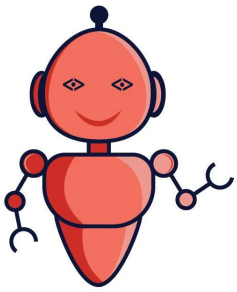
2018



KidzCanCode

IGNITE, INSPIRE, INNOVATE

www.kidzcancode.com

Space Invaders

KIDZCANCODE

TREVOR WARREN

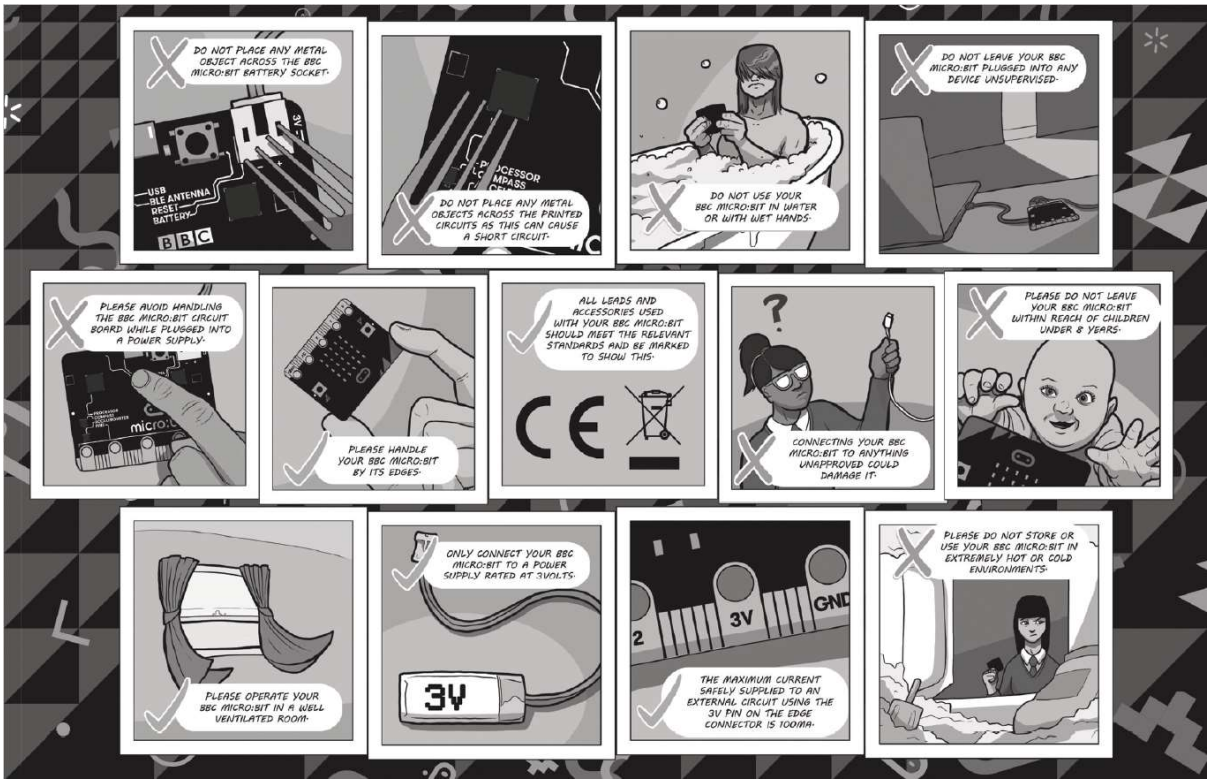
Tutorial 2.1w – Space Invaders

1. Safety Warnings

THE MICRO:BIT IS AN EXPOSED BOARD, TO BE USED WITH CARE PLEASE RETAIN THIS INFORMATION FOR FUTURE REFERENCE. You can read the detailed document at - http://microbit.org/guide/safety-advice/

a. General Safety Warnings

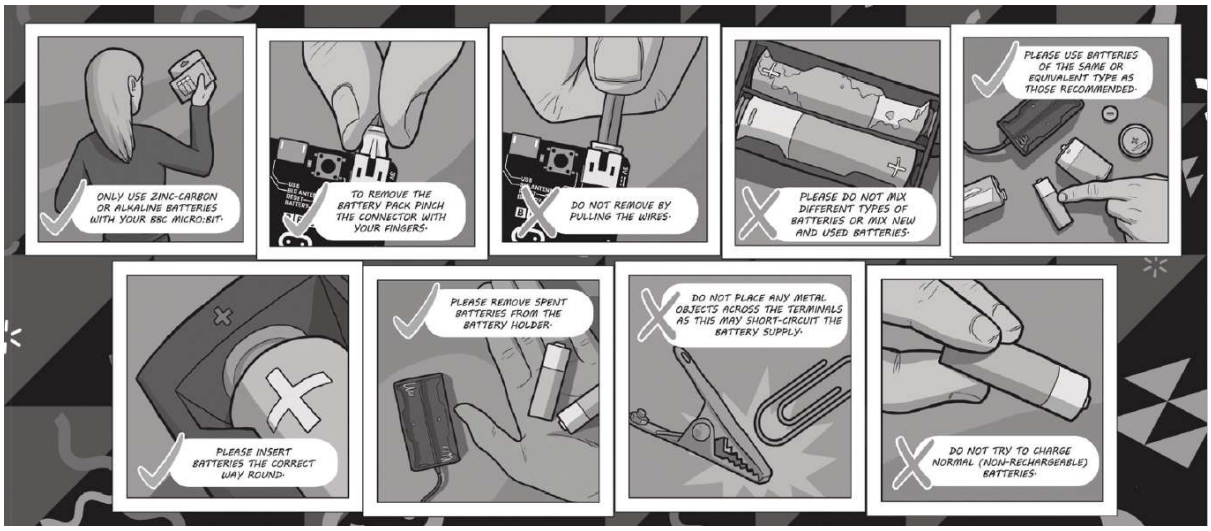
Using the BBC micro:bit is easy to use but is designed to have all the electrical parts on display. This does mean there's a small risk that the parts can be damaged and even overheat with a risk of injury but a little bit of care and caution will ensure you and your micro:bit will stay fit and healthy.



