
                    //turn LED off
                    pixels.clear();
                    pixels.show();
                }
                //otherwise restart at level 0
                else{
                    score=len_i-2;
                    len_i=2;
                    //LED 0 = red
                    pixels.setPixelColor(0,pixels.Color(70, 0, 0));
                    pixels.show();
                    delay(500);
                    state=0; //change states
                }
            }

            if(a1>HAL_1+stride ||a1<HAL_1-stride){
            //if the value in the sequence = button press turns LED green
                if(step_S == 1){
                    len_i=len_i+(j==len_i-1); //increment level
                    //LED 1 = green
                    pixels.setPixelColor(1,pixels.Color(0, 70, 0));
                    pixels.show();
                    //next value in the array
                    j=j+1;
                    step_S = sequence[j];
                    delay(500);
                    //turn LED off
                    pixels.clear();
                    pixels.show();
                }
                //otherwise restart at level 0
                else{
                    score=len_i-2;
                    len_i=2;
                    //LED 1 = red
                    pixels.setPixelColor(1,pixels.Color(70, 0, 0));
                    pixels.show();
                    delay(500);
                    state=0; //change states
                }
            }

            if(a2>HAL_2+stride ||a2<HAL_2-stride){
             //if the value in the sequence = button press turns LED green
                if(step_S == 2){
                    len_i=len_i+(j==len_i-1); //increment level
                    //LED 2 = green
                    pixels.setPixelColor(2,pixels.Color(0, 70, 0));
                    pixels.show();
                    //next value in the array
                    j=j+1;
                    step_S = sequence[j];
                    delay(500);
                    //turn LED off
                    pixels.clear();
                    pixels.show();
                }
                //otherwise restart at level 0
                else{
                    score=len_i-2;
                    len_i=2;
                    //LED 2 = red
                    pixels.setPixelColor(2,pixels.Color(70, 0, 0));
                    pixels.show();
                    delay(500);
                    state=0; //change states
                }
```

```
            }

            if(a3>HAL_3+stride ||a3<HAL_3-stride){
            //if the value in the sequence = button press turns LED green
                if(step_S == 3){ //the correct led
                    len_i=len_i+(j==len_i-1); //increment level
                    //LED 3 = green
                    pixels.setPixelColor(3,pixels.Color(0, 70, 0));
                    pixels.show();
                    //next value in the array
                    j=j+1;
                    step_S = sequence[j];
                    delay(500);
                    //turn LED off
                    pixels.clear();
                    pixels.show();
                }
                //otherwise restart at level 0
                else{
                    score=len_i-2;
                    len_i=2;
                    //LED 3 = red
                    pixels.setPixelColor(3,pixels.Color(70, 0, 0));
                    pixels.show();
                    delay(500);
                    state=0; //change states
                }
            }
            //if all the tiles have been correctly pressed don't wait
            //for the counter
            if(j==len_i){
                c=0;
            }
            //if times up then change the counter accordingly for the
            //next sequence
            if (c==0){
                pixels.clear();
                pixels.show();
                c=10000+3000*(len_i);
                state=0; //change states
            }
    } //END switch
} //END loop

/*--------------------------------------------------------------------*/

// function for creating the pattern and storing it in sequence array
void populate(){
    for (int i=0; i<len_i; i++){
        sequence[i]= random(NUMPIXELS);
    }
    //print the sequence in the serial monitor for debugging
    for (int k=0;k<len_i; k++){
        Serial.println(sequence[k]);
    }
}

//function for displaying the sequence on the cube and generating fake
//flashes
void pattern(){
    for(int i=0; i<len_i; i++) {
        //show pattern in BLUE
        pixels.setPixelColor(sequence[i],pixels.Color(0, 0, 70));
        pixels.show(); //Send the updated pixel colours to the hardware
        //determines the speed of the flash
        delay(delayVal);    //pause before next pass-through loop
        pixels.setPixelColor(sequence[i],pixels.Color(0, 0, 0));
        pixels.show(); //send the updated pixel colours to the hardware
        delay(delayVal);
        toss=random(3); //probability of fake flash 1/3

        //if fake flash, then turn on a random LED orange
        if(toss==0 && fake_flash>0){
            pixels.setPixelColor(random(4),pixels.Color(70, 30, 0));
            pixels.show();
            //determines the speed of the flash
            delay(delayVal); // Pause before next pass-through loop
            pixels.clear();
            pixels.show();
            //keep track of the number of fake flashes
            fake_flash=fake_flash-1;
        }
    }
    //turn on all LEDs magenta
    pixels.fill(magenta,0,NUMPIXELS);
    pixels.show(); //send the updated pixel colours to the hardware
    delay(200);
    //turn all LEDs off
    pixels.clear();
    pixels.show(); //send the updated pixel colours to the hardware
}
```

Chapter 8: **Appendix 2 – testing code**

8.1 RGB LEDs test code

