

# CS3099 B-1

Peter Goad, Daniel Key, Nicola Macdonald, Joanna Moreland, Oli Solomons

*School of Computer Science  
University of St Andrews, Fife, KY16 9SX, Scotland*

*18/04/2019*

# Abstract

*This project concerns a “minimal internet device” which allows its users to communicate with their peers and maintain access to important web content without interfering with or distracting them from their current tasks and/or surroundings to an excessive degree. This was accomplished by programming a BBC micro:bit to communicate with other similarly programmed micro:bits over radio and creating a server that could access designated web content and communicate it to a micro:bit via USB or radio, along with a website that allows users to manage login information and configuration settings for the product.*

We declare that the material submitted for assessment is our own work except where credit is explicitly given to others by citation or acknowledgement. This work was performed during the current academic year except where otherwise stated. The main text of this project report is 11154 words long, including project specification and plan.

In submitting this project report to the University of St Andrews, we give permission for it to be made available for use in accordance with the regulations of the University Library. We also give permission for the report to be made available on the Web, for this work to be used in research within the University of St Andrews, and for any software to be released on an open source basis.

We retain the copyright in this work, and ownership of any resulting intellectual property.

# Contents

<b>Contents</b>	<b>3</b>
<b>Introduction</b>	<b>5</b>
The Problem	5
Solution Summary	6
<b>Development of the Project</b>	<b>7</b>
Our Journey	7
Software Development Methodology	8
<b>Product Functionality</b>	<b>10</b>
The Product's Features	10
User accounts	10
Website	10
User Interface - Pattern Lock	12
User Interface - Menus	13
User Interface - Keyboard	14
Weather	15
Twitter	15
Twitter - Spam Filter	16
Piano	16
Implementation	17
Server	17
Microbit connection to server	17
User accounts	18
Website	19
User Interface - Menus	20
User Interface - Keyboard	20
Weather	20
Twitter	20
Twitter - Spam Filter	21
Piano	22
Messaging	23
Protocol	23
Security	23
Mesh Network	25

<b>Testing</b>	<b>26</b>
Website	26
Messaging	30
Weather	32
Tweets	32
Mesh network online access	33
Artificial Intelligence	33
Confusion matrix	33
Classification report	34
User Interface	35
<b>Evaluation and critical appraisal</b>	<b>36</b>
Our Project Compared to Original Objectives	36
Our Project Compared to Others' Work	37
<b>Conclusion</b>	<b>39</b>
Our Summary	39
Key Achievements and Drawbacks	39
Future Work	40
<b>Appendix A: References</b>	<b>42</b>
<b>Appendix B: Our SCRUM Meetings This Academic Year</b>	<b>44</b>
<b>Appendix C: User Interface Questionnaire</b>	<b>51</b>
<b>Appendix D: Code Building and Running the Server</b>	<b>54</b>

# I. Introduction

Contributed by Nicola Macdonald

## A. The Problem

The aim of this project was to create a ‘Minimal Internet Device’ which would allow users to stay connected to relevant information, without the distractions introduced by a smartphone. In an increasingly technologically driven world it is more relevant than ever to stay connected to digital information, but in many situations, for example in a professional environment, it is inappropriate and intrusive to achieve this with devices like phones which are also designed for personal communication and interests.

This is a serious and widespread problem, one example being that 60 percent of college students in the United States of America admit they are addicted to their mobile phones.<sup>[A]</sup> To help address this issue, we will be creating a device that will allow the user to keep in contact with friends and access important information on the web such as the news and weather without the temptation of using other apps that their smartphone may have. The limited nature of the BBC micro:bit, as shown in Fig. 1, allows it to fill this role of a minimal communication device, capable of providing the required functionality to stay connected to people and events remotely without intruding on daily life.



Fig. 1. A BBC micro:bit. Features a 5x5 LED matrix and 2 buttons. Retrieved from <https://www.sparkfun.com/products/14208>



Fig. 2. The official BBC micro:bit logo, adapted from <https://www.bbc.co.uk/mediacentre/mediapacks/microbit>

In the initial project specification<sup>[B]</sup>, a number of basic requirements were given for the device. This minimal functionality was to include social media or email access; weather or news updates; messaging between micro:bits; Artificial Intelligence (AI) to filter external data for relevancy; and a website to support the device. In further specifications, requirements were added to implement a mesh network for messaging, allowing messages to be forwarded between micro:bits of the same supergroup; to provide security by encrypting messaging between micro:bits; and to allow the micro:bit to function with and without an additional piece of hardware.

The product was to be created using a BBC micro:bit. The device's features include a radio, an accelerometer, 25 LED lights and two buttons. This came with a number of hardware challenges including the limited 16K RAM and lack of internet support from the micro:bit itself.

## B. Solution Summary

Over the course of the project we have met all of the basic requirements, in addition to a large amount of further functionality. Because of the hardware limitations of the micro:bit screen displaying large quantities of text was impractical, so we chose to integrate Twitter access with the micro:bit, with the ability to both send tweets and receive the latest tweets from a personalised feed, filtered by a random forest algorithm-based text classifier. The micro:bit can also access and display the latest weather updates from any chosen city. Both of these features are supported by the website, which provides user accounts to allow customisation of the previous features on an individual level, as well as providing further information and functionality. The user accounts also allow the micro:bit to be locked and logged into using a pattern lock on the device itself.

The micro:bit implements messaging with a mesh protocol which allows it to communicate with any micro:bit within the supergroup, or all of them across the broadcast channel. The protocol also enables the forwarding of messages if a recipient is out of range, extending the usable distance of the micro:bit radio. If none of the group's micro:bit devices is connected via serial to the computer, the micro:bit still operates but without online functionality. If at least one micro:bit is connected however, the mesh network is utilised to allow full online capabilities for all group micro:bits. The micro:bit also provides support for the :KLEF Piano by Kitronik<sup>[3]</sup>, functioning with different inputs and providing further functionality with customisable notification sounds.

For the purposes of connecting the micro:bit to the server please see **Appendix D**. This is a guide for accessing the server or running your own if you wish.

## II. Development of the Project

Contributed by Joanna Moreland

### A. Our Journey

We started the project with the intention of creating a device that would be capable of fulfilling the basic specification (see I. Introduction, A. The Problem). It was our intention to create a communication device that minimised distraction so that communication was possible and useful, but not as addictive as that of a smartphone. We intended to extend the functionality to improve the user experience and product usability with: additional security features; message templates for the user to send rather than typing out a message, along with the ability to customise these templates via the website; providing a history log of messages; and the ability to bookmark received messages so that important ones were readily available in the history log.

As the semester progressed, we wanted the user to have more control over the micro:bit and reduce the degree to which users' data crossed over on the server. As such, we introduced username and password sign in to the website, and with this came the opportunity for the user to personalise their micro:bit and their own data to be maintained on the server. The website developed further in the following semester, with the weather displayed on the device being specific to each user and an unlock pattern being settable on the website per user. The micro:bit remained a quick and unintrusive device, while also obtaining extended functionality.

After the first semester, we were given further requirements to consider. Encryption became compulsory as opposed to an extension and so had to be implemented among the supergroup as a whole, instead of just our team. As such, we began to consider including other security features, such as a lock screen and password, and such features were also added as requirements. Our intention to create a very minimal communication device with effective but limited functionality was further strained with the introduction of the :Klef piano requirement. This required that the team consider how to maintain minimal distraction while also utilising the new hardware and its many input possibilities. For all of the new requirements, we made decisions that preserved as much of our desired final product's minimalism as possible, while also ensuring that we implemented all the required functionality in an effective and user friendly way.

As the year progressed, many planned add-ons were dropped in order to prioritise improved basic functionality or because the team concluded that they went against our intended, non-distracting final product. However, some extensions that the team agreed would meaningfully improve the product were retained. Features such as message templates, utilising

the input pins for “swipe” inputs, an extended keyboard with more symbols and playing the piano hardware as an actual instrument have all been cut from the project as year progressed.

We initially planned to program the micro:bit in Python, as the team was more familiar with it than most other compatible programming languages. However, towards the end of the first semester, the micro:bit’s limited memory became a problem which we were only able to resolve by converting all of our code on the micro:bit into C++. The server side code was run on Node.js and Express.js, but as more functionality was required (such as AI and serial communication with the micro:bit) some added sections were written in Python and C++. The website was largely HTML with CSS styling with elements of JavaScript for interactive scripts to run.

## B. Software Development Methodology

The team used agile and SCRUM as the basis of our development. Agile is a leading method of software development which is fast and allows smooth evolution of software over time. This was a good approach for our team to take as it allowed us to adapt our design decisions without having to replan whole sections.

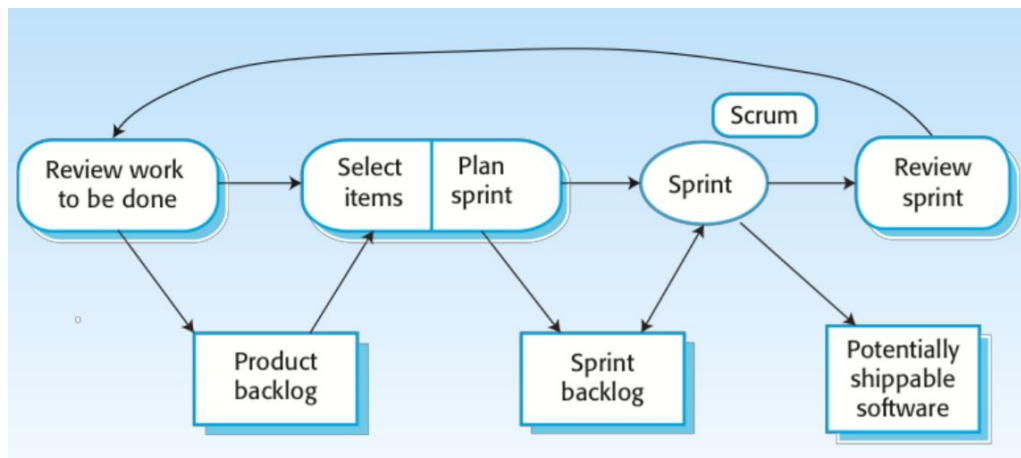


Fig. 3. Agile Methodology that describes our process

Linton, S. (2018, September 28). CS3099 JH Team Project SCRUM. University of St Andrews. <https://studres.cs.st-andrews.ac.uk/CS3099/Lectures/Lecture02.pdf>

The team met once a week for a SCRUM meeting, with each member informing the rest of the team what they had done, what they would do in the next sprint and any problems they had encountered. There was also a weekly meeting with the team’s supervisor. The team decided at the start of the project to rotate the role of SCRUM master so that everyone got an opportunity to gain experience and guide the meetings. All members but Peter had time to do the role but as he would have only have had 2 weeks as SCRUM master and given the imminent final submission



we did not push for a swap to occur. We produced minutes for our SCRUM meetings, which can be seen in **Appendix B**. The tasks for each sprint were assigned to members of the team, also recorded in the given appendix.

Nicola acted as the group's product owner and helped to ensure that the team remained focussed on producing a *minimal* internet device, as we often wanted to implement features which would lead to the device becoming distracting and hence run contrary to the overall aim of our project. Daniel took the role of supergroup representative, attending supergroup meetings on behalf of the group and feeding back relevant information to the group in SCRUM meetings.

SCRUM was effective for the project as we were able to build effectively on past sprint cycles. The requirement for regular meetings was also beneficial as the team was kept up-to-date with the progress of other team members. We were also able to test regularly having completed small, functionable sections of code. The only problem the team experienced with SCRUM was the lack of a larger plan, as our goals and tasks were created based on the timeframe provided by a sprint. As such, larger goals sometimes took longer to achieve because each sprint focussed on smaller, less relevant (though necessary down the line) achievements.

Given what the team learned over the past year of agile and SCRUM use, in retrospect we would make some adjustments. Firstly, the team would ensure that everyone got a proper chance to adopt the role of SCRUM master so that everyone could gain some experience in the role. Secondly, we would aim to achieve the more important requirements first and then build on them and add other features later rather than doing them simultaneously.

