**Queens University**

**Belfast**

Name: Shane Sally  
Email: ssally02@qub.ac.uk  
Student number: 40381276

‘UnSmart Home’

A dissertation submitted in partial fulfilment of  
The requirement for the degree of  
MASTER OF SCIENCE in Software Development  
in  
The Queen’s University of Belfast  
By  
‘Shane Sally’  
‘08/09/2023’

**CSC7057 – INDIVIDUAL SOFTWARE DEVELOPMENT PROJECT**

A signed and completed cover sheet must accompany the submission of the Individual Software  
Development dissertation submitted for assessment.  
Work submitted without a cover sheet will NOT be marked.

**Declaration of Academic Integrity**

(Insert Declaration)

**Acknowledgements**

I would like to thank my mentor Neil Anderson for helping with this Masters’ research project in  
Software Development, for his time taken during bi-weekly meetings and commitment to impart his knowledge and guidance.

Thank you to Queen’s University Belfast for the opportunity to take on and complete this degree, it has helped alter my career path.

**Abstract**

This is a summary (100 words) outlining the subject matter and conclusions of the dissertation.

**Contents**

Acknowledgements ...............................................................................................................................3  
Abstract .................................................................................................................................................4  
Introduction ...........................................................................................................................................6  
Understanding the problem ..................................................................................................................7  
Technology Stack ...................................................................................................................................8

Design Phase.........................................................................................................................................10

Functional & Non-Functional Requirements..........................................................................................19  
Architecture design and VR Experimentation.......................................................................................16  
Backend framework and APIs...............................................................................................................?  
Testing..................................................................................................................................................?  
Evaluation and Conclusion ..................................................................................................................?  
References............................................................................................................................................?

**Introduction**

Below is an outline of the structure of this dissertation and key components of each section:

* Understanding the Problem
* User interface design
* Architecture design and algorithm explanation
* Experimentation
* Testing
* Evaluation and Conclusion

When trying to understand the problem for this project it was important to review the philosophy behind it, whilst analyzing the current project that had been exhibited so that a solution could be developed and integrated with the current software. The new software technologies researched in this project included an entirely new web framework that would incorporate HTML, CSS and JavaScript and new languages that would collaborate with minicomputer technology.

As discussed at a later stage the technology stack used (Python, HTML, JavaScript, CSS, A-Frame, Raspberry Pi) and the justification of these.

The user interface was spread across a variety of technologies that encompassed experimental design decisions that would be tested regularly so the most appropriate resolution would be implemented to work with this project.

A high-level overview of the current project's user interface is displayed.

For the architecture and design a high-level overview of the project’s solution files is available. This covers the data storage methods, the documentation created for the external users to utilize the open API and how the finalized system is connected when looking at the front and back end.

For Testing the test-driven development is discussed and how it played a pivital part during the development lifecycle.

Finally, the Evaluation and Conclusion runs through the outlined requirements, how they were accomplished and what improvement could be made going forward.

**Understanding the Problem**

The escape room industry is one that is often suitable for all ages, providing the opportunity to solve unique and challenging problems whilst operating within a team-building environment. A common issue that will present itself when playing can often be the room's reusability and the means in which they can be updated or altered so that puzzle solutions remain distinctive to the room in play.

Escape rooms today have become part of a saturated market therefore the traditional escape room has been innovated to include new technologies and high-quality room environments that replicate fantasy and science-fiction genres for a more theatrical experience.

Theme inspired rooms allow for the use of actors and more advanced technology in the form of VR, interactive sets and new development channels that allow game masters to increase a rooms difficulty whilst never compromising the overall structure of the room.

VR is a relatively new technology that allows the user to immerse themselves in a simulated 3D environment whilst having the ability to interact with the created virtual world.

There are two types of VR:

* Native VR – developed using gaming engines for an established platform
* WebVR – A website programmed in HTML5 with open OpenGL that can recognize a device and display the contents of the website whilst optimizing it.

VR can be non-immersive, allowing the user to view the virtual environment through a screen or monitor. It can also be semi-immersive which offers a partial VR experience with the assistance of a VR headset (This will be the technology used for this project). Finally, fully immersive which offers an enhanced environment that allows for sound, sight, touch and on rare occasions smell.

*This technology many have agreed, could help to improve performance and conceptual understanding on a specific range of tasks* [2].

When reviewing the project from an educational perspective facing children and young adolescents, this can lend itself to creating a unique learning environment for players when attempting to learn about computer technology and software development for the first time.

As an educational motive this is a unique direction that allows the project to stand out as it attempts to conquer a growing market by reimagining the learning environment and transforming what it means to acquire new knowledge for younger generations. *Previous studies have shown advantages in applying this approach and this is expected as, educational escape rooms were found to promote teamwork and collaboration, as well as produce high levels of enjoyment and engagement.*[1]

The combination of these more recent technologies with how rooms are currently set up can provide many additional advantages. Some of these would include a more efficient use of space, where escape rooms themselves can require large areas, adding an additional virtual room to the existing room allows for a more efficient use of smaller spaces. More players could be accommodated, including those with disabilities, when trying to have the same experience which could be achieved solely through the use of a chair.

**Technology Stack**

Before confirming which technology and architecture design components to appoint, research was carried out in order to investigate their suitability. Back-end code was primarily handled by Python so that the game puzzles were able to interact effectively with the hardware. HTML was used in conjunction with the A-Frame framework in order to produce the VR interface that players would ultimately interact with.

* **Data storage needs to be mentioned briefly**
* **What hosts the API?**

The following key architecture aspects were explored:

* Programming languages
* Pi Project exploration and creation
* VR Development and creation

**Programming Language**

Careful consideration must be given to the programming languages use for any project and due to the nature of this exploratory project three factors were taken into consideration:

* VR development capabilities
* Embedded systems development
* Ease of development for building protypes and testing

VR Development Capabilities

In terms of VR there are two types that are recognized and useful when developing and building a room in virtual reality.

1. Native VR – these applications are generally optimized for a specific operating system, for example Android7 that would run in association with Oculus or HTC. The disadvantage with this is that they require their own unique hardware in the form of a headset that is not very accessible to the average user. These VR apps would usually be developed using a gaming engine like Unity that offers advanced developer tools to assist in the creation of virtual worlds. Although popular in industry the development path is complex and time consuming which isn’t relatable within a beginner learning environment.

2. WebVR – these applications work on all operating systems and present a cheaper less complex approach to the development process. Although visually it is less superior to what could be produced on Unity it can provide an ample learning experience whilst displaying detailed and fully functioning VR worlds. There are a variety of frameworks available such as react360, Three.js and for the purposes of this project we will be looking further into A-Frame.

A-Frame is a web-based framework for building VR experiences and at its core is a powerful entity-component framework. It has been developed so that it is easy to follow and accessible to the majority of VR headsets. Its aim is to define a fully immersive interactive VR experience that goes beyond basic 360° content.

Further research suggests that HTML and JavaScript are languages that are utilized when building an A-Frame specific project.

