micro:bit Workshop Notes

Overview

This workshop aims to introduce Key Stage 2 pupils to coding and some of its applications (specifically those related to particle physics) using a micro:bit.

Learning Outcomes

1. Be able to design and write programs and detect and correct any errors present
2. Be able to work with different forms of inputs and outputs
3. Understanding of the scientific method
4. Understand where coding and programming is used and what it can do

The Micro:bit

This is a small pocket-sized computer which can be easily programmed through a USB connection to a computer. It has many features including buttons, an LED display and accelerometer. For more information on the micro:bit visit: https://microbit.org.

The Code

In this workshop we will be using Scratch, an introductory coding language which uses colourful blocks that you can drag-and-drop to build a program (more info here https://scratch.mit.edu). This is freely available online and can be used to program a micro:bit. All example code given in this workshop has been put together using Microsoft's MakeCode platform (https://makecode.microbit.org/#), this also includes a micro:bit simulator so that if a micro:bit is unavailable pupils can still program.

The micro:bit website can create a classroom (https://classroom.microbit.org) where pupils can join a session by going to "microbit.org/join" on a web browser. In a classroom session the leader can send example code to the pupils and review their work live during the session. This is a helpful tool which can be used to run the workshop.

Workshop Activities

Name	*Required*	*Details*
Intro	Micro:bit and USB	Show the pupil's name and/or age after pressing a button on the micro:bit.
React!	Micro:bit and USB	Program a game to test reaction time. Application: Particle detectors data collection
Speed!	Micro:bit, USB, metal foil, 4 crocodile clip leads, cardboard, (optional: blocks for a ramp), sticky tape, toy car, ruler	Speed gates connected to a micro:bit record the time for a toy car to travel between them, then calculate the speed of the car. Application: Measuring the speed of particles round a particle accelerator.
Tracker!	6+ Micro:bits for each pupil, USB	A hidden message is sent to a micro:bit which then gets scrambled and sent out to some surrounding micro:bits, the group can then unscramble the message. Application: Many detectors pick up fragments of a signal which need decoding

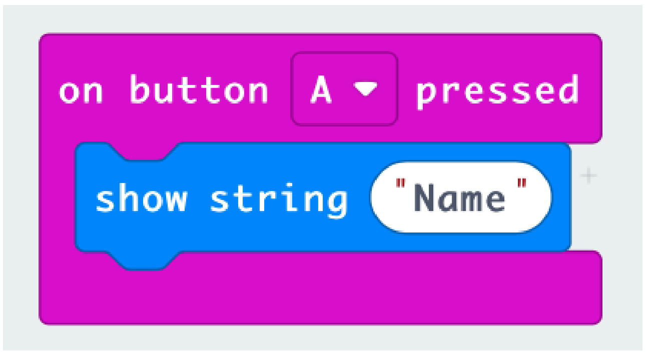
Activities in-depth

For these exercises the model code is made using Microsoft's MakeCode platform using the Scratch coding language (https://makecode.microbit.org).

Introduction to code

***Required**: micro:bit and USB cable*

The aim of this exercise is to ease pupils into using the micro:bit, the Scratch language, and some programming concepts. Only two blocks are needed, and these introduce the idea of input and output. The model code is shown below in Figure 1.



```
on button A ▾ pressed
    show string "Name"
```

Figure 1 Model code for the introduction exercise. When programmed this will show the pupil's name when they press the A button.

The pink block here tells the program when the code contained inside it should be run, it's sometimes known as a hat block. This is often how we decide the input, in this example we've chosen pressing the A button. There are several other types of input which can be used such as clapping nearby or shaking/tilting the micro:bit.

The blue block is a stack block, these are the main commands and it's often how we produce an output on the micro:bit. In this example we want to show a string – this is just what we call some text in code. The pupil can find this in the 'Basic' tab and drag this block into the pink block (like below) and type their name into the white box.

The final step is to transfer this to the micro:bit, if this is unavailable the website has a simulator available. Click the **Download** button and the code is downloaded as a `.hex` file. Connect the micro:bit to the computer with a USB cable and move the `.hex` file into the **MICROBIT** drive.

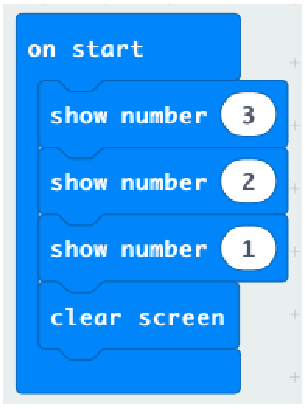
Exercise 1: React!

***Required**: micro:bit and USB cable*

In this section pupils can program a game that tests their reaction speeds. Here we introduce several new ideas: mathematical operators, loops, variables and logic statements. These can all be added from the tabs on the left with the corresponding name.

