Girls' Programming Network

Bop It with Micro:Bits!

FOR TUTOR EYES ONLY

This project was created by GPN Australia for GPN sites all around Australia!

This workbook and related materials were created by tutors at:

Sydney and Canberra

Girls' Programming Network

If you see any of the following tutors don't forget to thank them!!

Writers

Renee Noble
Courtney Ross
Alex McCulloch
Belinda Wong
Rowena Stewart
Kim Apted
Kassandra di Bona
Jennifer Henoch

Testers

Mikaela Goldstein
Michelle McPartland

Intro to Micro:Bit

Part 0: Setting up

Task 0.1: Micro:Bits and pieces

Let's set up the Micro:Bit for programming today! You should have:

- **1 Micro:Bit chip**
- **1 USB cable**

1. Connect the small end of the USB cord to the middle port of the Micro:Bit
2. Connect the big end of the cord to your computer
3. Go to **python.microbit.org**

TUTOR TIPS

Students *must* use Google Chrome or Microsoft Edge to run code directly from microbit.org

If students do not have access to Chrome/Edge - they will need to download their code and move it onto their micro:bit manually.

Task 0.2: Micro playground

First we're going to play around with the displays on **microbit.org** and test them on our Micro:Bits.

1. Make sure `from microbit import *` is at the top of your code.
2. Change the code under the `while True:` loop to display a duck and scroll your name instead
3. Click the **'Send to Micro:Bit'** button to try this out. Then follow the steps on the screen.
4. Try this out with other words and pictures.

Hint

Don't forget you have cheat sheets to help you code!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 1:

- ☐ You are have connected your Micro:Bit to the computer
- ☐ You can display different pictures and words

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.DUCK)
sleep(1000)
display.scroll('name here!')
```

Today's Project Plan - Bop It

We're going to make a Bop It game!
It will prompt the user to press Button A or B to get points!
Get as many points as you can in the time limit.

1. Start off the game by showing a starting image!
2. List your actions, and choose a random action to be the first move!
3. Display different images on the screen depending on what action you chose!
4. Add a loop to make it choose and display actions over and over again!
5. Make the game wait for you to complete the action. Get a smiley and new move when you're correct!
6. Add scores to the game and show the final score at the end of the game!

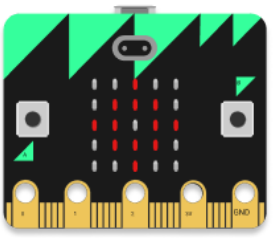
+ more

Once your base game works add cool extensions!
There's extensions sounds, making your own buttons out of foil, using radio communication to make multiplayer games and many more!!

Part 1: Ready! Set! Go!

We need to know the game is starting!

Display an image that tells the player the game is starting!



Task 1.1: Name your file!

Now you're used to working with your Micro:Bit, let's start working on the project!

1. On your Python Editor on **microbit.org**. At the top of the page, edit your project name to be 'bop_it'.
2. Delete all the code except for `from microbit import *`
3. At the top of the file add a new line (above the import line) use a comment to write your name.

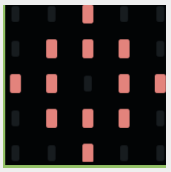
Hint

Remember comments start with a **#**

```
# This is a comment
# Comments don't actually do anything - they are just notes!
```

Task 1.2: Starting your game

To show that the game is starting let's show a target image for 1 second!



1. After your **`comment`** use **`display.show()`** to show a target. (called **`Image.TARGET`**)
2. Make the program **`sleep`** for **`1000`** milliseconds.
3. Then **`clear`** the display.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 2:

☐ Your program shows a target at the start of the game for one second and then clears the display.

☐ You tried it on your real life Micro:Bit

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()
```

Random

Part 2: Choosing a move

Our game is about doing random actions!

Let's start by chosing the first move!
What will be Button A or B?

Task 2.1: Making a list of actions

We need to make a list of actions to refer to later.

1. At the end of your code, create a `list` called `actions`.
2. Inside the `list` store the two actions `"button a"` and `"button b"`.

Hint

Remember a `list` looks like this:

```
fave_foods = ["pizza", "curry", "nutella", "omelette"]
```

Task 2.2: Get random

To randomly select actions in our game we'll need to import a special library.

Underneath `from microbit import *`, add a new line of code that says `import random`

Task 2.3: Selecting the next action

Now we'll use the library to choose a move from our list of actions.

