**Introduction to micro:bit Programming**

**Example 1: Name Badge**

You will never need to worry about forgetting to bring your name badge to PDST LCCS CPD events again! (just don’t forget to wear your micro:bit!)

|  |  |  |
| --- | --- | --- |
| *Version 1*  Drag the show string block (Basic) into the program workspace and change the string so that it will display your own name. | | |
|  | | |
| The simulator will scroll your name – once. We want it to scroll your name continuously. | | |
| *Version 2*   * Let’s place a duplicate of the show string block inside the forever container. * To make a duplicate just right-click on the 9-dot menu (shown here) and select Duplicate. * Snap the duplicated block to the forever container | |  |
|  | | |
| Voila! The simulator now scrolls your name continuously.  Let’s evaluate our solution. Can we make our program any better? Is the on start block really necessary? | | |
| *Version 3*   * Discard the on start block by dragging it off the workspace (don’t forget you can select Ctrl-Z to undo). The final solution is shown. |  | |

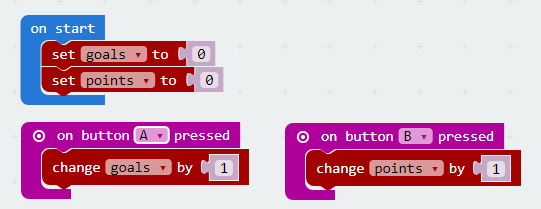
**ADDITIONAL NOTES**

**Example 2: GAA Score Board (counter)**

The program below keeps is used to keep track of the score (goals and points) in a GAA game.

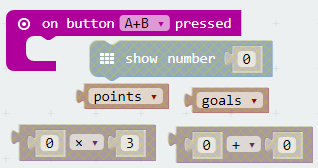
The user interface was designed to work as follows:

* Button A is used to increase the number of goals scored by one. The end-user will press this button whenever the team scores a goal. A goal is worth three points.
* Button B increases the number of points scored by one. The end-user will press this button whenever the team scores a point.



Download the code and run it.

Can you spot the problem? The programmer forgot to complete the code to display the total points scored. What the programmer had in mind was to display the score when the end-user pressed the A+B button. The necessary code bocks were left dangling beneath the on botton A+B pressed container. Can you assemble the hatched out code blocks below to complete the programmer’s task?



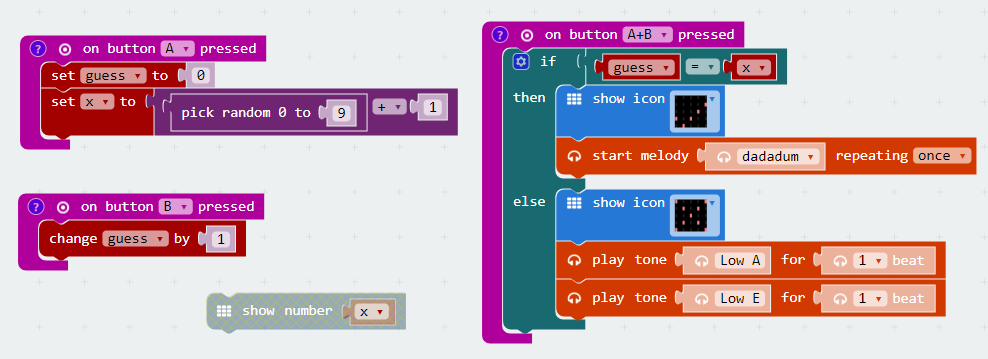
|  |
| --- |
| Can you think of an alternative way to display the total score? If so, how? |

|  |
| --- |
| What is the main problem with this implementation?  Can you suggest a design that would allow the end-user to keep track of the score of both teams? Implement your design. |

|  |
| --- |
| The variables goals and points are examples of *running totals* i.e. every time they are changed the updated value is based on the previous value plus a certain amount (one or three).  Can you think of any other everyday software systems where a running total pattern would be used? Identify as many as you can. |

**Example 3: Guess the Number**

Study the program carefully and answer the questions that follow.



