Design Document

Chime-In

Date: November 2, 2021

Written by: #160

Operational Concept Description

Chime-In is an interactive, memory and speed-based game guaranteed to provide an enjoyable experience for all players, no matter their age. To maximize Chime-In's flexibility, we designed the game to be hand-held and wireless so that the player can interact with it throughout all areas of the escape room. This design resulted in a small, compact box with four different coloured light-up pushbuttons on the outside and all internal circuitry (Nucleo Board, battery pack, and wiring) enclosed within and secured to the bottom plate. To allow for optimal replayability and customizable difficulty, the escape room operator is provided with multiple top plates, which each offer a different button layout for Chime-In. Since all of the major electronic components are attached to the bottom plate, the operator can simply detach the pushbuttons, lift off the top plate, and reattach a new top plate with a new and unique button layout.

When the player initially discovers Chime-In within the escape room, the system will be in "idle mode," where the four light-up push buttons will be flashing in a pattern meant to grab the player's attention. To initiate Chime-In, the player is required to enter a four-colour sequence code that they received from a previous challenge in the escape room. Once they enter the correct code, Chime-In will begin displaying a randomized series of different coloured flashes, which the player must replicate using the corresponding buttons. As the game continues, the colour sequence grows longer, and the speed of the lights increase. If the player fails to enter the sequence quick enough or enters the incorrect sequences, Chime-In will indicate failure by flashing its lights three times, waiting ten seconds, and restarting the game back to the first sequence. The player is expected to continue playing until they have successfully completed the required number of sequences (set by the operator). Once completed, Chime-In will indicate that you have finished the puzzle by a celebratory sequence of light flashes, and the player is free to complete the next puzzle.

Design Specifications

Hardware

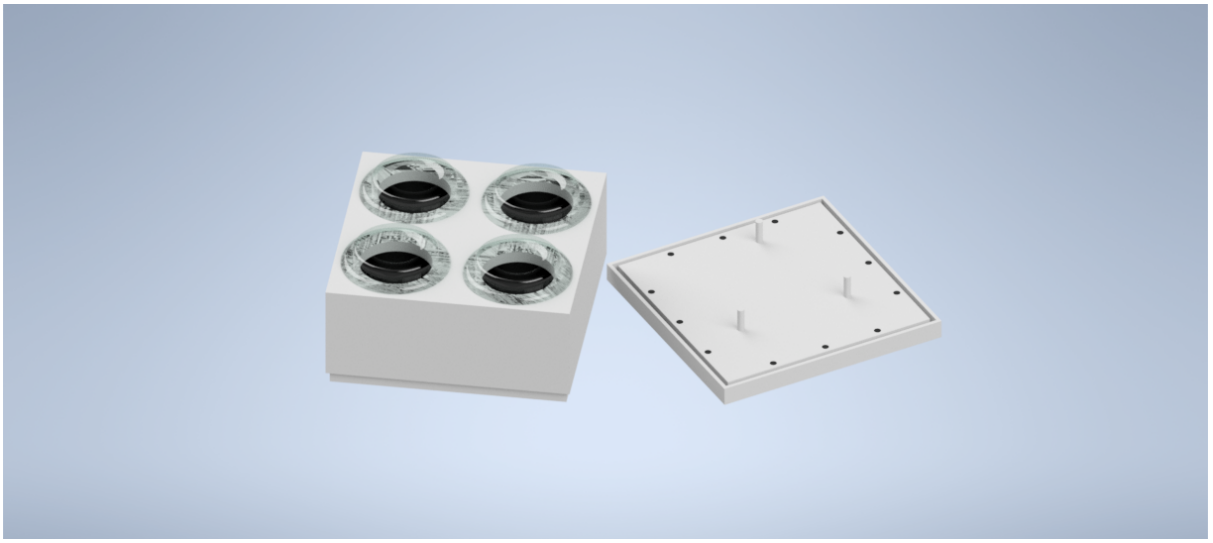

Figure 1: Overall Design

Our hardware setup is quite simple yet developed to be as flexible as possible so that a large number of game modes can operate on the hardware platform. The design consists of a base plate (Figure 2) and a series of interchangeable top plates (Figure 3). By using this design method, the game hardware is contained within the base while still allowing the button layout to be customized and upgraded using various top plates.