1. Make a new line after our list of `actions`.
2. Choose a random action from the list of `actions`. Assign it to a variable called `action`.
3. Then `print` the `action` so we can see what it is.
   It will print to the serial under the simulator Micro:Bit (make sure you click "show serial to see it!)

Hint

Remember we can choose something randomly from a list like this:

```
fave_foods = ["pizza", "curry", "nutella", "omelette"]
dinner = random.choice(fave_foods)
```

Task 2.4: Check that it works!

Now you need to run your program a few times to check that it is working!

1. Run your code multiple times. See what action it prints out.

Do you get different actions? You might get the same one a few times in a row.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 3:

- ☐ You have a list of **actions**
- ☐ You choose an action using **`random`**
- ☐ You have the next **action** stored in a variable
- ☐ You have a print statement that **`print`**s out the **action**

TUTOR TIPS

The code should look like this (no bonuses):

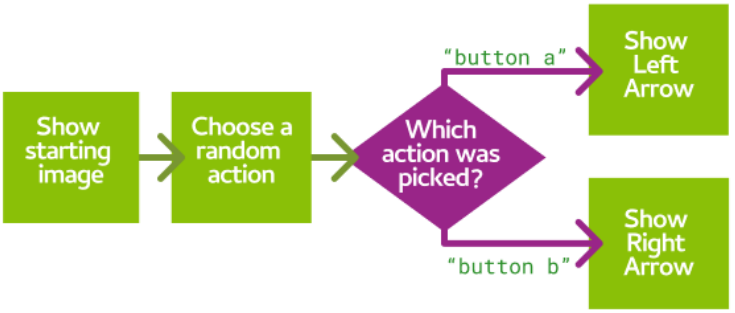
```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["button a", "button b"]

action = random.choice(actions)

print(action)
```

If statements

Part 3: Light it up!

Let's tell the user which action we chose!

Display different images for each action!



Task 3.1: What's your action?

In part 2, you made two **actions** your game can choose from.
Do you remember what they were called?

Write their names down below:

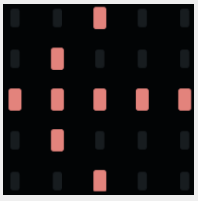
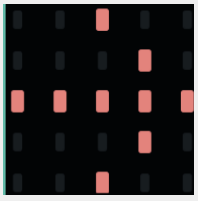
1) .....

2) .....

Task 3.2:

We want to point to the button the player should press.

Which action will we show these images for?

 **Image.ARROW_W**	 **Image.ARROW_E**
Action:	**Action:** .....

Task 3.3: Giving the first action a picture

Let's check what action was selected and display a picture!

1. At the bottom of your code create an `if` statement
2. The if statement should check if the action the computer chose is `"button a"`.
3. Inside the `if` statement, use `display.show()` to show the arrow that points to **Button A.** Make sure it's indented.

Hint

Remember `if` statements have indentation. Here's an example about the weather:

```
if raining == True:
  print ('oh no!')
```

Task 3.4: Giving the second action a picture

Now we'll do the same for the other action

1. After that `if` statement, create another `if` statement.
2. This time check if `"button b"` is the action.
3. Inside this `if` statement, display an arrow that points to **button B**.

Task 3.5: Testing time!

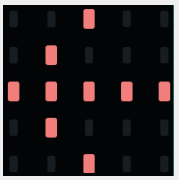
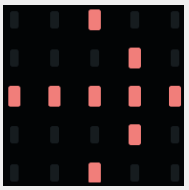
Run your code!

1. Check the terminal to see which `action` was selected.
2. Does it `display` the correct picture for the randomly chosen action?
3. Run your program multiple times to check both actions!

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 4:

☐ When you run the program, it shows **one** of the pictures below

Picture 1:		Picture 2:	

☐ If you run the program multiple times, it shows the **other** picture sometimes. (This might take a few goes)

★ BONUS 3.6: Choose your own pictures! ★

Waiting for the next lecture? Try adding this bonus feature!!

Instead of showing left and right arrows, let's choose our own pictures!

- Replace the code that shows the **left arrow** with `Image.SQUARE`
- Replace the code that shows the **right arrow** with `Image.HEART`

What do you see when you run the program? What if you rerun the program a few times?