|  |
| --- |
| What do you think would happen when button A is pressed? What about button B? |

|  |
| --- |
| Predict what you think would happen when button A+B is pressed? |

|  |
| --- |
| Download and run the program. Were your predictions correct? Describe what the program was designed to do? |

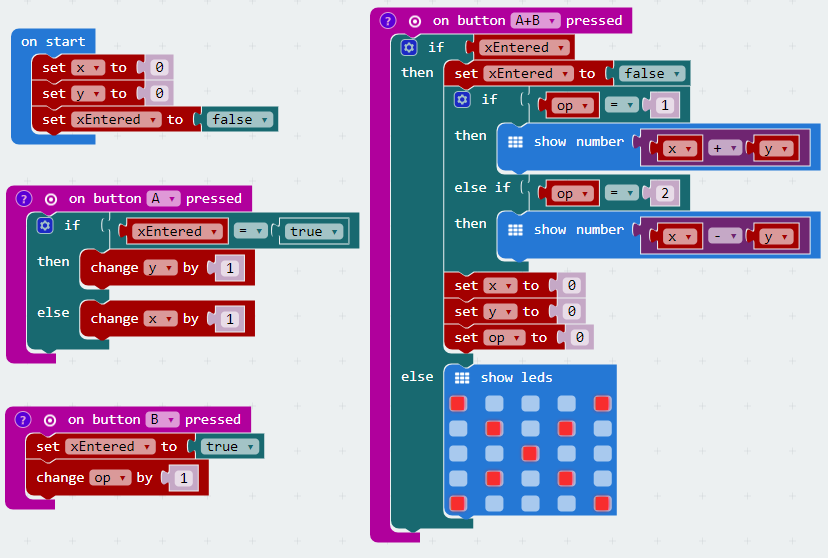
|  |
| --- |
| The code block show number x is *hatched out*. The hatch pattern indicates that the block will not be executed at runtime i.e. it is not part of the *runtime system*.  Snap the hatched block to a place which would make testing this program easier? Explain your choice. |

|  |
| --- |
| Evaluate the program. Suggest (and implement) any changes that might make this program more user-friendly. Explain. |

**Example 4: Simple Calculator**

This example turns your micro:bit into a simple calculator (performs addition and subtraction only).

|  |
| --- |
| Study the code carefully and see if you can explain the design.  In order to do this, it might be useful to describe a number of use cases. A *use case* is a list of precise instructions a user would have to follow in order to achieve a particular goal. In this case the goal would be to perform a specific calculation e.g. or . |



Download and run the code. Test your use cases.

|  |
| --- |
| Devise a general notation that could be used to describe how the system works? |
| The datatype of the variable xEntered is Boolean. This means that it can be used to store one of two values - either true or false. Boolean variables are often referred to as *flags* because they can be used to indicate or flag whether or not a particular runtime event has occurred or not.  Examine the code carefully and explain what event the variable xEntered is used to flag. |
| Compare the logic contained in the two code blocks shown below   |  |  |  | | --- | --- | --- | |  | and |  | |

|  |
| --- |
| Once simple enhancement that could be made to the program is the addition of multiplication and division. See if you can implement these operations without using the Block Category list in the centre of the IDE i.e. restrict yourself to duplicating and changing existing code blocks only.) |
| Evaluate the program by suggesting any improvements that could be made to it. Implement. |
| Suggest other systems that could be developed by adapting patterns used in this implementation. |

**ADDITIONAL NOTES**

**Example 5: Binary Number Converter**

This would be a good example to use with students to follow a binary unplugged activity.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| The purpose of the program is to convert from binary to decimal on the fly.  As each binary digit is entered the decimal equivalent is calculated.  The abstractions are:   * button A represents a 1 and * button B represents a 0   The table shown here on the right illustrates the first ten binary numbers and their decimal equivalents. | |  |  | | --- | --- | | **Binary** | **Decimal** | |  | 0 | |  | 1 | |  | 2 | |  | 3 | |  | 4 | |  | 5 | |  | 6 | |  | 7 | |  | 8 | |  | 9 | |

*Exercise.* Use your knowledge of decimal and binary numbers to fill in the binary values in the table below. The even numbers are on the left and the odd numbers are on the right

|  |  |  |  |
| --- | --- | --- | --- |
| **Binary** | **Decimal** | **Binary** | **Decimal** |
|  | 0 |  | 1 |
|  | 2 |  | 3 |
|  | 4 |  | 5 |
|  | 6 |  | 7 |
|  | 8 |  | 9 |
|  | 10 |  | 11 |
|  | 12 |  | 13 |
|  | 14 |  | 15 |

Do you notice any patterns?