# III. Product Functionality

Contributed by Oli Solomons

## A. The Product's Features

The product the team created meets all of the basic requirements stated in the specification and updated specifications since the beginning of the year. Features were extended so they had more useful functionality and also new features were added that the team agreed was appropriate for the device we aimed to make.

### User accounts

Video: <https://www.youtube.com/watch?v=AbxY0UzaVgo>

The video provided above is a visual guide to this subsection.

User accounts are used to store individual preferences and data for users of the micro:bit. The account is consistent across the website and micro:bit, allowing users to register and log in on the website and make changes which will then be reflected on the next micro:bit they log in on.

Users can use their username and lock pattern to log in to a micro:bit for the first time, after which only the lock pattern is required. Once logged in, users can access all of the micro:bit's functionality without requiring further authorisation, as requests to the server are automatically and securely authorised with the existing credentials.

### Website

Video: [https://www.youtube.com/watch?v=9iRFpv5\\_-00](https://www.youtube.com/watch?v=9iRFpv5_-00)

The video provided above is a visual guide to this subsection.

The micro:bit website<sup>[E]</sup>, the front page of which can be seen in Fig. 4 below, provides an easy way to start using the micro:bit. It has a minimalist design, comprising of a maximum of three 'sections' per page: the header, the navigation menu and a main content box. Its colour scheme is similar to the micro:bit, which can be seen in Fig. 1. The light blue accent colour is taken directly from the official BBC micro:bit logo, which can be seen in the top left corner in Fig. 4, or in more detail in Fig. 2, above. It provides a number of customisation options and features. Before authenticating, users can view detailed instructions for the micro:bit from the home page, and after registering or logging into a user account they have easy access to further pages from the menu.

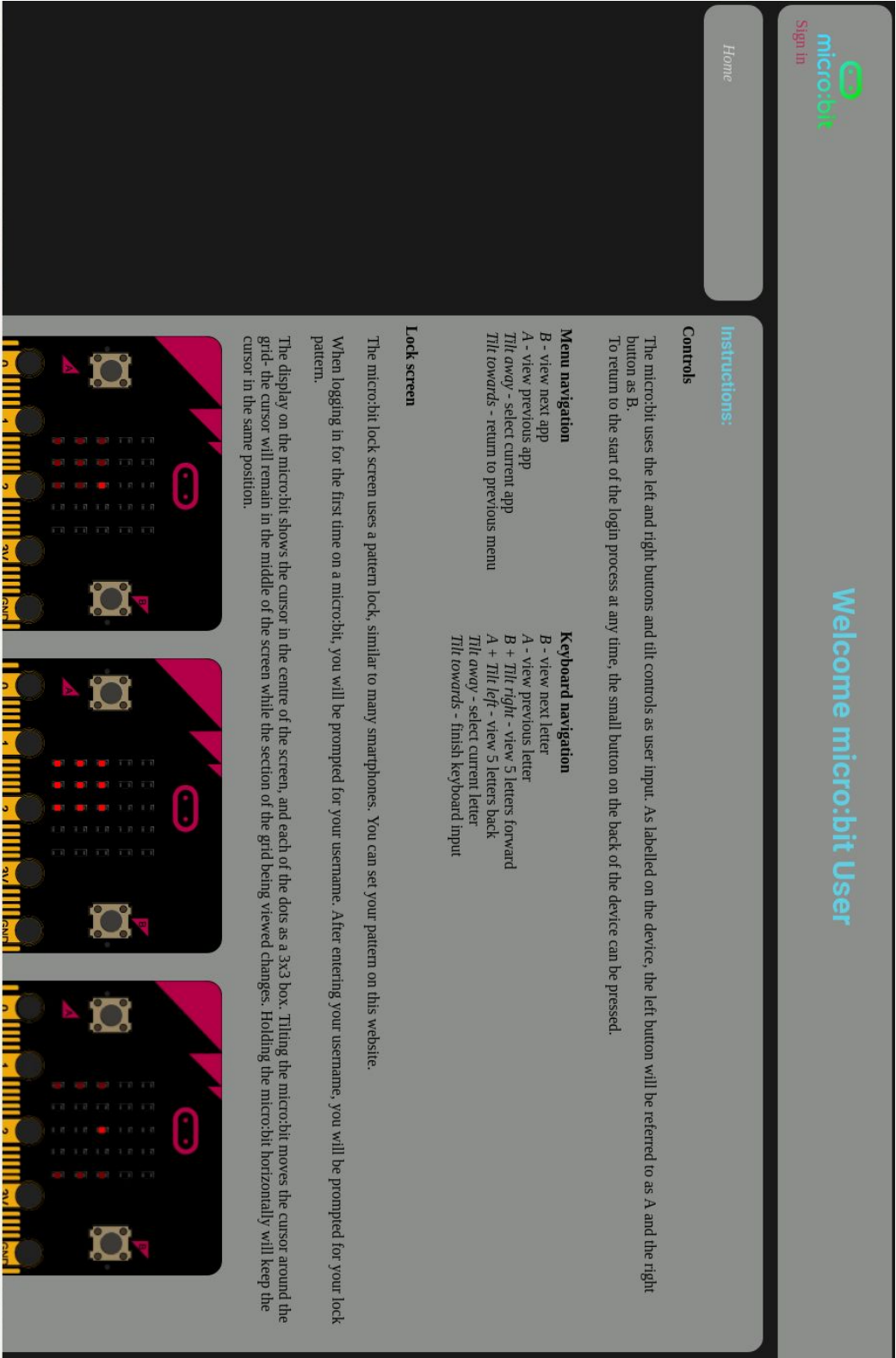


Fig. 4. The website home page, cropped.

The navigation menu provides access to various web pages in the website and allows for further micro:bit functionality. The web pages are: home, history log, weather, lock pattern and Twitter. Furthermore, in the top left corner, a log in/log out option is available, with the option changing depending on whether the user has logged in or not to a user account.

Users can use the website to understand how to use the micro:bit. A user manual is provided in the form of a control flow diagram and some text explanations, and an example of how to unlock the micro:bit is provided in the form of still and animated images. When browsing the website, the user is able to set a lock pattern using an interactive pattern lock simulator by accessing the 'lock pattern' page, or they can view logs of the messages they've received on any micro:bit device by accessing the 'history log' page. When selecting the 'weather' page, they can also choose which city to view weather for, and choose Twitter handles they wish the micro:bit to display tweets from.

A concern the user may have is the storage and use of their data. For example, they may set a city in the 'weather' page, or they may set a lock pattern they have also set for their smartphone. Similarly, they may use the same credentials when registering for their user account that may be used for other personal accounts, such as a social media account<sup>[G]</sup>. B-1 will not use this data for any purpose except for providing current weather, allowing the user to unlock their micro:bit etc., and informs the user as such in the log in/log out page.

### User Interface - Pattern Lock

Video: <https://www.youtube.com/watch?v=GhMrMuLZ02o>

The video provided above is a visual guide to this subsection.

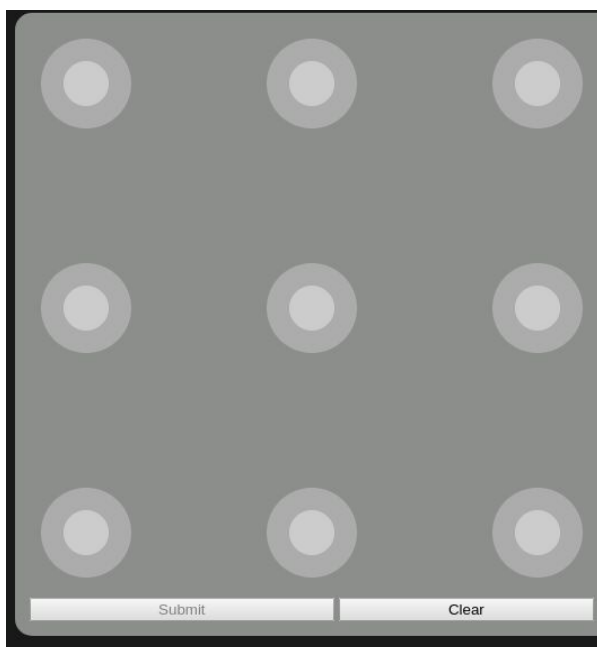


Fig. 5. An 'empty' pattern lock on the 'lock pattern' web page. See pg. 26 for extensive tests.

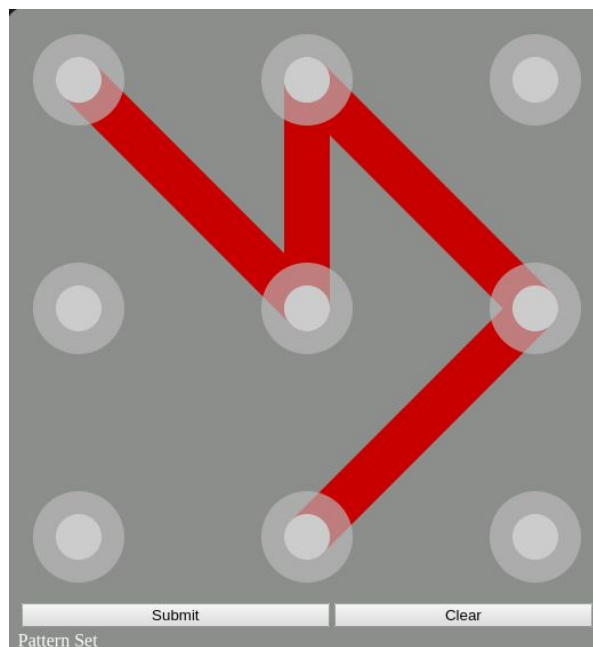


Fig. 6. A pattern lock that has been created and saved on the 'lock pattern' web page.

The pattern lock used to log into the micro:bit is a quick and intuitive<sup>[P]</sup> form of authentication which allows users to quickly resume a session on a micro:bit after switching it on. The pattern itself is formed of connections between unique nodes on a 3x3 grid and is set on the website. Fig. 5, above, shows an 'empty' pattern lock, whereas Fig. 6, above, shows a pattern lock with a saved pattern. The user may not have a pattern of less than 3 nodes.

To replicate the pattern on the micro:bit, the device is tilted to move a cursor around the grid, and either button is used to select each node in order. As the nodes are selected the connections are displayed on the screen. An example of this is shown on the website home page in Fig. 4.

## User Interface - Menus

Video: <https://www.youtube.com/watch?v=bMaq2rVmYrQ>

The video provided above is a visual guide to this subsection.

The micro:bit interface uses a hierarchical menu system, with the main pieces of functionality accessed from the top level menu through letter symbols representing the applications. The main pieces of functionality are messaging, weather and Twitter. For example, selecting the 'M' symbol on the top level menu accesses the messaging application, as seen in Fig. 7, and selecting the 'T' symbol accesses the Twitter application as seen in Fig. 8. Pressing the A and B buttons

scrolls left and right respectively through each menu. Tilting the device away from the user selects the current option, while tilting the device towards the user returns to the previous menu level. The control scheme is consistent throughout the menu system to allow navigation to be as easy and intuitive as possible.