Find more images on the *Micro:Bit Image Cheat Sheet*: http://bit.ly/images-microbit

TUTOR TIPS

The code should look like this (no bonuses):

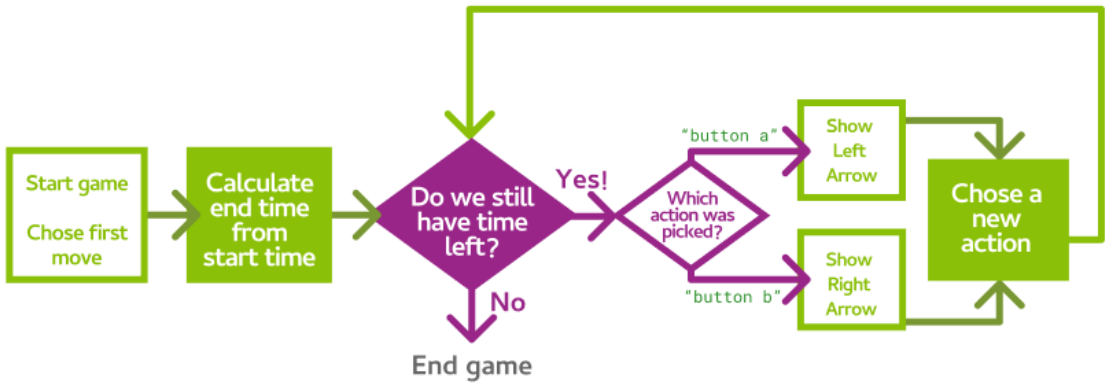
```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["button a", "button b"]

action = random.choice(actions)

print(action)

# If the action is button a
if action == "button a":
  display.show(Image.ARROW_W)

# If the action was button b
if action == "button b":
  display.show(Image.ARROW_E)
```

While Loops

Part 4: The more actions the merrier!

One action is not enough for a game!

Let's make it play on a loop and show new actions for 10 seconds!



Task 4.1: Looping for 10 seconds

To know when to stop the game, we need to know when it started!

1. Make a new line after you randomly choose an **`action`**.
2. Ask the Micro:Bit how long the game has been running by using **`running_time()`**.
3. Store the **`running_time()`** in a variable called **`start_time`**.
4. On the next line, create a variable called **`end_time`**, set it to **`start_time`** plus 10,000 milliseconds (10 seconds). *You can change this later if you want a longer game!*

Task 4.2: Here we go again!

Now let's add the loop that goes until the end_time!

1. Go to the next line after you set the **`end_time`**.
2. Add a **`while`** loop with a condition that checks that the current **`running_time()`** is less than the **`end_time`**.
3. **`Indent`** all the code that is below this line (your **`if`** statements), so they are inside the **`while`** loop.

Hint

Your `while` loop should have a structure similar to this example:

```
while raining == True:
  print ('Raindrops keep falling on my head')
```

Task 4.3: Wait a second and then change the action

We already show the image for the first action we choose! Let's wait 500 milliseconds, then choose a new action.

1. Go to the end of your code, and make a new line. It **should be indented** inside the `while` loop, **but not** inside the last `if` statement.
2. `sleep` for `500` milliseconds.
3. After the `sleep`, update the value of `action` by choosing a new one from the list of `actions` again.

Hint

To update a variable, just assign something new to it! You can use the same code you used in `Task 2.3` to pick the first random move.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 5:

- ☐ Your game runs for 10 seconds
- ☐ Your game keeps choosing new random actions
- ☐ Your game updates to the correct picture for the new action