1. Always keep your BBC micro:bit in the anti-static bag when not in use. It's good practice for students to earth themselves before handling it.
2. Please handle your BBC micro:bit by its edges. This minimises the risk of damage through an electrostatic discharge.
3. Please use the battery pack and the USB lead provided to power your micro:bit. Do not use portable battery chargers or USB charging ports (often marked with a lightning bolt or 'SS'), to power your micro:bit. Using these may damage your micro:bit and stop it working properly.
4. Please avoid handling the BBC micro:bit circuit board while plugged into a power supply.
5. All peripherals (for example: USB cable, battery holder, sensors) used with your BBC micro:bit should comply with the relevant standards and should be marked accordingly.
6. Connecting your BBC micro:bit to any unapproved peripherals could damage your BBC micro:bit
7. Please do not attempt to keep using faulty micro:bits. If a school-issued micro:bit develops a fault, contact the vendor immediately.

8. The maximum current safely supplied to an external circuit using the 3V pin on the edge connector is 100mA. Please make sure this limit is not exceeded.
9. Please do not store or use your BBC micro:bit in extremely hot or cold environments.
10. Do not place any metal objects across the printed circuits on the board as this can cause a short circuit damaging your BBC micro:bit. This can cause risk of burn or fire.
11. Do not use your BBC micro:bit in water or with wet hands.
12. Do not leave your BBC micro:bit plugged into a computer or any other device unsupervised.
13. Please do not leave your BBC micro:bit within reach of children under 8 years of age.
14. Please operate your BBC micro:bit in a well ventilated room To remove the battery pack, pinch the connector with your fingers. Do not remove by pulling the wires.

b. Battery Warnings



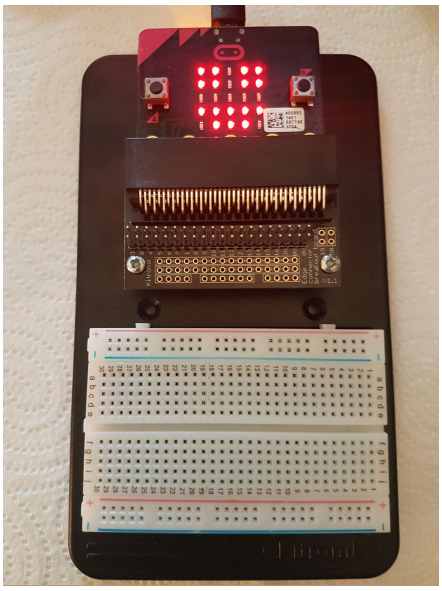
1. Do not try to charge normal (non-rechargeable) batteries
2. Please do not mix different types of batteries or mix new and used batteries.
3. Please use batteries of the same or equivalent type as those recommended.
4. Please insert batteries the correct way round (with the correct polarity).
5. Please remove spent batteries from the battery holder.
6. Do not short-circuit the battery supply terminals, for example by placing a metal object across the terminals.
7. Only use Zinc or Alkaline batteries with your BBC micro:bit.
8. Please do not use rechargeable batteries

2. Pre-requisites

If you need assistance with the assembly of the micro:bit, Edge connector breakout board, mounting board and the breadboard please speak to one of the volunteers on duty.

To be able to perform this tutorial you will need the following components –

1. Parts required –
    a. 1 x BBC Micro:bit
    b. 1 x Mounting Plate
    c. 1 x Edge connector breakout board
    d. 1 x Bread board
2. Assembly required –
    a. Bread board mounted on top of the mounting plate
    b. BBC Micro:bit inserted into the Edge Connector breakout board



Before proceeding please check your setup and confirm that all the required parts are configured as demonstrated in the above picture.

3. Learning Objectives

The objectives of this tutorial are to introduce the student to the following concepts –

- Use code to turn LED's on and off across the board
- Understanding the use of custom functions for purposes of programming
- Understanding how to use sensors, reading the value from sensors and making decisions based on the values obtained
- Understanding the importance of making decisions based on events
- Using logical programming constructs to perform actions based on occurrence of different events
- Declaring variables and assigning values to variables at different times of execution in the program
- Coding for use of the buttons on the micro:bit, perform a given action when a button is pressed.

The BBC micro:bit is a powerful little computer. Through programming these games kids explore more advanced computer science concepts. Along the way kids are encouraged to share, create and extend the games using their own imagination and creativity.

This tutorial builds upon concepts introduced in previous tutorials so please make sure you have covered the previous tutorials before you dive into this one. So overall this tutorial intends to build upon concepts learnt in previous tutorials while exploring new concepts.

In future tutorials we will continue to build upon the concepts learned here and will build more complex interactive games using the functionality provided by the micro:bit.

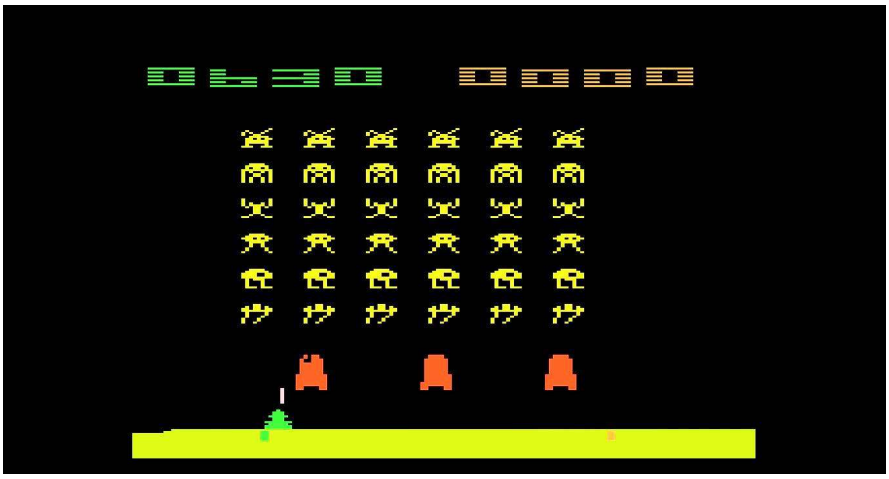
4. Activity

a. Activity

Space Invaders is an arcade video game created by Tomohiro Nishikado and released in 1978. It was originally manufactured and sold by Taito in Japan, and was later licensed for production in the United States by the Midway division of Bally. Space Invaders is one of the earliest shooting games and the aim is to defeat waves of aliens with a laser cannon to earn as many points as possible. In designing the game, Nishikado drew inspiration from popular media: Breakout, Gun Fight, The War of the Worlds, Space Battleship Yamato, and Star Wars. To complete it, he had to design custom hardware and development tools.