To get to the model code seen below, we can break it down as follows:

1. The Countdown: this is not entirely necessary, but it improves usability, and it can be a good way to show what a loop is and discuss variables. It is possible to create a simple countdown without a "for" loop, as shown in the image to the right. If you would like to spend time introducing the "for" loop you can do so here following the instructions below, otherwise just substitute the 3 "show number" blocks for the "for" loop in the model code below.



```
on start
    show number 3
    show number 2
    show number 1
    clear screen
```

   - The countdown is the first thing we want to happen when we start the code, so we put this in the "on start" block.
   - First add a "for" block from the Loops tab and drag it into the "on start" block (found in the Basic tab, and also generates automatically at the start of a project). The first line of the "for" block declares a variable ('index') which will take values from 0 to a number that we can type in.
     - This "for" block will run the blocks we drag inside it for the number of times we put in, after each run the variable ('index') will increase by 1, until it reaches the maximum value – then the loop has ended.
     - The "show number" block can be found in the Basic tab and the mathematical operator can be found in the purple Math tab and dragged into the "show number" block. The 'index' variable block is in the Variables tab and should be dragged into the second box in the purple Math block.
   - The model code below will show a '3, 2, 1' countdown with 0.5 seconds pause (N.B. time is entered here in milliseconds), the loop will run as follows
     - Run 1: index = 0, so the micro:bit shows (3 – 0 =) 3
     - Run 2: index = 1, so the micro:bit shows (3 – 1 =) 2
     - Run 3: index = 2, so the micro:bit shows (3 – 2 =) 1
       - The index has now reached the maximum value we specified so the loop ends
   - Before we move onto the next section, we need to get rid of the 1 that will be on the screen, so we add a "clear screen" block from the Basic tab after the "for" block.
2. The Signal: In the next 4 blocks in the "on start" code we will program the micro:bit when to show a signal and what to do when that happens.
   - We want this to happen as soon as the countdown finishes which is why we put this in the "on start" block
   - The first thing to do is declare a variable which we'll call 'delay', this will be the time from the countdown finishing to the signal showing. From the Variables

tab we need to click on the ‘Make a Variable…’ box and type in ‘delay’. We now can drag the “set (variable) to” block into the “on start” block making sure that the ‘delay’ variable is selected in the drop-down menu

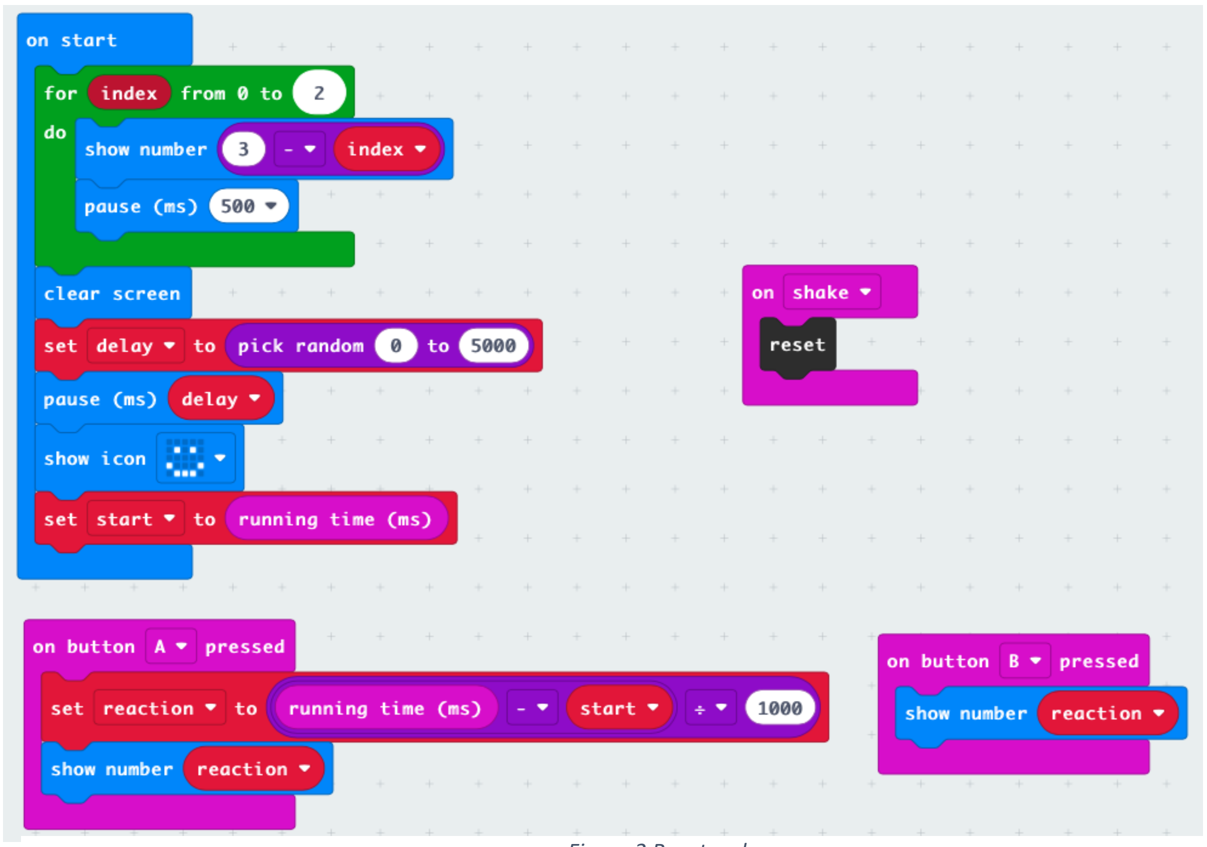
    - We want this to be randomised so that the player doesn’t know when the signal will appear. To do this we use the “pick random” block from the Math tab and set it from 0 to 5000. This will set the delay variable equal to a random number between 0 and 5000.

