Lightning ‘Bugs’

A micro:bit makecode project (mentor guide)

In the real world, no code is perfect - especially when it’s first written. Maintaining most software involves an ‘endless cycle’ of discovering and fixing bugs (artistic license for a pun goes here).

This workshop project is designed to follow on from the Firefly coding workshop, where hopefully several micro:bits were programmed to create an awesome light display for a local public event.

Young coders are told that 2 problems have been found with the code - one that would allow ‘Mischievous Malory’ to take control of the artistic light display and another that makes the device unresponsive when Alice keeps pressing button A.

Perhaps as competing teams they are challenged to complete each of the following stages. How they achieve this is up to them.

Perhaps in ‘competitive’ teams, their **4 challenges** are:

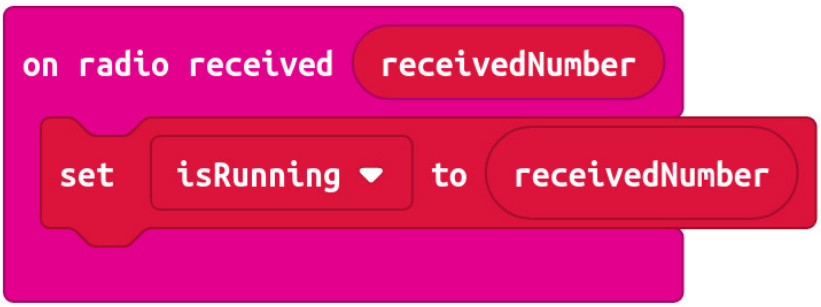
1. Use a Firefly coded micro:bit to demonstrate the Alice problem (including verbally).

2. Program a micro:bit with Malory’s code, then demonstrate control of the Firefly display.

3. Rewrite the Firefly code to fix the Alice problem, then demonstrate that it has been fixed.

4. Rewrite the Firefly code so that Malory’ problem code no longer affects it.

Explaining the Alice problem:



Looking at the Firefly code, the radio is listening for a number. It will accept ANY number it receives. When a number is received, the program immediately sets the *receivedNumber* variable to the value of the number and then sets our *isRunning* variable to the value of *receivedNumber*.

But what happens when several more numbers are then received by the radio before the micro:bit has even done what that program asks it to do with the first one?
Apparently, the micro:bit handles this situation in much the same way we might handle being continuously given sheets of paper with instructions to read, then carry out:

So, you're in the middle of reading an instructions sheet, when someone hands you another one, then another, then another...

You're not allowed to turn them away (just yet), so you store them on your desk in a sort of queue (buffer), to be read and acted upon in the order received (first in first out). When you've finished with the current instruction page, you can then deal with the next one and so on.

Also, you have a limited amount of space on your desk for this queue of piece of paper, so whenever the queue gets to 10 pieces of paper, you're allowed to refuse any more (you can say "go away!"). But as soon as the length of the queue goes back down below 10, you must resume accepting those instruction sheet onto your desk.

At some point, someone comes in with an urgent instruction (the button B press), that is expected to be acted upon the moment it's been received and read. Unfortunately, as there's no way to mark our sheets of paper a urgent, it's just accepted onto the end of the queue as normal.

The instruction that *the sender* is expecting to happen immediately won't get read and acted upon until all the others ahead of it in the queue have been dealt with first.

Resolving the Alice problem:

So, to change this behaviour into something more like what we expect, we could do any of a number of different things like:

- Give incoming messages a ‘priority label’, so urgent ones go straight to the front of the queue.
- Get an assistant to check incoming messages and decide whether to either throw them straight into the bin (if that instruction is already being carried out) or add them to the queue (if the instruction changes something).

Explaining the Malory problem:

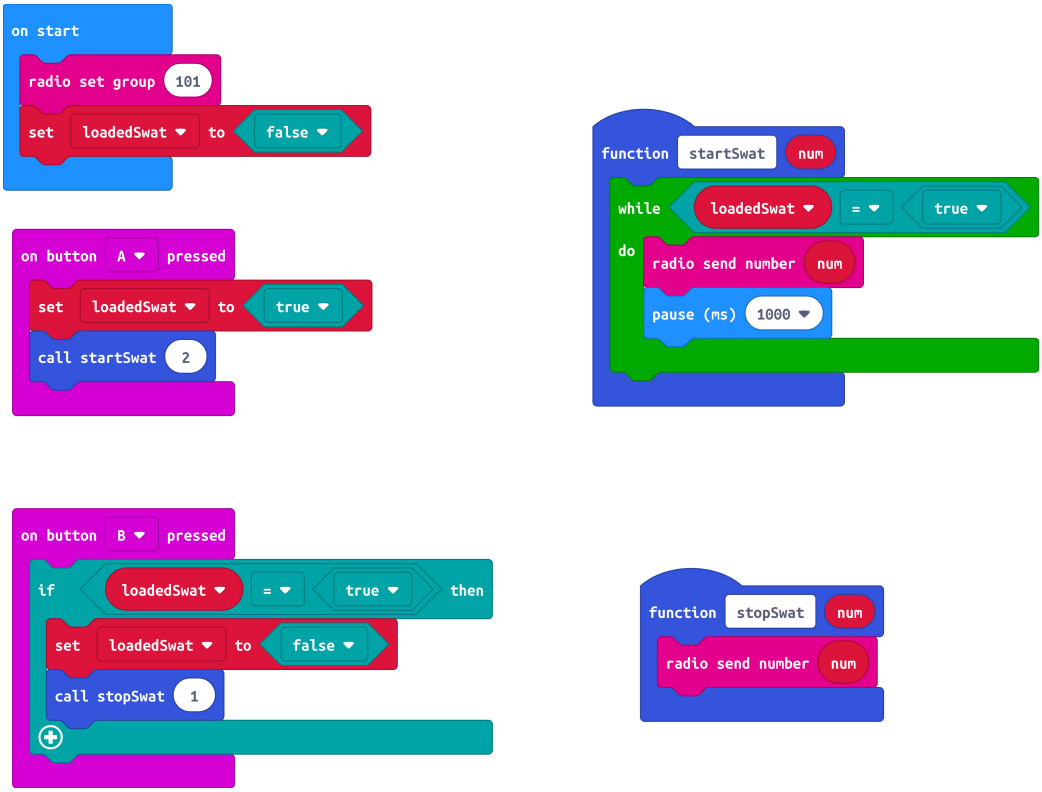
As mentioned in the Alice problem explanation, the Firefly code tells the micro:bit radio to accept ANY number and immediately start doing stuff with it, without even checking that it's one of the only two numbers it knows what to do with.

So, when Malory writes her program to send a number other than zero or one, it's like she's hired someone to go into Halton library to return a 'bag of fish and chips'! During the time it's being politely explained that the library could not have issued this bag of fish and chips, nothing else can happen.

Malory smiles as she then thinks - "what if I hire everyone in Leeds to buy bags of chips, any try to return them to the Halton library desk?" (All politely queued of course). "I reckon Halton library won't be able to do anything until I issue the instruction for them to stop".

Malory's program:

Malory designed her program to use button A to start a Denial of Service (DoS) attack on the Firefly display, and button B to stop the attack (whenever she feels like it).



Resolving the Malory (and Alice) problem:

Similar to the Alice problem, the program needs to check what's coming in by radio and immediately throw away any number that we're not expecting.

Librarians could recruit a volunteer to stand at the library entrance and check what the person at the front of the queue is trying to return. If it's anything other than the expected items, that person is ejected from the queue, so cannot enter the library to tie up valuable resources.
(Note: As these queued people have agreed to being controlled by Malory, it's technically anti-social behaviour!)