It was one of the forerunners of modern video gaming and helped expand the video game industry from a novelty to a global industry (see Golden age of arcade video games). When first released, Space Invaders was very successful.

This activity involves designing a replica of the space invaders game using our micro:bit.



The activity is designed to have additional challenges which allow the developer to keep pushing the boundaries.

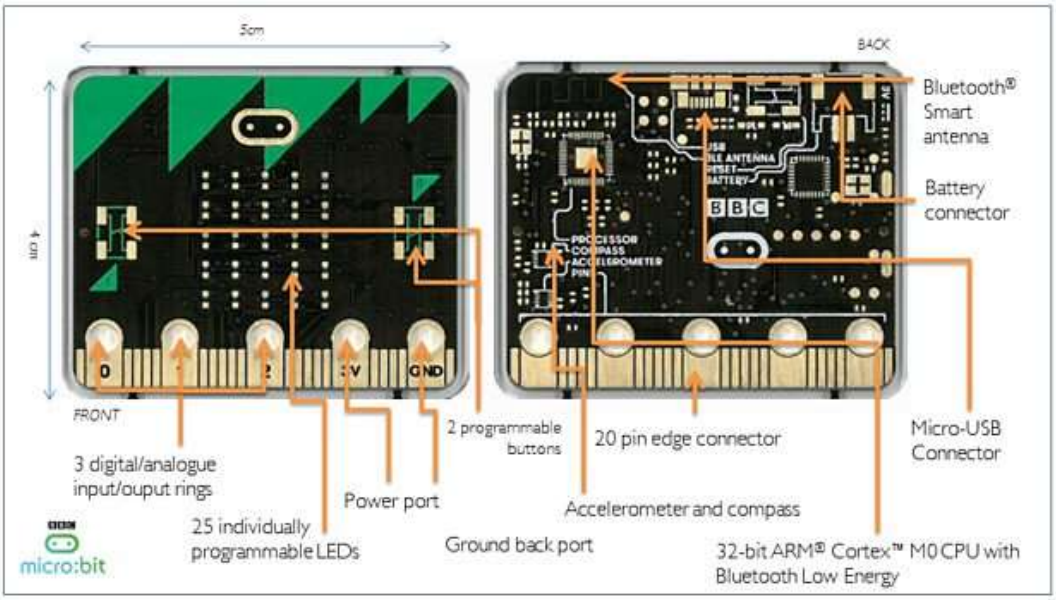
This tutorial is based on the space invaders game created by Digimakers. All the steps used in section 5 of this tutorial are from the Digimakers handout and created by the Digimakers team. See following link for additional details - http://www.digimakers.co.uk/digi-core/uploads/2016/09/microbit-space-invaders-worksheet.pdf

b. How Does It Work

This section talks about the BBC micro:bit and what it's made up of. If you have already read through this section then feel free to skip directly to the next section. The BBC micro:bit is a powerful little board and has various types of sensors on board. Here's what makes up the BBC micro:bit.

1. Size: approx. 5cm x 4cm.
2. Weight: 8g.
3. Processor: 32-bit ARM Cortex M0 CPU.
4. Bluetooth Low Energy.
5. Digital Compass.
6. Accelerometer.
7. Micro-USB controller.
8. 5x5 LED matrix with 25 red LEDs.
9. Pins for connecting external sensors, LED's, etc.

Here's what the micro:bit looks like –



Front of the board (left hand side)

1. Button A (left button with edge connector at the bottom) – labelled A on the board
2. Button B (right button with edge connector at the bottom) – labelled B on the board
3. P0 (left large pin (crocodile clip port) with edge connector at the bottom) - labelled 0 on the board
4. P1 (middle large pin (crocodile clip port) with edge connector at the bottom) - labelled 1 on the board
5. P2 (right large pin (crocodile clip port) with edge connector at the bottom) - labelled 2 on the board
6. +3V - labelled 3V on the board. This is 3V PWR OUT
7. GND
8. P3 – P22 pins from left to right with edge connector at the bottom. Referred to as Pins when referencing that part of the board. Text will talk about 'pins' when referring to

individual connections or the general way of connecting to the board – not labelled on the front of the board
9. LED matrix referred as the 'screen' - not labelled on the board
10. LED coordinates starting at 0,0 top left corner and ending at 4,4 at the bottom corner - not labelled on the board

The order of the large pins as follows: P0 P1 P2 3V GND labelled 0, 1, 2, 3V GND on the board

Rear of the board (Right hand side)

1. 1. USB Plug (Micro-USB plug) – labelled USB on the board
2. Button R (reset button) – labelled Reset on the board
3. Status LED – not labelled on the board
4. Battery socket – labelled Battery on the board

Other components on the board include

1. Accelerometer
2. Compass
3. Bluetooth Smart Technology Antenna
4. AAA Battery Holder - not labelled on the board
5. Processor (Cortex M0)

The BBC micro:bit is programmable in a few different languages. You can write code for the micro:bit using the Makecode block coding interface, Javascript, Python of even in C. Most of our tutorials will cover the use of the Makecode block coding interface built by Microsoft for the micro:bit.

5. Let's write some code

It's time to write some code and get going with coding our game. So let's head over to the micro:bit block code editor page (https://makecode.microbit.org/) and get coding!!!

a. Lighting LED's

We're going to start simple, by just getting an LED to display on the screen. Then we'll see how we can make a pattern of LEDs blink. First, we need a loop. A loop is a block of code which repeats. Some loops repeat forever, others repeat for only a certain number of times. We want a forever loop.