  - In the next line we use the “pause” block and put the ‘delay’ variable in there. This means the code will wait for a random amount of time between 0 and 5 seconds before it continues (time is input in milliseconds, 5000ms = 5s)
  - As soon as the delay is over we want to show a signal, in the model code we have a smiley face icon appear. Pupils can choose one of these from the Basic tab or fill in their own LED pattern using the “show leds” block.
  - In the final line we set another variable called “start” which will be useful in a later calculation. It will record the time that the signal is shown by using the “running time” block – this can be found in the more section of the Input tab. It gives the time elapsed (in milliseconds) at that point since the micro:bit turned on.

3. The Input: We can handle the inputs with these extra pink blocks which will run the code inside them when a button is pressed. In the model code we’ve chosen button A to be the required reaction.
    - Pressing the A button should make the micro:bit calculate a reaction time and then display it. We create a new variable called ‘reaction’ and set it as follows in the model code below. Note, it is divided by 1000 to convert from milliseconds to seconds. To do this we put one mathematical operator inside another, effectively using brackets and putting the subtraction first according to BIDMAS.
    - The B button and shake can also be added as inputs. These can also both be found on the Input tab. The B button will simply show the reaction time again, in case you missed it; shaking the micro:bit will reset the game – the “reset” block in the Control tab which is in the Advanced section of tabs (there is actually a reset button on the back of the micro:bit which does this already so it’s not necessary but makes it a bit more fun)

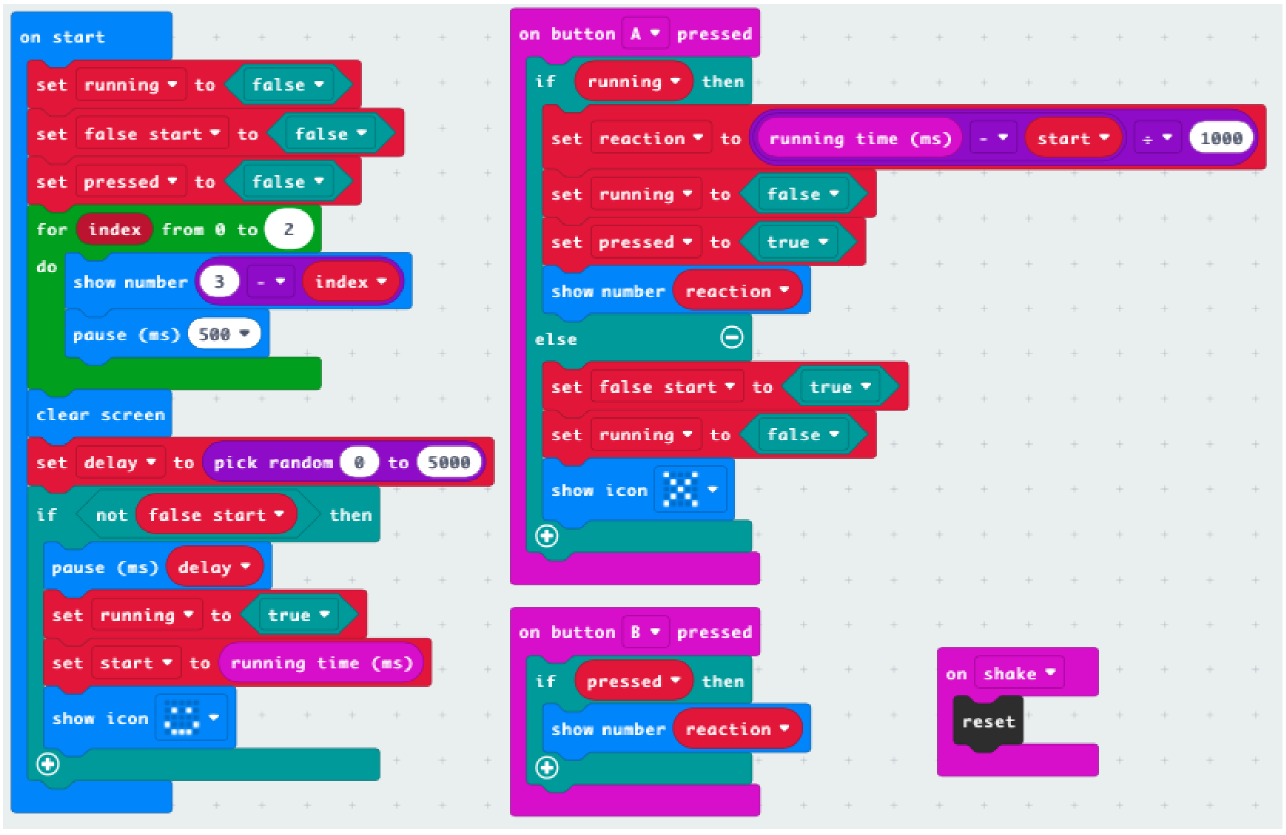
React! Extension:

Some pupils may notice that the above code has an issue – if you press the buttons before the signal appears, then the game won’t work. This isn’t a big problem as we can just reset it with a shake and try again the next round, but we can add some extra code to deal with this problem. On the right there’s an example of code that does this (there are many different approaches that could also work). In this example we introduce some extra variables and some “if” blocks which will only run if the statement in its top line is true.
So, the result of this in the example above is that if the A button is pressed before the signal shows we get a ‘X’ and the game is effectively over, as the signal will only show if there is *not* a false start



```
on start
    for index from 0 to 2
    do
        show number 3 - index
        pause (ms) 500
    clear screen
    set delay to pick random 0 to 5000
    pause (ms) delay
    show icon
    set start to running time (ms)

on shake
    reset

on button A pressed
    set reaction to running time (ms) - start ÷ 1000
    show number reaction

on button B pressed
    show number reaction
```

Figure 2 React code



```
on start
    set running to false
    set false start to false
    set pressed to false
    for index from 0 to 2
    do
        show number 3 - index
        pause (ms) 500
    clear screen
    set delay to pick random 0 to 5000
    if not false start then
        pause (ms) delay
        set running to true
        set start to running time (ms)
        show icon

on button A pressed
    if running then
        set reaction to running time (ms) - start ÷ 1000
        set running to false
        set pressed to true
        show number reaction
    else
        set false start to true
        set running to false
        show icon

on button B pressed
    if pressed then
        show number reaction

on shake
    reset
```

Figure 3 React Extension

Exercise 2: Speed!

***Required**: micro:bit, metal foil, 4 crocodile clips, cardboard (the 'track'), sticky tape, toy car, and ruler*

In this section the micro:bit can be used to record the speed of an object between two points where we set up timing gates.

Making the timing gates:

The micro:bit can receive signals from the pins along the bottom row, if we create a circuit from one of them and the GND pin the micro:bit will receive a signal which we can use in the code. The timing gate is just two strips of foil on the track with a small gap between them, and connected to one of the micro:bit pins and the GND pin with crocodile clips. Then when the toy car (with some foil attached beneath it) passes over it, the circuit is completed and we get a signal. An example of this can be seen in the picture below.

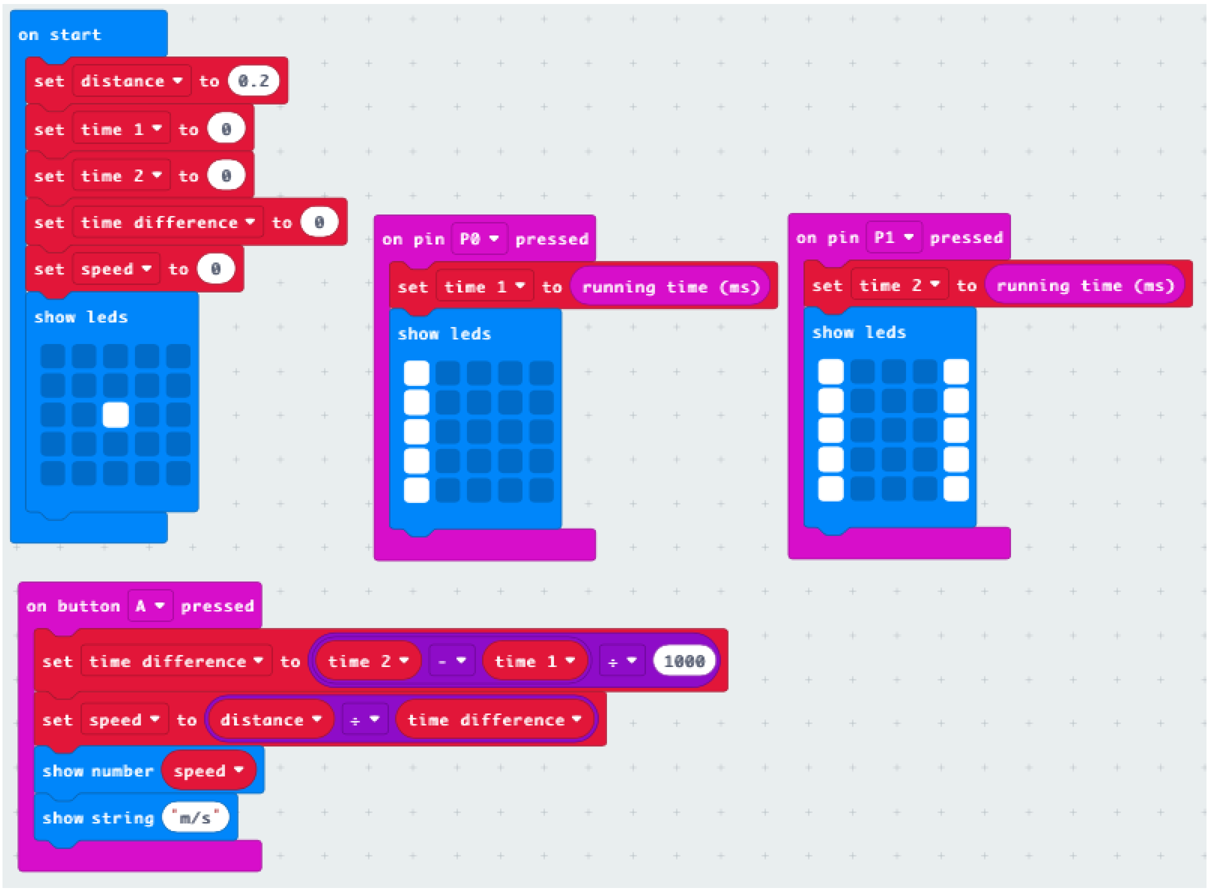
The toy car can be made to move over the gates manually, or a ramp can also be added (as seen below).

