OCR GCE A

COMPUTER SCIENCE PROJECT

H446-03

Name: Fredric Pashley

Candidate Number: 2010

North Leamington School: 31155

Title of Project: Micro:bit Display

H446-03 – Project CONTENTS

Table of Contents

[A. Analysis 5](#_Toc154058938)

[Description of the problem 5](#_Toc154058939)

[Outline of the project 5](#_Toc154058940)

[Stakeholders 5](#_Toc154058941)

[How the problem can be solved with computational methods 5](#_Toc154058942)

[Thinking abstractly & visualisation 5](#_Toc154058943)

[Thinking ahead 5](#_Toc154058944)

[Thinking procedurally & decomposition 5](#_Toc154058945)

[Thinking logically 6](#_Toc154058946)

[Thinking concurrently 6](#_Toc154058947)

[Research 6](#_Toc154058948)

[Similar products 6](#_Toc154058949)

[Interview with stakeholder to establish potential solution 7](#_Toc154058950)

[Features of proposed solution 9](#_Toc154058951)

[Hardware & software requirements 9](#_Toc154058952)

[Success Criteria 10](#_Toc154058953)

[Stakeholder response to proposed solution 10](#_Toc154058954)

[B. Design 10](#_Toc154058955)

[Structure of the solution 10](#_Toc154058956)

[Systems diagram 10](#_Toc154058957)

[Proposed usability features 11](#_Toc154058958)

[Summary of the process 12](#_Toc154058959)

[Server 12](#_Toc154058960)

[Client 15](#_Toc154058961)

[Objects 15](#_Toc154058962)

[Test data for development 16](#_Toc154058963)

[Algorithms 16](#_Toc154058964)

[Message character allocation 16](#_Toc154058965)

[Client display 17](#_Toc154058966)

[Alpha testing 18](#_Toc154058967)

[Test data 18](#_Toc154058968)

[C. Developing the coded solution (“The development story”) 18](#_Toc154058969)

[Development stage #1 – The principle 19](#_Toc154058970)

[Aim 19](#_Toc154058971)

[Development 19](#_Toc154058972)

[Evaluation 19](#_Toc154058973)

[Development stage #2 – Encoded data format 20](#_Toc154058974)

[Aim 20](#_Toc154058975)

[Development 20](#_Toc154058976)

[Evaluation 21](#_Toc154058977)

[Development stage #3 – Serial numbers 21](#_Toc154058978)

[Aim 21](#_Toc154058979)

[Development 21](#_Toc154058980)

[Evaluation 22](#_Toc154058981)

[Development stage #4 – Creation of classes 22](#_Toc154058982)

[Aim 22](#_Toc154058983)

[Development 22](#_Toc154058984)

[Development Stage #5 – Persistent memory of locations 23](#_Toc154058985)

[Aim 23](#_Toc154058986)

[Development 23](#_Toc154058987)

[Development stage #6 – Pre-set instructions 23](#_Toc154058988)

[Development stage #7 – distributing characters 23](#_Toc154058989)

[D. Evaluation 23](#_Toc154058990)

[Project Appendixes 25](#_Toc154058991)

# A. Analysis

## Description of the problem

Students are likely to benefit from having a display board in their classroom for multiple reasons. Firstly, they will be able to receive additional information set by a teacher, such as class statistics. Secondly, they will potentially find the class more exciting, as the idea of a custom display board on the wall is quite novel and unique.

## Outline of the project

This project’s aim is to create a display board consisting of BBC micro:bits. The reason behind using these micro:bits is to demonstrate the usability of the device to younger students, to inspire them to create their own projects using the micro:bits. The display board will consist of multiple rows (roughly 2/3 rows of 20 micro:bits) to display longer messages. These micro:bits will be mounted onto a permanent frame which will be attached to the classroom wall.

## Stakeholders

The stakeholder designated for this project is a teacher in my school who will use the display board in their classroom.

## How the problem can be solved with computational methods

### Thinking abstractly & visualisation

Using micro:bits to build the display simplifies the problem as they are very simple devices, with basic input and output systems. They contain their own in-built radio function which can be used to communicate with other micro:bits in the display, removing the need for serial communication (which would involve physical wire connections).

### Thinking ahead

The display board will consist of many micro:bits (‘clients’), all connected to another micro:bit acting as a ‘server’.

Planned data input/output:

|  |  |  |  |
| --- | --- | --- | --- |
| **Server** | | **Client** | |
| **Input** | **Output** | **Input** | **Output** |
| Message to display from user,  List of micro:bits in display | Character sent to each micro:bit in display to show on screen | Character sent from server to show on screen |  |

### Thinking procedurally & decomposition

#### Server

The server will have to perform distinct tasks during run-time. Namely, sorting the message from the user into a list of characters assigned to each micro:bit in the display (i.e., formatting), as well as creating effects such as the swipe transition between messages. Each of these can be executed individually, therefore decomposing the problem into smaller areas.

#### Client

There is not much need for decomposition for the client, as when the message is received from the server it only needs to display the relevant output.

### Thinking logically

#### Server

The server will repeat depending on the type of input given by the user. For a single message, once the server has sent it to the clients, it will not run any more code. However, for multiple messages, the server will continuously send the next message to the clients, with the swipe transition in between. This will therefore mean that the server will have to repeat this process until either a new message is loaded onto the server by the user, or the power source is removed.

#### Client

The client will have to repeatedly check for new messages from the server using its radio.