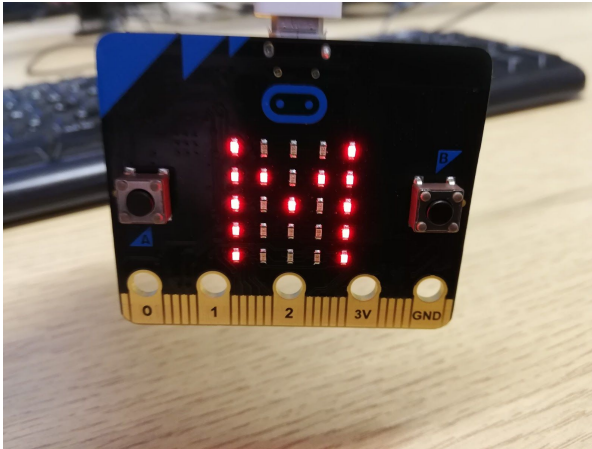


Fig. 7. Messenger application on the menu

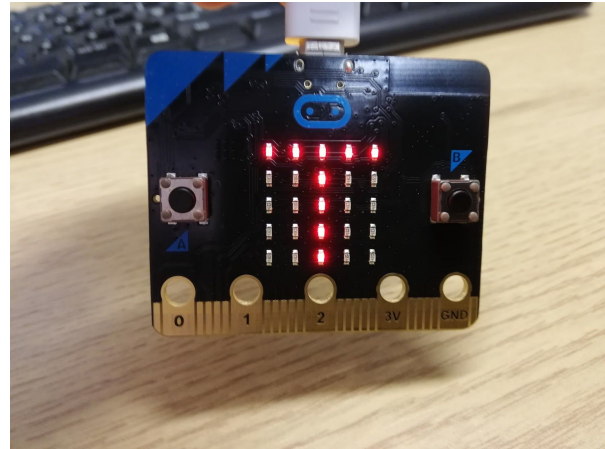


Fig. 8. Twitter application on the menu

### User Interface - Keyboard

A simple keyboard is used for text input, consisting of letters in alphabetical order followed by a space character. Similarly to menu navigation, pressing the A and B buttons scrolls left and right respectively through the letters, tilting the device away from yourself selects the current letter, and tilting the device towards yourself concludes text input. Additionally, pressing the A and B buttons while tilting left and right respectively scrolls 5 letters at a time to allow faster traversal of the alphabet.

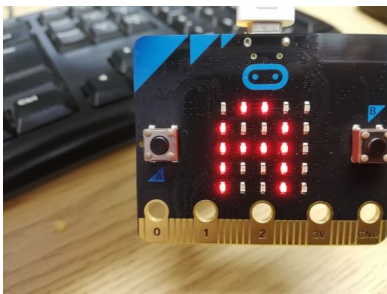


Fig. 9. Starting letter A

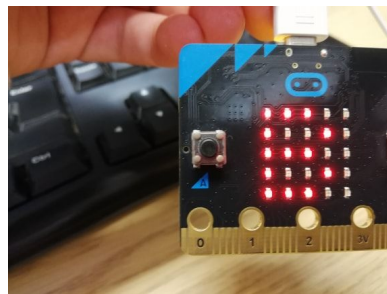


Fig. 10. Press button B, move one letter right to letter B

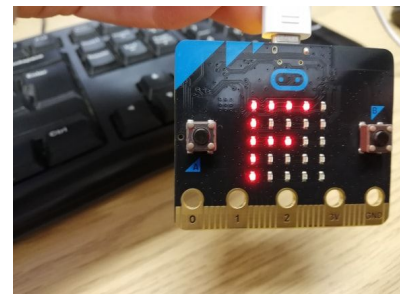


Fig. 11. Start from Fig. 9, tilt right and press button B, move 5 letters right to letter F

## Weather

Video: <https://www.youtube.com/watch?v=Z7q1v3kx5P8>

The video provided above is a visual guide for this section.

After selecting ‘W’ on the micro:bit’s top level menu, the micro:bit informs the user of the current weather in the user’s selected city. The default city is St. Andrews, though this can be customised by users as explained in the ‘website’ section above. There are numerous weather descriptions, including but not limited to: broken clouds, scattered clouds, light rain, moderate rain, haze and clear sky.

## Twitter

After selecting ‘T’ on the micro:bit’s top level menu, the user is taken to a second menu. The menu consists of an up arrow and a down arrow. If the user selects the up arrow, they are taken to the keyboard to enter a message. Selecting this message by tilting the micro:bit towards the user will send the Tweet to the B1’s Twitter account. Due to the implementation of the mesh network, a Tweet to be uploaded has a hard limit of 84 characters. The reasoning behind this is described further in the subsection *III. Product Functionality b. Implementation: micro:bit connection to server*. If the user selects the down arrow, they are shown the most recent Tweet from their subscriptions. Subscriptions can be managed via the website, and tweets shown on the micro:bit are also filtered to ensure they aren’t spam. If there are no Tweets to be shown, the user will see the text “0 tweets” and will then be taken back to the send/receive menu.

## Twitter - Spam Filter

Twitter’s own API endpoints are fairly static and offer a limited number of filtering conditions, especially in the case of the free APIs we are currently using. Additionally, their categorical conditions are mostly concerned with such more general and less subjective categories, such as the time a tweet was posted or geographic location it was posted from. These conditions may not be sufficient to stop distracting spam messages from being sent to the micro:bit, and in order to address this, we implemented an additional machine learning-based spam filter, which is invoked as a back-end feature upon a micro:bit tweet request being made. This spam filter is not directly visible to the user.



## Piano

Video: <https://youtu.be/oMJ0dM4IHwM>

Process of choosing a notification sound

Video: <https://youtu.be/97vmfSrQN5I>

Typing on the piano

We were given an add-on feature for the micro:bit, the :KLEF Piano by Kitronik <sup>[C]</sup>, which provided a set of 15 new input buttons. It also has a “buzzer” which can emit different frequencies to represent musical notes. We decided to use this new hardware as a stand for the micro:bit when it wasn’t in use, which would also make a notification sound whenever a plugged-in microbit receives a message over radio. The user is also permitted a further level of personalisation as they can select the P option on the menu and choose a specific tune to sound as their notification.

When P is selected the user is prompted to choose a tune. Each key then plays a “melody”, and the user is asked if this is the tune they want. If they confirm that it is, that tune will be saved to the micro:bit’s memory so that the same sound is used the next time the micro:bit is plugged in. If the user decides against that tune, then they are asked again what tune they want.

The piano is also able to replace the functionality of the tilt controls for the purposes of selecting menu items and typing out messages. The up and down keys are used in place of the forward and back tilts that are used for selection. The tilt commands (bar the pattern unlock) are disabled when plugged into the piano.

The user should be aware that the piano hardware can be temperamental. Power must be supplied to the piano itself via the back port not to the micro:bit. If there is a problem, the micro:bit will print “Reset” and attempt to restart the connection with the piano. When the piano and micro:bit are successfully connected the word “Piano” will print and the piano will emit a welcome tone.

## B. Implementation

This section covers the implementation of the features explained above, including summaries of methods used and underlying data structures, explanations of under-the-hood functionality, and design decisions made due to hardware or software limitations. .



## Server

The server is written in Node.js using express.js. It serves static files for the website, and also has a restful API for accessing the database and forwarding Twitter and weather requests. Some product features, such as the real-time message log, require server-side pushing, for which WebSockets were used.

## Microbit connection to server

The Python serial program allows a USB-connected microbit to access the server. The server runs a binary program with this project's mesh protocol code connected to mocked micro:bit hardware. This program simulates the behaviour of the radio protocol on a PC and sends and receives data over stdin and stdout.

Micro:bits which are not connected to the server via USB instead send an encrypted radio message containing the user's username and unlock pattern in addition to a message payload. The recipient ID will be the server's ID, 4. If another micro:bit is connected to a PC by USB, it sends all messages addressed to the server over serial rather than radio, including packets relayed over the mesh network from other microbits.

The Python serial program connects to the server via a WebSocket and relays mesh network messages both ways, while the server runs the mesh protocol binary in a child process and relays WebSocket messages to and from it.

This setup allows micro:bits to communicate with the server via the mesh network, even when not directly connected to a PC.

Micro:bit-to-server messages are then sent to a Python subprocess running an instance of the serial computer program, which makes server requests. Responses from this subprocess are sent back to the microbits over the mesh network.

The supergroup mesh protocol and our encryption system result in a message size limit of 93 bytes (see the *Messaging* section below). In order to send the username, pattern and message type, an additional  $(9 + \text{username})$  length bytes are needed, which will result in  $93 - 9 - \text{username length} = 84 - \text{username length}$  available bytes for the actual message payload.

## User accounts

The data for each user is persistently stored in a secure MariaDB database hosted on the school's servers. When a user enters their password, it is securely hashed before being stored in or checked against the database, preventing attackers from finding account details by intercepting the communication between the server and database or by gaining access to the database itself.

Once a user logs in to the website, a session key is created which allows access to protected, user-specific sections of the website for 20 minutes. Whenever the user accesses a protected section a new session key is generated, with the countdown set back to 20 minutes.

When a user logs in to the micro:bit, the username and lock pattern are stored in persistent memory until the micro:bit is reflashed. These are sent to the server whenever the micro:bit makes a request. The server passes this to the Python module, which checks the credentials against the database and then returns a temporary session key which is used to authenticate the request. This session key is not persistent; a new one is created for each micro:bit request.

## Website

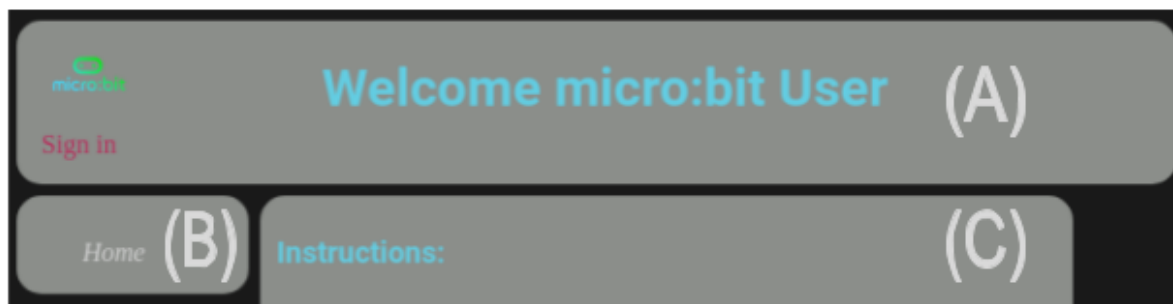


Fig. XX. A zoomed in screenshot of the website home page with the header, navigation menu and main text body labelled (A), (B) and (C) respectively.

The website, in addition to providing an interface for the user, also hosts the server. The former will be described in this subsection, while the latter will be discussed in the next subsection.

The website is built on HTML files that are linked to each other and to a single CSS file. The HTML files make extensive use of heading tags and table tags, particularly in the front page where clear sectioning of instructions for the user manual is important. Each 'section' of a web

page, labelled in Fig. 5, above, is clearly labelled using comments in order to ensure readability and a clear structure.

The CSS file uses a muted version of the general micro:bit's colour scheme, with a blue accent from the official micro:bit logo (seen in the top left corner of Fig. 5 or in more detail in Fig. 3). It also includes rectangular shapes which can be pieced together in an HTML file to replicate the appearance of the connector pins, which can be seen in Fig. 6, below.

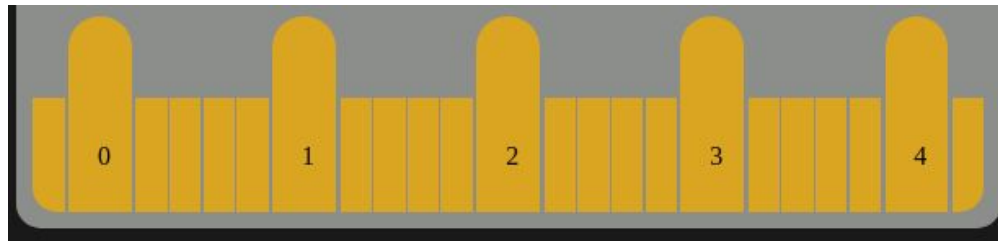


Fig. 6. A screenshot of the connection pin replications on the web pages.

Scripts were written in Javascript and JQuery and were used to repeat features that were shared between web pages, such as the navigation menu, where the script displays 'Home' if the user is not signed in or the full menu if the user is, and the connection pins seen in Fig. 6. These were implemented in the HTML files.

### User Interface - Menus

The menus and UI components are organised as applications with each collection of related functionality being put inside a subclass of `InteractiveComponent`. Events for the tilt controls and button presses are passed to the current `InteractiveComponent` subclass, and functions for switching between applications are exposed. Menus are implemented as subclasses of the `Selector` class, allowing menu items to be registered so that the user can switch to apps easily. See the homepage of the website for a diagram of UI components.

### User Interface - Keyboard

The micro:bit library includes methods for displaying letters on the screen, so the keyboard was relatively simple to implement. Methods were implemented to scroll left and right through the keyboard, scrolling 5 at a time if the micro:bit is tilted in the same direction. We experimented early in development with entirely tilt-based scrolling, featuring a scroll speed which increased over time, but this was slower to use and less accurate than the final method.