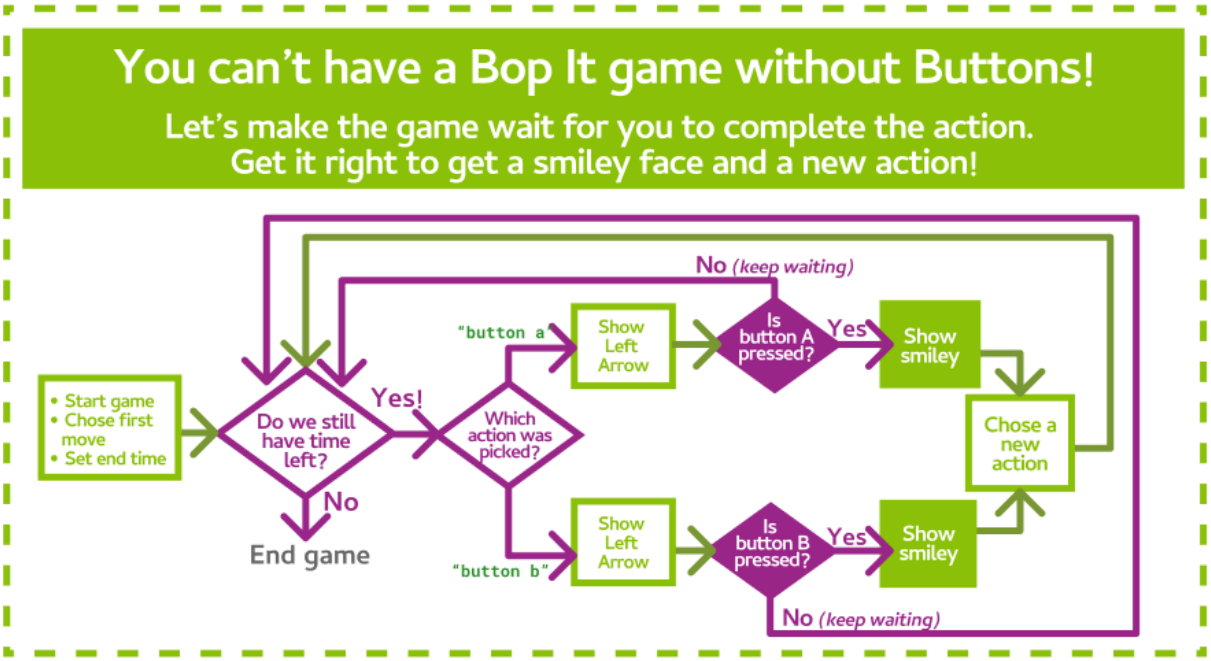
TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["button a", "button b"]

action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000

print(action)

while running_time() < end_time:
    # If the action is button a
    if action == "button a":
        display.show(Image.ARROW_W)

    # If the action was button b
    if action == "button b":
        display.show(Image.ARROW_E)

    sleep(500)
    action = random.choice(actions)
```

Micro:Bit Buttons

Part 5: Button Presses



Task 5.1: If Button A is pressed

If the action is "button_a", then we want to check whether "button_a" has been pressed.

1. Go to where you show the image for when the action is `"button_a"` and create a new line, after you display the arrows.

2. Create another `if` statement on the new line.
   Make sure this line is indented inside your existing `if` statement.

3. Make this new `if` statement to check whether `button_a.is_pressed()`.

4. Indented inside the new `if` statement, display a smiley face to celebrate!

Hint - Showing a Happy Face

The image for a smiley face is: `Image.HAPPY`

Hint - errors with is_pressed

Look at the end of this line of code, **notice the brackets at the end**:
`button_a.is_pressed()`

Make sure you include the brackets!

The brackets make it so we ***call*** the function and check if the button is pressed!

Task 5.2: Else, when Button A is not pressed

When button_a has not been pressed, we should continue the game.

1. Add an `else` statement for if the button is not being pressed.
2. Inside that `else` statement, add `continue`.

Hint

Your `if-else` statement should have a structure similar to this example:

```
if raining == True:
  print ('oh no!')
else:
  print ('Yay!')
```

Don't forget that indentation is important!

Task 5.3: Button B Pressed?

Now it's button_b's turn!

1. Inside the `if` statement that checks to see if the action is `button_b`, add an `if` statement that checks to see if `button_b.is_pressed()`.
2. Add an `else` to the `if` statement, that has a `continue`.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to Part 6:

- ☐ When the action is `"button a"` and you press `button_a`, a smiley face is displayed.
- ☐ When the action is `"button b"` and you press `button_b`, a smiley face is displayed.
- ☐ Your game waits on the same move until you press the correct button.

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

actions = ["button a", "button b"]

action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000

print(action)

while running_time() < end_time:
      # If the action is button a
      if action == "button a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
        else:
          continue

      # If the action was button b
      if action == "button b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
          display.show(Image.HAPPY)
        else:
          continue

      sleep(500)
      action = random.choice(actions)
```

Part 6: Scoring

Task 6.1: Let's get this scoring party started!

Create a variable to keep track of the score.

1. Create a new line in your code before the `while` loop is.
2. Here, create a variable called `score` and set it to `0`.

Task 6.2: Get those points!