### Thinking concurrently

There is no true concurrency for this project as the server can only send one radio message at a time. However, on some occasions multiple clients are contacted in the same command – e.g., when the display must be cleared the *clear* command will be sent to ALL clients rather than individual messages to each.

## Research

### Similar products

#### The 1,000 BBC micro:bit Display - Kitronik

<https://kitronik.co.uk/blogs/resources/building-the-bbc-microbit-matrix-display>

Kitronik developed a large display board out of micro:bits in a near square shape. Their system worked by communication via serial communication, which uses wires attached to the relevant ports on the micro:bit to transfer data (Fig. 1).

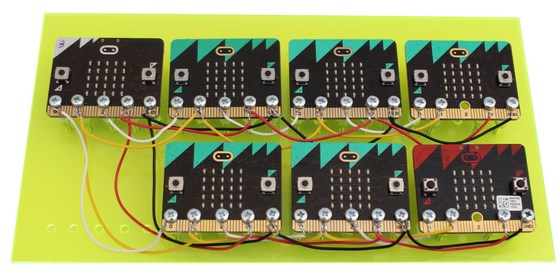


Figure 1 - Serial communication

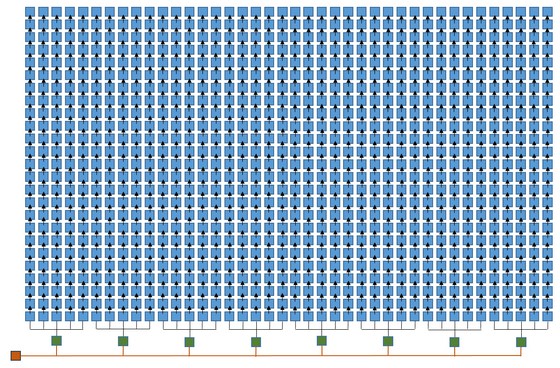


Figure 2 - Direction of communication

Kitronik’s method was to transfer data from one micro:bit to the next in sequence (Fig. 2), rather than sending data to all at the same time.

### Interview with stakeholder to establish potential solution

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Question No.** | **Question** | **Reason for asking question** | **Stakeholder response** | **Conclusion based on response** |
| 1 | What method would work best for you to interact with the display board (e.g., a menu screen or drag-and-drop a file)? | Does the user want a menu screen to type in their message, or do they just want to drag-and-drop a text file onto the server? What is easiest for them? | I think a drag and drop would be really easy to use. | Being able to drag-and-drop a file is the easiest for the user, as they would simply put the messages to be displayed in a text file and copy it across. The code will therefore have to accommodate reading external files. |
| 2 | How large would you want the display board to be? | To gauge how many micro:bits to use. | I’d like the board to be useful to students during their lesson. They need to be able to read the messages without leaving their seats. I think the height of each letter is fine, it should be readable. I think we would probably need about 20 characters wide so that decent length words could be displayed. I’m not 100% certain on this so it would be good if the x and y dimensions was adaptable. This way we could expand it in the future. | Once the frame is built, adaptable x and y dimensions would be difficult to achieve. Micro:bits could be ‘popped’ out of the frame if required. |
| 3 | How portable do you want the display board to be? | To determine the power source of the display board – i.e., mains or battery. May also suggest what material to make the frame out of. | Portability is less of a concern. I see this as being something attached to the wall above my whiteboard. Power could come from the mains. | Mains power is handy as this will remove the need for batteries. |
| 4 | Would you want the display board to provide dynamic data (e.g., temperature, light, time, etc.)? | This establishes whether there is a need to use the micro:bit’s sensors in the future. | Environmental values such as temperature could be useful. We could display this between messages. It would add interest for the students. | Using the micro:bit’s sensors, light levels and temperature can be monitored. These can be displayed. |
| 5 | Anything else you would like to add? |  | We have lots of key messages for students we could display such as “homework due on Wednesday” or “today’s keyword is iteration” or “average class attendance 98%”. If would be good if we could loop between messages. For example the three examples I gave in the previous sentence could all be interleaved. | Messages can be put in a text file on different lines for the server to read. Maybe a swipe transition between them? |

### Features of proposed solution

#### Features

##### Types of micro:bit

There will be 2 ‘types’ of micro:bit – server and client:

###### Server

This will be a single micro:bit with a text file containing the message(s) to display.

###### Client

These will be multiple micro:bits creating the display board.

##### Display board

This will be made from multiple ‘clients’ arranged in a 20x3 grid.

##### Logic

The server will format the message(s) into characters specific for different clients, and then send out the relevant character to the client via radio. The client will be waiting for the transmission and display the character when it receives it.

##### Sending message(s) to server

The user will drag a file onto the drive created by the server and it will then run the code as appropriate.

#### Limitations

Sending out transmissions on the same radio channel may result in overloading the clients with transmissions to decode, especially considering the radio module of the micro:bit can only hold so many messages in its queue. This would mean that the client may miss the relevant transmission as it has to decode irrelevant transmissions in the queue. The number of errors as a result of this would likely increase exponentially with the increase in clients on the board.

The server must have a consistent power supply when the display board is displaying messages, as the server will constantly be sending out characters to clients to display. If the server lost power the clients would endure a freeze effect, where they would continue to display the last character sent to them.