This example demonstrates how recognising patterns can be used to devise solutions.

The implementation is based on a pattern that can be found in the above table. Did you spot the pattern? How does each new binary digit impact on the magnitude of the decimal number? Only two possibilities exist for each digit; 0 or 1. Let’s take a closer look at what happens the decimal number when we add a 0 to the binary number.

e.g. when we append a 0 to 1 the new binary number becomes 10.

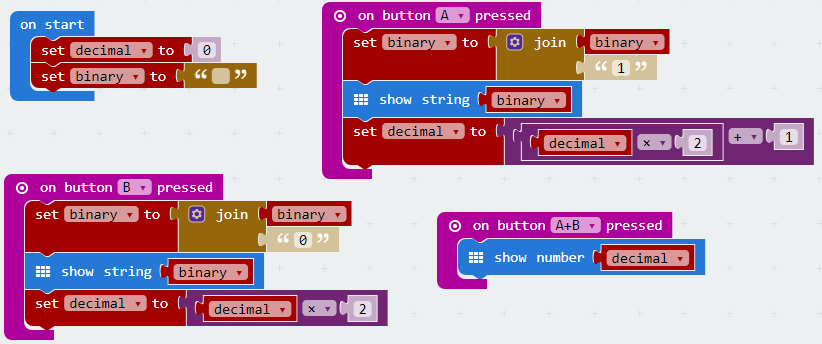
e.g. when we append a 0 to 10 the new binary number becomes 100.

e.g. when we append a 0 to 11 the new binary number becomes 110.

In each case ask what are the old and new decimal values?

It should be evident that each time a 0 is added the decimal value doubles. In a similar way we can discover that the effect of appending the digit 1 is to double the decimal value plus one.

Our solution, shown here, exploits these two patterns:



Try it out yourself!

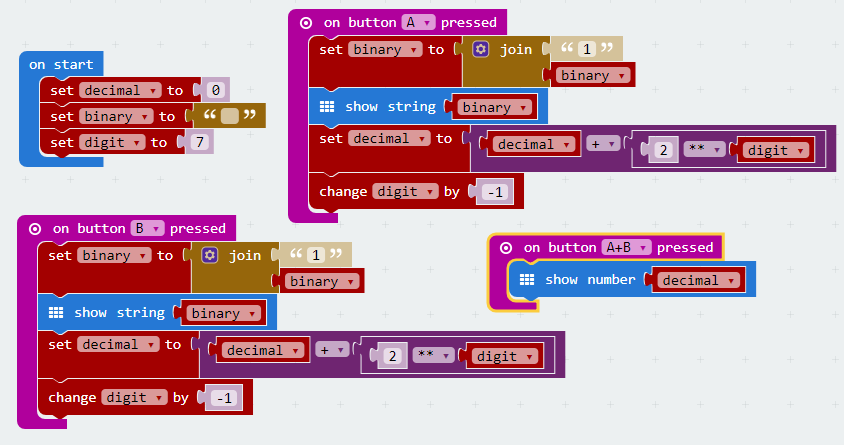
Download the program and run it.

Make some predictions – devise some use cases.

Can you think of any improvements that could be made?