## Weather

The micro:bit can make weather requests to the server, which will then fetch the current weather from the OpenWeatherMap API. Requests must be made when logged in (i.e. with a session ID cookie), which allows the weather to be specific to each user's location setting.

## Twitter

During the course of developing this product, our group created a developer account for Twitter and used it to generate API keys for the product, allowing it to access Twitter data via developer APIs. Users are then able to view tweets on, or post tweets from, the micro:bit by having the server access the API endpoints on its behalf, forwarding content and parameters from the microbit and/or the server database as appropriate.

To view tweets, the user specifies what handles they would like to subscribe to on the website while logged into their product account. The server verifies these handles before registering them by attempting to request one of their tweets from the GET statuses/user\_timeline API endpoint, which prevents nonexistent or protected handles from being subscribed to. Then, as mentioned in the Product Functionality section, a micro:bit user can then attempt to view their most recently subscribed tweets, which prompts the server to access the timelines associated with each handle this user has subscribed to via GET statuses/user\_timeline and retrieve the most recent tweet from this collection of timelines, which is then sent back to the micro:bit.

Earlier in development, we considered using endpoints such as statuses/filter to access a continual stream of real-time tweets, which could then be put into a server-side queue and sent to micro:bit users on request. However, this would have been difficult to implement, as our server would need to maintain a continual connection to the endpoint and ensure that it can consume and store tweets at a sufficient rate (as Twitter closes the connection if too many queued tweets accumulate on their side of it). Additionally, the free version of this API allows for only one connection (through which traffic for *all* subscribers would need to be routed) whose filter conditions cannot be altered without closing it. These additional problems, combined with the fact that a constant stream of tweets might run contrary to our one of our product's intended aims of alleviating distraction by social media, lead us to focus on a RESTful based implementation.

## Twitter - Spam Filter

Our product's spam filter was implemented using a random forest algorithm-based text classifier, which is created and accessed using Python. We used Python to implement this feature due to the large body of publicly available and relevant code, documentation and discussion concerning machine learning problems, mainly provided by Scikit-Learn and its userbase, and because the

lightweight and system-agnostic nature of Python should allow the classifier to be independent of the server code or architecture.

This classifier is created and trained using code from the Scikit-Learn library. It is based on a modified version of example tutorial code <sup>[Q]</sup> along with a dataset of tweets featured in a journal article also concerning automated Twitter spam classification <sup>[F]</sup> which was manually categorized as spam or non-spam. Since the tweets in this dataset are referenced using IDs, we implemented additional scripts which used the python-twitter and OpenPyXL libraries to trawl through this spreadsheet, retrieve the textual content of each tweet where possible and save it as an individual text file in an appropriate folder based on category, allowing it to be accessed using Scikit-Learn's load-files methods and used as training/test data. Note that some IDs in the dataset have had their associated tweets deleted or otherwise rendered inaccessible since the journal was published, and so it was not possible to retrieve text fields for all tweets.

Once trained, the classifier and its associated vectorizer are tested (the results of these tests are discussed in the Testing section) and stored as pickle files, allowing them to be used later without having to re-create the classifier from scratch. These pickle files are then loaded as part of a separate Python class, which also offers a function that invokes them over a given string and returns its predicted label. This label, then, corresponds to whether or not the given string represents the text field of a spam tweet. A python script then instantiate this class, and repeatedly takes tweets into stdin and responds to them by printing 0 or 1 to stdout. The server runs this model querying script in a child process, and uses it to classify incoming Tweets.

The version of the classifier provided in our submission was trained with all 3,991 successfully retrieved spam tweets, along with 3,728 non-spam tweets. Earlier versions made use of all retrieved spam tweets, but this produced a classifier wherein the labels did not accurately correspond to the provided categories, making it unable to properly distinguish spam and non-spam.

## Piano

We found the :Klef piano's hardware to be unreliable, with some keys being pressed and not producing any input without persistent interaction, while other keys were found to be very sensitive and behaved as though they were being pressed even after the interaction has ceased. This informed our decision to not let the user "play" the piano, as key presses would often be missed or added unexpectedly, resulting in a potentially very frustrating experience.

The piano's current design can be seen described above (*III. Product Functionality, A. The Product's Features: Piano*). Functions for the piano are called whenever a message is received or when the user wishes to edit the notification sound.

Melodies were programmed into the micro:bit so that when a key is pressed on the piano in the "P" application, a tune is played. Three of the keys play melodies from actual songs, while the rest play simple tones. The melodies were made by effectively transposing the notes of the song into a period of the note, i.e. how long a single wave of the note's frequency takes to make a sound. The note's period was calculated by inverting the frequency of the note and multiplying by one million.

$$\rho = \frac{1}{frequency} \times 1000000$$

For example, the period of middle C (C4) is:

$$\rho(c_4) = \frac{1}{261.63} \times 1000000 = 3822$$

The buzzer then outputs a sound based on the period value.

Each melody is stored as an element in a vector of tuples. Each tuple represents a different note of the tune, and holds the period of the note and the length of time the note is held for.

## Messaging

### Protocol

The protocol specification<sup>[O]</sup> was created by group representatives at the supergroup B meetings, and its implementation was left to each group individually.

The protocol defines each packet as consisting of a header and payload, where the data in the message being sent is stored in the payload. The header is 4 bytes long, and is formed of the recipient ID, sender ID, message type and flags, where each value is 1 byte long. The flags byte is further subdivided into 2 bits for the priority of a message, 3 bits for the packet ID used to identify separate messages from the same sender, 1 bit to signify whether or not the message is encrypted and 2 bits for the hop count, used in the mesh network as detailed below.

To send a message from micro:bit A to micro:bit B using the protocol, micro:bit A should first send a request to communicate using a packet with a request message type and an empty

payload. Micro:bit B should respond with a packet with a response message type, and a payload containing its public key. Micro:Bit A should then read the payload of the acknowledgment packet and encrypt the payload of the data packet before sending the message.

### Security

The supergroup opted to use RSA encryption which applies public and private keys. First, two prime numbers must be chosen,  $p$  and  $q$ . These two numbers are the basis of the public and private keys. The first part of the public key is  $n$ ,

$$n = p \times q$$

We also needed to make an exponent  $e$ , which must be an integer such that  $n$  is not divisible by  $e$  and  $1 < e < \phi(n)$ . This exponent forms the second part of the public key.

Then a value  $\phi(n)$  must be made,

$$\phi(n) = (p - 1)(q - 1)$$

The private key is then formed by calculating (for some integer  $k$ ),

$$d = (k \times \phi(n) + 1) \div e$$

Where  $d$  is the private key.

Now the public and private keys can be used to encrypt and decrypt data. Data is encrypted to ciphertext  $c$  in the following way, where  $m$  is the message being encrypted,

$$c = m^e \bmod n$$

And then to decrypt,

$$m = c^d \bmod n$$

The supergroup agreed on 3 bytes of plaintext being encrypted into 4 bytes of ciphertext for ease of implementation, since the microbit offers 64 bit integer types, and modular exponentiation of an 8 byte number that would fit in 4 bytes can be done relatively easily without overflow.

Thus the message payload grows by a factor of  $4 \div 3$  when it is encrypted. This means that a maximum size encrypted message is has a payload of size:

$$\begin{aligned}
 & (128 - \text{header size}) \times 3 \div 4 \\
 & = (128 - 4) \times 3 \div 4 \\
 & = 93
 \end{aligned}$$

The initial implementation was influenced by a sample code base <sup>[L]</sup>, though it was significantly altered by the end of the implementation.

### Mesh Network

The mesh network is included in the supergroup specification<sup>[O]</sup>, and allows a micro:bit to send a message to another micro:bit which is out of range, provided that both devices are connected to the mesh network. This is achieved by requiring micro:bits to forward any packets not addressed to them to all micro:bits in range, whilst decrementing a ‘hop count’ variable, allowing each message to be forwarded a small number of times before being dropped. Because of the mobile and inconsistent nature of a micro:bit radio network, it is infeasible to keep a network map, so packets are delivered over the mesh network using a best effort policy and do not require acknowledgment from any micro:bit except the intended recipient (which still uses the protocol outlined above).

The nature of the mesh network requires micro:bits to acknowledge each unique message only once, as they may receive multiple copies from different micro:bits. To prevent repetition, our micro:bit keeps a cache of recently received messages, identified by a tuple of the sender ID and packet ID, stored up to half a second after the last time a packet is received.



## IV. Testing

Contributed by Nicola Macdonald

Evidence is provided below of extensive testing of all micro:bit and website features available in our final product. While each feature was tested thoroughly in development, by performing comprehensive tests we can confirm the expected functionality of our product and highlight edge cases which would require further development for an enterprise-ready product.

### a. Website

Due to the variety of features to be tested on the website, the tests have been organised to fit a table, and they can be seen in Table XX, below.

**Table X**

Tests for the micro:bit website: <https://oas.host.cs.st-andrews.ac.uk/JH-project/>

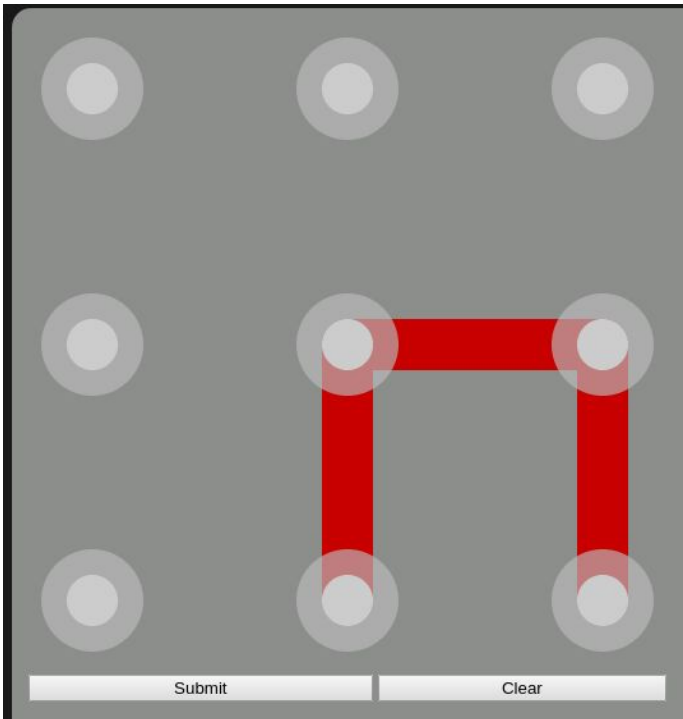
Test No.	Test Explanation	Result + Pic/Video
1.	<p>Set a pattern for the lock screen.</p> <p>This will be a feature all users will need to use at least once, so it's important the process is not faulty.</p> <p>There are three types of tests done: acceptance, boundary and extreme value.</p> <p>Acceptance testing is done to show that saving lock patterns works when used as expected. This is setting a pattern of at least 3 dots, then saving it. Then, clearing the pattern.</p> <p>The boundary tests are done to demonstrate the 3 - 9 dot length requirement for any pattern. An image example of one of the tests can be seen in Fig. 7, right, where a 4 dot pattern is about to be saved.</p>	

Fig. 7. A screenshot of the 4 dot pattern saved in the boundary tests video.

The extreme value test demonstrates attempts to make patterns with complex characteristics.

## 2. Registering user accounts.

In order to make full use of the micro:bit the user must create an account. The account's identity is the username, so a given username cannot belong to more than 1 user account.

There are three types of tests used: acceptance, boundary and extreme value.

The acceptance test demonstrates basic functionality, such as registering a unique username fulfilling an alphabet-only criterium.

The boundary tests demonstrate how the database reacts to users registering usernames and passwords below the limit of 3 characters.

The extreme value tests, split into two videos, demonstrate how the database does not allow a username of one thousand characters, but it does allow a password of one thousand characters. Fig. 8, right, demonstrates this.