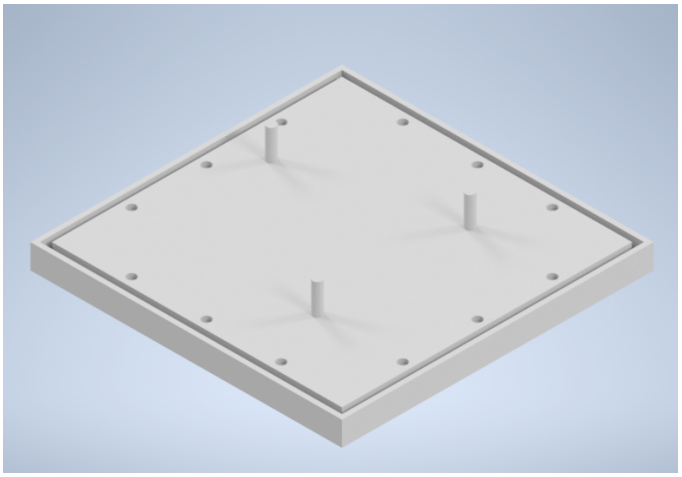

Figure 2: Base Plate

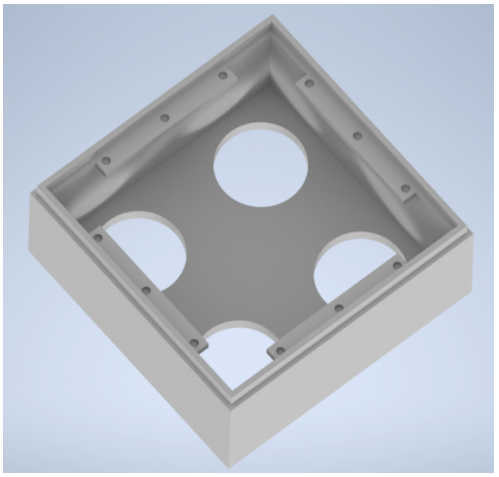

Figure 3: Top Plate

We will begin by looking at the base plate. Figure 4 shows the base plate design and specifications.