HTML is an easily understood language that is accessible to everyone without requiring specialized software or hardware. It provides a straightforward and easily manageable syntax that contains a wide range of libraries and frameworks, like A-Frame that can simplify large portions of the development process. Its compatibility stretches across numerous devices that would include desktop computers, smartphones and a variety of VR headsets that support web browsing.

JavaScript is well supported and the preferred choice due to its compatibility with HTML and web development. However, as its syntax is considered to be quite complex it can be time-consuming during the development process.

Embedded systems development

In the past embedded systems development worked primarily with specialized software in conjunction with a micro controller that needed a specialist who was literate in C or C++. As technology progressed the Raspberry Pi was introduced containing a System on a Chip (SoC) powerful enough to run its own operating system. This meant high level languages like Python could be utilized to create new and highly efficient embedded systems [5].

A Raspberry Pi is a small, inexpensive minicomputer that can plug into a monitor or TV. It allows users of all ages to experiment and explore computing in an interactive and structured manner using a variety of languages. More interestingly *it has the ability to interact with the outside world and has been used in a wide array of digital maker projects, from music machines and parent detectors to weather stations and tweeting birdhouses with infra-red cameras* [3]. In conjunction with the manufacturing of these small computers the Raspberry Pi foundation (A registered educational charity) was founded in order to *advance the education of adults and children, particularly in the field of computers, computer science and related subjects* [4].

A Raspberry Pi computer can be used in a variety of ways to benefit an escape room, including reading types of sensors, playing back video or sound, controlling lights within a room and of course unlocking doors.

The use of these Raspberry Pi lends themselves to the overall use case of this project as this technology is available to the public at a low cost with the necessary learning resources made available for free to younger generations' who wish to explore further.

Ease of development for building protypes and testing

Python is a high-level language considered by many to be more manageable than most other languages today, especially for beginner programmers who are trying to break into the world of programming for the first time. There are many online resources for creating new Python projects that are time efficient and easy to follow. This would mean that when exploring new ideas there is an extensive library of low-level projects that are available to try online that can be used as a springboard into more complex projects.

**Pi Project Exploration and Creation**

When developing an embedded system, it is important that the hardware and software can effectively work together, to achieve functionality they must interact precisely with one another. This will produce several challenges:

* Exploring relevant projects – completed projects that are specific to the puzzles
* Enhancing – Building upon the code so that it suits the puzzle design
* Connecting – Linking the puzzles via the code so that they work symbiotically

Each of the above points would provide a basic structure for the development of a new escape room puzzle and how one might consider their design before proceeding with development.

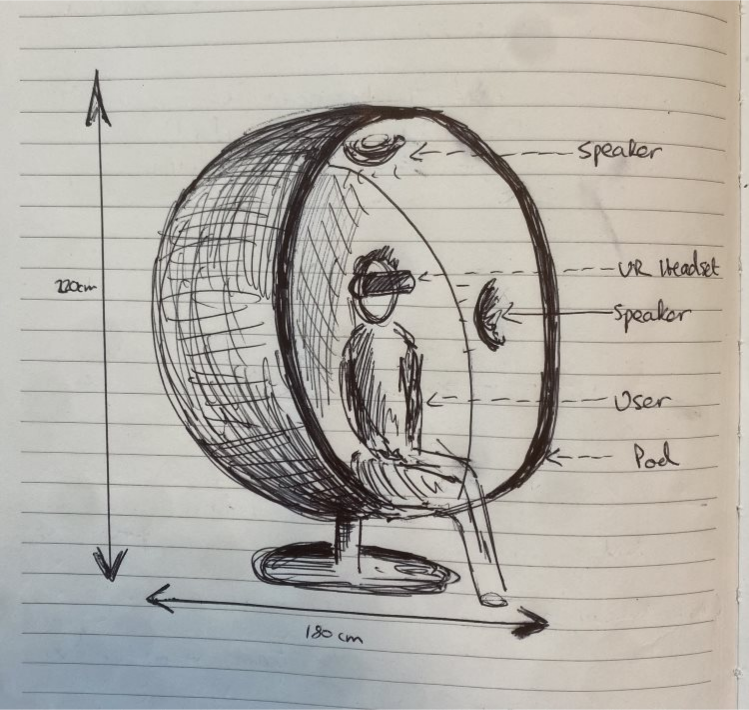
**Design Phase**

**Design 1**

The initial concept for the design was to have a chair that could be made interactive as part of the escape room so a VR element could be integrated into the design allowing for experimentation with this additional technology. This chair could potentially act as an extension of the room that would supply a code to the user that they could input into a GUI, ultimately unlocking a latch so that a key item would be presented to the user.

The chair would potentially need to have an activation button that could switch on the VR headset and allow the user to see a completely new room in VR space.

From this point the user would have to look for the code they would need to input into the GUI. The GUI itself could be located in the VR world or outside of it within the physical room, allowing a unique communication task between the team of players trying to complete the escape room.



(Design 1, Image 1)

From this original concept there were several questions that need to be answered:

1. How will the chair be activated?
2. Will activation require more than one additional piece of information.
3. Does the user need to interact with the VR environment to locate the code?
4. Once the code has been located does the VR element become obsolete?
5. If so, how can this be conveyed to the user to avoid any confusion?

After reviewing these questions, it was decided that the focus was falling too heavily on the VR element of the puzzle and that it should only be used as an extension of not just the proposed escape room but also the puzzle design itself. I liked the idea that it could be used as a team building exercise, possibly as the first puzzle in the room, pushing the players to work together as one player would communicate with the others to solve a puzzle. I also liked the idea of a GUI opening a lock mechanism, the next would be to come up with a puzzle suitable to implement these elements.

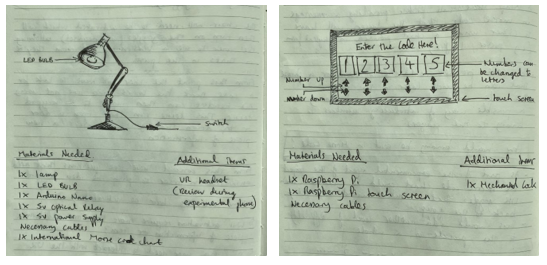
**Design 2**

As part of a revised concept, I thought it would be interesting to add in a unique puzzle element that would get players thinking and collaborating. This would come in the form of a flickering lamp using an Arduino Nano (Image 1).

The lamp would flicker providing a message in morse code that the players will have to cipher using an international morse code chart. The code provided could be a word or a phrase that could be used to open a lock using a GUI. The chart itself could be presented within the VR world, incorporating an additional puzzle element or simply as an interactive poster to develop interest.

The code itself could also be sent as numbers as to introduce the subject of binary to players keeping in line with the theme of the escape room.

This could lend itself to the difficulty level requirement of the project also, with the VR element acting as the 2nd difficulty level whilst a simple physical chart could be difficulty level 1.



(Morse Code Lamp, Image 2) (GUI Design, Image 3)

To input the code that’s sent to the user via the flickering lamp, the user should be able to input the details into a GUI. For this to be incorporated as part of the working puzzle I felt an independent touch screen would be perfect.