### Acceptance Test:

<https://www.youtube.com/watch?v=FdfPENxI2qM>

### Boundary Tests:

<https://www.youtube.com/watch?v=QuRTMIKkaJg>

### Extreme Value Tests:

<https://www.youtube.com/watch?v=gx4TNPT-6CM>

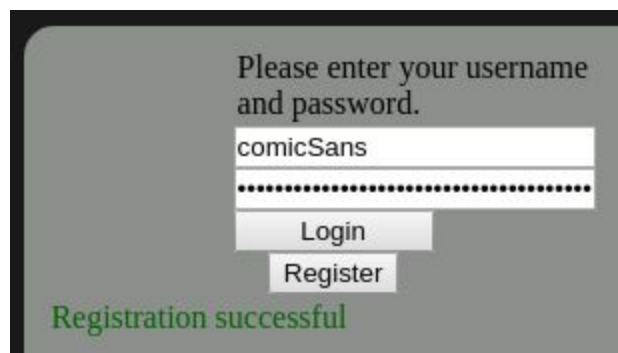


Fig. 8. A screenshot showing the successful registration of a user account with a password 1000 characters long.

### Acceptance Tests:

[https://www.youtube.com/watch?v=\\_7nMsMhwoOw](https://www.youtube.com/watch?v=_7nMsMhwoOw)

### Boundary Tests:

<https://www.youtube.com/watch?v=NanevFVQegg>

### Extreme Value Tests

Part 1:

<https://www.youtube.com/watch?v=ihOnBAnQNdE>

Part 2:

<https://www.youtube.com/watch?v=Zjrfojuk6ds>

### 3. Log in.

After creating a user account, the user must be able to sign in using the same credentials they used to create their account.

Fig. 9, right, shows the ‘Incorrect username or password’ message provoked by a failed login attempt.



Fig. 9. A screenshot demonstrating a user attempting to log in using the wrong credentials.

### Acceptance Test:

<https://www.youtube.com/watch?v=ec5rjLtReMg>

### 4. Log out.

For security reasons users must be able to successfully log out of their account.

Figures 10 and 11 show how the navigation bar and the ‘sign in’ link changes depending on whether the user is currently signed in or signed out.

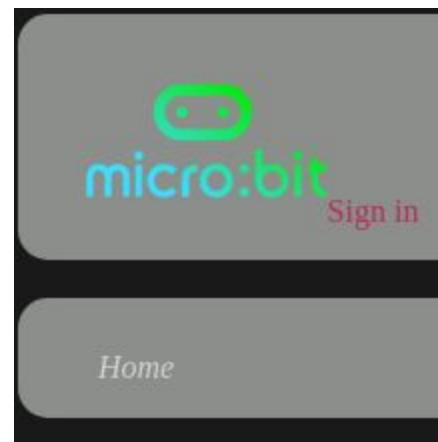


Fig. 10 A screenshot of the website’s ‘sign in’ link and navigation bar before the user is signed in.



Fig. 11. A screenshot of the website's 'sign out' link and navigation bar after the user has signed in.

### Acceptance Test:

<https://www.youtube.com/watch?v=O1IC1Z6B4Iw>

#### 5. Change city to display weather for.

This is a core micro:bit feature; the user is expected to be able to select a city to display the current weather for, and should function as expected.

Fig. 12, right, is a still from the video listed demonstrating the auto-complete function of the drop-down list.

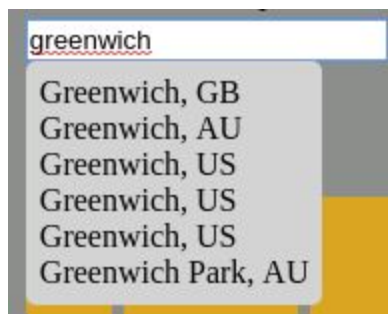


Fig. 12. A screenshot of the 'weather' page demonstrating the auto-complete drop-down list feature. Currently no options are selected.

- Acceptance Test
  - Valid city name
  - Invalid city name
- Autofill city name example

#### 6. Manage Twitter subscriptions.

This is a core micro:bit feature; the user will be able to choose whose Twitter handles they want to get Tweets from when using the Twitter feature on the micro:bit, and should function as expected.

Fig. 13, right, shows that the user may subscribe to a handle with non-alphanumeric characters but they cannot subscribe to a handle they are already subscribed to.



Fig. 13. A screenshot of the 'Twitter' page following the boundary test.

### Acceptance Test:

<https://www.youtube.com/watch?v=bLXsJqACKk4>

### Boundary Tests:

<https://www.youtube.com/watch?v=qbEF3TdYJQ0>

#### 7. View message history log.

This is a core micro:bit feature and should function as expected.

Videos to be uploaded:

- Acceptance Test
  - Receive a message on the micro:bit and view it on the log
  - Clear message log

## b. Messaging

In order to more easily test the protocol implementation during development, a radio simulator was made as part of this project. It allows simulated micro:bits to be placed in 2-dimensional Cartesian space, and only allows messages to be broadcast to other simulated micro:bits within a fixed Euclidean distance as an approximation of real life (see Figures 14 and 15 below).

For example it can be used to demonstrate the protocol working correctly with multiple hops and encryption in Figures 16 and 17, below.

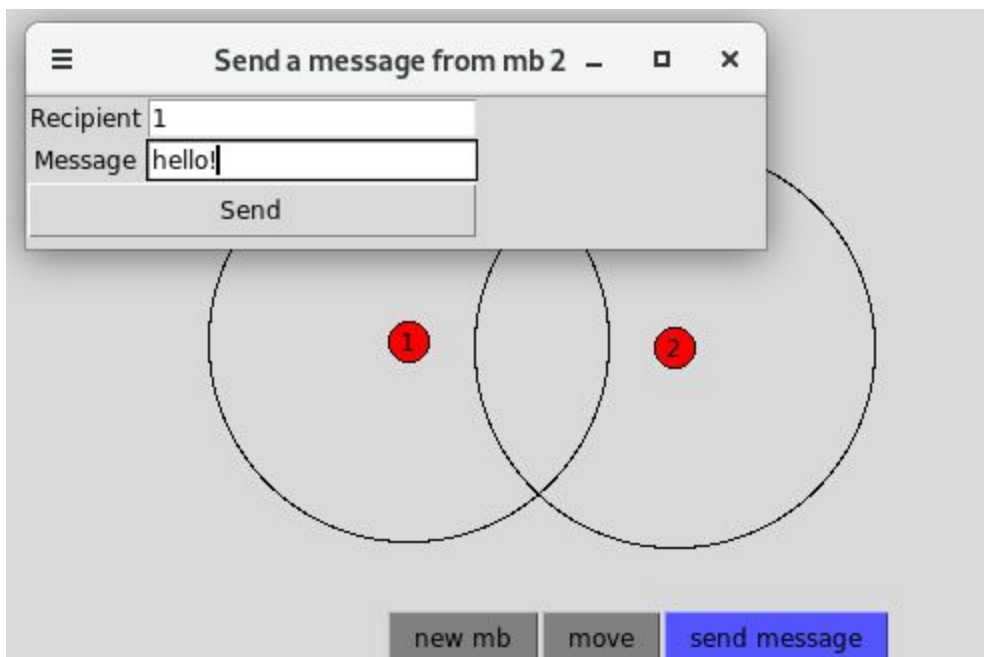


Fig. 14. The radio testing simulator - attempting to send a message between out-of-range micro:bits

```
mb 2 sent: b'\x01\x02\x00\x07'
```

Fig. 15. The output from running the simulation in Fig. 14. A request is visible, showing that the first byte is 1, the recipient ID, the second is 2, the sender ID, and the 3rd byte is 0, signifying that it is a request. The 4th byte being 7 (00000111 in binary) means that the priority (first 2 bits) is 0, packet ID (next 3 bits) is 0, encryption is enabled (next bit) and the hop count is 3 (last 2 bits).

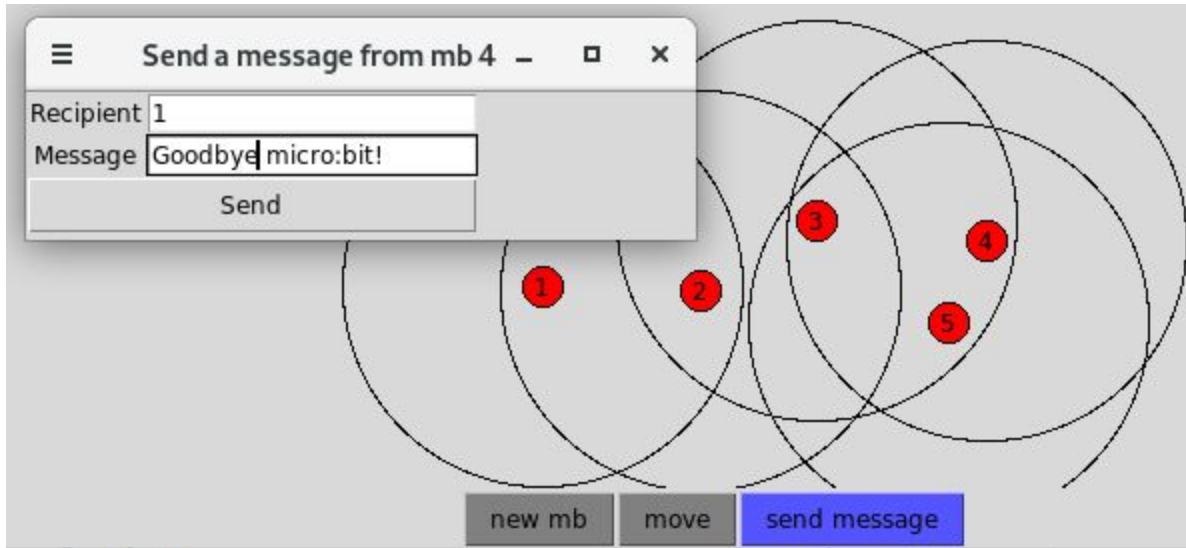


Fig. 16. A setup sending between micro:bits out of radio range (from ID 4 to ID 1) with intermediary micro:bits.

```
mb 4 sent: b'\x01\x04\x00\x07'
mb 3 sent: b'\x01\x04\x00\x06'
mb 4 sent: b'\x01\x04\x00\x05'
mb 2 sent: b'\x01\x04\x00\x05'
mb 1 sent: b'\x04\x01\x01\x07=+3\xd8\x00\x00\x00\x00\x8bz\xc9\xec\x00\x00\x00\x00'
mb 5 sent: b'\x01\x04\x00\x05'
mb 2 sent: b'\x04\x01\x01\x06=+3\xd8\x00\x00\x00\x00\x8bz\xc9\xec\x00\x00\x00\x00'
mb 3 sent: b'\x04\x01\x01\x05=+3\xd8\x00\x00\x00\x00\x8bz\xc9\xec\x00\x00\x00\x00'
mb 5 sent: b'\x04\x01\x01\x04=+3\xd8\x00\x00\x00\x00\x8bz\xc9\xec\x00\x00\x00\x00'
mb 1 sent: b'\x04\x01\x01\x05=+3\xd8\x00\x00\x00\x00\x8bz\xc9\xec\x00\x00\x00\x00'
mb 4 sent:
b"\x01\x04\x02\x0fy\x08dV\xaeVj\x99\xd4\xb5I'\x93G\xca\xcd\xb3\xb8\xa7\xc7L\xcc\x07k"
mb 3 sent:
b"\x01\x04\x02\x0ey\x08dV\xaeVj\x99\xd4\xb5I'\x93G\xca\xcd\xb3\xb8\xa7\xc7L\xcc\x07k"
mb 5 sent:
b"\x01\x04\x02\x0ey\x08dV\xaeVj\x99\xd4\xb5I'\x93G\xca\xcd\xb3\xb8\xa7\xc7L\xcc\x07k"
mb 2 sent:
b"\x01\x04\x02\ry\x08dV\xaeVj\x99\xd4\xb5I'\x93G\xca\xcd\xb3\xb8\xa7\xc7L\xcc\x07k"
mb 1 got from 4: Goodbye micro:bit!
```

Fig. 17. Output from running the simulation in Fig 16. Radio data can be seen flooding the network, with the intended micro:bit receiving the message on the last line. Note that data is sent encrypted.