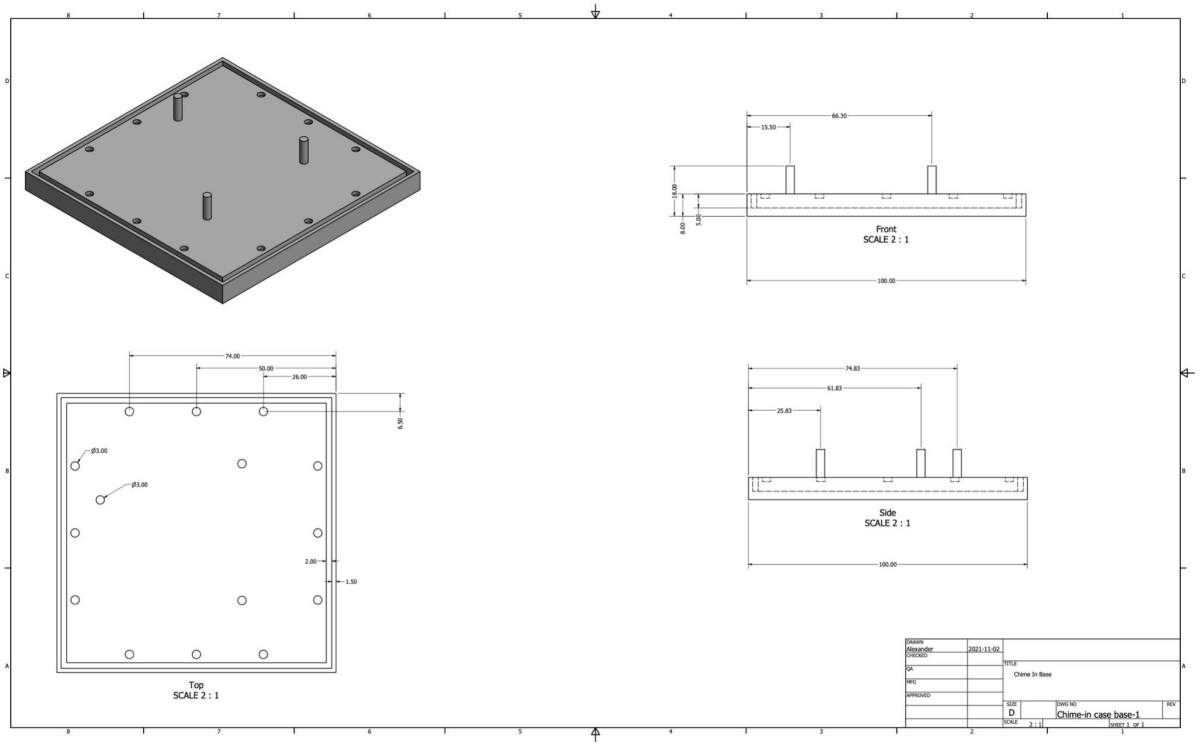

Figure 4: Base Plate Specifications

The base plate is a solid piece of 3D-printed plastic and has a groove around the edge so that the top plate can be fitted into it. Twelve magnets are recessed within the base plate and connect to magnets in the top plate to hold the whole system. The middle of the case contains a series of pegs that hold the STM32 board up allowing for the battery pack to fit underneath. The IO that is required to drive the buttons and LEDs in the top plate connects to the bottom plate and the STM32 using a series of Pogo Pin connectors. These connectors include one pin for ground, one for voltage, one for the Neopixel LED light data, and four for the buttons IO. There are also an additional three pins that are used to identify the top case. This makes for a total of 10 connectors between the top and bottom cases.

Figure 5: An example of a Pogo Pin Connector.
Source:
https://www.newark.com/mill-max/858-22-004-80-011101/btb-connector-header-4pos-1row/dp/23AH3873

In terms of the electrical wiring within the base plate, the STM32 is connected to a battery power source (5V) using the Vin and GND pins. The board then connects to the pogo pin connector. A GND is sent over, as well as a 5V line to power the LEDs. The rest of the pins are connected to the board's digital IO. Of particular interest are 3 of the Pogo Pins labelled green in Figure 6. These pins simply

connect to ground on the case lid and the specific pins that are jumpered allow the board to have the capability to recognize what lid is being used. As there are 3 pins, the board can identify nine different case lids.

