# Lesson 9 – Activity Sheet

Setting the Scene

In this project we will explore a more complicated game that involves some logic to determine the winner. The game is the classic rock:paper:scissors.

For this project we will make a programme on two micro:bits so that you can play rock:paper:scissors (also known as Roshambo) <https://en.wikipedia.org/wiki/Rock%E2%80%93paper%E2%80%93scissors>

## Success criteria

* Make a game where A beats B, B beats C and C beats A
* The program will randomly select A, B or C when shaken
* The program will transmit via radio the selection and will determine if it has won or lost the match
* The program will keep count of wins and losses until reset by a button press (A and B)

|  |  |
| --- | --- |
| Rock-paper-scissors.svg  <https://en.wikipedia.org/wiki/File:Rock-paper-scissors.svg> | **Bback propagation**  **C**  **Aback propagation** |

## Getting Started

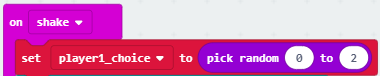
To start with we will solve the second success criteria. We need to create a variable to store the randomly generated A, B or C.

Open <https://makecode.microbit.org/>

## **Random**

We can’t get the micro:bit to make a random choice between A, B or C so we need to get it to randomly generate a number (0,1 and 2) which we can then use to represent the A,B and C. We also can’t compare characters (chars) using logic so we must use numbers for this.

Here you can see we use an ***on shake*** block with a variable called ***player1\_choice***with a **Math** block to create the random number from 0 to 2, we will use 0 as A, 1 as B and 2 as C.

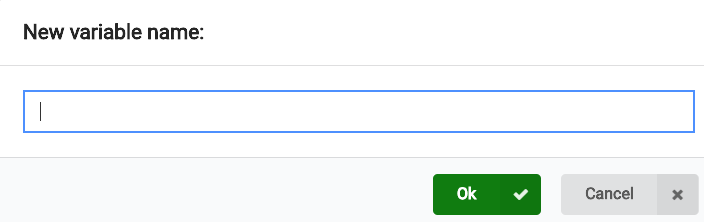
**

To create a variable you need to open the **Variables** menu in MakeCode and select “Make a Variable” button

A close up of a sign

Description automatically generated

this will open a menu to name the variable:



Pro-tip

Name your variables wisely! A good variable name describes what it is for or what it contains.

We will now use some logic to tell the user (and the other player’s micro:bit) what our choice was.

**Two Players Are Better Than One**

For this game to be realistic we will need two players and the code for each will be very similar. Think about how the code for the second player will need to change.

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | There are three “tests” with three different outcomes:   * If the random number is **0** then the micro:bit shows **A** and also broadcasts the number **0** * If the random number is **1** then the micro:bit shows **B** and also broadcasts the number **1** * If the random number is **0** then the micro:bit shows **C** and also broadcasts the number **2**   This gives us the basic game where the shake produces either A, B or C. You could play the game just with this program as you can work out the winner yourself but we can make the micro:bit work it out for us! |

## Pro-tip

We need to ensure that the two micro:bits are talking to each other on the same channel. To do so we need to set the **radio group** to the same number on both micro:bits.

A screenshot of a cell phone

Description automatically generated

So far we have met the following success criteria:

* The program will randomly select A, B or C when shaken
* The program will transmit the selection via radio

## **Keeping Score**

We will now make the program keep score for each user and also allow the user to reset the score to zero.

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | Here we create the initial user interface by telling the user to “Shake to play”. Then we use the ***set score*** block (in the ‘Game’ blocks menu) to set the score to 0 when the program first runs and also when **A+B** are pressed. **The Score Variable** ‘Score’ is a built-in **variable** in the micro:bit used for making games. |

## **The Game Logic**

Now for the slightly tricky bit! We have made the program transmit the choice but we now need to make it receive the other players choice and determine whether we have won, lost, or whether it’s a draw.

We are going to do some comparrisons of the variable ***player1\_choice*** and ***receivedNumber.*** We have just created the ***player1\_choice*** variable and it will contain a number (0-2) if it has been shaken. We now need to make the program listen for the other players choice and to do this we need the ***on radio received*** block from the Radio blocks.

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | This block receives the radio signal from the other micro:bit and stores the number in a variable called ***receivedNumber*** |

We can now compare the number transmitted from the other micro:bit (which represents their choice) and compare it to our number (choice) to determine who has won.

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | To do this we need an **if then else** block from the logic blocks. Inside that block we will put an = block. We are going to compare whether the ***receivedNumber*** is the same as ***player1\_choice*** and if it is then the game is a draw. |

We now need to add an **=**logic block so we can see if the variable is = to ***receivedNumber***.

|  |  |
| --- | --- |
| A screenshot of a cell phone  Description automatically generated | Here you can see the = block with the variables inside them.  The **then** part of the **if then else**  block shows the string “Draw!” because if both player choose the same the game is a draw. |

Now we need to add some more logic to deal with the other possible outcomes of the game. To simplify the logic we are going to use another ***if then else*** block to test if the ***receivedNumber*** is less than (<) the ***player1\_choice*** . This takes care of 2 of the possible outcomes. If the ***receivedNumber*** is less than (<) the ***player1\_choice*** then we see if the ***receivedNumber*** is equal to the ***player1\_choice*** as 2 beats 0. If this is true then the player has won and their score can be increased by changing the in built ***score*** variable by 1, otherwise they lose.

A screenshot of a computer

Description automatically generated

You could also program this logic by testing for each possible outcome individually. This would work but ends up with a lot more code. If there were more variables and comparisions then this would make the code unmanageable and so we use the logic blocks to minimise the number of ***if then else*** blocks we need.

Pro-tip

Check these blocks carefully. It is easy to miss a block or not change a number and this will make the program behave strangely.

## **The Full Solution**

A screenshot of a computer

Description automatically generated

We have now met all the success criteria:

* Make a game where A beats B, B beats C and C beats A
* The program will randomly select A, B or C when shaken
* The program will transmit the selection via radio and will determine if it has won or lost the match
* The program will keep count of wins and losses until reset by a button press (A and B)

## **Two Player**

Don’t forget that this is a two player game and so this program needs to be uploaded to two micro:bits to play properly. You may want to change the ***player1\_choice*** variable to ***player2\_choice*** on the second micro:bit but the program will work fine without this change.

## Test Time

We have used a few different blocks here and lots of blocks within blocks, so now is the time to test your program and make sure that it behaves as you would expect. Make sure to check that the game works correctly a few times!

## Stretch Tasks

* Change A, B, C to icons for rock paper , scissor
* Make the program include three players
* Add D that beats A but looses to C and B)
* Adapt this new program to make it fair
* Change the program so the player has to press a button that start a 3 second countdown before the choice is made
* Adapt the game to use functions to remove the nested ***if*** blocks

## Final Thoughts

This has been the most complicated program so far and has used some complicated logic. There are other ways to make this game and this is just one of them.