### Hardware & software requirements

User – requirements for the user to interact with the display board  
Solution – requirements for the solution to work correctly

|  |  |  |  |
| --- | --- | --- | --- |
| **Hardware** | | **Software** | |
| **User** | **Solution** | **User** | **Solution** |
| -USB port for server micro:bit (for power & data transfer)  -Standard peripherals (e.g., mouse, keyboard, etc) | -A certain number of micro:bits to make up the display, each with a Micro USB cable for power supply to each micro:bit | -Ability to convert .py to .hex (potentially made possible with automatic menu system)  -File editor | -Code (.hex) onto each micro:bit relevant to if it is a client or the server |

### Success Criteria

* User should be able to write a message and transfer it to the server quickly and easily
* Server should automatically display any new message
* Client locations should be stored as non-volatile
* Display board should be able to be attached to a wall for use

### Stakeholder response to proposed solution

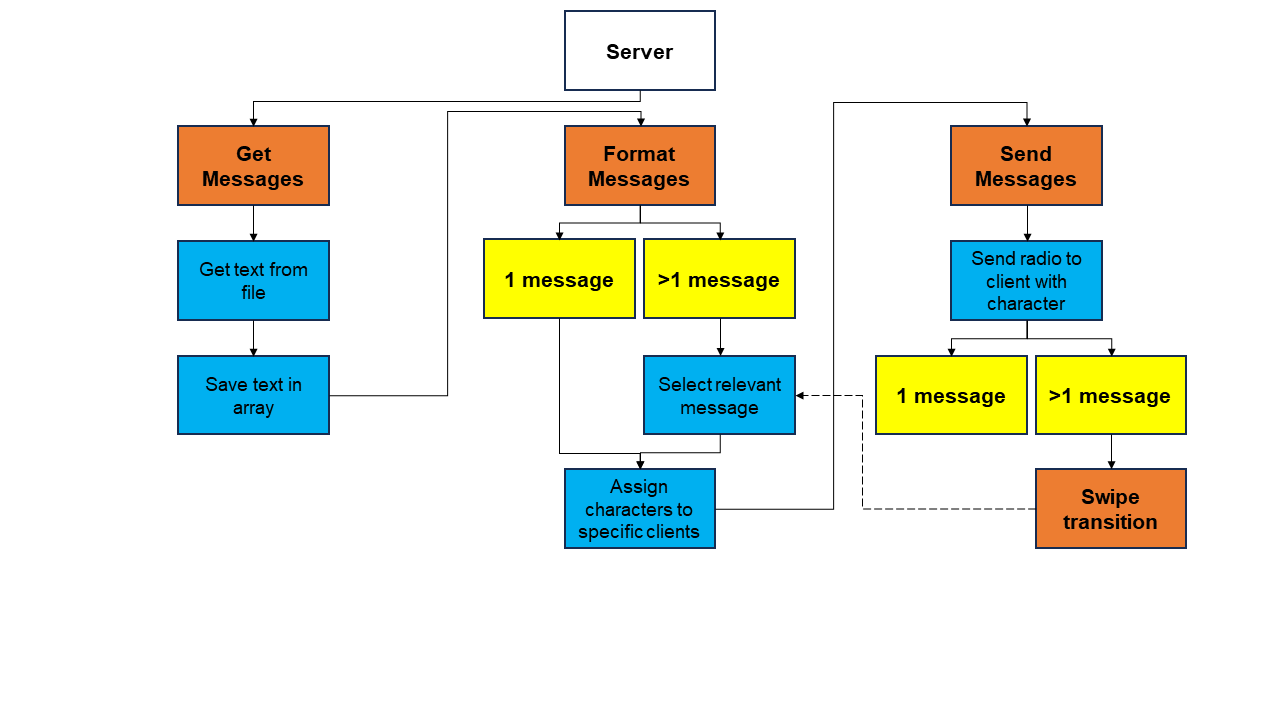
The size of the letters needs to be suitable for students to read across a room. Scrolling adds interest to maintain attention. Altering the message also helps with attention.