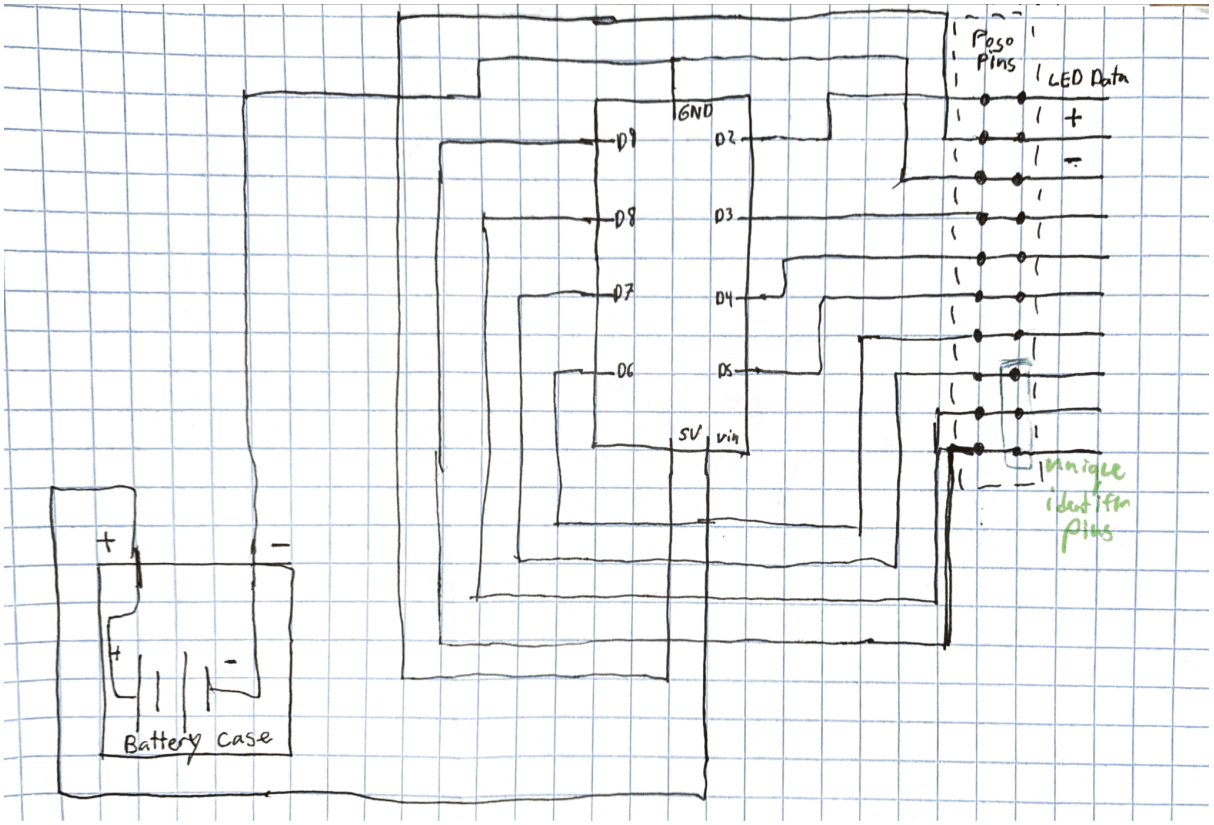

Figure 6: Base Plate Wiring Design

Next, we move on to the top plates. Each top plate contains the same components and only differs in the layout of these components. For our purposes, within this design document, we will be walking through a basic top plate with all buttons on the top. Each top plate is 3D printed and includes four buttons as well as four RGB LED lights. The LEDs are installed within the buttons by disassembling the buttons, using a 3D printed mount from Adafruit.com to hold the LED in place, and then reassembling the buttons (Figure 8).

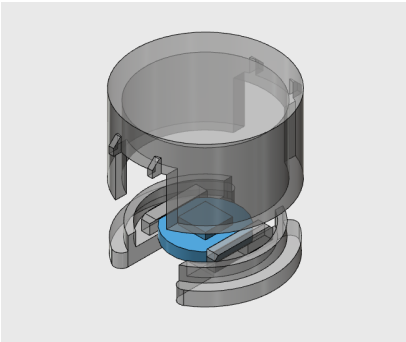

Figure 8: Switch Modification
Source: https://learn.adafruit.com/neopixel-arcade-button/3d-printing
The top plate is held in place using a ridge and 12 magnets that line up with the magnets in the case. See Figure 9 for detailed drawings and measurements.