```
#include <Adafruit_NeoPixel.h> //library for RGB LEDs
#define PIN        7 //LED-IN pin (PCB) connected to the NeoPixels
#define NUMPIXELS  4 //number of NeoPixels
//when setting up the NeoPixel library, we tell it how many pixels,
//and which pin to use to send signals.
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
//set Adafruit_NeoPixel object as pixels
#define DELAYVAL 500
/*--------------------------------------------------------------------*/

void setup() {
    pixels.begin(); //initialise NeoPixel strip object
}
/*--------------------------------------------------------------------*/

void loop() {
    pixels.clear(); //turn all the pixels off

    // turn all LEDs orange one at a time
    for(int i=0; i<NUMPIXELS; i++) {
        pixels.setPixelColor(i, pixels.Color(30, 10, 0));
        //send the updated pixel colours to the hardware
        pixels.show();
        //time (in milliseconds) to pause between pixels
        delay(DELAYVAL);
    }

    //turn all LEDs turquoise one at a time
    for(int k=0; k<NUMPIXELS; k++) {
        pixels.setPixelColor(k, pixels.Color(0, 30, 10));
        //send the updated pixel colours to the hardware
        pixels.show();
        delay(DELAYVAL);
    }

    //turn all LEDs purple one at a time
    for(int j=0; j<NUMPIXELS; j++) {
        pixels.setPixelColor(j, pixels.Color(10, 0, 30));
        //send the updated pixel colours to the hardware
        pixels.show();
        //time (in milliseconds) to pause between pixels
        delay(DELAYVAL);
    }
}
```

8.2 Hall-effect sensors test code

```
//library for Hall-effect sensors
#include <Adafruit_MCP3008.h>
#define CS 5                //ADC CS pin
long const ADC_REF = 3.0; //set voltage used for VREF
int num_hall=4;             //number of hall effect sensors
Adafruit_MCP3008 adc ;     //set Adafruit_MCP3008 object as adc
/*---------------------------------------------------------------------*/

void setup() {
    //put your setup code here, to run once:
    Serial.begin(9600); // =initialise serial monitor
    //use custom pins
    adc.begin(SCK, MOSI, MISO, CS);
}
/*---------------------------------------------------------------------*/

void loop(){
    //read all Hall-effect sensors
    for(size_t i=0; i<num_hall;i++){
        int adc_Reading = adc.readADC(i);
        //print to the serial monitor
        Serial.print(adc_Reading);
        Serial.print(" = ");
        //convert binary to decimal voltage
        Serial.print((adc_Reading * ADC_REF)/1024.0);
        Serial.print("V,   ");
    }
    Serial.println(""); //print a space
    delay(100); //wait for 100ms
}
```

8.3 7-segment display test code

```
#include "TM1637.h" //library for the 4 digit 7-segment display
#define CLK1 9      //pins definitions for TM1637 dis1
#define DIO1 10
#define CLK2 11     //pins definitions for TM1637 dis2
#define DIO2 12
//initialise two object of type TM1637
TM1637 dis1(CLK1,DIO1);
TM1637 dis2(CLK2,DIO2);
float level = 0; //counter variable
/*---------------------------------------------------------------------*/

void setup() {
    // put your setup code here, to run once:
    dis1.init(); //initialise display 1
    //BRIGHT_TYPICAL = 2,BRIGHT_DARKEST = 0,BRIGHTEST = 7;
    dis1.set(BRIGHT_TYPICAL);
    dis2.init(); //initialise display 2
    dis2.set(BRIGHT_TYPICAL);
}
/*---------------------------------------------------------------------*/

void loop() {
    // put your main code here, to run repeatedly:
    int8_t NumTab[] = {0,1,2,3,4,5,6,7,8,9}; // 0-9 config
    //digit one score (right to left)
    dis1.display(3,NumTab[int((level))%10]);
    // digit two score (right to left)
    dis1.display(2,NumTab[int(floor(level/10))%10]);
    // digit three score (right to left)
    dis1.display(1,NumTab[int(floor(level/100))%10]);
    //digit four score (right to left)
    dis1.display(0,NumTab[int(floor(level/1000))%10]);

    //digit one high score (right to left)
    dis2.display(0,NumTab[int((level))%10]);
    //digit two high score (right to left)
    dis2.display(1,NumTab[int(floor(level/10))%10]);
    //digit three high score (right to left)
    dis2.display(2,NumTab[int(floor(level/100))%10]);
    //digit four high score (right to left)
    dis2.display(3,NumTab[int(floor(level/1000))%10]);
    delay(500); // wait for a while
    level=level+1; //increment counter
}
```

8.4 Magnet polarity test code