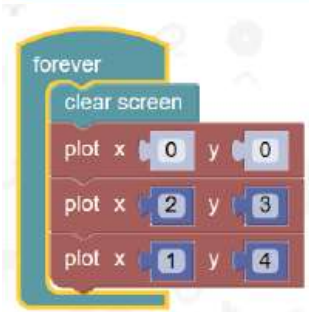
Actions

1. Click the "Basic" tab on the left
2. Click and drag a "forever" block into the editing area

Each time around the loop, we're going to clear the screen and then light up some LEDs again. Later, we'll see how this allows us to move the defender and the space invaders around the display.

Actions

3. Click and drag a "Clear screen" block from the "Basic" tab to the inside of your "forever" block
4. Click and drag a "Plot x/y" block from the "LED" tab to just after the "Clear screen" block
5. Add more "Plot x/y" blocks and change the x/y values to plot more pixels. You can choose any number between 0 and 4 (including 4)!
6. Click "compile" at the top of the screen and download the file to your Microbit.



Goals

You should now see your pattern of LEDs on the display of your Micro:bit!

b. Defenders Position

Now we know how to make the LEDs light up, we want to be able to use the bottom row of LEDs to display the defender. The defender will be represented by a single LED in the column the defender can shoot in. We need to track the defender's position - 0 for left-most column, 4 for right-most and 1,2,3 for the columns in between. To keep track of which column the defender is in, we can use a variable.

Notes

A variable named thing which can be set to a whole number (0,1,2,3,.....) or a logic value (true/false). We can change the value of the variable at different points in the code.

Actions

1. Click and drag an "Item" block from the "Variables" tab into the "x" value of a "plot" block
2. Click the down arrow next to the word "item" in the block
3. Click "Rename variable..."
4. Rename the variable to "DefenderX" (without the quotes)
5. Set the "y" value of the "plot" block to 4
6. Remove any other "plot" blocks from your "forever" loop

Plotting the Defender

Goals

Click run to see a simulation of your program in the editor. You should see the bottom-left LED light up.

c. Moving The Defender

We now know how to use a variable to track the defender's position and make the LED light up. But the value of the variable is never changed - our defender always stays in the bottom left!

We're going to use the tilt sensor to move our defender around. When the board is tilted left, the defender will move towards the left hand side of the display. When the board is tilted right, the defender will move towards the right hand side of the display.

Actions

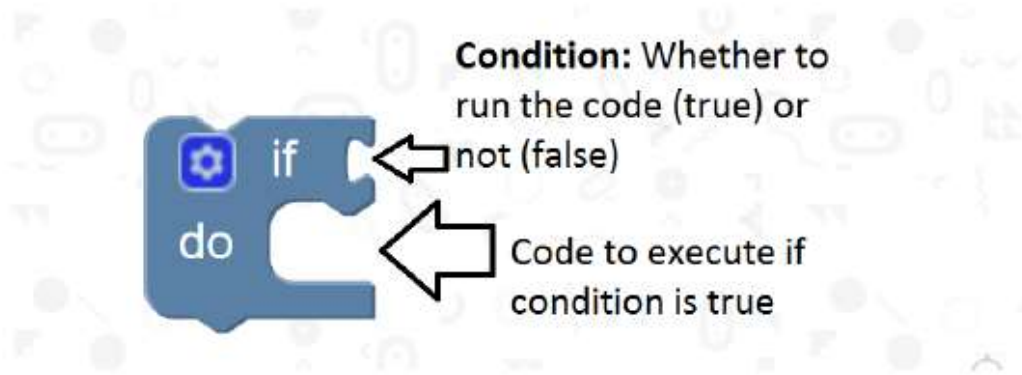
1. Click and drag another "forever" block into the editing area (from the "Basic" tab)
2. Click and drag an "if" block from the "Logic" tab into the "forever" block

Notes

An "if" block is a decision block (often called a Conditional block). You test something, and if that test passes, then the code inside the block executes. Otherwise, the code inside the block doesn't execute and the programs goes to the block following the if block.

Notes

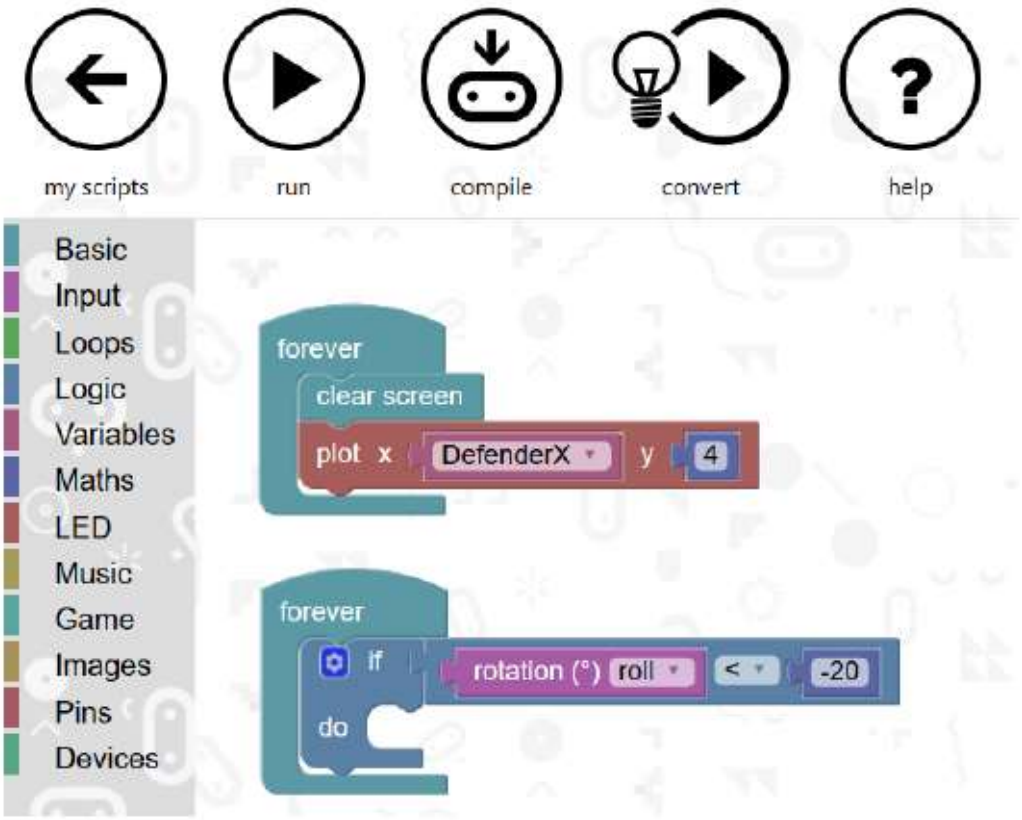
The "test" the if-block uses is called the "condition". It is a logic equation. For example, "is five greater than four" is a condition (to which the answer is always "yes" - usually called "true"). If the condition is "true" the code inside the if block executes, otherwise the condition is "false" so the code inside doesn't execute.