Every time the correct move is made, add 1 to the score.

1. Go to your `if` statement where you check if **button A** was pressed.
2. Create a new line after you show the smiley face.
3. On the new line, add `1` to the score variable.
4. Repeat for the other action.

Hint - Keeping Count!

When we want to add to an existing variable it looks like this example:

```
num_apples = 5
```

```
num_apples = num_apples + 1
```

Task 6.3: How did you do?

Now we need to tell the player how well they did!

1. Got to the end of the program, after the `while` loop finishes.
2. Convert the final score to a string and then make it `scroll` across the display.

Hint - String theory!

To scroll a number on the screen we need to convert it to a string.
We can use `str` to convert to a string inside our scroll, like this:

```
fave_num = 317
display.scroll(str(fave_num))
```

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the Extensions:

☐ You have a score variable that is set to 0 at the start of the program

☐ At the end of the game, the score scrolls across the display

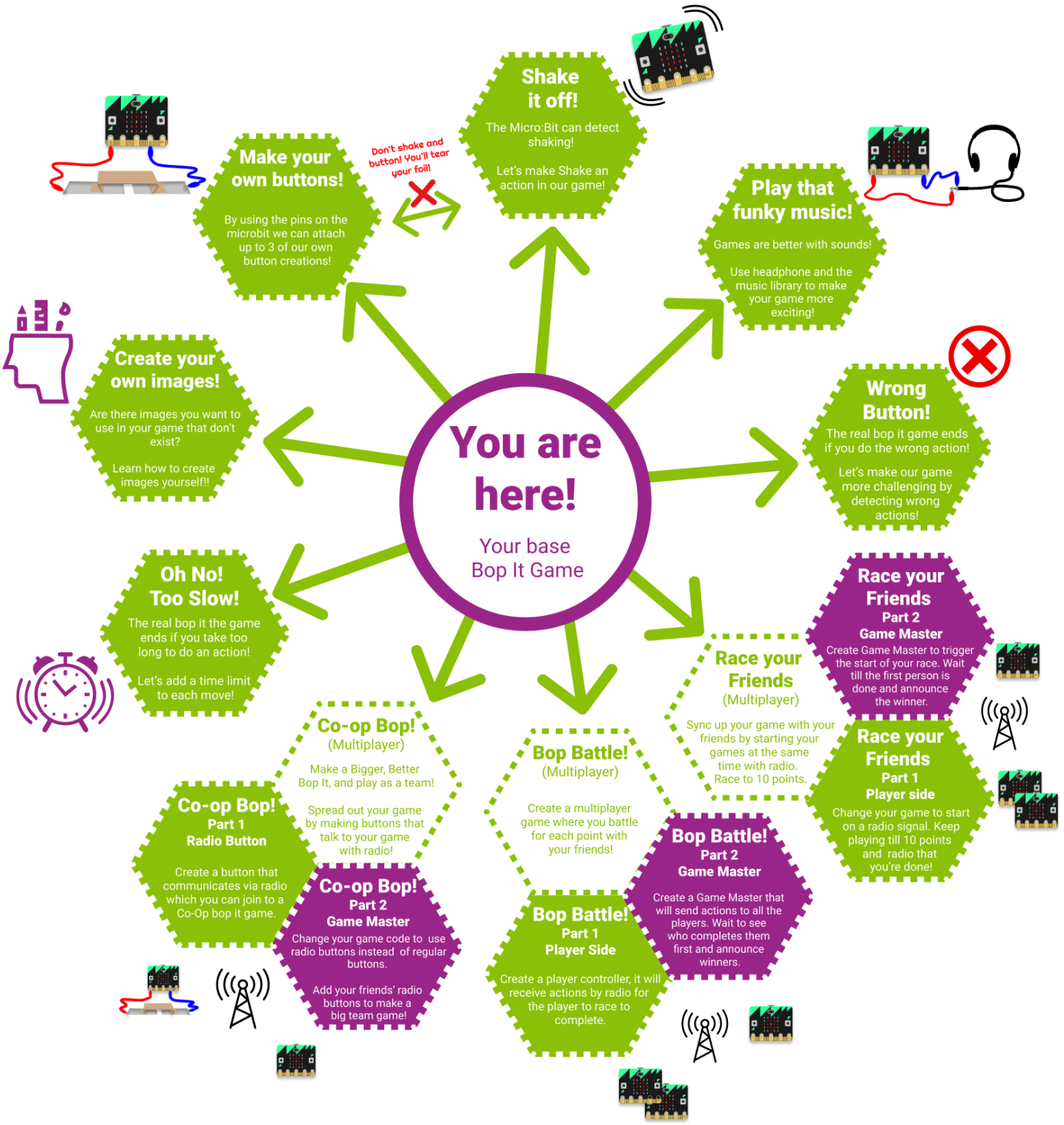
☐ You have made sure that the score counts to the right number