The supergroup communication will be demonstrated at the supergroup B demo session, as while the mesh network has been successfully tested it's very difficult to show this through screenshots or videos. We do however have a video demonstrating our micro:bit successfully receiving an

encrypted message sent from another group's micro:bit in our supergroup:

<https://youtu.be/Vdjl2kQoMKQ>

### c. Weather

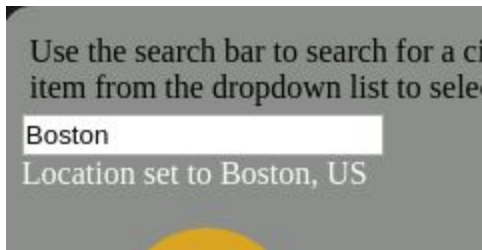


Fig. 18. A screenshot of the website showing the user has selected their city to be Boston, US.



Fig. 19. A screenshot of the api the server uses to send the current weather of a selected city to the micro:bit.

The video shown in *III. Product Functionality a. The Product's Features: Weather* demonstrates acceptance testing. Figures 18 and 19 above show via still images what information the server receives before it is sent to the micro:bit.

### d. Tweets

**Table 1**

*Test table for sending and receiving Tweets*


Test No.	Test Explanation	Result + Pic/Video
1.	<p>Sending a Tweet.</p> <p>The user is expected to be able to send Tweets to the group's developer Twitter account.</p> <p>The acceptance test demonstrates that the user can send a Tweet via the group's Twitter account. Fig. 20, right, shows as a still image the Tweet sent in the video provided.</p>	

Fig. 20. A screenshot of a Tweet that was sent via micro:bit that has successfully been posted on the group's timeline.

**Acceptance:**

<https://www.youtube.com/watch?v=ReU2KC18suo>





correctly labeled as non-spam, while the bottom left represents non-spam tweets incorrectly labeled as spam.

<b>Table 2</b>		
<i>Confusion matrix for the current model</i>		
Actual Label		
Predicted Label	0	1
0	689	61
1	158	636

### Classification report

This table lists the results of the `classification_report` function for our classifier, which calculates three main accuracy metrics for each label to be classified: precision, which is the ratio of true positives to all positives; recall, which is the ratio of true negatives to all negatives; and F1 score, which is the harmonic mean of the prior two values. Also listed is support, i.e. the number of tweets in the test data corresponding to each label, and average values for each metric. (The function normally provides micro, macro and weighted average values, but these were all equal for this classifier, so they have been collated into one row.)

<b>Table 3</b>				
<i>Classification report for the current model</i>				
Label	Precision	Recall	F1-Score	Support
0	0.81	0.92	0.86	750
1	0.91	0.80	0.85	794
Average:	0.86	0.86	0.86	1544

Overall accuracy, as calculated by `accuracy_score` = 0.8581606217616581

Our current classifier has an overall accuracy of 86%, according to the results of these Scikit-Learn evaluation methods. Of note is the fact that the lower left entry in the confusion matrix is around twice the value of the upper right entry, and that label 1 has a significantly

higher precision value than label 0 (and vice versa for label 0). This indicates that most of the current classifier's inaccurate predictions result in non-spam tweets being incorrectly labeled as spam. The alternative, which would result in more genuine spam tweets making it past the filter, and hence more sources of distraction, is much less desirable based on our project goals.

#### g. User Interface

Having received permission to collect anonymous feedback on our project, we created a series of tasks for a user to complete followed by a short feedback form, both of which can be found in **Appendix C** along with the responses to the form. These were used as the basis of our user test for the micro:bit, through which we aimed to receive and respond to constructive feedback from users to improve the overall user experience. The user tests were carried out in Jack Cole labs, using lab machines and the provided micro:bits and hardware. Participants were encouraged to ask questions at any point and a member of the group was on hand at all points to respond to any queries.

The first wave of user tests was completed with three participants. Participants were able to complete all website-related tasks and use the micro:bit log-in system, but due to a software bug were unable to proceed further with the tasks. Because of this we didn't ask for feedback through the feedback form as the experience was incomplete, but we were able to receive informal feedback and improved aspects of the website instructions and task phrasing in response.

The second wave of user tests was completed with six participants. The process was the same as before but the software issue was fixed so participants were able to complete all tasks and fill in the feedback form.

The responses to the feedback form were varied but largely positive, and would prove very helpful for further iterations of the product. Two main problems were highlighted from the responses to the second written question; the unlocking mechanism being unintuitive and unclear initially, and the piano hardware being unresponsive. Given more time we would improve either the pattern unlock on the micro:bit or the tutorials for it on the website. Other functions of the UI were complimented in the first written question, including the use of buttons and tilting, the messaging functionality and the messenger and weather applications.

## V. Evaluation and critical appraisal

Contributed by Daniel Key

### A. Our Project Compared to Original Objectives

The original objectives as understood from the lecture slides provided<sup>[H]</sup> and the emails sent are as follows:

Build a minimal internet device that allows the user to communicate with other people via internet or texts. The user is expected to be able to access social media or view their emails and read the news and/or the weather. A website is expected to be designed and created to assist these goals.

Security must be implemented; there must be a way to ‘lock’ the device such that only its owner can access information in it; messages must be encrypted; and measures must be taken such that micro:bits do not impersonate one another when, for example, a particular micro:bit is the expected receiver of a message.

Artificial intelligence (AI) is expected to be implemented such that it decides what information the user sees. It is up to the group to decide what information gets processed by the AI.

The user may also use a piano-like device as an add-on for the device. It is up to the group to decide how to incorporate the piano, however it must not be a requirement for the device’s basic functionality.

We achieved the majority of original objectives but some either have limited implementations or were not implemented at all. As a team we agree that Twitter can act as an effective news feed should the user choose to follow the appropriate Twitter feeds. There is no direct implementation to a news API.

With reference to micro:bit impersonation the supergroup protocol states that micro:bits should keep the same public key, and if someone replies to a send request with a public key, the device assumes that they are who they say they are. There is no way of knowing if a message comes from who it says it comes from, since there is no support for message signing in the protocol - the supergroup agreed to include passphrases in messages in the form of public keys, as explained in the protocol section. Such passphrases are included in the message body and can be used to prevent impersonation since they are originally shared in person and kept secret. This group’s implementation does not automatically insert or check passphrases.

Email access was not implemented at all as we had already established two effective methods of communication, and it was understood that email was not required. Furthermore, emails are often longer than messages on other modes of communication, making an email application unsuitable for the micro:bit due to previously discussed text display and message size issues.

## B. Our Project Compared to Others' Work

While there is no evidence of a project of this scale being attempted on the BBC micro:bit, similar projects have been completed on a more powerful microprocessor, the Raspberry Pi. Some of these projects have been briefly summarised and compared below.

A series of blog posts entitled 'Mobile Mesh Networks with the Raspberry Pi'<sup>[K]</sup> outlined some basic steps taken to create a mesh network allowing internet access through a network of Raspberry Pi devices. This allowed a download speed of 250kb/s on a device two nodes away from a device connected to ethernet, compared to 400kb/s when directly connected. Similarly our micro:bit is able to upload and download data to and from our server across the mesh network, across a radio connection between the micro:bit nodes.

The 'Dress for the Weather' project for the Raspberry Pi<sup>[J]</sup> uses the same OpenWeatherMap API to access weather data as is used in our weather application. This project aims to retrieve a weather forecast for a future date and time, which could be a potential extension to our existing weather application. However, our website provides autofill and data validation for the inputted city names, which this project does not.

There are some similarities between our project and existing projects on the micro:bit, for example the Micro morse phone<sup>[M]</sup> is a simple demonstration of the micro:bit's radio functionality.

It can be observed that significantly larger scale implementations of communication can be made using the Raspberry Pi as written by Jason Malliss, Vaibhav Patel, Sushilkumar Yadav, Himanshu Varun, Dr. Suprava Patnaik in their paper Real Time Communication (RTC) Device using Raspberry Pi<sup>[I]</sup>. These are much larger than we were intending to make with the micro:bits, but comparison between the micro:bit and the Raspberry Pi must take into consideration both devices' capacity to reach such a scale. Given our device's memory issues, it may not be possible to make a larger network with so many other technologies involved. The Raspberry Pi also has a much larger body of add-on technologies that allow it to implement more effective communication methods.

## VI. Conclusion

Contributed by Peter Goad

### A. Our Summary

The final product produced reflects our intended initial product, providing simple interactions and few sources of distraction. This shows that we had a clearly-defined goal and kept to our design choices effectively. The final product satisfies the basic requirements and has additional features implemented which ensures that our micro:bit offers a comfortable user experience.

The micro:bit and its hardware limitations presented an array of development challenges, especially its limited memory. There were multiple occasions where the code that we were running or our data structures caused the device to crash due to memory errors. Despite these challenges, we were able to complete the project successfully with a fully functional minimal communication device.

### B. Key Achievements and Drawbacks

The project maintained its streamlined, distraction free approach throughout to produce a product that is functional and not a problem to users who have other tasks which require their attention.

The design of the user interface is simple, with used letters to clearly represent each application and to allow the user to smoothly navigate between them. One of our greatest achievements beyond the original specification is the implementation of a pattern unlock system which grants greater security to the micro:bit, as the chances of guessing the password correctly are extremely low (see III. Product Functionality, B. Implementation).

Our AI spam filtering of Tweets is also a significant feature as it allows us, as developers, to have a finer degree of control over what web content is sent to the micro:bit, and to compensate for the absence of more complex filters in the (free) Twitter APIs. This extra control makes it easier to maintain the product's unintrusive properties without reducing the user's social connectivity.

Our implementation, however, is still limited as there are no options to return to previous states from the current one without resetting the micro:bit. For example, if you start writing a message and decide against it then you must either proceed with the message or reset the micro:bit. This is a simple fix, but it may not be intuitive to some, and it may also disrupt the user experience.

## C. Future Work

In previous hand-ins for this project, our group planned additional features and improvements that could be added to the product at that stage, some of which were not implemented due to time constraints and prioritization of other features. A brief summary of unimplemented features which our group views as suitable additions to a future version of the product is as follows:

- Text message templates
  - A system of pre-written messages or message templates which could be sent from the micro:bit instead of having to manually type every message with the keyboard saving the user time typing long messages.
  - Users should be able to customise the pre-written messages available to them via the website.
- An expanded character set for micro:bit text display and keyboard, including:
  - Lower case letters and punctuation.
  - Replacements for certain emojis.
  - Icons to represent weather conditions (e.g. sunny, cloudy, raining).
- Utilising the input pins underneath the micro:bit's LED display to enable additional input options, potentially including menu navigation.
  - During the course of development, it was found that the input pins on the bottom of the microbit could be activated using one's fingers as opposed to another device.
- Expanded filtering
  - Alternate means of filtering, such as Bayesian evaluation, could be explored and implemented if they prove to be more efficient than our current filtering system.
  - Stricter filtering based on whitelisting or blacklisting message sources could be implemented, as certain users may prefer this more clearly defined option.
- Impersonation prevention and Encryption
  - The supergroup had hoped to extend the implementation so that when you receive a message from a new public key, you would verify that it is from the person it claims to be from, by mapping public keys to the ID of each micro:bit, preventing impersonation.
  - Larger keys and block sizes and adding nonces to make encryption more secure.

- Spam filter refinements
  - As mentioned earlier in the report, the current version of our spam classifier uses a small subset of the non-spam tweets we were able to retrieve in order to ensure proper binary classification. An alternative way to ensure this behaviour would be to modify the weight associated with the spam label, potentially in a programmatic fashion based on the ratio between the number of tweets in each category. This would allow the classifier to be trained with a larger dataset, which in turn should help to improve its accuracy.
  - The dataset used to train our model, though comprehensive, was collected in mid-2016, and hence may not be representative of modern Twitter traffic and what features mark it as spam or non-spam. More up-to-date models could be created by searching for more recent studies and data sets or collecting our own data, potentially informed by user feedback.
  - More efficient ways of saving and/or pipelining the classifier do exist, such as joblib utilities, which may reduce the time needed to access the model or the space needed to store it.
  - We made very few alterations to the classifier attributes provided by the example code given earlier, as we were still able to produce reasonably accurate predictions without modifying them. We could experimentally determine if other attributes, such as a larger set of features recognized by the vectorizer, would result in a more accurate classifier.
  - The article which we sourced our dataset from (Chen, Yeo, Lau & Lee, 2018) does mention that features besides text can be used to identify spam tweets, such as the posting account's recent activity or interactions with other users. A more advanced spam filter could also take these features into account.
  - If necessary, the provided code could be readily extended to provide filters for other forms of online content, such as Facebook messages, assuming a suitable dataset could be found or assembled.
  