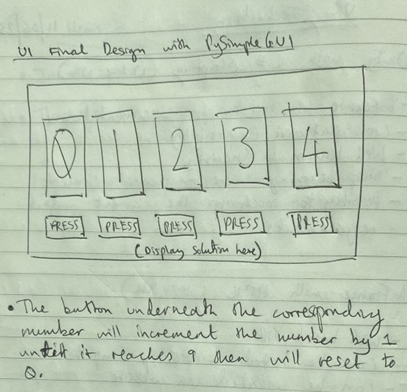
The GUI itself could also be used to meet the difficulty level requirement. Younger players would simply input the code to proceed, whilst older players would have to type in the code with an added level of difficulty. Additional functionality could be added to the button events, whereby one button could change multiple numbers forcing the player to figure out a sequence of button presses in the correct order.

**Design 3**

To facilitate many of the ideas for *Design 2* some changes were needed so that it could be functional. Although the design for the Morse Code Lamp would remain the same, I opted to use a Raspberry Pi so that the GUI and Puzzle could work in unison.

The Morse Code Lamp would not necessarily need to be a physical lamp but an LED or Bulb that could be placed within an object of choice later in development.

The GUI would have some slight changes whereby a single button would be used to increment the numbers rather than arrows as there are ten numbers per button. The need to decrease is not a priority.



*(Final GUI design, Image 4)*

The VR element would be created as an extension of the escape room in the form of a VR classroom. It would be here the virtual monitor could display the Morse Code chart with the potential for experimentation into creating new features further into development.

The final puzzle design before development would consist of a flashing LED in the physical room that displays a Morse code message. It would then be up to the player using the VR headset playing the virtual room to view a morse code chart. At this point team members in the physical room would communicate with the player in the virtual room to solve the morse code and then input the code into a GUI touchscreen in the physical room. This would then release an electromagnetic lock opening a door.

**Final Design**

During development, there were a few issues concerning the requirements and how the puzzle design could run and produce 2 difficulty levels for the user to choose from. This would mean that changes were made during the project so that these requirements could be fulfilled whilst remaining true to the previous concept. Two things were decided at this stage:

1. The LED light would be substituted to have an LED strip.
2. The difficulty levels would be decided by the GUI.
3. The VR would display a puzzle that would convey numbers and colors being symbiotic.

The LED strip would be used as a means of sharing colors to the user. There would be a total of three colors, each color would represent a number, red being 1, green being 2 and blue as three.

A color sequence would be created at random producing five colors that would translate to their designated number. It would be then up to the user to use the VR application to figure out how to translate the meaning behind the flashing colors.

The GUI would remain the same using a difficulty setting discussed in Design 2. The easy setting will allow the user to simply input the numbers whilst the hard setting would require additional functionality. This would mean then that one button could change multiple numbers forcing the player to figure out a sequence of button presses in the correct order.

**Functional and Non-Functional Requirements**

### **Functional Requirements for Raspberry Pi 3 (Client):**

1. **GUI Display Requirement**:

* The system must display a GUI on a 3.5-inch touch screen.
* GUI should allow user input for a 5-digit code.
* Provide visual feedback for connection status and validation responses.
* **Communication Requirement**:
* The system must establish a socket connection with the Raspberry Pi 4.
* Transmit the 5-digit code over the connection.
* Handle errors or exceptions during communication.

### **Non-Functional Requirements for the Client (Raspberry Pi 3 with GUI):**

1. **Usability**:

* The GUI should be intuitive and user-friendly, allowing users to enter code with minimal confusion.

1. **Performance**:

* The GUI should respond promptly to user touch inputs without noticeable lag.

1. **Reliability**:

* The client should consistently establish a connection with the server

1. **Maintainability**:

* The client software should allow for updates and modifications without causing disruptions to its basic functionality.

### **Functional Requirements for Raspberry Pi 4 (Server):**

1. **Communication Requirement**:

* The system must accept socket connections from Raspberry Pi 3.
* Receive and validate the 5-digit code.
* Send appropriate success or failure responses.
* System serves the guessing game

1. **Lock Control Requirement**:

* Upon receiving the correct 5-digit code, the system must release the solenoid lock.
* Handle any failures in releasing the lock appropriately.

1. **Morse Code Display Requirement**:

* The system must continuously flash a light in Morse code, representing the correct 5-digit code.
* Ensure that the Morse code flashing does not interfere with other system operations.

1. **Logging and Monitoring Requirement**:

* The system should log key events in the form of print statements to be viewed in the terminal for monitoring
* Provide means for monitoring the status of the system, such as the connection status.

1. **Error Handling and Recovery Requirement**:

* The system should handle unexpected errors and recover from failures.

1. **Responsiveness and Performance Requirement**:

* The system must respond to user inputs and commands in a timely manner.
* Ensure that simultaneous operations (e.g., Morse code flashing and communication handling) do not cause significant performance issues.

### **Non-Functional Requirements for the Server (Raspberry Pi 4):**

1. **Performance**:

* The server should handle multiple client requests without causing performance lags.
* The Morse code display in the VR headset should be real-time with minimal latency.

1. **Reliability**:

* The server should have high uptime, ensuring that it's consistently available when the client attempts to connect.

1. **Maintainability**:

* The server software should be modular and well-documented to facilitate updates or troubleshooting.

1. **Interoperability**:

* The server should be compatible with different versions or models of Raspberry Pi

1. **Safety**:

* The lock type must be safe for young children

1. **Responsiveness**:

* The server should quickly process the received code and send back appropriate responses to the client.

### **Functional Requirements for VR Display:**

1. **VR Hardware Compatibility Requirement**:

* The system must be compatible with the specific VR headset model being used.
* Ensure a consistent display resolution and refresh rate suitable for the VR headset.

1. **VR Initialization and Calibration Requirement**:

* Upon starting, the system must correctly initialize and calibrate the VR environment.
* Provide the user with clear instructions or guides for adjusting the VR headset for optimal viewing.

1. **Visualization Requirement**:

* The system must display the puzzle hints in a visually clear and easily discernible manner within the VR environment.

1. **User Interaction Requirement**:

* Implement intuitive controls or gestures for user interactions within the VR space that don’t require controllers.

1. **Safety and Comfort Requirement**:

* Ensure the VR display doesn't cause discomfort, dizziness, or nausea.
* Include regular breaks or reminders if continuous viewing is expected for long durations.

1. **VR Environment Aesthetic Requirement**:

* Design the VR environment to be immersive and free from distractions.
* Incorporate thematic or aesthetic designs that complement the puzzle visualization.

1. **Audio Integration Requirement**:

* Ensure that audio is spatially accurate and synced with the visual display.

1. **Performance and Responsiveness Requirement**:

* Ensure that the VR display maintains a consistent frame rate to avoid user discomfort.

1. **Exit and Transition Requirement**:

* Ensure that transitions between VR and non-VR modes are smooth and clearly communicated to the user.

### **Non-Functional Requirements for VR Display:**

1. **Usability**:

* The VR interface should be intuitive and user-friendly, requiring minimal training for users.
* Icons, text, and visual cues in the VR environment should be easily discernible.