An if-block: Condition determines if the code inside runs.

Actions

3. Click and drag a "less-than" block ("¡" block) from the "Logic" tab to the connector on the right of the word "if"
4. Click and drag a "rotation" block from the "Input" tab to the left-hand "0" of the "less-than" block
5. Click the drop down and select "roll" instead of "pitch"
6. Set the "0" on the right-hand side to "-20"



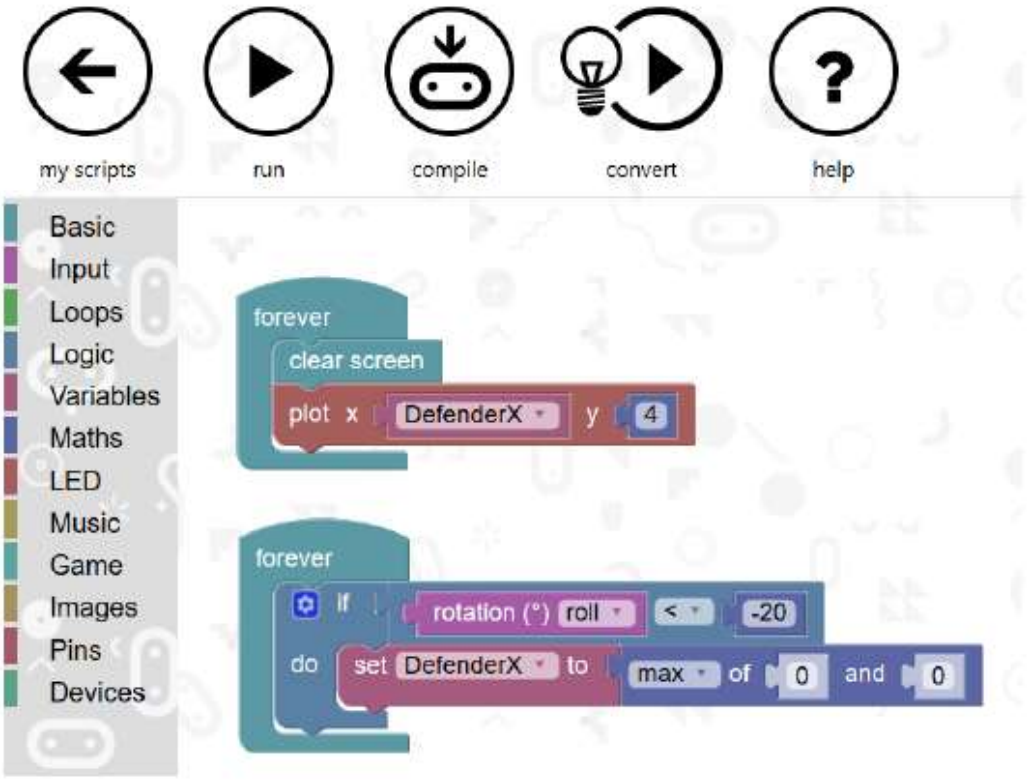
First condition for rotation of the Micro:bit

Notes

This code continuously loops, checking to see if the tilt of the Micro:bit is 20 degrees or more to the left. If it is, it will execute the (empty) code inside the if block.

Actions

7. Click and drag a "set item to" block from the "Variables" tab to inside the "if" block
8. From the drop down, change "item" to "DefenderX"
9. Click and drag a "max" block from the "Maths" tab to the right-hand connector of the "set" block



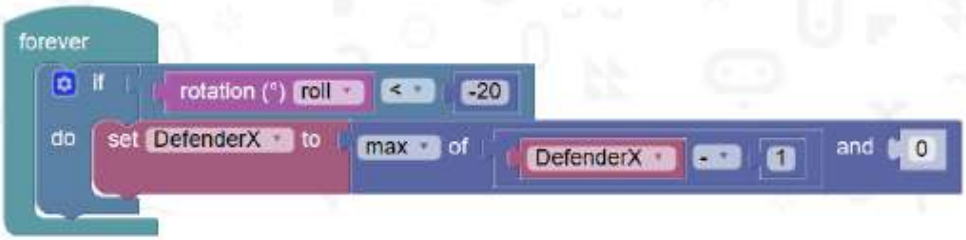
First use of a max-block

Notes

"Max" is a clever maths operation which picks the larger of the two numbers it is given.

Actions

10. Click and drag a "subtract" block ("-" block) from the Maths tab to inside the left-hand value of the "max" block
11. Click and drag a "DefenderX" item block to inside the left value of the "subtract" block
12. Set the right side of the "subtract" block to 1



Code for moving the defender left

Notes

The code inside the if block will now set the "DefenderX" variable to the maximum of "DefenderX - 1" and "0". In other words, if the defender is in the left-most column (column 0) it will stay there. If it is not in the left-most column, the defender will move one column to the left. E.g. if it is in column 3, it will move to column 2.

Actions

13. Copy and paste the "if" block to after itself (use Ctrl+C to copy, Ctrl+V to paste)
14. Set the condition to use ">" (greater-than) instead of "<" (less-than)
15. Set the right hand comparison value to "20" instead of "-20"
16. Set the maths operation to "min" instead of "max"
17. Change the "subtract" operation to "addition"
18. Set the right hand "and" value of the "min" block to "4"

Code for moving the defender left and right

Notes
The tilt direction in the simulator in the editor is the opposite to the real Micro:bit.