The code:

This program needs to handle the input from the pins on the micro:bit by recording the time difference between when they receive a signal, and then work out the speed of the object that caused that.

1. Start: The "on start" block will be used here to set up all the variables we'll need to use. These are: distance, time 1, time 2, time difference, and speed. These are all made in the Variables tab, and we set them all to 0 to begin with except for distance, the distance is found by measuring the distance between the two timing gates (measuring from the gap in the middle of the two foil strips) and you can type the number straight into the variable box. Make sure to remember the units you've used – in the model code we've used metres so the final speed will be metres per second, but if you'd rather use centimetres or millimetres you can, just remember the units of speed need to match. It can also be helpful to have a signal showing the micro:bit is in this state, so in the model code there is a block that makes a single LED in the centre light up.
2. The Pins: When the micro:bit receives a signal from the pins we've selected we need to record the time that this happens so that it can work out the time difference. To do this grab two of the pink 'on pin P0 pressed' block from the Input tab. Choose the pins that are connected to the timing gates from the drop-down menu on the Input block – in the example we have used P0 and P1. For the pin that the car will go over first we need to set the value of the time 1 variable. This can be done using a red "set variable" block from the Variables tab and choosing time 1 from the drop-down menu. Then, find the "running time (ms)" block from the more section of the Input tab and set the variable to that. It is also useful to have something on the micro:bit show that it's registered a signal so we've added a "show leds" block to let us know the micro:bit has recorded the time the toy car passed through the gate. Repeat this for the second pin but using the time 2 variable instead of time 1.
3. The Calculation: In the model code we've chosen pressing button A to get the micro:bit to calculate the speed. To calculate speed we need to divide the distance by the time taken to cover that distance. The micro:bit already has the distance from when we

measured it and put it in the “on start” block. We need to calculate the time difference by subtracting the time 1 variable from time 2. The red “set variable” block is used to set the time difference variable with some mathematical operators as seen in the model code below. Note: it’s divided by 1000 to get the speed in metres per second, as the micro:bit measures time in milliseconds. The next calculation sets the speed variable to ‘distance’ divided by ‘time difference’. Finally, to get the micro:bit to show the speed we use a “show number” block from the Basic tab and put in the speed variable, we can also use “show string” to add the units here too. In the model code we’ve used metres per second (m/s), but yours may be different so make sure it matches the units for distance you put in at the beginning!



```
on start
    set distance to 0.2
    set time 1 to 0
    set time 2 to 0
    set time difference to 0
    set speed to 0
    show leds

on pin P0 pressed
    set time 1 to running time (ms)
    show leds

on pin P1 pressed
    set time 2 to running time (ms)
    show leds

on button A pressed
    set time difference to time 2 - time 1 ÷ 1000
    set speed to distance ÷ time difference
    show number speed
    show string "m/s"
```

Figure 4 Model code for the Speed workshop

Speed! Extension:

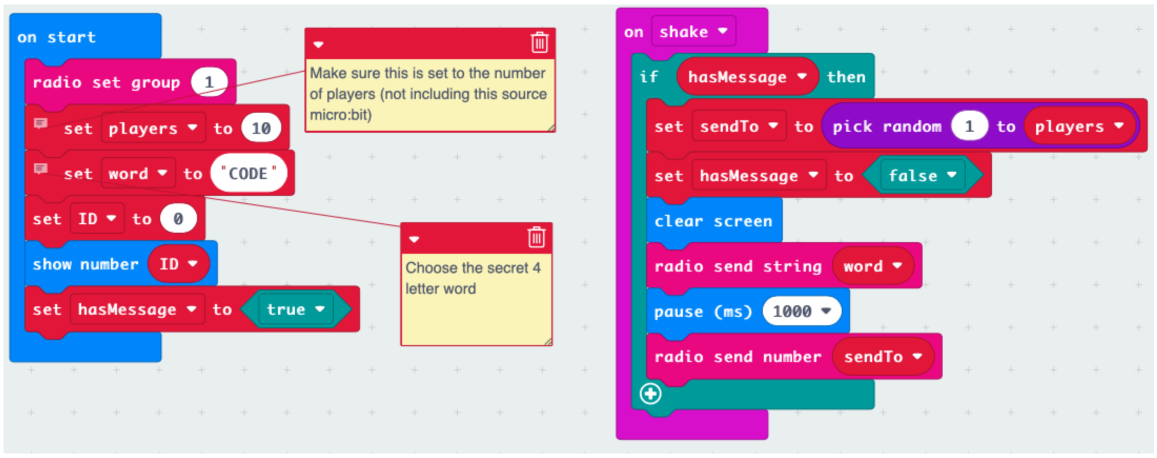
Different surfaces for the ramp, or different cars could be used. The speed can be recorded and plotted on a graph to discuss the effect of friction or car mass on an object’s speed, and also help in understanding a fair test.