1. **Performance**:

* The VR display should maintain a consistent frame rate, preferably at or above the recommended rate for the specific VR headset.
* The system should have a latency of less than 20ms to ensure real-time interaction.

1. **Reliability**:

* The system should be highly reliable, meaning it should be functional almost every time it's used.

1. **Maintainability**:

* The software should be modular and well-documented, allowing for updates, bug fixes, and enhancements without major system overhauls.

1. **Interoperability**:

* The system should be compatible with different models or brands of VR headsets.

1. **Durability**:

* The VR environment should remain stable and not crash during prolonged usage.

1. **Portability**:

* The system should be adaptable to different VR platforms or operating systems with minimal modifications.

**Architecture Design & VR Experimentation**

**VR Design & Experimentation**

For this project the decision was made to develop the VR element using A-Frame. This was due to a variety of reasons, one being the unnecessary complexities involved with using other more recognized software like Unity. A-Frame has all the necessary components to produce a fully working VR world that is easily accessible to the user with functionality on a vast majority of headset brands.

A-Frame has its own learning space available on its main webpage called, A-Frame School and it is here that I was able to learn the basics of Web VR and fully understand how it is possible to build a basic world that could be later enhanced to fulfil my requirements.

**Hello, Web VR**

To get started with learning to use A-frame, I was able to take advantage of the A-frame School. It shared step by step lessons that would explain how to get the most from the framework as part of an interactive course for Web VR.



(Hello, Web VR Tutorial, Image 1)

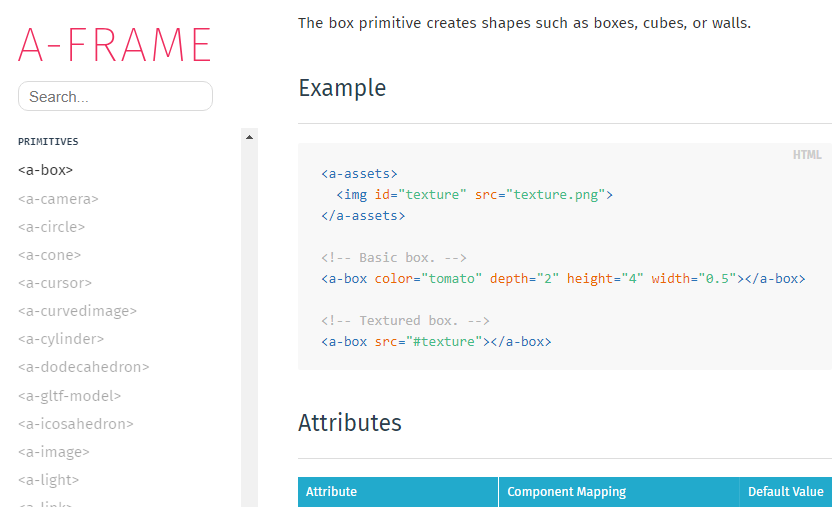
From here I was able to take the lesson and open it using an online platform called glitch. Gitch provides a space to host projects without downloading any additional software. It offers developers the chance to instantly remix (edit) and create projects so that the user can alter them to their liking and save them to their own personal user account to be opened and edited at any stage. The platform is completely free to use and all that is needed to set up a new VR environment with A-Frame.

Glitch was an incredibly useful tool for learning especially when going through each of the tutorials, with each one I could view the source code to avoid continuously copying and pasting from the A-Frame library. All I had to do was click the Remix button, and it would transfer the code into a new project so that I could learn from it.

To include A-Frame within a new HTML file a <script> tag should be inserted pointing to the CDN build, this will be included when cloning (remixing) any tutorial or example project.

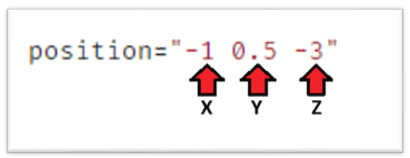


Upon cloning the first lesson and renaming the project, I was introduced to `Primitives`. These would allow me to render simple objects such as boxes, spheres, cylinders, the sky and the floor. Within each of these primitives are attributes which can be specified and altered, such as position, rotation of the object, scale and color. Within the A-Frame schools' main webpage they have their own sub-section that lists all the available primitive types and the attributes that can be specified for that type. These can be viewed in image 2 below:



(Reference, Image 2)

The first thing that I was able to learn was how to change the position of an object, which proved to be difficult at first as the coordinates in A-Frame for objects are defined by X, Y, Z. ClareCreate was able to explain via her YouTube channel that co-ordinates have both positive and negative directions and how each of these would impact the position of any primitive object.

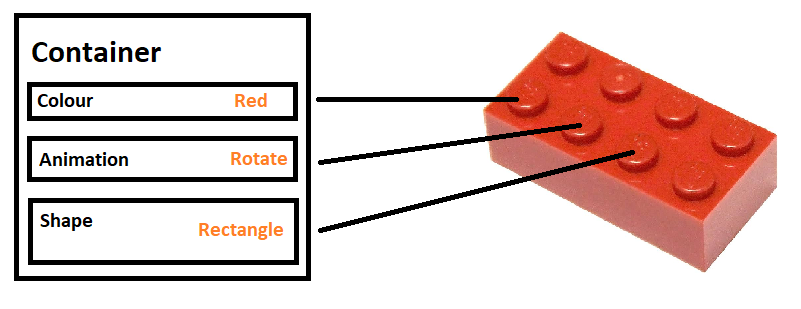


(Reference, Image 3)

The next step was to grasp what A-Frame is and how it runs as an Entity Component System Architecture (ECS). ClaireCreate was able to explain that *`ECS is like a box of Lego’s, the set of Legos itself is the framework and the ECS is the little bumps and cavities that allow you to put the Legos together. You can then use these cavities and bumps to make endless models of Lego` (Reference).*

This was a perfect analogy as to how A-Frame works. The ECS architecture is made from entities, components and system.

An entity is an empty container, it holds reusable modules that are the components that will define what the entity will look like and what it possibly can do. It is important to note that it is needed to specify the type of component and also the properties of that component.

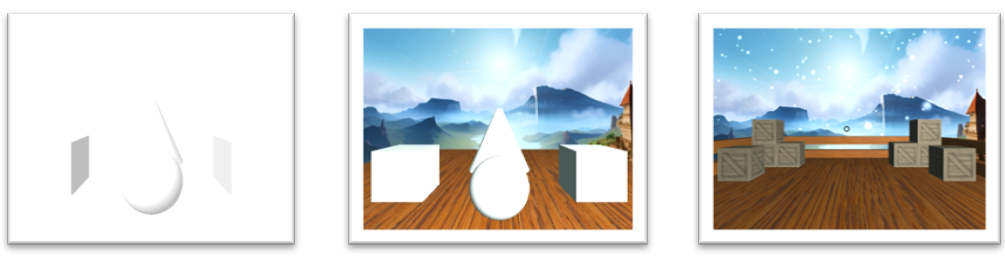


(Reference, Image 4)

Syntactically this is the same as adding an attribute to a HTML tag with the many properties that can be specified in the same fashion as when writing in CSS. When creating an entity this will sit between the `*<a-scene>*` tags similarly to the primitives as everything that is placed between these tags are considered an entity and primitives are no different, being entities with pre-determined components.