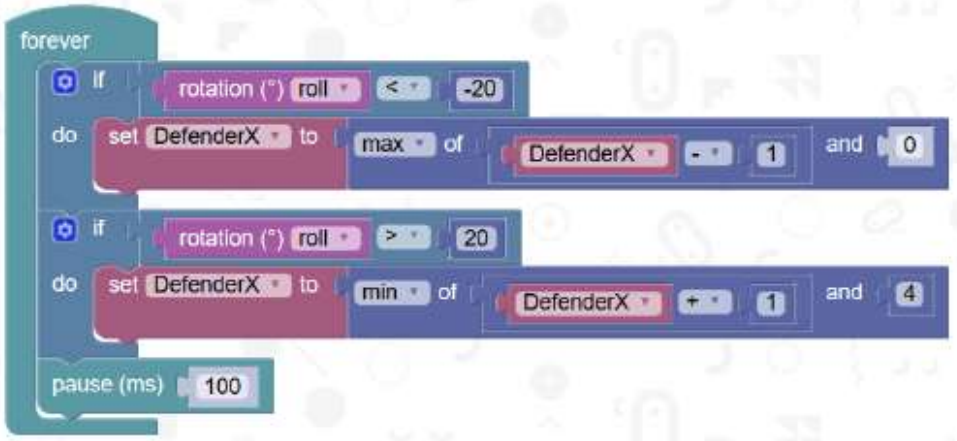
Goals
Compile and save your code to your Micro:bit - you should now see the Defender move left and right on the bottom of the display when you tilt the Micro:bit left and right.

Notes
You may notice the defender moves very quickly - too quickly! We can fix that by adding a delay to the loop.

Notes
A delay pauses the execution of the blocks for a length of time. The length of time is in milliseconds. There are 1000 milliseconds to 1 second.

Actions

19. Click and drag a "pause" block from the "Basic" tab to the end of the "forever" loop after your "if" blocks
20. Set the value to "100" (if it is not already 100)



Improved code for moving the defender left and right

Goals

Compile and save your game to your Micro:bit. You should now be able to control the defender position so that you can put it in a particular column by tilting left and right.

d. Starting The Game

At the moment, as soon as the Micro:bit switches on, the game will begin. This is a problem because you may want to wait until the player is ready. We can achieve this by using another variable and waiting for the A and B buttons to be pressed before starting the game.

When both the A and B buttons are pressed together, we will set a new variable called "Playing" to "true" to indicate the start of the game. When the player dies during a level, it will reduce their number of lives by 1. When they have no lives left, we will set "Playing" to false to signal the end of the game.

Let's start by doing the game start signal - buttons A and B being pressed together.

Actions

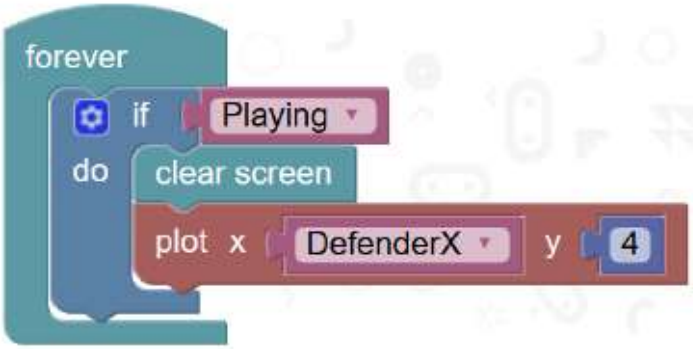
1. Click and drag an "on button A pressed" block from the "Input" tab onto the editing area
2. From the drop down, change "A" to "A+B"
3. Add a "set item to" block (from the "Variables" tab) into the "do" section of the button press block
4. From the "item" drop down, select "New variable..."
5. Call the new variable "Playing" (without quotes)
6. From the "Logic" tab, click and drag a "true" block to the right hand connector of the "set Playing to" block.

Code to detect user pressing buttons A and B to start the game.

We also only want to draw the player and invaders on the screen when the game is being played. We can make sure they are only drawn when the game is being played by putting an if-block around our draw code. The condition of the if block will be our variable - "Playing".

Actions

7. Click and drag an "if" block from the "Logic" tab into the editing area
8. Click and drag the "clear screen" block (which should drag the blocks after it too) from the "forever" loop into the "if" block
9. Click and drag a "Playing" variable block from the "Variables" tab into the right-hand connector of the "if" block condition
10. Click and drag the "if" block (and all its contents) back into the empty "forever" loop



Only draw the player / invaders when playing the game

Goals

Compile and save your game to your Micro:bit. When the Micro:bit powers on, the screen should be blank. Press the A and B buttons together - you should now see the defender at bottom of the display.

e. Invader Positions

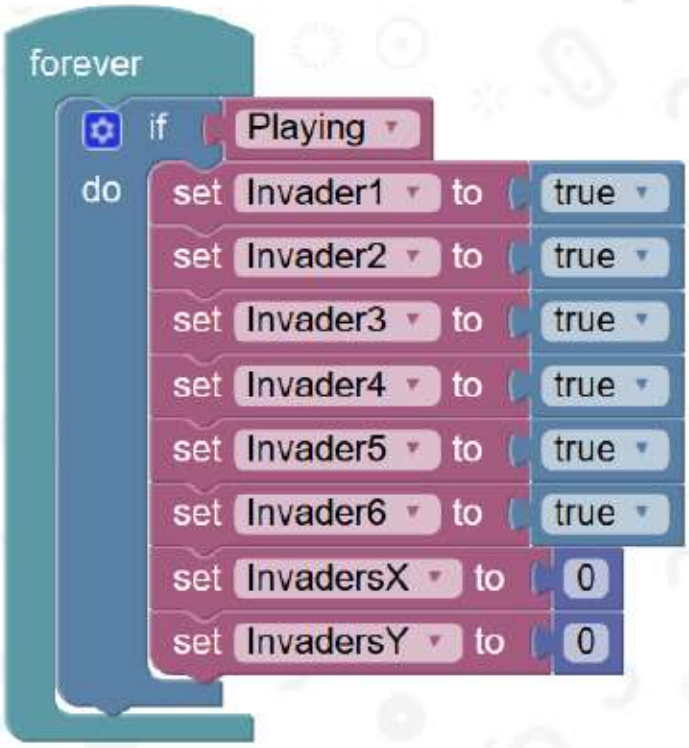
There will be six invaders, that will start at the top-left and work their way across as a block from left to right. When they reach the right hand side, they will jump back to the left again.