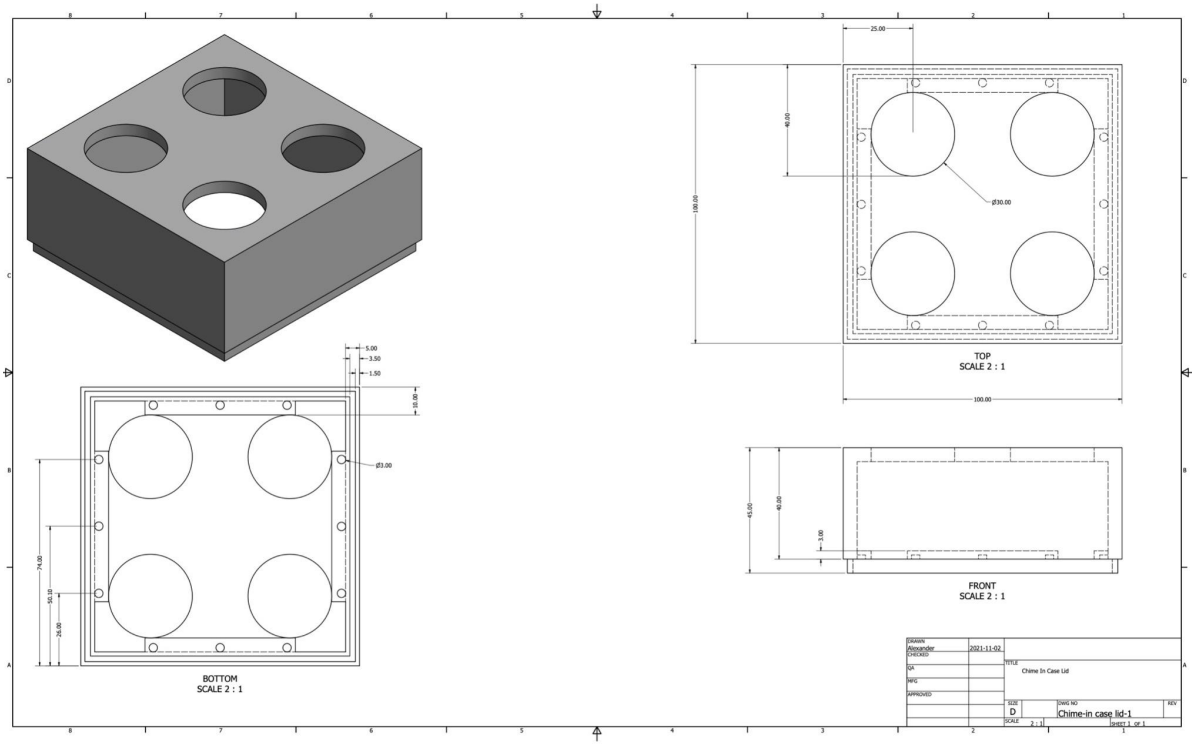

Figure 9: Top Plate Specifications

In terms of the electronics for the top plate, it is quite a simple setup. The LEDs are connected to a 5V power line and GND. They then are daisy-chained onto one data line (as they are addressable LEDs). The four switches are also connected to the data line and then to the corresponding data Pogo Pins. Lastly, the three pins left as unique identifiers are jumpered to identify the board. In this case, no pins are jumpered, identifying it as board 0.

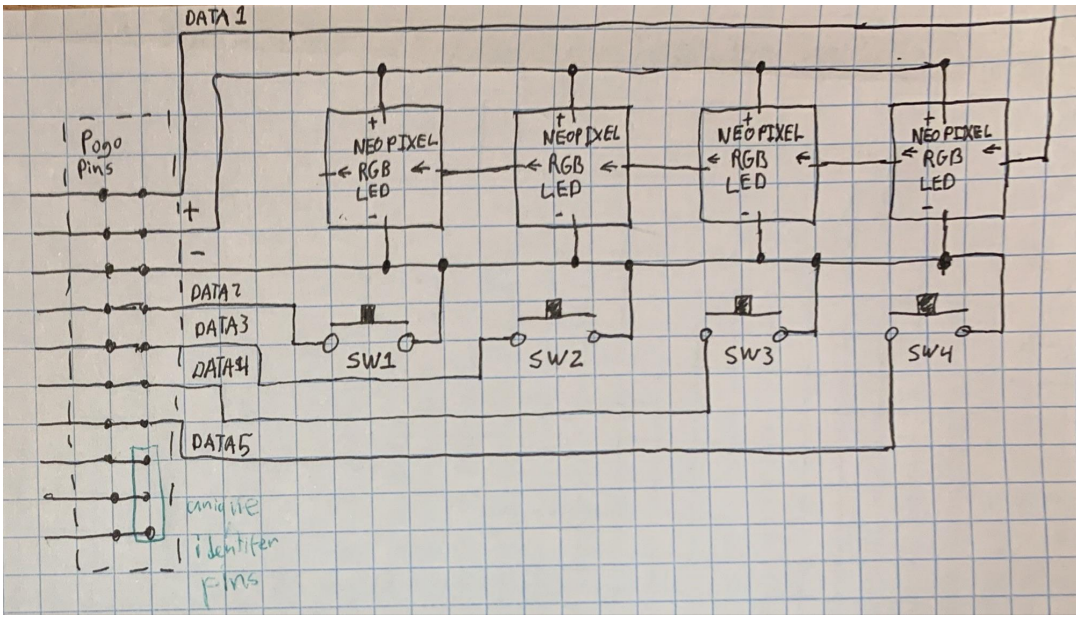

Figure 10: Top Plate Electrical Wiring

Software

Because Chime-In runs mainly on software, our team has thoroughly planned nine major functions in charge of the main functionality of the game.

1. idle
idle is a void function that receives no values. It is the first function that is called from main at runtime. Its purpose is to display different colours and sequences to attract the player's attention and ultimately get them to start the game. *idle* cycles through a randomly-generated colour sequence until it detects an input from any one of the four pushbuttons. Once an input is detected, *idle* terminates.