I found that when initially learning to use A-Frame using these primitives made it a lot easier for me to pick up how to use the framework as a first-time user continuing to use them throughout most of the development of my VR world.

Upon learning all these new techniques I was able to create my first VR world, building a small plane with a surrounding landscape and sky that held four shaped objects. Everything that can be seen below is built using primitive types that could be manipulated to display a small platform holding objects in images 5, 6 and 7.



(Reference, Image 5, 6, 7)

The changes between images 5 & 6 are very noticeably different as textures have been added to the plane and sky really highlighting the shaped objects that now sit comfortably on the plane.

It is important to note that the image used for the sky was one taken from Blockade Labs, where they have created a tool called Skybox AI. This will take prompts in the form of text descriptions that will then create a 360-degree image based on the user's creative vision. This is an amazing new concept as it features the ability to change a description based on style, genre and remixing existing prompts to create completely new results.

In Image 7 the box primitives have been given their own textures so that they replicate wooden boxes. The background has added animations that signify snow falling from the sky alongside a light setting that allowed the objects to produce shadow and reflections.





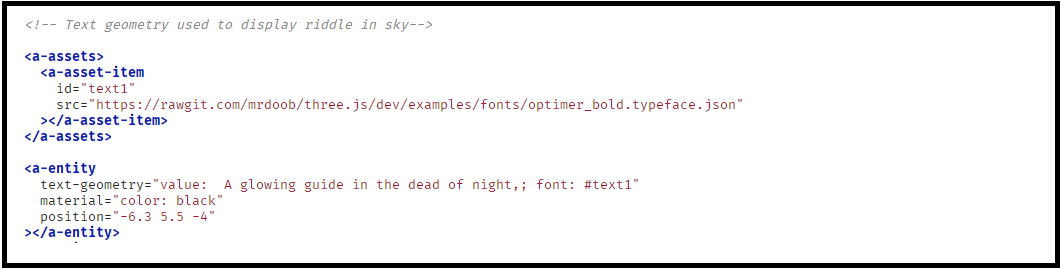
(Image 8, 9, 10,11)

The above images show my first VR puzzle that would collaborate with the flashing LED puzzle. The user would read the riddle then turn 180 degrees to find nine rotating boxes. Each box would display an image as a possible answer to the riddle when the cursor hovered over a box. The user would then click the box to reveal if the answer is correct with one of the Pico controllers. The correct box would then display a large Morse Code chart signifying to the user that Morse code should be used as part of the puzzle in the physical room.

Please note that it is assumed at this point the LED would be eventually updated to a bulb that could be placed inside a household lamp.

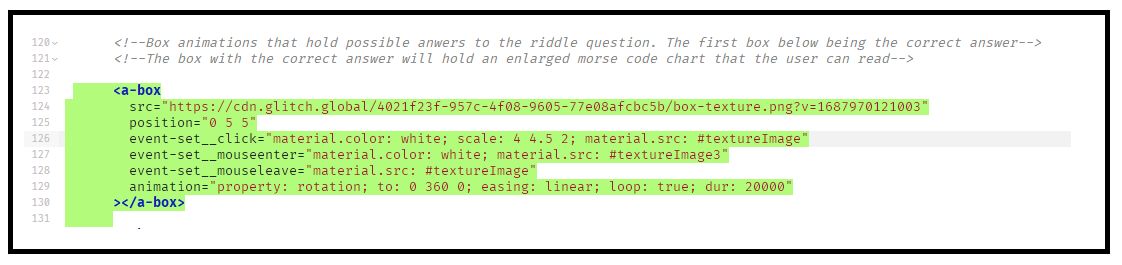
To create the floating text that would display the riddle, I began by creating a new entity. An <a-assets> tag would be used as a way of defining a text type that could be used multiple times with the id #text1. The `src` attribute would refer to a JSON font file from a list of Three.js examples for the type “optimer\_bold”.

Within the text entity I could use text-geometry as explained on A-Frame school to render text in 3D and define the properties of the text. The property `value` was used to set the text as can be seen in the image below.



(Image 12)

The box primitives were animated to rotate in mid air and act as an animated element for the user to interact with as part of an immersive VR experience. To understand how this was achievable it is essential to look at the code used for one of these primitive objects.

(Image 13)

Here we can see the texture for the box primitive being set to an image saved in the assets folder that’s included as part of a new project so that the PNG image can be hosted by Glitch. The position has been set so that it’s placed 5 units above the plane level and 5 units forward from its origin point. Events have been set up to allow the image to change once the cursor hoovers over the box (mouseenter), once the cursor leaves the box (mouse leave) and lastly when the user decides to click on the box (click). The animation determines the rotation property whereby the box will rotate around the Y axis from 0 to 360 degrees. As the loop is set to true this will loop indefinitely and complete a full rotation in 20,000 milliseconds (20 seconds).

When reviewing this completed VR puzzle it was important to ask myself a number of questions:

1. Should controllers be used as part of the experience?
   * They could be easily damaged during use, especially with young children.
   * They present the option of exiting the VR application accidentally which could confuse the user and impair the immersive experience.
2. If a controller is not used, how would the player interact with the environment?
   * Is there an alternative action whereby the user could use a gaze function to click on potential objects?
   * Could the player possibly not need to click on items and just hoover over them?
3. Does the Morse Code chart contain too much information?
   * The chart could possibly be displayed with reduced content, only displaying numbers.

From here I decided to review the design later in development upon completing the completed raspberry pi section design. This was so that I could decide if changes were needed to make the puzzle more accessible for multiple age groups as a whole and at a reduced difficulty level. It was clear at this point that my final design may need to be updated at a later stage.

**Raspberry Pi Set UP**

When setting up the Pi3 and Pi4 they required a blank micro-SD card that could be formatted using the Raspberry Pi Imager APP, enabling me to have a working desktop upon turning on each Pi computer. This would then give access to the Thony IDE which would be the main platform used for the majority of coding throughout the project.

**Touch Screen**

When initially attaching the touch screen it displayed a white screen, realizing additional set up was required I turned to YouTube and found a clear and concise approach to quickly setting up the screen, this would be paired with the `*Instructables*` Webpage where it had a step-by-step manual. Connecting the Pi 3 to the monitor allowed me to open the terminal and clone the necessary settings from GitHub using a set of commands provided by the video and associated article. The decision was made to use the Pi3 for the touchscreen, putting aside the Pi4 to interact with the future circuit that would be required.

**User Interface**

When trying to understand how to build a simple interface for the puzzle I used PySimpleGUI, a python module that was suggested by a well-known YouTuber alongside the `*Cookbook*` section of the PySimpleGUI main webpage, where it went into further detail. *The PySimpleGUI Cookbook is meant to get you started quickly. But that's only part of the purpose. The other, probably most important one, is coding conventions. The more of these examples and the programs you see in the Demo Programs section on GitHub, the more familiar certain patterns will emerge. [Reference number].*

It was using the Cookbook that I was able to understand how to use buttons effectively, change the theme of UI and modify the UI as the project developed.