*Version 2*

Now take a look at an alternative implementation of our binary to decimal converter program.



|  |
| --- |
| Study the above code blocks carefully.  Describe how the approach taken here differs from that in the earlier version. Which version do you think is better? Why? |

**Other things to try!**

Here are some additional modifications you might try:

1. Add a way to clear the binary and decimal values so you can start over.
2. Add a way to erase the previous value.
3. Consider how the program could convert the binary number if it didn’t do it ‘on the fly’.
4. Create a decimal-binary converter that allows you enter a decimal value and see the binary equivalent when you press A+B.
5. Modify the program to implement a hexadecimal to decimal converter

**Example 6: Using the micro:bit display and accelerometer**

|  |  |
| --- | --- |
| The micro:bit’s display is made up of a *light emitting* *diode (LED)* *matrix*.  Each LED in the matrix makes up one pixel so the micro:bit display has a total of 25 pixels. (Contrast this with a screen resolution of pixels used currently by many desktop monitors.)  Each pixel is capable of displaying a single colour in varying levels of brightness. |  |
| *Display co-ordinate system of the micro:bit* |

Pixels can be displayed using the plot code block or as sprite objects.

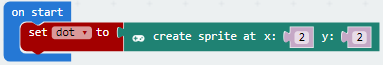
The plot code block is used to display pixels using absolute co-ordinates. The illustration below depicts three examples of the use of plot.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |

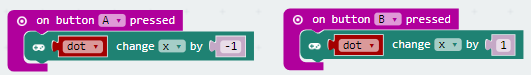
*Exercises*

1. Write a program that starts up with the LED light at the centre of the display turned on.
2. Extend the program so that button A causes the position of the light to move left by one pixel, and button B causes the position of the light to move right by one pixel.
3. Can you outline any design decision you would have to make about what should happen when button B is pressed and the light at co-ordinate (2, 4) is already on?

The code illustrated below depicts how to display pixels as sprite objects. In this example the name of the object is the variable dot. Initially, dot is set to refer to a sprite object at co-ordinates (2,2).



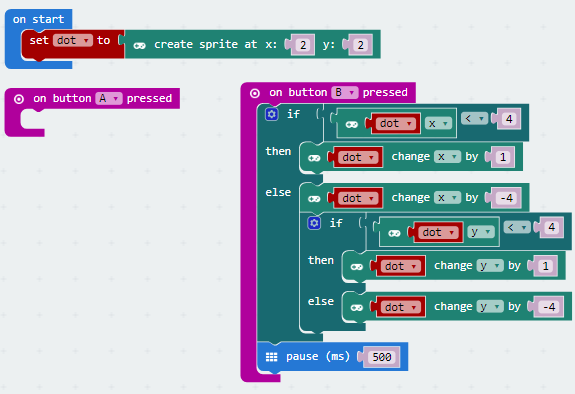
Once a pixel has been created in this manner it can be manipulated using the dot reference. For example, the change command can be used to alter the position of the dot object as follows.



The above illustration shows how the horizontal position of dot can be changed by 1 (left or right) in response to a button event (A or B).

*Exercise*

Download and run the program below (called ‘Display Exercise’) and see if you can figure out how button B causes the dot to move in a rightward/downward direction.



Add the necessary code blocks to the button B event so that when it is pressed the dot will move in the opposite direction i.e. leftwards and upwards.

**Accelerometer[[1]](#footnote-1)**

An accelerometer chip measures the forces acting on a tiny weight in each of three dimensions. If you imagine the micro:bit sitting flat on a desk, the x dimension is left to right; the y dimension front to back and the z dimension into and out of the desk.