2. passcodeCheck
passcodeCheck is a function that receives no values and outputs a positive integer. *passcodeCheck* is called immediately after *idle* terminates. The purpose of *passcodeCheck* is to continue checking pushbutton inputs until:

A. The player inputs the correct 4-colour passcode (given by a previous challenge in the escape room, determined by global variable: *passcode*)
    i. If the player inputs the correct password, the four pushbutton lights will indicate success, and *passcodeCheck* will terminate with a return value of **1**.
B. The operator enters one of three of the following 4-colour special passcodes to change the difficulty of Chime-In[1]:
    1. Easy passcode (determined by global variable: *easy_passcode*)
        i. If the operator enters the easy passcode, the four pushbutton lights will blink green, and *passcodeCheck* will terminate with a return value of **2**.
    2. Moderate passcode (determined by global variable: *moderate_passcode*) [default difficulty at startup]
        i. If the operator enters the moderate passcode, the four pushbutton lights will blink orange, and *passcodeCheck* will terminate with a return value of **3**.
    3. Difficult passcode (determined by global variable: *difficult_passcode*)
        i. If the operator enters the difficult passcode, the four pushbutton lights will blink red, and *passcodeCheck* will terminate with a return value of **4**.

If none of the passcodes that the user enters correspond to *easy_passcode*, *moderate_passcode*, or *difficult_passcode*, the four pushbutton lights will indicate failure, and the function will restart. *passcodeCheck* will continue to loop through this operation until the user enters the correct passcode.

3. difficulty[1]
difficulty is a void function that receives a *difficulty_level* value from *main*. Depending on the received *difficulty_level* (2 = easy, 3 = moderate, 4 = difficult), the function will change the global variables that determine the difficulty of the game (i.e. the time between light flashes [double *between_time*], the time that Chime-In waits for a pushbutton input [double *wait_time*], and the amount of sequences the user must complete [int *sequence_amount*]).

4. randomNumber
randomNumber is a function that receives no values and outputs a positive decimal value from 0.0 to 1.0. This function creates a random number seeded from time, meaning it will generate a new value every time the function is called.

5. randomColour
randomColour is a function that receives no values and outputs a positive integer. *randomColour* is called every time *gameplay* needs one of four random colours to display to the corresponding

[1] The addition of the *difficulty* function and all related components will be added if time allows.

pushbutton. This function takes a value returned from the *randomNumber* function to generate a random integer from 1 to 4 (1 = colour #1, 2 = colour #2, 3 = colour #3, 4 = colour #4). This integer will then be returned from the function and *randomColour* will be terminated.

6. gameplay

gameplay is a void function that receives no values. This function runs the major game aspects of Chime-In, and is, therefore, one of the most important functions within the software. *gameplay* runs through a certain number of colour sequences (determined by the global variable *sequence_amount*)[2] and will continue to loop until the player successfully completes all of the sequences or fails. As the amount of completed sequences increases, the time between light flashes (determined by the global variable *between_time*)[2] and the time that Chime-In waits for a pushbutton input (determined by the global variable *wait_time*)[2] also increases.

7. deactivate

deactivate is a void function that receives no values. This function is called as soon as *win* terminates. The purpose of *deactivate* is to lock Chime-In and turn off the four pushbutton lights so that the player can no longer play the game. The only way for *deactivate* to terminate is if the operator inputs a special passcode determined by the global variable *restart_passcode*. When the operator inputs *restart_passcode*, *deactivate* will terminate, and Chime-In will completely restart back to its *idle* phase.

8. win

win is a void function that receives no values. This function is called when the player successfully completes all of the required sequences in *gameplay*. When called, *win* flashes the four pushbutton lights in a special sequence to indicate that the player has won. After flashing this congratulatory sequence for ten seconds, the function terminates and *deactivate* is called.

9. lose

lose is a void function that receives no values. This function is called when the player:

A. Fails to enter the colour sequence quick enough
B. Enters the incorrect colour sequence

When *lose* is called, it deactivates the four pushbuttons on Chime-In, indicates that the player has lost with a special light sequence, waits 10 seconds, and unlocks the buttons. *lose* then terminates, and Chime-In restarts from its first colour sequence.

Safety

As Chime-In is open to the public, maintaining a safe design is of utmost importance. To ensure that Chime-In is as safe as possible, we first performed a comprehensive risk analysis. Through this risk analysis, we identified two major hazards that could occur during operation:

1. The possibility of electronic short-circuiting which would lead to electric shock or fire.
    a. This could occur if Chime-In is used aggressively by the user, causing violent shaking and vibration.
        i. This intense movement could cause the internal electronics (Nucleo Board, battery pack, and wiring) to come loose and ultimately come in contact with one another, leading to short-circuiting.
2. The possibility of causing epileptic seizures.
    a. This could occur if the flashes from the four pushbuttons become too rapid.

[2] The addition of the *difficulty* function and all related components will be added if time allows.