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

score = 0
actions = ["button a", "button b"]
action = random.choice(actions)

start_time = running_time()
end_time = running_time() + 10000
print(action)

while running_time() < end_time:
      # If the action is button a
      if action == "button a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        else:
          continue

      # If the action was button b
      if action == "button b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        else:
          continue

      sleep(500)
      action = random.choice(actions)
display.scroll(str(score))
```

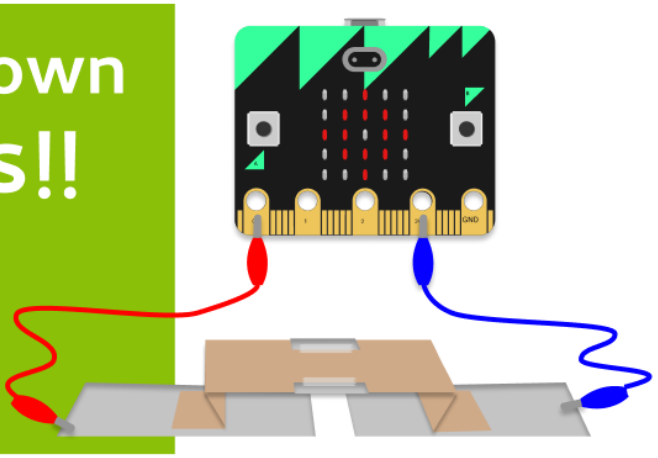
Extension Map



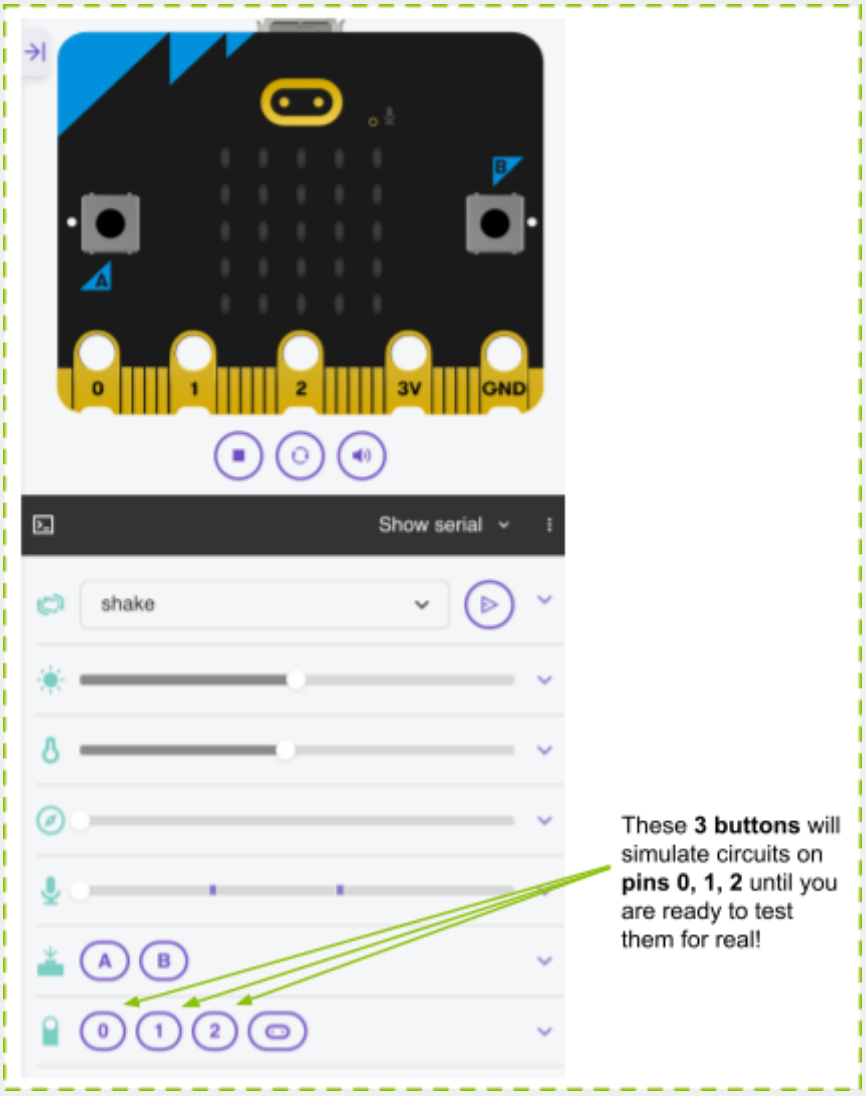
Extension

Make your own buttons!!

By using the pins on the microbit we can attach up to 3 of our own button creations!



Task 1.0: Getting ready for pins



Task 1.1: Get the pins ready

We need to start by resetting the pins so they are ready to read

1. Go to your code and add a new line after the import section.
2. Add this line of code to prepare pin 0 by resetting it.
   `pin0.read_digital()`
3. Repeat for pin1 (and pin2 if you want an extra button!)

Task 1.2: Goodbye microbit buttons, hello my buttons!

We'll edit our code to use handmade buttons instead of microbit buttons.
You can copy add to this later to use both the microbit and handmade buttons.

1. Go to the line where you check if Button A is pressed.
2. We want to check if there is current in the circuit on pin0 (instead of checking if the button is pressed). **Replace** `button_a.is_pressed()` with `pin0.read_digital()`
3. Repeat by replacing **Replace** `button_b.is_pressed()` with `pin1.read_digital()`
4. Run your code in Grok and test it out using the first pin button!

Task 1.3: Build a button!

1. Pick up a **Build a Button** cheat sheet!
2. Learn how to make a basic button and connect it to your Micro:Bit to use your code in real life!
3. Come up with your own ideas for make circuits! We've got a lot of different things to craft fun buttons like rubber bands, popsicle sticks and more!

★ Bonus 1.4: Want more actions?! ★

★ Use third pin! ★

Create another action and a button on pin2

★ Use the Micro:Bit buttons again! ★

With 2 buttons and 3 pins, you could have up to 5 actions!
So *add back in your original Micro:Bit button code, but make some changes.*
Make sure you have different action names and pictures for each button/pin.

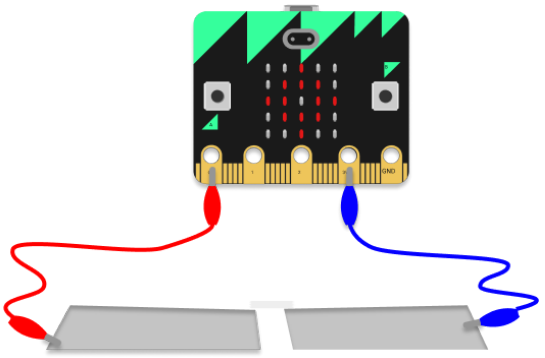
☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You made buttons/contraptions that complete circuits for you game
- ☐ Your game completes actions based on buttons connected to pins

Build a Button

Build the broken circuit

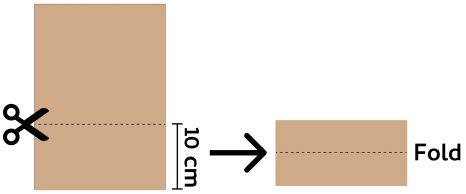


1. Get 2 alligator clips, connect one to **Pin 0** and the **3V Pin**.

2. **Connect** the other ends of the alligator clips to 2 pieces of aluminum foil.

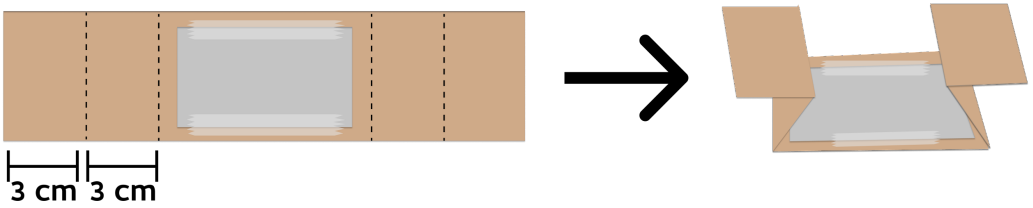