In the below image we can see a simple design and color scheme that allows the numbers to easily stand out with each corresponding button that will increment each number by one until it reaches nine, then it will return to zero.



(GUI first draft, Image 1)

Should the code be presented here to display some for inner workings of the design decisions.

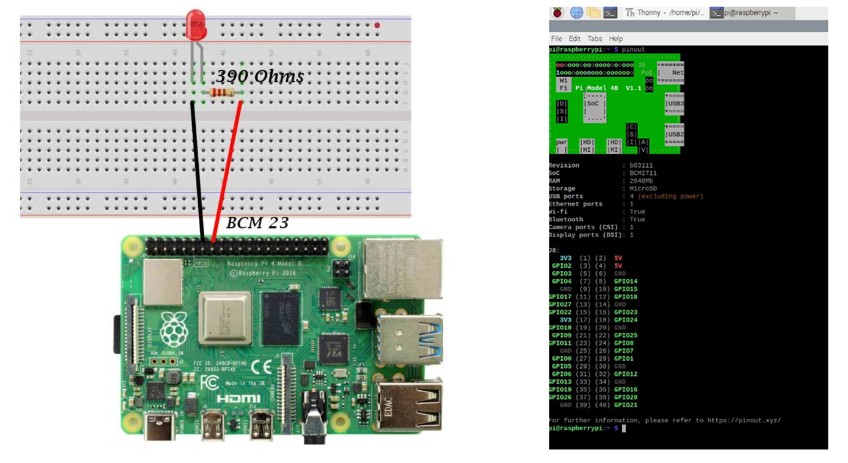
The numbers that can be seen at the bottom of the image reflect the code that the user will enter to solve the completed puzzle as a means of allowing the developer to easily test the code throughout development and would be removed upon completion. **(Test Case 1)**

**Blinking LED & Morse Code Integration**

To allow for a light to flicker and resemble morse code the first phase for development would be to figure out how to blink an LED light. This would essentially be my first mini project using a Raspberry Pi.

I began by following a tutorial article published by Robotica DIY that would claim, *“once you learn how to blink LED, you can literally control anything from LED to Air-Conditioner". [Reference No]*

After installing the relative libraries, the first step was to connect the circuit, connecting the wiring to the GPIO as it's directly connected to the CPU of the Pi. To find the correct BCM number and Board number located on the Pi as a beginner, there were additional terminal commands available to assist with this process.



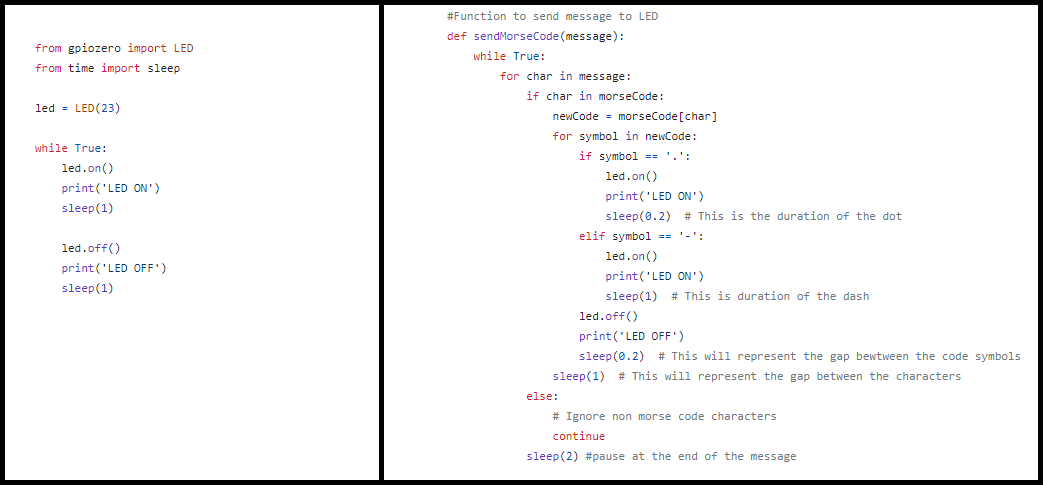
*(Led Set up on Pi4 & terminal command with result for board numbers, Image 2)*

Importing LED from the GPIOZERO library allowed me to create an LED object in the Thony IDE and using a while loop I could set the LED to `on` and then `off` separating them with a sleep module.

The second step would be to update the code so that the blinking light would send out a message that could be understood using morse code. For this to work I began by mapping a dictionary of characters to its retrospective morse code string. From here I wrote a function that would effectively reuse the previous blinking LED code with a range of updates.

This function would initially loop through each character in the message and check to see that the character is within the morse code dictionary of characters mapped at the start of the application. If located, it will retrieve the morse code string for that character. From here the function will loop through each of the symbols and the LED will flash accordingly. The morse code `Dot` represented by the LED remaining on for 0.2 seconds and the `Dash` by the LED remaining on for 1 second. After each represented character there is an additional pause of 1 second. **(Test Case 2)**

Each code representation for both the blinking LED and updated morse code function can be seen below.



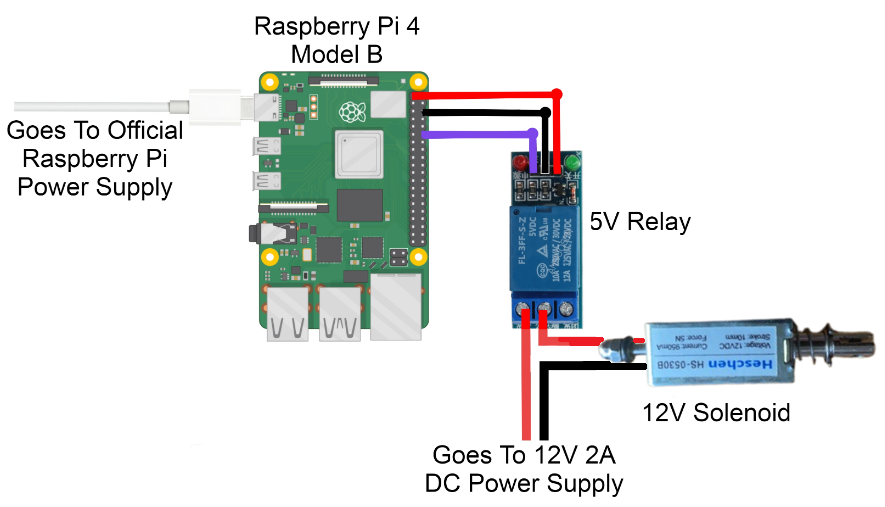
*(Original flashing LED code (left) & updated code as Send Morse Code function, Image 3)*

**Solenoid Lock**

When reviewing a working lock mechanism, I first wanted to understand how to control a 12-volt Solenoid with the Raspberry Pi 4. Core Electronics provided a cohesive tutorial on its main webpage explaining that solenoids are an immensely useful way to turn electrical energy into a linear motion.