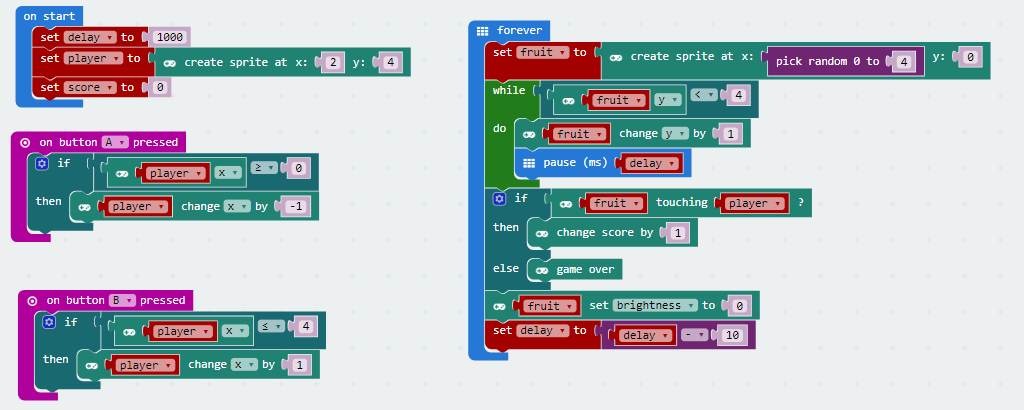
|  |  |  |
| --- | --- | --- |
|  | The accelerometer detects if it is speeding up or slowing down in three directions labelled as X, Y and Z.  Each axis can measure between -4096 and 4096 which correlates with -4G and 4G respectively | |
| If the micro:bit is perfectly flat, the forces on the x and y dimensions will be zero. The force acting on the z axis will have a value (when read in your JavaScript Blocks) of about -1000 due to the pull of gravity.  If you then tilt the back of the micro:bit up then the force in the y dimension will increase a little. | |  |

|  |  |
| --- | --- |
| The code blocks shown here to the right demonstrate how the accelerometer can be used to move the position of a LED light both horizontally and vertically in response to tilting the micro:bit on its X and Y axis respectively.  Can you suggest any ways in which this idea could be developed by LCCS students? |  |

**ADDITIONAL NOTES**

**Example 7: Fruit Ninja Game[[2]](#footnote-2)**

Download the program called ‘Fruit Ninja’ and study the code carefully.



|  |
| --- |
| Describe how to play the game. |
| Evaluate the game. Can you suggest any changes that might make the game more enjoyable and/or challenging? |

|  |
| --- |
| Explain how the code works. |

*Exercise*

Modify the code so that the accelerometer (as opposed to the buttons) is used to control the horizontal movement of the fruit object.

**Example 8: Raindrop Catcher**

Browse to <https://nccacurriculum.azurewebsites.net/Senior-cycle/Senior-Cycle-Subjects/Computer-Science/Support-Material-draft-for-T-L/2-ALT-Resources/ALT4-Support> and scroll down to the bottom of the page where you will see links to nine levels of scaffolded resources in editable HTML files to create a ‘drop and catch game.

The progression in functionality (and programming concepts) that exist across the levels are evident from the table below

|  |  |
| --- | --- |
| *CT Level 0 Challenge* | Write code, using variables, to simulate a raindrop. |
| *CT Level 1 Challenge* | Write a program to simulate a raindrop dropping in the centre every second. When it reaches the bottom of the screen, it must return to the top. |
| *CT Level 2 Challenge* | Include a Catcher on the bottom row, controlled using the A and B buttons. The Raindrop and Catcher do not interact at this level |
| *CT Level 3 Challenge* | If the user catches the raindrop, game over and happy face. If the raindrop gets through, then fall from the top but slower |
| *CT Level 4 Challenge* | Raindrop falls from a random location at the top of the screen. If the drop is caught, the next round starts from another random location. If the raindrop gets through, then show number of completed rounds. |
| *CT Level 5 Challenge* | A limited number of rounds, each round the raindrop speeds up as it falls. If the raindrop gets through, then sad face. Show number of rounds completed out of total |
| *CT Level 6 Challenge* | 2 or more raindrops, stored in an array, drop from random locations. On reaching the bottom, they return to top random locations. No catcher interaction. |
| *CT Level 7 Challenge* | 2 or more raindrops, a limited number of rounds, each round the raindrop speeds up. If a raindrop gets through, catcher loses. Show number of rounds completed out of total. |
| *CT Level 8 Challenge* | The same algorithm as the previous level, but functions used where appropriate. Identify blocks of code that can act as modules, enhancing abstraction and re-usability. |

1. https://learn.adafruit.com/micro-bit-lesson-1-using-the-built-in-sensors/accelerometer [↑](#footnote-ref-1)
2. The Official BBC micro:bit user guide, G. Halfacree, Wiley, 2018 [↑](#footnote-ref-2)