3. **Stick** those to the table so there is a gap in the middle.

Build the button

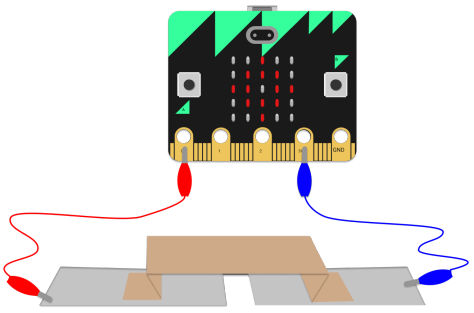


4. **Get** an A4 piece of paper.
   Cut a 10cm wide strip.
   Fold it in half long ways, for added strength.

5. **Take** the folded strip.
   Stick aluminium foil to the center.
   Crease the paper to make the button shape.



Add the button



6. **Place** the button on the broken circuit.
   Align it so that when you squish the button the aluminum patch closes the circuit.
   Stick down the button

7. **You're ready to try your code on the Micro:Bit!**

What next?

Now that you've made one button think about other cool circuits you can make!
Can you make a twist, pull, flick or spin like the original Bop It game you could buy?
What other ideas can you create?

Extension: Oh No! Too Slow!!

The real bop it the game ends if you take too long to do an action!

Let's ad a time limit to each move!

Task 1.1: Start the timer!

First, we need to prepare the timer.

1. At the start of your program, make a new variable called `turn_length` and set it to `1000` (1 second). This is how long a turn will be.

2. In your program, after we choose the first action, create a new variable called `turn_start` and set it to the `running_time()`.

3. At the end of the loop is where we set the next action, when they move on to the next action we want to restart the timer, so after we pick the action set the `turn_start` to `running_time()` again.

Task 1.2: Oh No, Too Slow!

Now, the turn needs to end if the correct action isn't completed in time.

1. Create a new `if` statement inside the `while` loop but before we check all of the actions.

2. To find out if we have run out of time, we want to see if the current `running_time()` *minus* the `turn_start` is greater than the `turn_length`.

3. If we have run out of time in our turn, `display` a sad face for one second.

4. Now that the turn is over we need to reset the `turn_start` and keep going. Set the `turn_start` to `running_time()` and add a `continue`.

Task 1.3: Speeding up each turn

To make the game get harder as it goes, make the turn length shorter and shorter.

1. After we reset the `turn_start` at the end of the loop, minus `100` from the `turn_length`.

2. Play around with how much you minus from the `turn_length` and find a number that you're happy with!

Task 1.4: If or elif?

Right now, we have a lot of if statements. Let's clean them up.

1. You can change the `if` statements (where you check which action is chosen) to `elif` statements.

This is just a little neater. It means **only one** of these options can be chosen.

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You have 2 new variables: `turn_length` and `start_turn`.
- ☐ You have a new if statement that checks if it has been too long since the turn started.
- ☐ You have changed all the action `if` statements to `elif` .
- ☐ You have made the game speed up as it goes.

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
```