Exercise 3: Tracker!

***Required**: 6+ micro:bits and USB cable*

This final part involves a hidden message being sent from one micro:bit using the radio feature to all the others, the message is scattered and the pupils can then unscramble the signal received to work out the original message.

The code for this is a bit more involved so it may not be worth making the pupils create the code from the start, but it may be worth talking through how it works. The code is shown below.



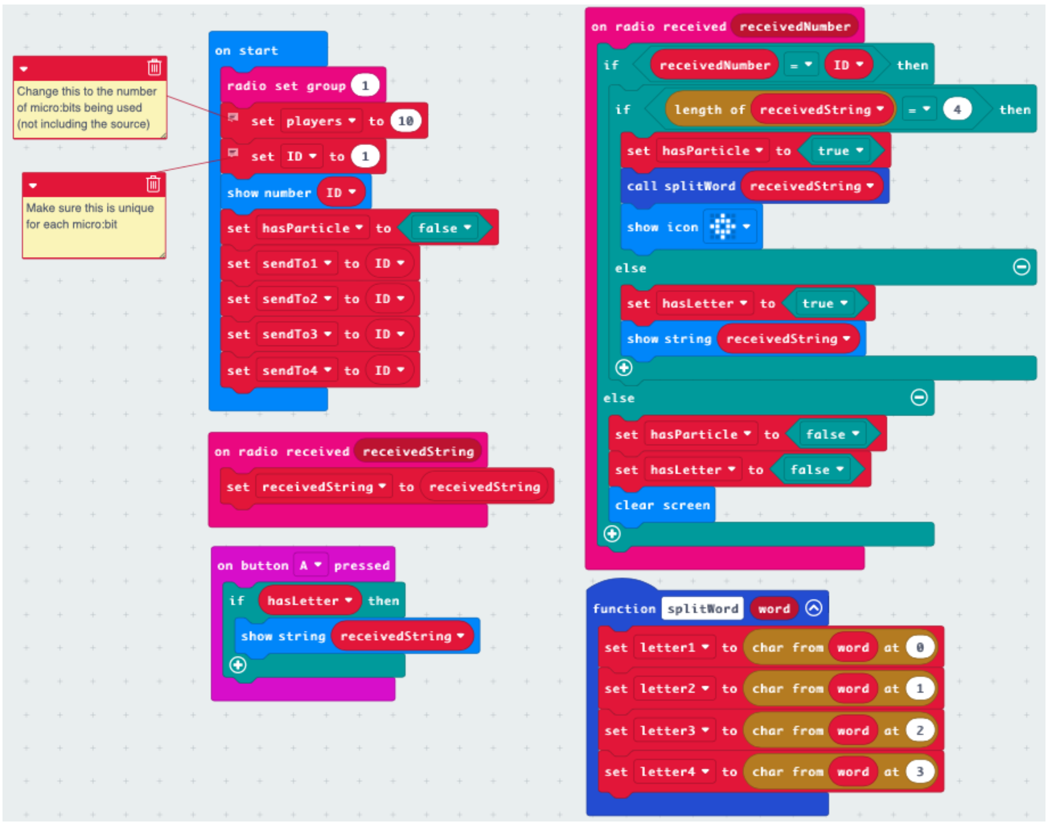
```
on start
    radio set group 1
    set players to 10
    set word to "CODE"
    set ID to 0
    show number ID
    set hasMessage to true
```

Make sure this is set to the number of players (not including this source micro:bit)

Choose the secret 4 letter word

```
on shake
    if hasMessage then
        set sendTo to pick random 1 to players
        set hasMessage to false
        clear screen
        radio send string word
        pause (ms) 1000
        radio send number sendTo
```