# B. Design

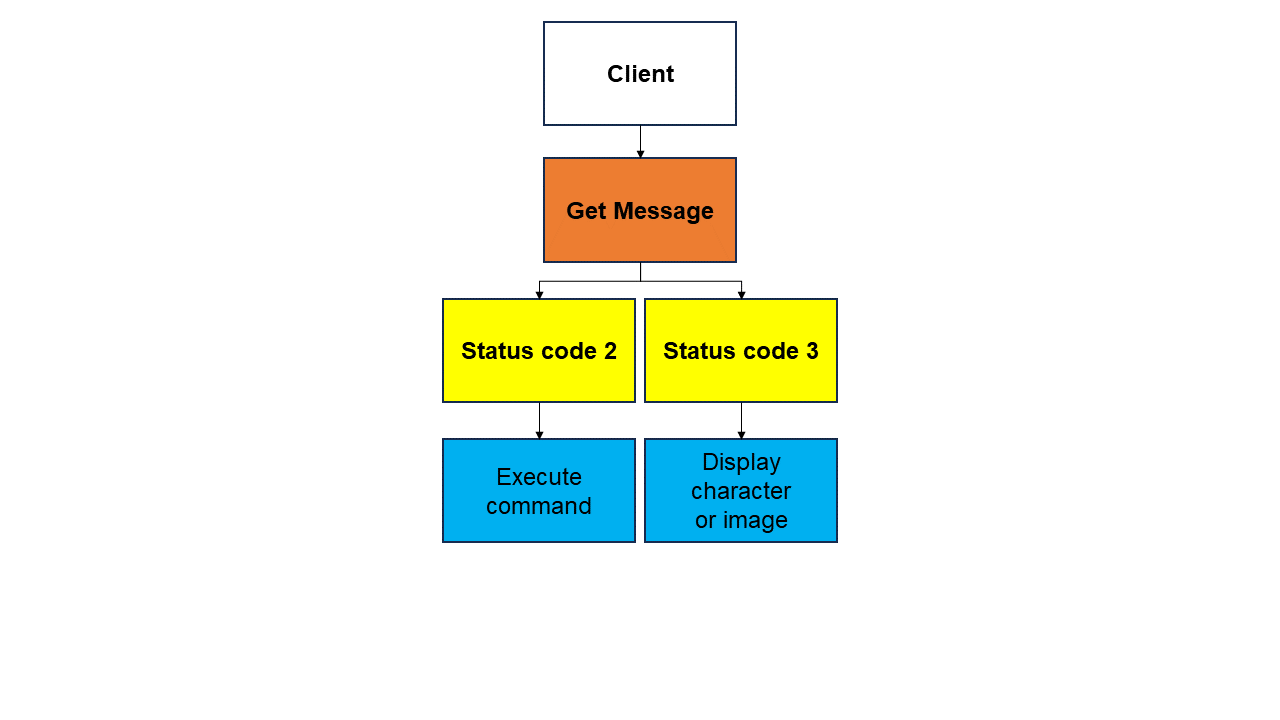
## Structure of the solution

### Systems diagram

#### Server



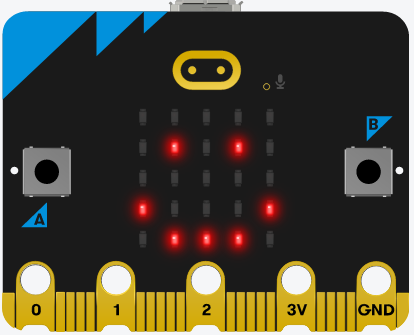
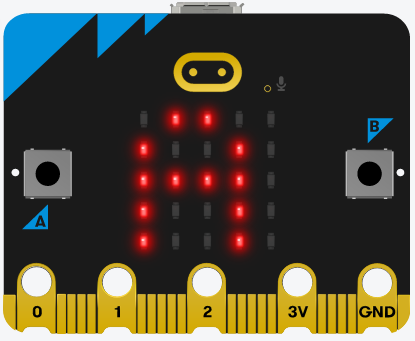
#### Client



### Proposed usability features

#### Screen designs

The only screens used will be the micro:bit’s in-built 5x5 pixel display. This will display either a character or a preset image.



#### dimensions

The display board will have 60 clients in total, arranged in a 20x3 grid. Each micro:bit is 51.6mm(w) x 42.0mm(h) x 11.7mm(d). Therefore, if the dimensions of each micro:bit is approximated to 52mm(w) x 42mm(h) x 12mm(d), the display board will have the dimensions of 1040mm(w) x 126mm(h) x 12mm(d), which is 104cm(w) x 12.6cm(h) x 1.2cm(d). There will have to be ‘wiggle room’ considered, such as the display’s frame’s thickness and any space left for wiring. Therefore, a generalised dimensions would be about **110cm(w) x 15cm(h) x 2cm(d).**