```
//libraries for Hall-effect sensors and RGB LEDs
#include <Adafruit_MCP3008.h>
#include <Adafruit_NeoPixel.h>
/*---------------------------------------------------------------------*/

#define PIN        7 //LED-IN pin (PCB) connected to the NeoPixels
#define NUMPIXELS  4 // How many NeoPixels are attached to the Arduino?
#define NHALL      4 // number of hall effect sensors
#define CS         5 //ADC CS pin
#define DELAYVAL 500 // Time (in milliseconds) to pause between pixels
//hall effect sensors base value
#define HAL_0      556
#define HAL_1      522
#define HAL_2      545
#define HAL_3      531

long const ADC_REF = 3.0;  // Set voltage used for VREF
//when setting up the NeoPixel library, we tell it how many pixels,
//and which pin to use to send signals.
Adafruit_NeoPixel pixels(NUMPIXELS, PIN, NEO_GRB + NEO_KHZ800);
//set Adafruit_MCP3008 object as adc
Adafruit_MCP3008 adc ;
/*---------------------------------------------------------------------*/

void setup() {
    // put your setup code here, to run once:
    pixels.begin(); //initialise NeoPixel strip object
    Serial.begin(9600); //initialise serial monitor
    adc.begin(SCK, MOSI, MISO, CS); //custom pins
}
/*---------------------------------------------------------------------*/

void loop() {
    // put your main code here, to run repeatedly:
    pixels.clear(); //set all pixel colors to 'off'
    pixels.show();  //send the updated pixel colors to the hardware
    //read all the Hall-effect sensors
    int adc_Reading0 = adc.readADC(0);
    int adc_Reading1 = adc.readADC(1);
    int adc_Reading2 = adc.readADC(2);
    int adc_Reading3 = adc.readADC(3);
    // print the values read from the sensor 0
    Serial.print(" Hall 0 --> ");
    Serial.print(adc_Reading0);
    Serial.print(" = ");
    Serial.print((adc_Reading0 * ADC_REF)/1024.0);
    Serial.print("V, ");
    // print the values read from the sensor 1
```

```
Serial.print(" Hall 1 --> ");
Serial.print(adc_Reading1);
Serial.print(" = ");
Serial.print((adc_Reading1 * ADC_REF)/1024.0);
Serial.print("V, ");
// print the values read from the sensor 2
Serial.print(" Hall 2 --> ");
Serial.print(adc_Reading2);
Serial.print(" = ");
Serial.print((adc_Reading2 * ADC_REF)/1024.0);
Serial.print("V, ");
// print the values read from the sensor 0
Serial.print(" Hall 3 --> ");
Serial.print(adc_Reading3);
Serial.print(" = ");
Serial.print((adc_Reading3 * ADC_REF)/1024.0);
Serial.print("V, ");
//new line
Serial.println("");

//upper bound of sensor 0
if (adc_Reading0 > HAL_0+10){
    pixels.setPixelColor(0, pixels.Color(100, 0, 0)); //red
    pixels.show(); //send the updated pixel colors to the hardware
}
//lower bound of sensor 0
if (adc_Reading0 < HAL_0-10){ //base value minus stride
    pixels.setPixelColor(0, pixels.Color(0, 100, 0)); //green
    pixels.show(); // Send the updated pixel colors to the hardware
}

//upper bound of sensor 1
if (adc_Reading1 > HAL_1+10){
    pixels.setPixelColor(1, pixels.Color(100, 0, 0)); //red
    pixels.show(); // Send the updated pixel colors to the hardware
}
//lower bound of sensor 1
if (adc_Reading1 < HAL_1-10){
    pixels.setPixelColor(1, pixels.Color(0, 100, 0)); //green
    pixels.show(); // Send the updated pixel colors to the hardware
}

//upper bound of sensor 2
if (adc_Reading2 > HAL_2+7){
    pixels.setPixelColor(2, pixels.Color(100, 0, 0)); //red
    pixels.show(); // Send the updated pixel colors to the hardware
}
//lower bound of sensor 2
if (adc_Reading2 < HAL_2-10){
    pixels.setPixelColor(2, pixels.Color(0, 100, 0)); //green
```

```
        pixels.show(); // Send the updated pixel colors to the hardware
    }

    //upper bound of sensor 3
    if (adc_Reading3 > HAL_3+10){
        pixels.setPixelColor(3, pixels.Color(100, 0, 0)); //red
        pixels.show(); // Send the updated pixel colors to the hardware
    }
    //lower bound of sensor 3
    if (adc_Reading3 < HAL_3-10){
        pixels.setPixelColor(3, pixels.Color(0, 100, 0)); //green
        pixels.show(); // Send the updated pixel colors to the hardware
    }

    delay(DELAYVAL); //pause before next pass-through loop
}
```