As the 12-volt solenoid needed would break the Pi if connected directly, a relay was introduced by the suggested teachings of Core Electronics to protect it. The reasoning behind this is due to the maximum voltage a Raspberry Pi is able to take, 5 volts. The relay can then provide a safe connection for communicating between the Pi4 and the solenoid.

Below you will see an image displaying the final build with each item labeled used for this section of the project.



*(Solenoid Lock build with relay, Image 4)*

When establishing the code for this piece of circuitry Core Electronics were able to provide the main body of code that would demonstrate the solenoid lock simply locking and unlocking on a continuous basis. The GPIO 18 pin on the Raspberry Pi was made the output pin, following this a while (true): statement was set up, starting by setting the GPIO 18 Pin to high, which in turn sends out the maximum voltage (Approximately 3.3 volts). As explained in the Core Electronics tutorial page, *`this voltage will not trigger the relay and thus the solenoid will remain deactivated`.* After a brief pause using the sleep function, *`the GPIO Pin 18 is turned to low, thus sends out the min voltage. This will send out the lowest voltage an output pin can. This is approximately 0 Volts). This will trigger the Relay to switch on thus activating the Solenoid*`. **(Test case 3)**

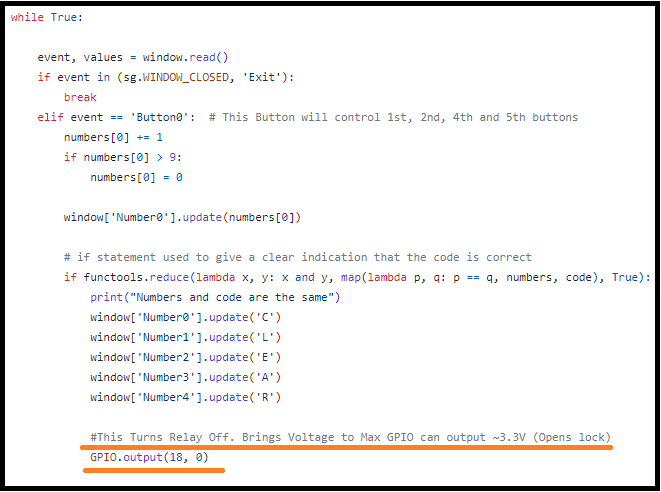
It is important to note that once the code stops running whilst the solenoid is active, it will remain active, and this could cause potential issues. When the solenoid is active for longer than a few minutes it produces quite a lot of heat externally making it difficult to physically touch the solenoid lock. As this project is one which is aimed at children, this may not be suitable within a learning environment for young children and therefore other types of lock should be tested for the final build.

**Solenoid & GUI Integration.**

When integrating the solenoid with the GUI the first question that needed to be answered was when do I want to activate the relay?

This was relatively straight forward as a means of updating the current code for the UI by installing the relative libraries once again for the relay to run, then setting up the GPIO 18 Pin so that it sets the lock.

The second question was, when do I want to deactivate the relay? This could be done by adding the GPIO output to the already completed if statement that would let the user know the correct code was completed. This is highlighted in orange in the image below.



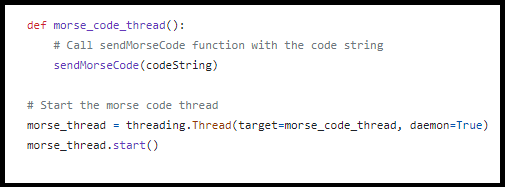
(Reference, Image)

As the above piece of code demonstrates, it is checking whether the list of numbers displayed on the GUI matches the secondary code list. If they match the GUI elements will update to read, `CLEAR`. It is below this line of code I have placed the *GPIO.output(18, 0),* where it will interact with the relay as it sets the GPIO Pin 18 to 0 signifying that the lock is now deactivated.

**Morse Code, Solenoid & GUI Integration.**

To complete the basic functionality of the design, I needed to find a way to allow the LED to flash in the background as the user would attempt to use the GUI to complete the puzzle. The issue here would be that when adding the Morse Code functionality, it would run as an infinite loop before the GUI event loop therefore making the GUI unresponsive (**Test Case 4).**

To fix this problem threading was introduced, creating a separate thread that would run the morse code functionality. This in turn would allow the morse code to be sent concurrently whilst the main event loop listens for each of the button events.



(reference, image)

Additionally, the *`daemon = True`* argument will ensure that the thread is properly terminated when the main program is exited.

**LED Strip**

During the development of this project changes were made, hardware was upgraded, and design changes were made to work with the new hardware to have a complete working puzzle. As explained in the Design Phase section under *`Final Design`,* the LED was upgraded to a LED strip otherwise known as a WS2812B Strip.

As for the next steps, I had to understand how to turn on the strip and control it on its own. I was able to review an article provided by Core Electronics to dig deeper into the subject. They were able to explain that *`the name WS2812B Strip usually refers to a long length of flexible PCB with many specific RGB LED Nodes evenly spaced and dotted along the top side of the PCB. Furthermore, WS2812B strips are fully addressable. This means each RGB LED node can display a different color and intensity than its Neighbours.*

The following packages first had to be installed onto the Raspberry Pi OS so that it could operate the LED’s:

* sudo pip3 install rpi\_ws281x
* sudo pip3 install adafruit-circuitpython-neopixel
* sudo python3 -m pip install --force-reinstall adafruit-blinka

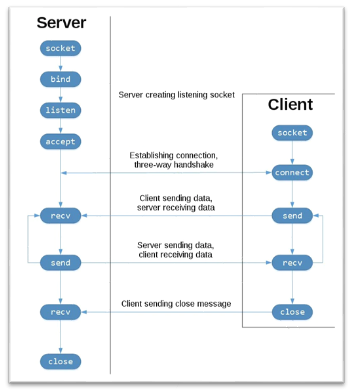
These would directly power and drive the strip allowing me to control them using Python in the Thony IDE.

**Socket Implementation**

As part of the project, I wanted to allow the user interface to act as its own application that could communicate with the server independently from a 3.5-inch touchscreen, as opposed to the player using a monitor, keyboard and mouse. This would make the UI feel like one complete unit, whilst benefiting the gamemaster also, as there would be less equipment to be concerned with and present more space within the escape room.

To achieve this, I experimented with sockets as a way of sending messages across a network. This method would provide inter-process-communication where a local network to the computer could be set up using a WIFI connection or ethernet cable.

To understand this in finer detail we can look at a sequence of socket API calls and data flow for TCP below:



(Insert reference, Image 1))

The server here is represented on the left collaborating with a listening socket that will listen for a connection from a potential client. Once connected with the client the server will `accept` to complete the connection.

The client will then request a connection to the server as part of the three-way handshake, which is crucial as this will determine if a viable connection is reachable from each side. As can be seen in the middle of the diagram this is where data is exchanged between the server and client, the bottom also showing each corresponding socket being closed after use.

This allows for the creation of a basic implementation called an Echo Client and Server, whereby the server will echo what it receives returning it back to the client. This was the first step into understanding python socket modules and how they work.