We will need to keep track of which invaders are alive and which are dead. We can do this using six variables named "Invader1", "Invader2" and so on up to "Invader6". If an invader variable is "true", it will mean the invader is alive. If it is "false", the invader is dead.

We will also need to know where the invaders are. We will use two variables for this: one for the X-location (column number) that the left-most invader is in, and one for the Y-location (row number) that the top-most invader is in.

Actions

1. Add blocks to your game's code so that it matches the screenshot below. You will need to add a new "forever" block.

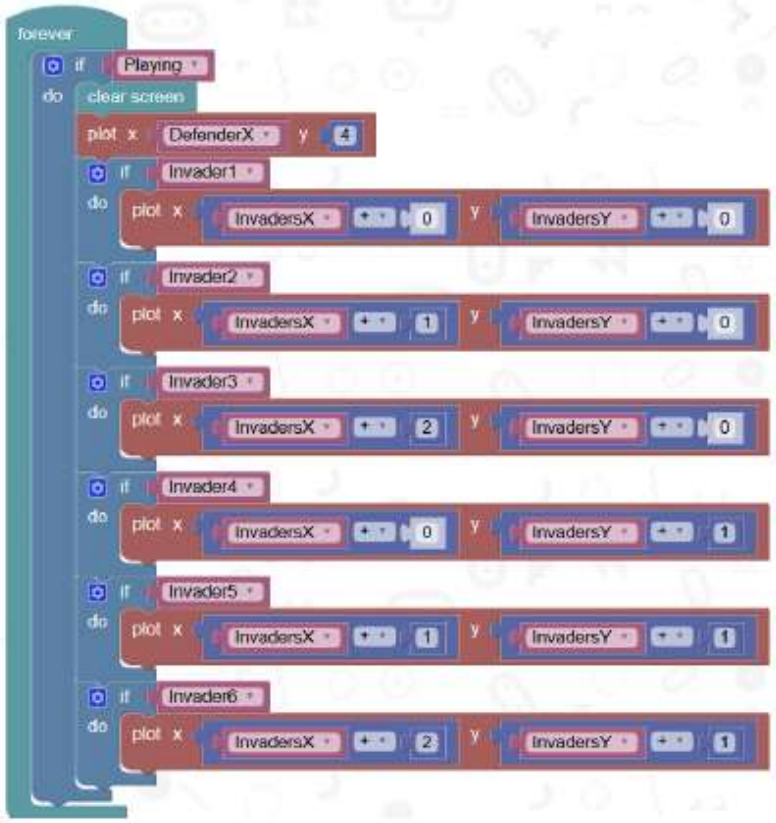


Initialising variables for tracking the invaders

Now we need to plot the invaders - i.e. to display them on the LEDs. We can do this by using the current location of the invaders and whether each invader is alive. If the invader is alive, we turn on the relevant LED. Otherwise, we leave the LED off.

Actions

2. Add blocks to your game's code (extending the section we made earlier for displaying the defender) so that it matches the screenshot below.



Code for displaying the invaders and the defender

Goals

Compile and save your game to your Micro:bit game to your Microbit. When the device powers on, press the A and B buttons together. You should now see the defender moving as before and the six invaders (staying still) in the top-left corner. (We'll make them move in the next section!)

f. Controlling The Game

Our game is almost complete - we just need to add the final two sections of code. The first one is what we call the "Game Logic". The second is the section to enable the defender to shoot players.

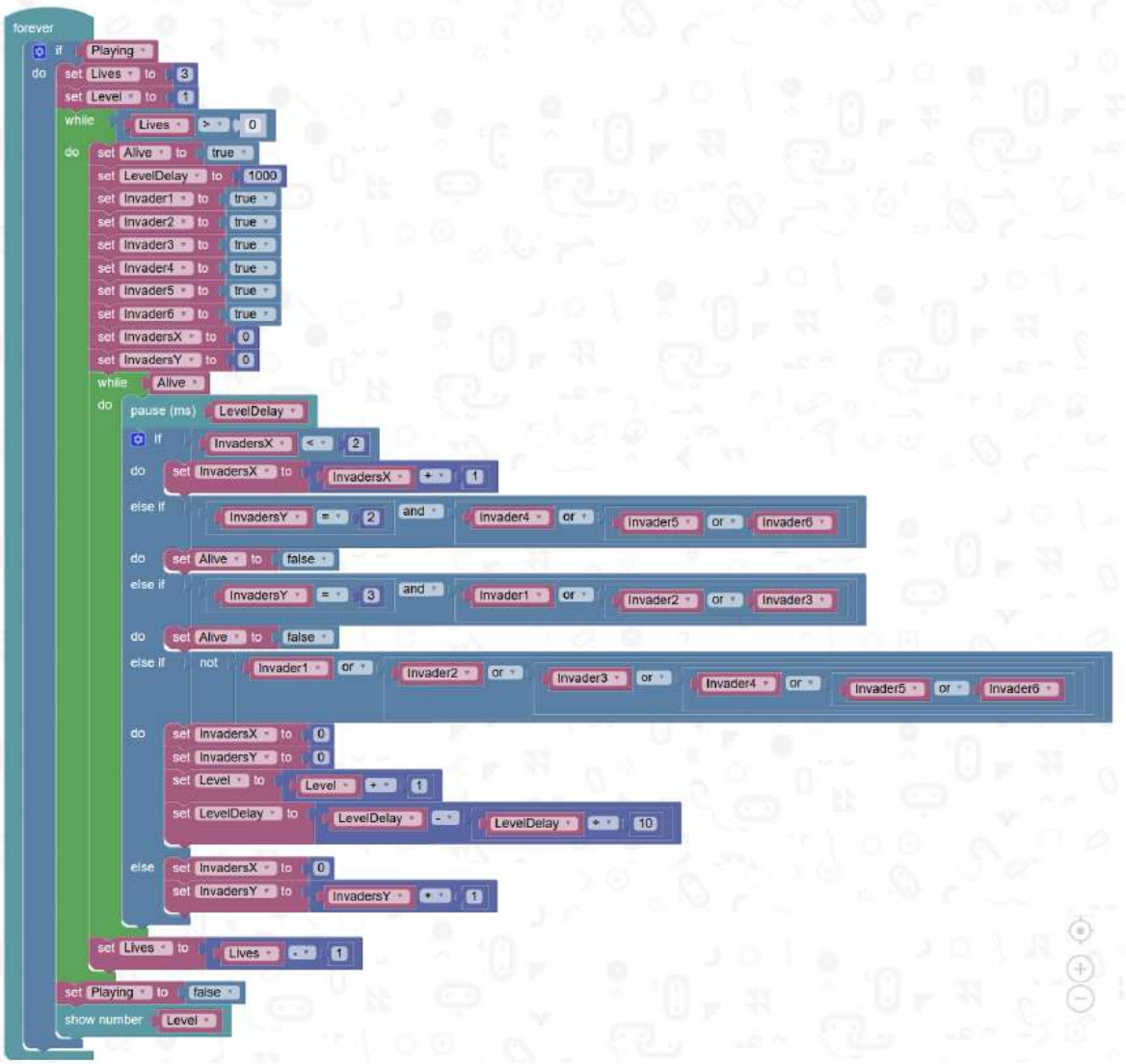
The Game Logic is the main section of control code which determines how the game proceeds (as opposed to how its drawn or how the player moves). Game Logic also includes the number of lives a player has and what determines when a player dies.