Figure 5 Code for the source micro:bit in the Tracker workshop

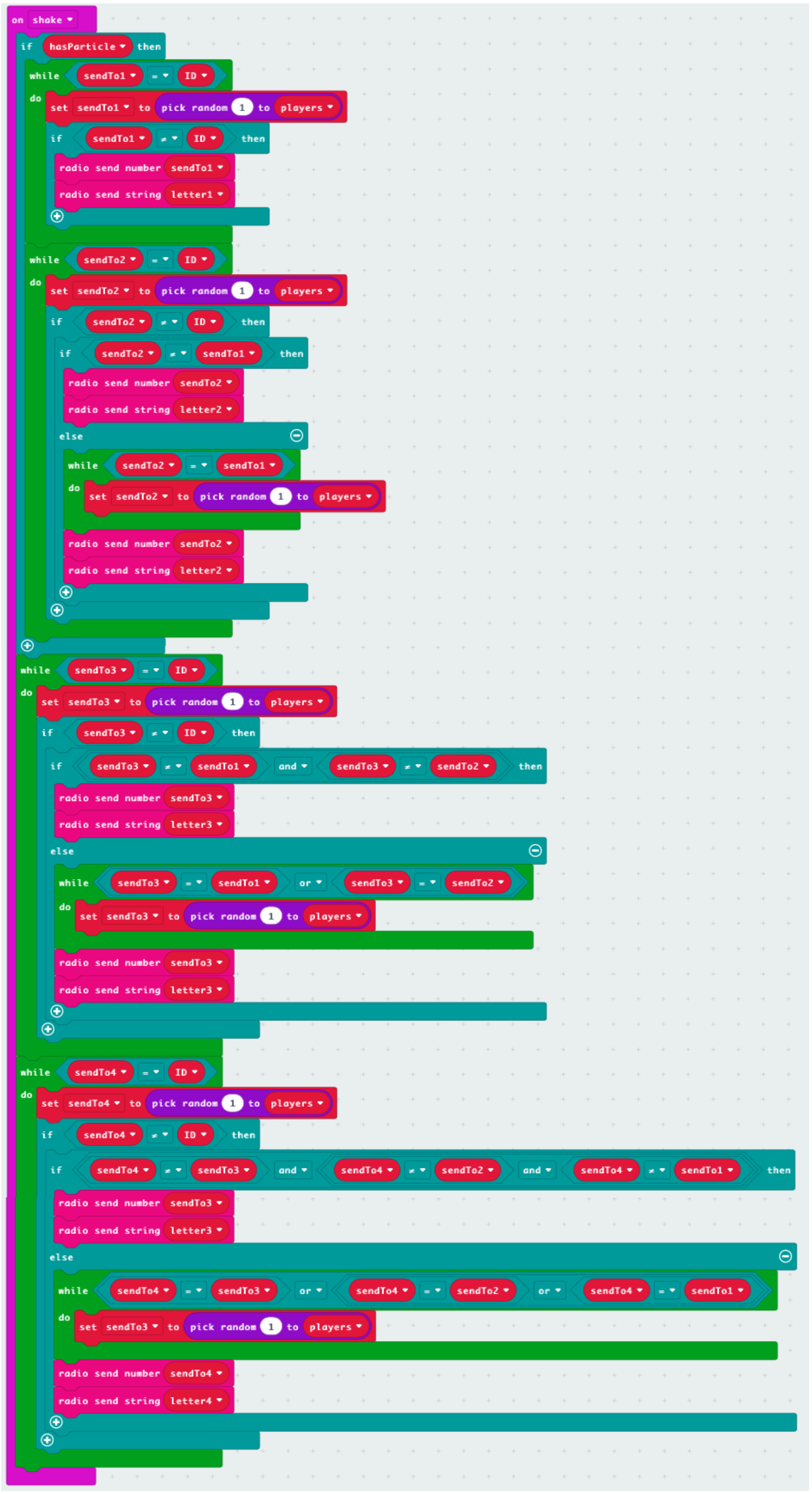


Change this to the number of micro:bits being used (not including the source)

Make sure this is unique for each micro:bit

```
on start
    radio set group 1
    set players to 10
    set ID to 1
    show number ID
    set hasParticle to false
    set sendTo1 to ID
    set sendTo2 to ID
    set sendTo3 to ID
    set sendTo4 to ID

on radio received receivedString
    set receivedString to receivedString

on button A pressed
    if hasLetter then
        show string receivedString

on radio received receivedNumber
    if receivedNumber = ID then
        if length of receivedString = 4 then
            set hasParticle to true
            call splitWord receivedString
            show icon
        else
            set hasLetter to true
            show string receivedString
    else
        set hasParticle to false
        set hasLetter to false
        clear screen

function splitWord word
    set letter1 to char from word at 0
    set letter2 to char from word at 1
    set letter3 to char from word at 2
    set letter4 to char from word at 3
```

Figure 5 1. The code for the 'source' micro:bit



```
on shake
  if hasParticle then
    while sendTo1 = ID
    do  set sendTo1 to pick random 1 to players
        if sendTo1 ≠ ID then
          radio send number sendTo1
          radio send string letter1
    while sendTo2 = ID
    do  set sendTo2 to pick random 1 to players
        if sendTo2 ≠ ID then
          if sendTo2 ≠ sendTo1 then
            radio send number sendTo2
            radio send string letter2
          else
            while sendTo2 = sendTo1
            do  set sendTo2 to pick random 1 to players
            radio send number sendTo2
            radio send string letter2
  while sendTo3 = ID
  do  set sendTo3 to pick random 1 to players
      if sendTo3 ≠ ID then
        if sendTo3 ≠ sendTo1 and sendTo3 ≠ sendTo2 then
          radio send number sendTo3
          radio send string letter3
        else
          while sendTo3 = sendTo1 or sendTo3 = sendTo2
          do  set sendTo3 to pick random 1 to players
          radio send number sendTo3
          radio send string letter3
  while sendTo4 = ID
  do  set sendTo4 to pick random 1 to players
      if sendTo4 ≠ ID then
        if sendTo4 ≠ sendTo3 and sendTo4 ≠ sendTo2 and sendTo4 ≠ sendTo1 then
          radio send number sendTo3
          radio send string letter3
        else
          while sendTo4 = sendTo3 or sendTo4 = sendTo2 or sendTo4 = sendTo1
          do  set sendTo3 to pick random 1 to players
          radio send number sendTo4
          radio send string letter4
```