## Summary of the process

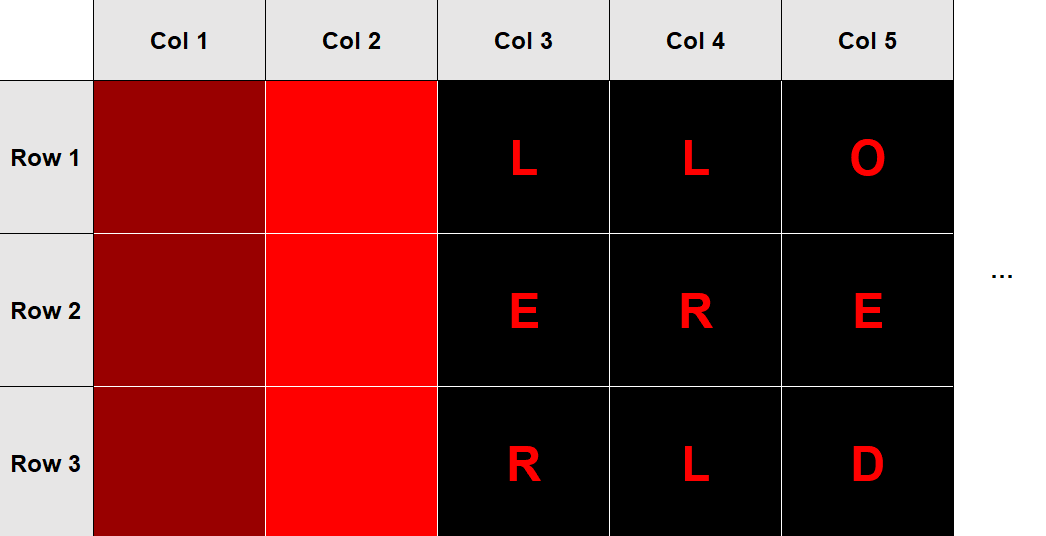
### Server

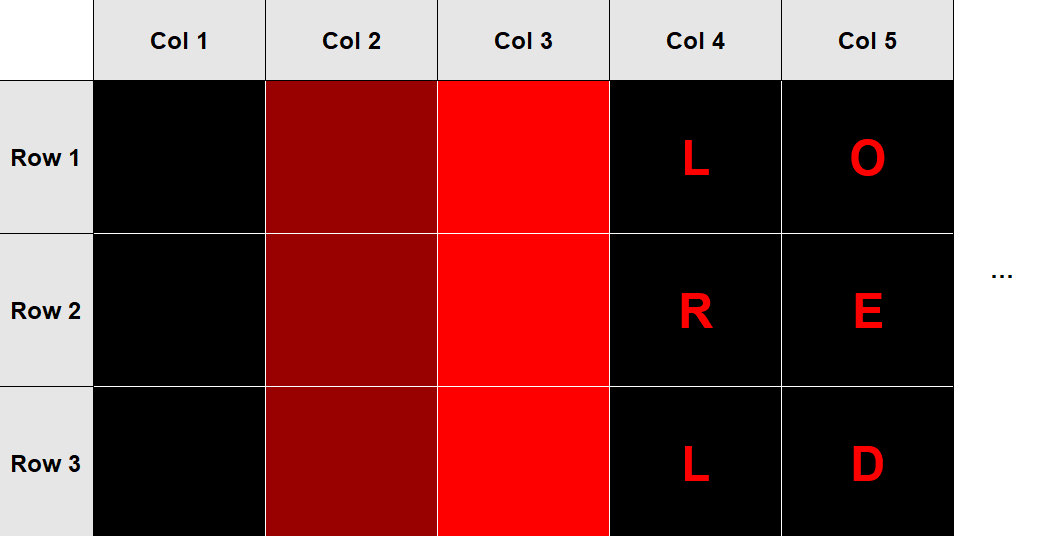
Open the text file stored on the micro:bit and read the lines. These lines represent the different messages to be displayed. These lines are then saved in an array for future use. To format each message, the number of messages needs to be established. If there is more than one message, the relevant message first needs to be specified. This will then increment in turn. Once the message has been selected, it is split into individual characters (or images if they are included in the message). This would look something like: [“A”, “B”, “C”, “ ”, “duck”]. The server will then assign each character/image to the correct client in its 2D array of locations. In practice, this means that “A” would be assigned to Client #1, “B” would be assigned to Client #2, etc. If the message text is longer than 20 characters/images, the server will wrap the remaining text onto the next row. If the message text is longer than 60 characters/images, the server will truncate the remaining text. Once each client has been paired with a character, the server will send out a radio transmission to the clients with the data encoded as appropriate. The client will then receive this message and display the character received if it applies to them.

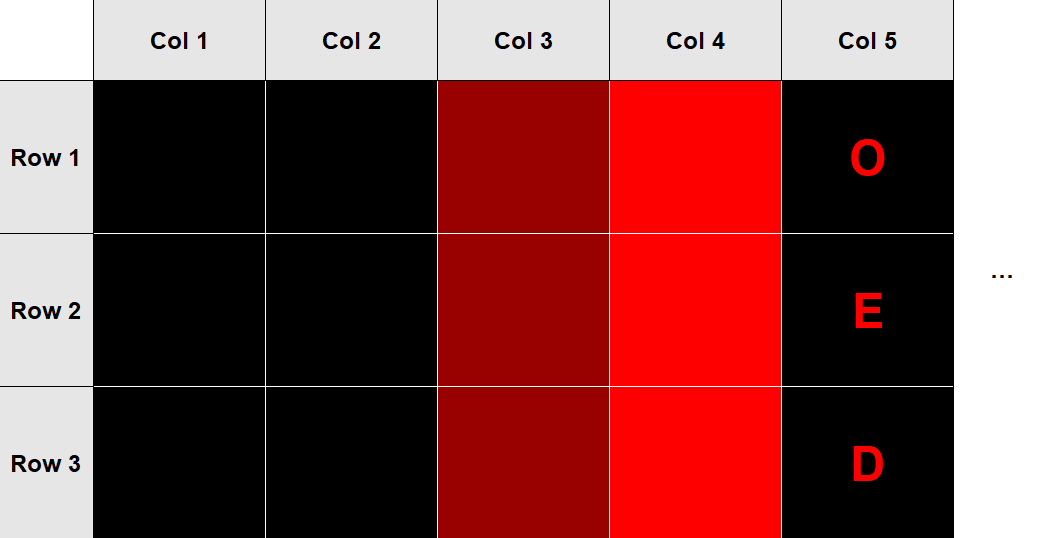
If there are multiple messages, this process will repeat with a swipe transition in-between messages. This works in a similar way to pairing clients with text, but it will be a custom image of a block with a certain brightness (to simulate swipe history). This will then move from left to right across the screen as shown below:

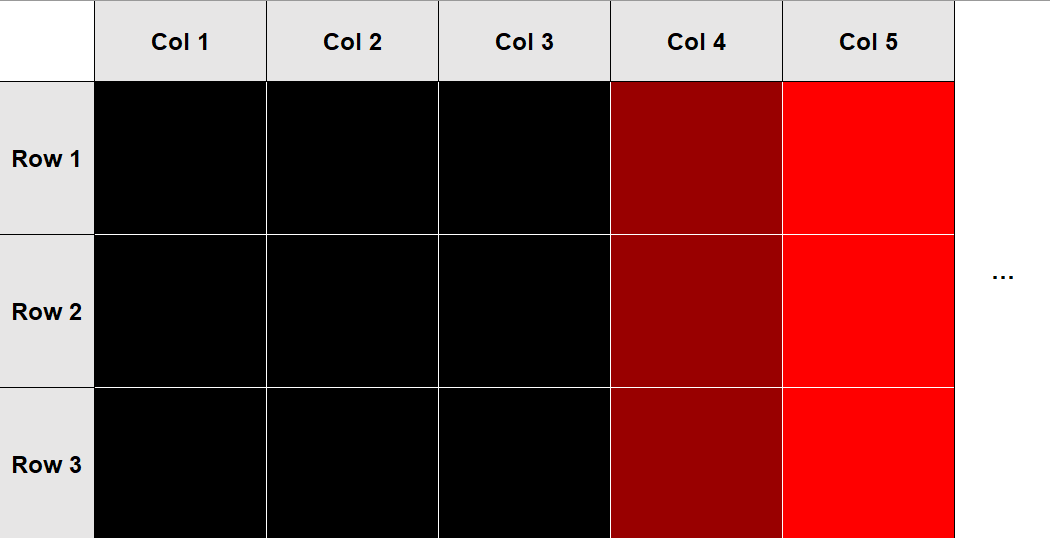


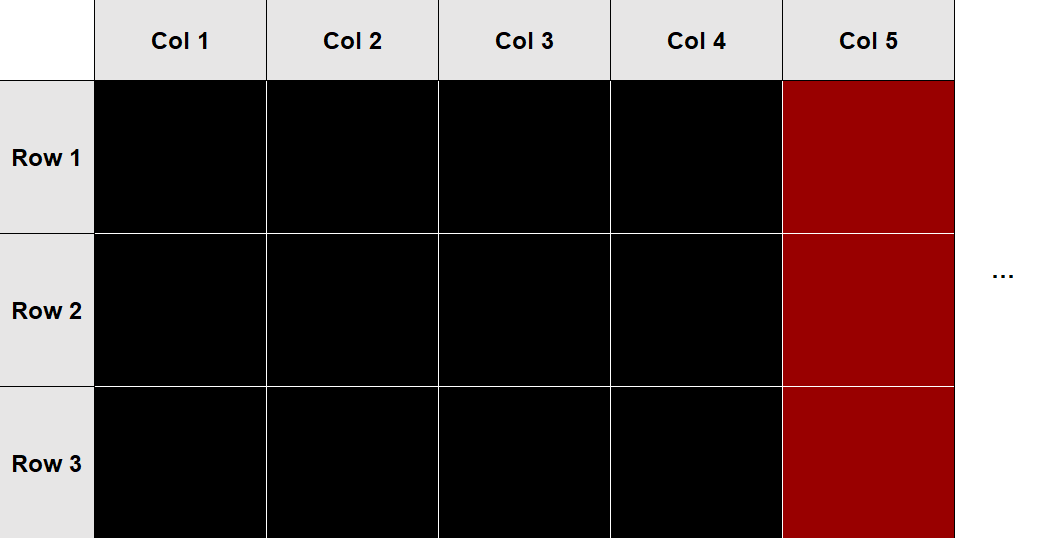


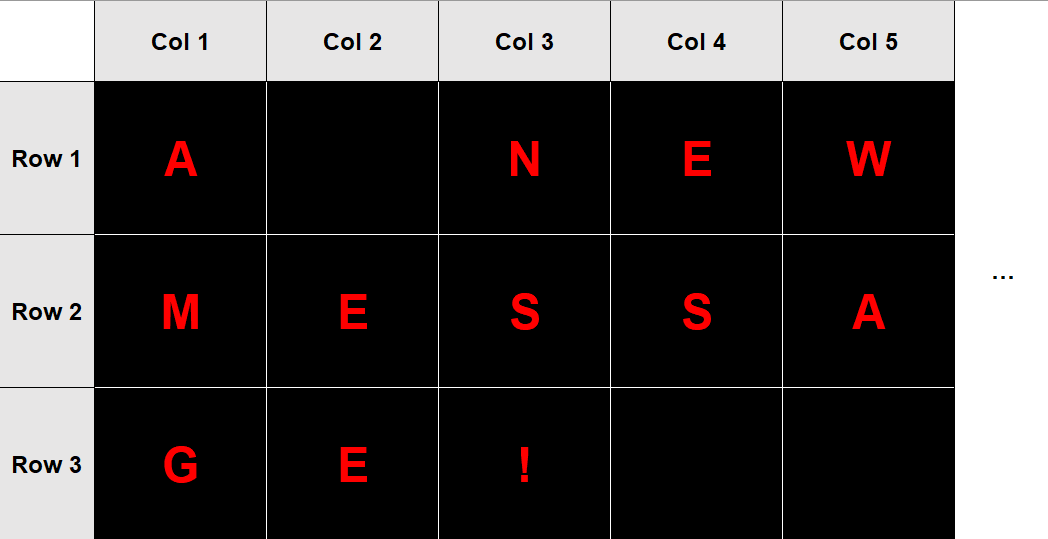










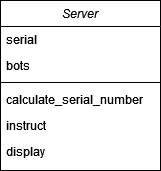


The total time for the swipe transition should be ~1s. This means that each client should show their block for .

### Client

The client will be constantly listening for messages from the server. When it receives a message, it will decode the data from the message to establish its intended recipient along with the data transmitted. If the intended recipient is itself, then it will either follow the command or display the data depending on the status code sent. Once this has completed, the client will return to listening for new messages.

## Objects

## Test data for development

Throughout the development process, random messages will be sent to the server to check if the display is working correctly.