(Insert Reference, image 2)

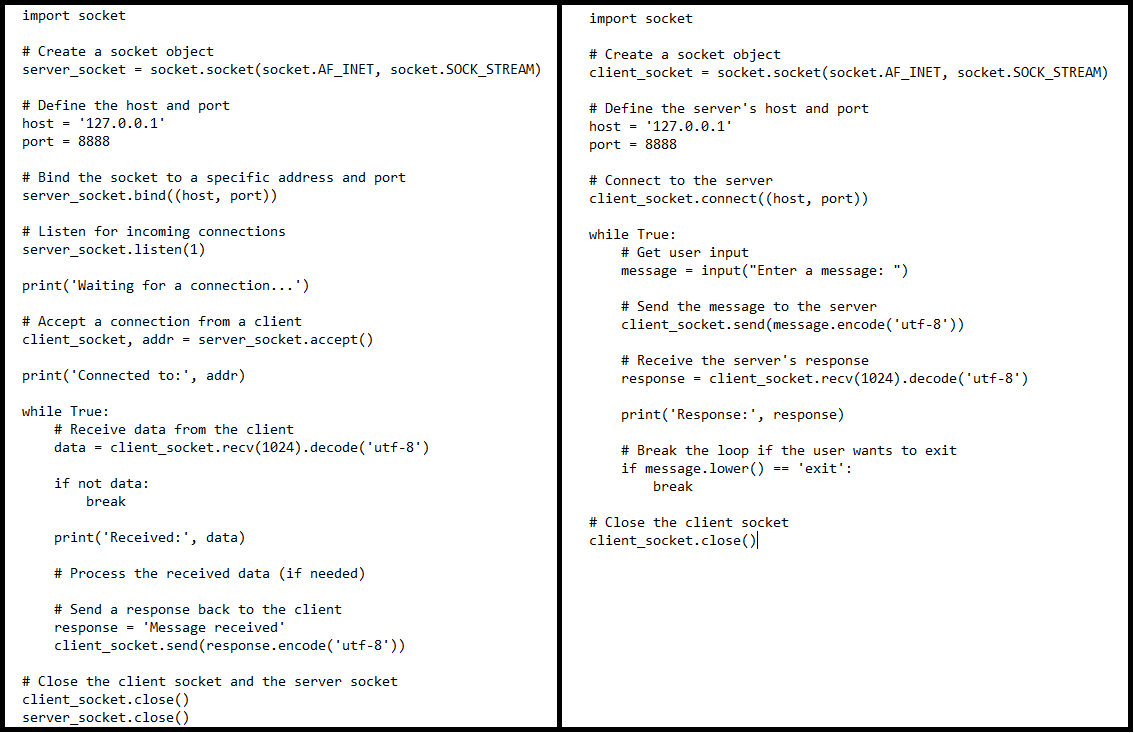
Starting with a basic socket server and client I created a socket object specifying the socket type, thus creating a default protocol called the Transmission Control Protocol. This was an acceptable starting point as TCP is reliable and has in-order data delivery meaning that the application is able to read data in the order it was written by the sender.

The script above demonstrates the echo server using the socket library allowing Python to interact with its networking functionalities. The two constants are used to specify the address family collaborating with the bind method so that they are bound to the socket via a unique IP address and port number.

It is important to understand that at this point the server and client socket connections are running on the same machine, therefore the IP address is set to `127.0.0.1`. A key point to note is that the client itself does not necessarily bind to this IP address or Port the same way the server will. The client instead uses them as its destination when confirming and establishing a connection.

The client server should then be able to send 5 bytes of data (`b”1234” `) to the server, upon receiving this the server will then echo it back to the client. The client waits to receive the data from the server, therefore receiving the same 5 bytes originally sent. A clear reflection of the previously mentioned three-way handshake.

The next step in the development process would be to then send a personalized message to the server from the client and for the client to then receive a response. In order to explore this further Chat GPT, was used to restructure the code from image 2 and create an example as to how this could be implemented.



(Reference, Image 3, server (left), client (right))

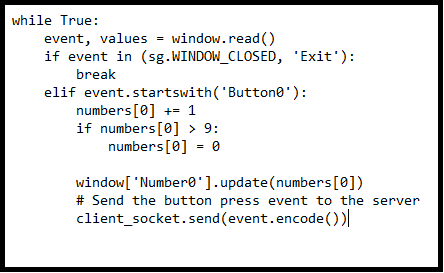
The image above displays the revised code generated by the AI. The code here is very similar, however exhibits some changes in order to achieve the desired outcome. The server is placed within an infinite loop, meaning it will continuously wait for messages from the client and respond to them indefinitely. This is useful as it could be implemented to the user interface for this project and enhanced with a condition so that when met the puzzle is complete. Noticeably on the client side, the user is given the option to input within the communication loop as the user is prompted to enter a message that will be sent to the server after encoding it to bytes. This also proves useful as the same concept could be applied to the UI when sending number solutions from the PySimpleGUI application.

From these learning outcomes I decided to experiment further by seeing how they could be implemented into my current code. This would have to be done in incremental stages to fully understand the inner workings of a working socket connection and how it could be inserted efficiently and effectively.

**Solenoid control**

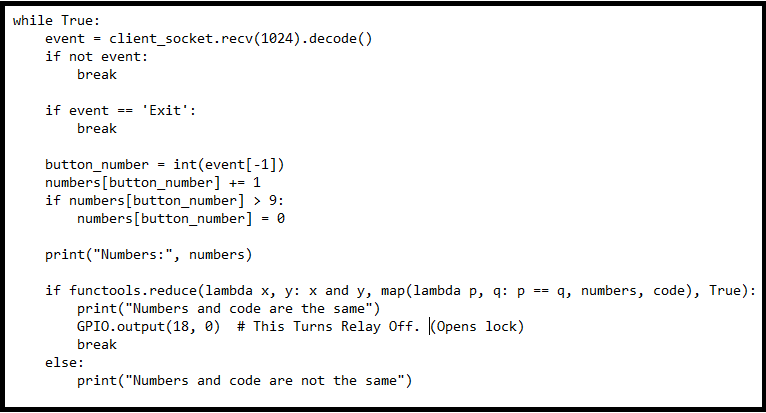
When applying the lessons learned previously attained the first step was to strip the code back so that the message sent on the client side could control the solenoid lock. The code will mainly stay the same from the previous example until we reach the while statement for both the client and server side.

The client side will use the same keypad functionality from the PySimpleGUI application with the event being sent to the server using `*client.socket.send*`. This will be implemented for each button press as it is now sent to the server. An exit button is included so that the application will continue to run until it has been pressed. This is demonstrated in the image below.



(Reference, image 4)

In image 5, an infinite loop is created so that it actively listens for incoming events from the client side. The event as we already know is represented by a button press. The server will then check that the current `numbers` list matches those of the randomly generated `code` list and if there is a match, the lock is then opened.



(Reference, Image 5)

To see if the correct numbers were being sent through correctly by the client-side, print statements were added to each button iteration so that the results of each button press could be seen in the terminal. (Test case, numbers list couldn’t be randomized. Hard coded to 0,0,0,0,0, so that they were always the same on both sides).