Figure 5 2 The code for the remaining 'receiver' micro:bits

Game setup:

One micro:bit should be chosen as the 'source' and programmed with the code seen in Figure 4, this is where the secret message will be typed in, it will need to be reprogrammed after each round with the next message. Make sure the players variable is correct – this is the number of micro:bits being used (not including this 'source' micro:bit).
The remaining micro:bits will be the 'receivers' and programmed with the code in Figure 5. These should all have a unique ID set as 1, 2, 3 … and so on. Once programmed these should show the ID number straight away, it is worth making sure everyone's ID number is different and there are no missing numbers otherwise the game may not work. The 'receivers' should also set the players variable correctly – this is the number of 'receivers'.

Important: All micro:bits need to be in the same radio group to be able to communicate with each other, this is set in the pink "radio set group" block at the beginning.

How to play:

1. The person with the 'source' micro:bit will choose a four letter word and type this into the 'word' variable as seen in Figure 4. Once programmed the signal can be sent by shaking the micro:bit.
2. One of the 'receivers' will pick up this signal and see their micro:bit light up – but they won't be able to see the message.
3. They can now shake their micro:bit to send the message to other 'receivers'.
4. Some 'receivers' will now see a letter appear on their micro:bit. They now have to work together to try and unscramble the letters to work out the original message.

The game can then be repeated with a new word, the person controlling the 'source' can be swapped around, this could be given to whoever works out the hidden message or to whoever gets the first signal (this is randomly chosen).
When programmed as above the micro:bits will only be able to handle four letter words. If you want to try for longer or even multiple words you can run the game for multiple rounds inputting just 4 characters of the word at a time. With the use of a whiteboard you could run the game similar to the Wheel of Fortune word puzzle.
Note: If you wish to include the space in a message this will count as a character, when you type it into the 'source' code it can then only take 3 more characters. A space will also be sent out to the 'receivers' but it obviously won't appear on the LEDs, pupils may need help to deduce that if there are only 3 letters and that a space must also have been sent out.

Code explanation:

The micro:bits can send out numbers and strings (words) by radio signals, these get sent out to all micro:bits in the same radio group, so we need IDs to decide which micro:bit will be able to do something when it receives a signal.

Source:

- The 'source' micro:bit is set with an ID of 0, so that it can be easily excluded when choosing where a signal should go, the secret message, number of players and radio group is also declared here.
- We initially set a variable 'hasMessage' to true, which allows the code to run when the micro:bit is shaken.

- When the micro:bit is shaken we want it to randomly choose the ID of a micro:bit that will get the message and send this ID with the message.
    - This is done by creating a variable ‘sendTo’ that is chosen randomly based on the number of players and sending it with the pink “radio send” block
    - The ‘hasMessage’ variable is then set to false so that if the ‘source’ micro:bit is shaken again nothing will happen.

Receivers:

- This code is significantly longer only because it has to deal with a lot more variables due to splitting the message into 4 characters.
- The initial setup is fairly similar but we introduce some more variables, there are 4 sendTo variables as we need 4 different micro:bits to use a signal from the first ‘receiver’.
- The pink “on radio received” block tells the micro:bit what to do when it detects a radio signal
    - When it detects a string (word) we want to set the receivedString to a variable ‘receivedString’ which can then be used in other blocks.
    - When it detects a number this will be the ID that either the ‘source’ or the first ‘receiver’ randomly chose. Only if this number matches the ID of the micro:bit being will any more code be run.
        - The micro:bit now needs to work out if this signal came from the ‘source’ or that first ‘receiver’, we do this based on the length of the receivedString
            - If the receivedString is 4 characters long then this came from the ‘source’ micro:bit, which means this micro:bit is the first ‘receiver’. This means that it has the message, so we set this variable to true – which you’ll later see means this micro:bit can now send a radio signal when shaken.
                - Before the message is sent a function we’ve called ‘splitWord’ is called which breaks the message into 4 new variables, which will be sent later.
            - If the receivedString isn’t 4 characters long we assume it must be 1 character and was sent from the first ‘receiver’. In this case we just need the micro:bit to show the string and then we can start unscrambling the message! (Note: we also have a variable ‘hasLetter’ that gets set to true in the example, this is so pressing the A button can make the letter show again)
        - If the receivedNumber doesn’t match the ID then this micro:bit has neither the message or the letter so we set these variables to false and this micro:bit won’t do anything with the radio signal.
- If a ‘receiver’ has the message it is the first ‘receiver’ and will need to send another signal, this is done “on shake”. This block is very long as it needs to run a lot of checks to make sure that of the four random IDs it needs to choose none of them are the same and it also doesn’t choose itself, otherwise the game won’t work.
    - For each letter a random ID is chosen and if it’s valid the ID and corresponding letter are sent out and the other ‘receivers’ handle it as explained above.