While both of these hazards have a very low probability of occurring, it is still essential to mitigate the potential risks. The most efficient and cost-effective way to reduce the risk of the first hazard is to ensure that all of the electrical components are held securely in place on the base plate. As well, it is crucial for the operator to ensure that the base plate and top plate are adequately fastened so that Chime-In does not disassemble during use.

To reduce the risk of the second hazard, there are two main strategies to consider:

1. Add an “accessibility mode” into the software to prevent the four push button lights from flashing too quickly.
2. Include an epilepsy warning at the escape room entrance to inform all players of the potential hazard.

The first strategy immediately seems like a more effective option. However, implementing an “accessibility mode” into the software could take a great amount of time and money. While the second option may not be as effective, considering the reduced cost and the overall low probability of the risk, the second strategy is a more practical—and ultimately better—way of mitigating the risk.

Testing and Validation Plan

While our game is simple and easy to play, the mechanics behind it are not. There are many parts in the workings of this product, and it is necessary to ensure that every part is working perfectly to ensure a happy experience for the player.

To guarantee maximum efficiency, we have created ‘checkpoints’ for different parts of the game to ensure that all aspects work as intended.

The first checkpoint is checking the software to make sure that it runs smoothly without the presence of any bugs.
The second is checking the functionality of each button to make sure they work as intended and that constant, repetitive use does not slow the game down.
The third checkpoint tests when the user starts the game and ensures that the passcode functionality works as planned.
During our testing, we will also check the different difficulty levels, making sure that they work as intended.

Design Process

When initially planning our project, we knew that we wanted to create a game that would be accessible to people of all ages and skill levels while still meeting the requirements of the project category. After brainstorming as a group, we settled on the idea of taking a retro game and adding new and modern features to it. This approach would give our older clients a sense of nostalgia and our younger players a fun challenge to solve. Using the suggested projects as inspiration, we decided to pursue a Simon game called “Chime-In.”

After deciding on this project, we began exploring the Internet to discover existing Simon games written in C. Not only did this show us that our project is, in fact, possible, but it also allowed us to foreshadow possible restraints that we may run into. For example, after viewing the additional hardware we will require (lights, buttons, and wires), we needed to ensure that we would have the proper skills to integrate each component into our Nucleo board. After discussing as a group, we decided that each member had enough experience to confidently take on this challenge. Another one of our immediate worries was that a typical Simon game would be much too easy to beat, causing an

anti-climatic experience for the user. To avoid this problem, we decided to implement a passcode that the player must enter to begin the game and an increasing difficulty level as the player advances throughout. Once we were satisfied with our general project outline, we began to discuss the feasibility of our Chime-In game.

We first considered whether or not we would be able to meet the given time constraints. Keeping each group member's strengths and weaknesses in mind, we concluded that we would be able to meet the deadline as long as we stayed organized and followed a strict schedule outlined by our Gannt chart. Next, by generating an estimated cost for each component of Chime-In, we ensured that the project was going to be affordable and that each member was willing to pitch in a small amount of their own money.

Now that we had confirmed the feasibility of our project, it was time for us to begin prototyping. After bouncing ideas off of each other, we each created multiple rough sketches of how we wanted the design to look. We all held a common notion to have Chime-In as a hand-held game that players could pick up and move around the escape room. Of course, this posed some challenges, as we would need to figure out a way to incase all of the components into a single, portable casing. We also needed to ensure that the electrical components could be powered using a battery pack rather than external wiring. We researched the power requirements of the board and confirmed that it is, in fact, possible to power it using small batteries. Despite these challenges, we collectively agreed that we still wanted to pursue our vision. Next, we created a list of the components we would need to make this design a reality. This list included the Nucleo Board, pushbuttons, LED lights, and of course, the software. Now that we had a list of components and an idea of the potential challenges ahead of us, we were ready to begin turning our vision into a reality.

Budget

Equipment	**Cost**	**Contingency**
Nucleo Board	$25 ($30)	$30
USB mini cable	$3 ($5)	$5
5 Buttons	$18 ($25)	$25
4 colour LEDs	$10 ($15)	$15
Wires	$2 ($5)	$5
3D Printer Material	$12.50/inch ≈ $20	$30
Pogo Pins	$10	$15
Neodymium magnets	$15	$25
Battery Pack	$6	$15
TOTAL COST		$165

Project plans and milestones