- Utilising the keys of the piano for more general input
  - Leftmost and rightmost keys used in place of button A and button B.

## Appendix A: References

- [A] Roberts, J., Yaya, L. and Manolis, C. (2014). The invisible addiction: Cell-phone activities and addiction among male and female college students. *Journal of Behavioral Addictions*, 3(4), pp.254-265.
- [B] Jefferson, C., Linton, S. (2018). Slideshow of B-1's Group Project for academic year 2018-2019, University of St Andrews, retrieved April 18, 2019, <https://studres.cs.st-andrews.ac.uk/CS3099/0-General/JH-induction.pdf>
- [C] :KLEF Piano for the BBC micro:bit. (n.d.). Retrieved April 18, 2019, from <https://www.kitronik.co.uk/5631-klef-piano-for-the-bbc-microbit.html>
- [D] Steve Linton. (2018, September 28). CS3099 JH Team Project SCRUM. University of St Andrews. Retrieved April 18 2019, from <https://studres.cs.st-andrews.ac.uk/CS3099/Lectures/Lecture02.pdf>
- [E] Solomons, O. (n.d.). Group B1 Website. Retrieved April 18, 2019, from <https://oas.host.cs.st-andrews.ac.uk/JH-project/>
- [F] Chen, W., Yeo, C. K., Lau, C. T., & Lee, B. S. (n.d.). A study on real-time low-quality content detection on Twitter from the users' perspective. Retrieved from <https://journals.plos.org/plosone/article?id=10.1371/journal.pone.0182487>
- [G] Das, A., Bonneau, J., Caesar, M., Borisov, N., Wang, X. (22 Feb 2014). The Tangled Web of Password Reuse. Retrieved April 18, 2019, from [http://www.jbonneau.com/doc/DBCW14-NDSS-tangled\\_web.pdf](http://www.jbonneau.com/doc/DBCW14-NDSS-tangled_web.pdf)
- [I] Jason Malliss, Vaibhav Patel, Sushilkumar Yadav, Himanshu Varun, Dr. Suprava Patnaik. (2018, February). Real Time Communication (RTC) Device using Raspberry Pi. Department of Electronics and Telecommunication, Xavier Institute of Engineering, Mahim, Mumbai, India. <https://www.ijser.org/researchpaper/Real-Time-Communication-RTC-Device-using-Raspberry-Pi.pdf>
- [J] Raspberry Pi Foundation. Dress for the Weather Python Project. Accessed 18 Apr, 2019, <https://projects.raspberrypi.org/en/projects/dress-for-the-weather/5>



[K] Eric Erfanian. (09 November 2012). Mobile Mesh Networks with the Raspberry Pi. Accessed 18 Apr, 2019, <http://www.ericerfanian.com/mobile-mesh-networks-with-the-raspberry-pi-part-4/>

[L] C++ program to encrypt and decrypt the string. Accessed 18 April 2019, <http://www.trytoprogram.com/cpp-examples/cplusplus-program-encrypt-decrypt-string/>

[M] Micro Morse Phone project. Accessed 18 Apr 2019, <https://make.techwillsaveus.com/microbit/activities/micro-morse-phone>

[O] CS3099 micro:bit Network Protocol Definition: Supergroup B. Accessed 18 April 2019, [https://oas.host.cs.st-andrews.ac.uk/JH-project/protocol\\_spec.pdf](https://oas.host.cs.st-andrews.ac.uk/JH-project/protocol_spec.pdf)

[P] Emanuel von Zezschwitz , Paul Dunphy , Alexander De Luca (2013, August 27) Patterns in the wild: a field study of the usability of pattern and pin-based authentication on mobile devices, Proceedings of the 15th international conference on Human-computer interaction with mobile devices and services, Munich, Germany [doi> 10.1145/2493190.2493231]

[Q] Malik, U. (2019, February 17). Text Classification with Python and Scikit-Learn. Retrieved April 18, 2019, from <https://stackabuse.com/text-classification-with-python-and-scikit-learn/>

## Appendix B: Our SCRUM Meetings This Academic Year

Table 1. <i>B1's sprint cycles 09/10/18 - 30/11/18</i>		
Sprint No.	Sprint	Sprint Progress
1	09/10/18 - 15/10/18	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Joanna as SCRUM master.</li> <li>2. Assigned Nicola as Product Owner.</li> <li>3. Assigned Daniel to attend supergroup meetings.</li> <li>4. Created product and sprint backlogs.</li> <li>5. Assigned Python as the group's programming language.</li> <li>6. Members assigned to write various parts of the first hand-in.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (6) Completed.</li> </ul>
2	15/10/18 - 29/10/18 (Includes Independent Learning Week)	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Daniel and Peter to work on binary input.</li> <li>2. Joanna to work on radio communications.</li> <li>3. Nicola to design the website.</li> <li>4. Oli to create a server for the website.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1) Binary input changed to alphabetical input using a scrollable keyboard. Otherwise completed.</li> <li>• (2) Basic radio communication developed and tested. To be continued in the next sprint.</li> <li>• (3), (4) Completed.</li> <li>• Additionally, Oli created a Makefile for pre-processing python files</li> </ul>
3	29/10/18 - 05/11/18	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Peter, Joanna and Daniel to merge radio, keyboard and messaging.</li> <li>2. Nicola to write HTML implementation of website.</li> <li>3. Oli to work on memos.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2) Completed.</li> <li>• (3) Serial communication implemented as a prerequisite</li> </ul>

		<p>to memos.</p> <ul style="list-style-type: none"> <li>• Additionally, Peter researched Twitter APIs.</li> </ul>
4	05/11/18 - 12/11/18	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Nicola to incorporate CSS into website.</li> <li>2. Joanna, Daniel and Oli to test communication between two micro:bits.</li> <li>3. Peter to apply for Twitter developer account.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (3) Completed.</li> <li>• (2) Completed with reworking of keyboard and menus for consistent gesture and button control.</li> </ul>
5	12/11/18 - 19/11/18	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Daniel to be SCRUM master.</li> <li>2. Switched to C++ as programming language.</li> <li>3. Joanna, Daniel and Oli to re-implement all code in C++.</li> <li>4. Peter to message supervisor and module co-ordinators concerning the Twitter API's Terms and Conditions.</li> <li>5. Nicola and Joanna to research AI filtering techniques.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (3) To be continued in the next sprint.</li> <li>• (4), (5) Completed.</li> </ul>
6	19/11/18 - 26/11/18	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Joanna to implement Tweet filtering.</li> <li>2. Daniel and Oli to implement supergroup protocol.</li> <li>3. Oli and Joanna to re-implement radio communication in C++.</li> <li>4. Oli to re-implement serial communication in C++.</li> <li>5. Peter to integrate Twitter API.</li> <li>6. Nicola to create micro:bit instructions page on the website.</li> </ol> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2), (4) (5), (6) To be continued in the next sprint.</li> <li>• (3) Completed.</li> </ul>
7	26/11/18 - now	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Nicola to create micro:bit instructions page on the website.</li> <li>2. Nicola and Joanna to work on implementing AI.</li> <li>3. Daniel and Oli to finish implementing supergroup protocol.</li> <li>4. Peter to work on linking Twitter to the server.</li> <li>5. Nicola to create and format the report.</li> <li>6. Group members to write subsections of report and proofread.</li> </ol>

**End of Sprint**

- (1), (2), (4) To be continued in the next sprint.
- (3), (5), (6) Completed.

*B-1's sprint cycles 29/01/19 - 06/03/19*

**Sprint  
No.**

**Sprint**

**Sprint Progress**

1

29/01/19 -  
05/02/19

**Beginning of Sprint**

1. Assigned Dan to create secure user accounts.
2. Assigned Joanna to tidy up website, clear user logs on server and download logs to client computer
3. Assigned Oli to pull current weather from a specific city.
4. Assigned Nicola and Peter to code posting or pulling Tweets to/from Twitter using Twurl.

**Problems encountered during Sprint**

- Twurl found to be too difficult to implement.

**End of Sprint**

- (1), (2), (3) Completed.
- (4) Not completed.
- Dan and Oli also created a functional login system to the website. The user accounts are made secure by using a hashing algorithm, and are currently stored in MariaDB.

2

06/02/19 -  
11/02/19

**Beginning of Sprint**

1. Oli to move message logs to database
2. Joanna to improve formatting of website.
3. Assigned Nicola and Peter to pull and post Tweets from Twitter as stated in (4) in the previous week.

**Problems encountered during Sprint**

- Work times reduced to the bio-medical building catching fire.

**End of Sprint**

- (1), (2), (3) Completed.
- Dan attended supergroup meeting.

3

12/02/19 -  
18/02/19

**Beginning of Sprint**

1. Assigned Dan to implement supergroup's flood mesh network protocol.
2. Assigned Joanna and Nicola to implement RSA encoding.
3. Assigned Peter to filter pulled Tweets such that it can be displayed on a micro:bit.
4. Assigned Oli to attach message logs to the database and implement micro:bit lock screen.

		<p>5. Members to consider the project's unique selling point.</p> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• C m:b code: button handlers throwing errors on lock screen</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2) not completed on account of illness and researching RSA and learning C++ respectively.</li> <li>• (3), (4) completed.</li> </ul>
4	19/02/19 - 25/02/19	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Dan to implement network protocol.</li> <li>2. Assigned Dan to research text to speech to use piano.</li> <li>3. Assigned Nicola and Joanna to implement RSA.</li> <li>4. Assigned Oli to implement setting locking patterns on the website.</li> <li>5. Assigned Peter to refine filtering.</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Server crashes randomly.</li> <li>• Functional text-to-speech application created but the micro:bit piano speaker isn't capable of playing speech</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1) Not completed.</li> <li>• (2), (4), (5) Completed.</li> <li>• (3) Not completed. Nicola implemented generating public and private keys, Joanna implemented decoding.</li> <li>• Members agreed to focus on user interface as the unique selling point. Members to also research during spring break any further possible unique selling points.</li> <li>• Piano extension agreed to be used as a siren if the user fails to login in x tries and be used to notify the user if they receive a message/Tweet.</li> </ul>
5	26/02/19 - 05/03/19	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Nicola to complete encryption using RSA.</li> <li>2. Assigned Nicola to head report.</li> <li>3. Assigned Joanna to complete encryption.</li> <li>4. Assigned Peter to complete refinement of filtering pulled Tweets.</li> <li>5. Assigned Dan to implement mesh network protocol.</li> <li>6. Assigned Dan to implement piano extensions as described in previous sprint.</li> <li>7. Assigned Oli to look into server crashes and fix them.</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Generating keys for RSA will randomly leave incorrect remainders, such as <math>ij = fi(n) - 1 \pmod n</math>, as described in CS3099 lecture slides. Implementing original RSA method leads to overflow where public key <math>&gt; 26</math>. May need to use Byte arrays, although issues such as supergroup protocol must be considered.</li> <li>• Very little information on how public keys will be shared and how private keys will be generated, so no implementation of encryption</li> </ul>

6	06/03/19- 12/03/19	<p>and decryption.</p> <ul style="list-style-type: none"> <li>Deadlines from other modules.</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>(1) Not completed on account of problems encountered during sprint.</li> <li>(2) Introduction complete. SCRUM meeting section complete to most recent SCRUM meeting.</li> <li>(3) Completed encryption and decryption.</li> <li>(4), (5), (6) Not completed.</li> <li>(7) Completed.</li> <li>Oli also was able to transfer lock patterns as described on the website to the micro:bit, so the user can now customise their lock patterns.</li> </ul> <p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>Members to fill out report: Nicola to finish SCRUM meeting summaries, Peter to head 'Current product', Oli to head 'Changes made this semester', Dan to head 'Super-group interaction' and Joanna to head 'Future implementation'.</li> <li>Dan to raise questions on how encryption is expected to be implemented in the next super-group meeting.</li> </ol> <p><b>End of Sprint</b> 12th March 2019</p>
---	-----------------------	---