Our game logic breaks down into two loops: An outer loop that allows the player to keep playing while they have lives left. And an inner loop: Which progresses through harder/faster levels until the player dies.

Actions

1. Add blocks to your game's code so that it matches the screenshot below.

Tutorial 2.1w – Space Invaders



Questions

1. Can you work out what is going on in this code?
2. Before you run the code, from your understanding of the code, do you think the space invaders will move from left to right or from right to left?
3. Before you run the code, from your understanding of the code, how many lives will the player get?
4. Before you run the code, from your understanding of the code, will the speed of each level increase? If so, by how much each time?

Tutorial 2.1w – Space Invaders

Goals

Compile and save your game to your Micro:bit. Press the A and B buttons together to start the game. You should see the invaders move around the display and the defender move at the bottom of the display.

Questions

5. Now you've tried running the code, which direction do the space invaders move?
6. How many lives did the player get?

The speed of the current level is reset when the player dies, so you won't have seen the level-speed increase yet.

g. Shooting Invaders

This is the last part to our code - shooting the invaders! We're going to program our game so that, after the game has started, when button A is pressed, the bottom-most invader that is in the same column as the defender will be shot.

Actions

1. Click and drag an "on button A pressed" block from the "Input" tab onto the editing area.
2. See if you can work out what if-blocks and conditions you will need to add to the "on button A pressed" block, to determine which invader the defender is underneath (if any)

See if you can work out the last section of code!

Notes

Hint: Think about the position of the defender (DefenderX) and the position of the invaders (InvadersX). You will want to compare them somehow!

Notes

Hint: The bottom-most invader should be killed first. Once it is dead, the top row invader can be shot.

Goals

Test your new code. Does it work? If not, keep trying - code, test, repeat - that's how real computer scientists and software engineers work.

Actions

3. Try to get a working version of shooting the invaders. If you're having problems, ask a workshop helper.

Goals

Congratulations! You've finished making the game!

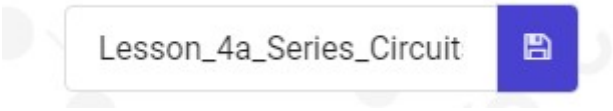
Questions

7. Are there any improvements to the game you can think of? Have a go at programming them. Remember: code-test-repeat.

You are encouraged to make changes, improvise and customize the game using your own ideas.

6. Downloading Your Code To The micro:bit

Once you have completed the program, enter a name for your program using the option provided below i.e. in the text box adjacent to the small save button.



Now that you have given your program a name and saved it you can download it your micro:bit. But before we do that let's confirm what drive your micro:bit shows up as. On most machines the micro:bit will show up as an additional USB drive. So head over into windows explorer and confirm what drive name (D:, E:, F:. G:, etc.) the micro:bit shows up as. You need to absolutely be sure what drive the micro:bit shows up as. Once you've confirmed what drive the micro:bit shows up as on your machine you can select the right drive when downloading the code to the micro:bit. If in doubt please ask the volunteer/mentor/session facilitator helping out.

To download the newly written code to the micro:bit, hit the download button shown above. You should now see a dialog box open up and you will be asked to save the file somewhere on your machine. Please choose the drive your micro:bit shows up as i.e. D: or E: or F: or whatever it shows up as on your machine. A sign of success is when you see the lights on the rear (orange) lighting up in quick succession suggesting that the code is being written to the micro:bit. On completion the micro:bit reboots and you should now see the code in action on the micro:bit.

If hitting the download button shown above does not open up a dialog box asking you to save to the micro:bit please save the file (you will have a <filename.hex> file) to your desktop. Then open up windows explorer and drag that file onto the drive which is your micro:bit. A sign of success is when you see the lights on the rear (orange) lighting up in quick succession suggesting that the code is being written to the micro:bit. On completion the micro:bit reboots and you should now see the code in action on the micro:bit.

Please feel free to customize the code blocks, have a play. Add your own custom code and re-download the code to the micro:bit. Give yourself a tap on the back, you've just completed your first circuit!!!!!

7. Challenges

Well done for completing the tutorial. There's a lot of ground we have covered in this tutorial so please feel free to make notes, come back to the tutorial at some point down the line and ask your learning facilitator any questions or doubts you might have on the concepts covered this far.

Let us now stretch it a bit further -

1. Implement code to allow use of button A to increase the pause time (between LED blinks) by 100 ms every time the button is clicked
2. Implement code to allow use of button B to decrease the pause time (between LED blinks) by 100 ms every time the button is clicked
3. Implement code to detect shake of the board and re-start the program i.e. re-run the entire program from the start.
4. Code the LED's to light up as a stream of lights moving around the board in a sequence to make it look like a snake is slithering around the screen. i.e. don't turn off the LED's as you progress through the inner and outer square
5. Why not try out a python version of the game. See the code at - https://github.com/hexkcd/micro-vaders