## Algorithms

### Message character allocation

function allocateCharacters(bots, message)  
 // bots is a 2D array representing the display board: [[r1bot1, r1bot2, …], [r2bot1, r2bot2, …]]  
 // message is a string of the desired message to display (e.g., “Hello world!”)  
 letters = "ABCDEFGHIJKLMNOPQRSTUVWXYZ"  
 characters = []  
 for character in message  
 characters.append(character)  
 endfor  
 current\_row = 0  
 current\_bot = 0  
 clients = {}  
 for bot\_row in bots  
 for bot in bot\_row  
 bot\_coord = letters[current\_row] + (current\_bot+1)  
 clients[bot\_coord] = [bots[current\_row][current\_bot], ""]  
 current\_bot += 1  
 current\_row += 1  
 endfor  
 current\_row = 0  
 current\_bot = 0  
 for character in message  
 if current\_bot == 20 then  
 current\_bot = 0  
 current\_row += 1  
 endif  
 if current\_row == 3  
 break  
 endif  
 character\_coord = letters[current\_row] + (current\_bot+1)  
 clients[character\_coord][1] = character  
 endfor  
 to\_send = []  
 for coord in clients  
 to\_send.append(clients[coord])  
 endfor  
 return to\_send  
endfunction

### Client display

function display(s)  
 // s is a string of either a character or an image name (e.g., “H” or “duck”)  
 if hasattr(Image, s) then // See comment outside of code block (\*)  
 show = getattr(Image, s) // See comment outside of code block (\*)  
 else  
 show = s  
 endif  
 display.show(show) // See comment outside of code block (\*\*)  
endfunction

\* The use of **hasattr** and **getattr** is specific to Python. The hasattr(obj, attr) function checks if the attribute exists in the object. For example, if Dog.name was an object with the attribute ‘name’, then hasattr(Dog, “name”) would return True. In context, this is checking if the image passed (e.g., “duck”) exists in the micro:bit ‘Image’ class (retrieved by using *from microbit import \**). If the image does exist, getattr(obj, attr) is used to retrieve the value of the attribute from the class. In context, this is assigning the variable ‘show’ to the relevant image to display.

\*\* ‘display’ is retrieved by using *from microbit import \**. display.show(x) will display whatever x is on the micro:bit’s screen, whether it is an Image or a character.

## Alpha testing

The program will be tested as it is developed. However, the final alpha testing will utilise a black box approach.

### Test data

The following lines of text represent what is inputted into the text file for the server.

#### Valid

Hello world!

#### Borderline

Hello world?  
I am a [duck] who is very loud.

#### Invalid

Hellø world! [duck]

# C. Developing the coded solution (“The development story”)

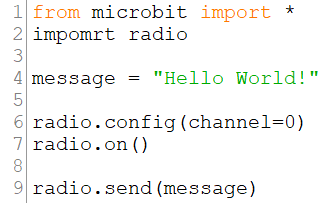
## Development stage #1 – The principle

### Aim

To successfully send a message via radio from one micro:bit to another. The rest of the program will then build off this working principle.

### Development

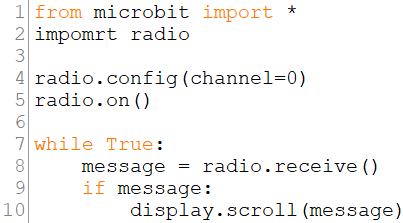
#### Micro:bit #1 (sender)



##### Line explanations

6: channel=0 ensures that both micro:bits are on the same channel, and therefore will receive the message.

#### Micro:bit #2 (receiver)



##### Line explanations

4: channel=0 ensures that both micro:bits are on the same channel, and therefore will receive the message.

### Evaluation

The principle shown works as expected.

IMAGE HERE

## Development stage #2 – Encoded data format

### Aim

To establish a standard format for sending data between server and client.

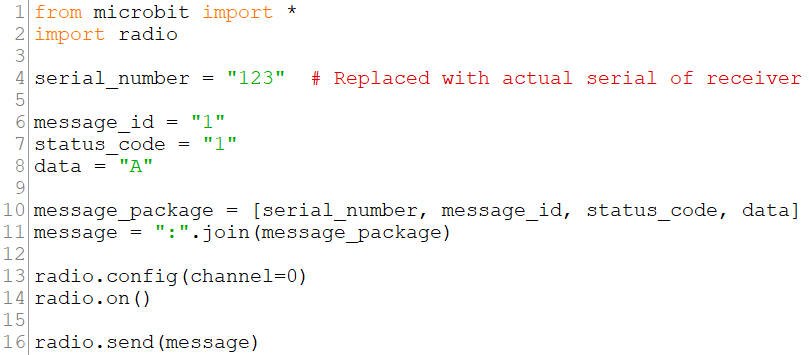
### Development

Each radio transmission will have multiple pieces of data, each separated with a ‘:’ (e.g., *data1:data2:data3*). This format will be defined as: **recipient:message\_id:status\_code:data**.

* recipient – serial number of micro:bit receiver (or ‘SERVER’ if the receiver is not a client)
* message\_id – random integer to distinguish new messages from old messages
* status\_code – determines whether the data is a command or a character to display
* data – this could be a command, a character, or an image name to display

The advantage of having a standardised format is that the code can split the message by the ‘:’ character, and it will always work.

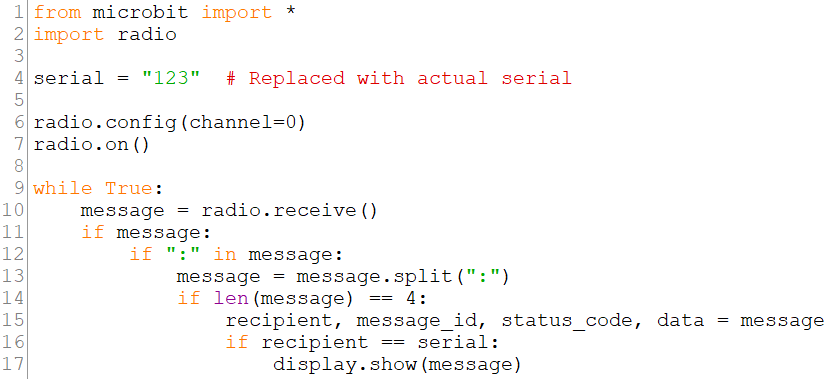
#### Microbit #1 (Sender)



##### Line explanations

4: Serial number of micro:bit ([to be calculated](#_Development_Stage_#3)) for identification.