Sprint No.	Sprint	Sprint Progress
1	12/03/19 - 18/03/19	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>Assigned Nicola to set accounts to access on Twitter and receive tweets.</li> <li>Assigned Joanna and Nicola to fix errors in encryption</li> <li>Assigned Joanna to start piano hardware solution</li> <li>Assigned Dan and Oli to make Mesh network</li> <li>Assigned Oli to make a large scale network simulation</li> <li>Assigned Nicola and Peter to create machine learning spam filter and research pre-made filters</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>Posting and retrieving tweets didn't work</li> </ul>

2	19/03/19 - 25/03/19	<ul style="list-style-type: none"> <li>• Away over Spring Break so no access to labs</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2) Completed.</li> <li>• (3), (5) Not Completed.</li> <li>• Oli also made micro:bit error message when not connected to the the python serial program and fixed server crashes</li> </ul> <p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Oli to make a basic simulator for protocol testing</li> <li>2. Assigned Nicola to test python encryption code</li> <li>3. Assigned Nicola to test Twitter code</li> <li>4. Assigned Nicola and Peter to continue AI filtering for retrieved Tweets</li> <li>5. Assigned Joanna to make method to test piano</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Ran out of micro:bit memory</li> <li>• Where to store trained AI model</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (4), (5) Not completed.</li> <li>• Completed further encryption debugging.</li> </ul>
3	26/03/19 - 01/04/19	<ol style="list-style-type: none"> <li>1. Assigned Oli to continue with the simulation</li> <li>2. Assigned Oli to try to free up some more memory on the device</li> <li>3. Assigned Peter to find training data for library appropriate for spam filtering, train the model and implement it onto the server</li> <li>4. Assigned Joanna to have the piano make a noise</li> <li>5. Assigned Nicola to continue on Twitter.</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Memory errors sending tweets</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2), (4), (5) Completed.</li> <li>• (3) Not Completed.</li> </ul>
4	02/04/19 - 08/04/19	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Nicola to continue generating public and private keys</li> <li>2. Assigned Peter to test spam data sets</li> <li>3. Assigned Oli to make dynamic protocol simulation work</li> <li>4. Assigned Joanna to add functionality to piano</li> <li>5. Assigned Dan to test mesh network and encryption with supergroup</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Memory error sending tweets</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1) Not completed due to absence.</li> <li>• (2), (3), (4), (5) Completed.</li> </ul>

5	09/04/19 - 15/04/19	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Peter to test Twitter filtering</li> <li>2. Assigned Joanna to extra piano functionality</li> <li>3. Assigned Joanna, Nicola and Oli to change encryption according to new supergroup standard</li> <li>4. Assigned Oli to make serial mesh</li> <li>5. Assigned Dan to make questionnaires and instructions</li> <li>6. Assigned Nicola to personalise where tweets come from</li> <li>7. Assigned Dan and Nicola to report writing</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Twitter python code gone wrong</li> </ul> <p><b>End of Sprint</b></p> <ul style="list-style-type: none"> <li>• (1), (2), (3), (4), (5), (6) Completed</li> <li>• (7) Not Completed.</li> </ul>
6	15/04/19- now	<p><b>Beginning of Sprint</b></p> <ol style="list-style-type: none"> <li>1. Assigned Oli and Peter to merge spam filter with server</li> <li>2. Assigned Oli to fix Twitter bugs</li> <li>3. Assigned Joanna to test piano</li> <li>4. Assigned Nicola to allow for chosen Twitter handles</li> <li>5. Ask around for user testers</li> <li>6. All to complete report</li> </ol> <p><b>Problems encountered during Sprint</b></p> <ul style="list-style-type: none"> <li>• Return value of spam filter</li> </ul> <p><b>End</b></p>



## Appendix C: User Interface Questionnaire

### **Group B1 Micro:Bit tasks**

Please complete the instructions below in order. You may look through the instructions page on the website at any time. If either the tasks or the instructions are unclear at any point please ask for clarification.

- 1) Register an account on the website.
- 2) Log into the website with the registered account.
- 3) From the website, choose a city to show weather from.
- 4) From the website, set a simple lock screen pattern to log into the micro:bit.
- 5) Log into the micro:bit. First enter your username, then the pattern that was set.
- 6) Navigate to weather on the main menu.
- 7) Select weather.
- 8) Check the latest tweet.
- 9) Check if any messages have been received.
- 10) Send a message to a micro:bit with ID 2.
- 11) Choose a new notification sound on the piano.

Fig. 1. Tasks for user study.

Please rate your level of experience with the micro:bit on the scale below: \*

- ☐ No experience
- ☐ Some experience
- ☐ Reasonable experience
- ☐ Development experience

Please rate your agreement with the following questions on the scale below: \*

	Strongly disagree	Slightly disagree	Slightly agree	Strongly agree
I quickly learned how to use the micro:bit controls.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the micro:bit controls intuitive to use.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I found the micro:bit interface easy to understand.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
I was able to easily access the features I wanted to use on the micro:bit.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
The features on the micro:bit provided the functionality I expected.	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Fig. 2. micro:bit questionnaire for user study, part 1.

I found the website  
clear and easy to  
navigate.

☐☐☐☐

I was able to easily  
access the  
information I  
wanted on the  
website.

☐☐☐☐

Were there any micro:bit features which you liked or found helpful?

Your answer

Did you encounter any problems with the micro:bit while using it? If so, how seriously did they impact your experience?

Your answer

Is there anything which you think should be added to the micro:bit? If so, how would this improve the user experience?

Your answer

SUBMIT

Fig. 3. micro:bit questionnaire for user study, part 2.

Please rate your level of experience with the micro:bit on the scale below:

6 responses

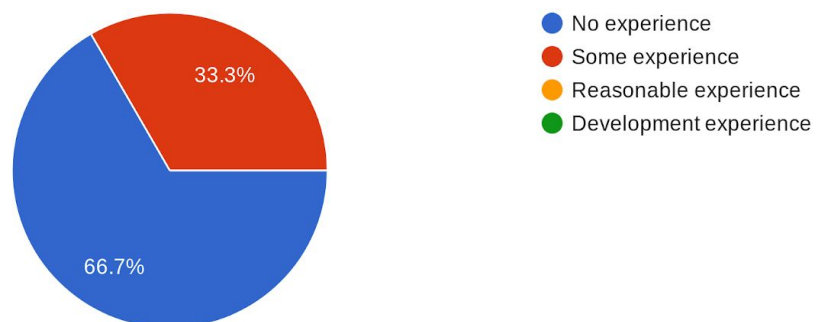


Fig. 4. Pie chart of responses to micro:bit experience

Fig. 5. Bar chart of responses to multiple choice questions

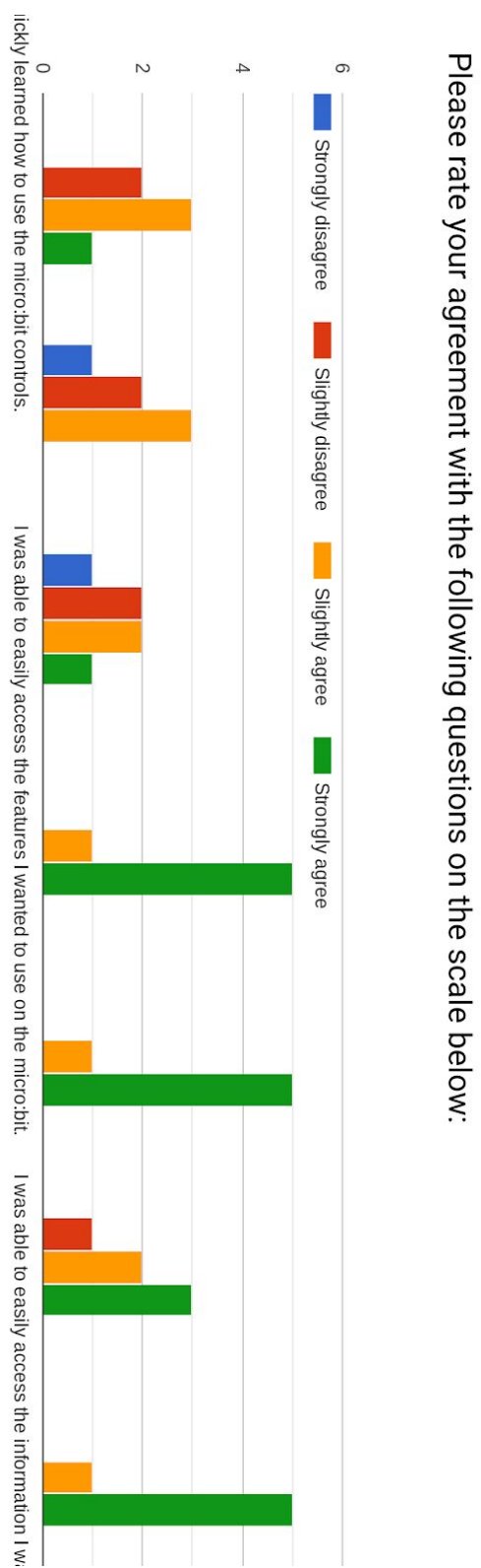


Fig. 6. Responses to first written question

### Were there any micro:bit features which you liked or found helpful?

6 responses

Using the buttons to navigate the menu was easier than tilting up/down to select things.

I liked the ability to send short range messages (and also the fun notification sounds).

I liked the unlock once it was explained how it worked.

messenger and weather were nice

no annoying piano sounds in the lab

I liked the tilting aspect. I found it fun.

Fig. 7. Responses to second written question

### Did you encounter any problems with the micro:bit while using it? If so, how seriously did they impact your experience?

6 responses

It was difficult to enter the login pattern because the grid was hard to visualise. I was confused by the use of arrows for sending/receiving tweets/messages because I had to tilt the opposite way to the arrow to select the function.

At first, took a while to work out how to handle the lock screen pattern, but once this was dealt with it did not impact my experience. Also, the piano keys were not always very receptive which, to some extent, made setting the notification sounds difficult

The unlock was initially unclear

piano is unresponsive, unlocking with longer lock patterns is hard

the hardware is not very responsive

Unlocking, but that's probably because the 3x3 grid isn't as easily translated to the micro:bit, so it's more of a hardware issue.

Fig. 8. Responses to third written question

Is there anything which you think should be added to the micro:bit? If so, how would this improve the user experience?

5 responses

More step-by-step instructions or a diagram of the graph for the lock pattern could be added to the website to make it easier to understand the microbit functions.

Make the unlock clearer.

news maybe, staying up to date on important topics is good

maybe a simplified user input

A tutorial for the 3x3 grid, but that's it :)

## Appendix D: Code Building and Running the Server

Assuming yotta is already installed:

Install python and node dependencies by running “make install-packages” from the project root.

There is a server accessible via <https://oas.host.cs.st-andrews.ac.uk/JH-project/>, but if you wish to run the server yourself, first open the file ‘serial\_comms/serial\_computer.py’, comment out line 433 and uncomment line 432.

Navigate to the ‘website’ directory and run ‘node index.js 8080 UNSECURE\_COOKIE’. The UNSECURE\_COOKIE argument allows cookies to be sent over HTTP, which is unsafe. This option is only present so that the server can be run on localhost, since traffic is usually proxied over NGINX which upgrades traffic to HTTPS.

This should allow the website to be accessible at <http://localhost:8080/>.

Subsequent steps will work with either <https://oas.host.cs.st-andrews.ac.uk/JH-project/> or <http://localhost:8080/> depending on the settings.

The python serial program should be run using ‘make py’ from the project root.

The microbit code can be built using the main Makefile, which uses yotta. Plug in one micro:bit, then run ‘make build-flash1’ to build the binary for the microbit with ID 1, then flash it to the microbit. Unplug this micro:bit, plug in another one, then run ‘make build-flash2’ for ID 2, and similar for ID 3.

The binary for the mock micro:bit can be built by navigating to ‘radio\_simulator/mock\_microbit’ and running ‘make mock\_mesh’. There is a copy of this run by the server and another copy used by the simulator GUI.