#### Microbit #2 (Receiver)



##### Line explanations

4: Serial number of micro:bit ([to be calculated](#_Development_Stage_#3)) for identification.

12-14: The micro:bit may have received a message that isn’t intended for this solution due to external factors, so it needs to check if it is actually what it wants to decode.

16: The micro:bit needs to check if the message is intended for itself rather than another in the board.

### Evaluation

The solution for encoding data works as expected.

IMAGE HERE

## Development stage #3 – Serial numbers

### Aim

To create a function that finds the serial number of a micro:bit, to enable accurate identification of unique micro:bits.

### Development

Finding the serial number of a micro:bit involves searching in the micro:bit’s memory.



<https://support.microbit.org/support/solutions/articles/19000070728-how-to-find-the-micro-bit-serial-number>

This solution requires the ‘machine’ library to be imported.

### Evaluation

The function works as expected. This can be tested by calculating the serial number, then getting the micro:bit to display the returned value:

IMAGE HERE

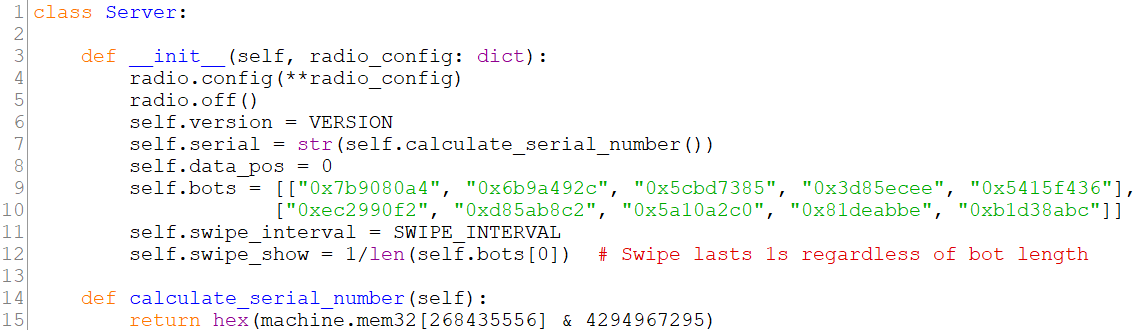
## Development stage #4 – Creation of classes

### Aim

Create two classes: Server and Client.

### Development

#### Server



##### Explanation

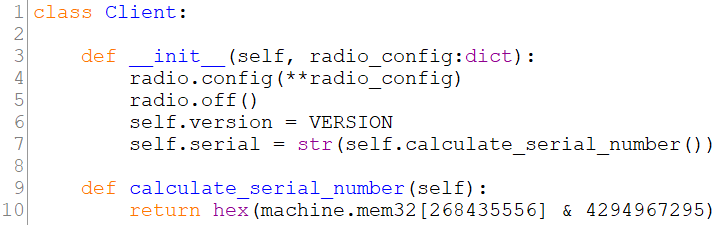
radio\_config (argument): This is a dictionary passed as an argument when the Server object is created. This enables the radio configuration to be edited, such as the channel it is listening on.

\*\*radio\_config (in \_\_init\_\_): This unpacks the dictionary into the radio.config() method. The radio is then turned off to save power.

The server also calculates and remembers its serial number, stored as self.serial.

self.data\_pos: This is initialised as an integer count for which character in the message data the server is looking at when it is [distributing the message](#_Development_stage_#7) accordingly.

#### Client



##### Explanation

radio\_config (argument): This is a dictionary passed as an argument when the Client object is created. This enables the radio configuration to be edited, such as the channel it is listening on.

\*\*radio\_config (in \_\_init\_\_): This unpacks the dictionary into the radio.config() method. The radio is then turned off to save power.

The client also calculates and remembers its serial number, stored as self.serial.

## Development stage #5 – Persistent memory of locations

### Aim

The micro:bit acting as the server must know where the other micro:bits are within the display board, in order to distribute the message’s character accordingly. This can be done by simply saving each serial number to a 2D array when the server micro:bit is turned on. However, the main issue with this solution is that if the power supply to the server micro:bit is lost for any reason then the 2D array is lost. There are two potential fixes to this issue: get each client micro:bit to remember its coordinates and then work out which character it must display; or, write the 2D array of serial numbers into the server micro:bit’s code so that it is a non-volatile array. The second option is lengthier in terms of time, however in the long run is more efficient.

### Development

.

## Development stage #6 – Pre-set instructions

.

## Development stage #7 – Distributing characters

.

# D. Evaluation

<See H446-03 Project Advice Booklet for help and guidance of what must go here.>

# Project Appendixes

Insert as many project appendixes as you need for your project.

These might include, but are not limited to:

* Complete Code Listing (ESSENTIAL)
* Interview Transcripts
* Meeting notes
* Observation notes or questionnaires

[github.com/FreddyPashley/Microbit-Display-Board](https://github.com/FreddyPashley/Microbit-Display-Board/)