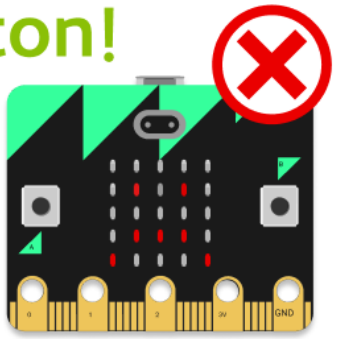
```
display.show(Image.TARGET)
sleep(1000)
display.clear()

#setting up variables
score = 0
actions = ["button a", "button b"]
action = random.choice(actions)

turn_length = 1000
turn_start = running_time()
start_time = running_time()
end_time = running_time() + 10000
#print(action)

while running_time() < end_time:

# Checking if ran out of time
    if running_time() - turn_start > turn_length:
        display.show(Image.SAD)
        sleep(1000)
        turn_start = running_time()
        continue

# If the action is button a
    elif action == "button a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        else:
            continue

# If the action was button b
    elif action == "button b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
            display.show(Image.HAPPY)
            score = score + 1
        else:
            continue

    sleep(500)
    action = random.choice(actions)
    turn_start = running_time()
```

```
    turn_length = turn_length - 100

display.scroll(str(score))
```

Extension: Wrong Button!

The real bop it game ends if you do the wrong action!

Let's make our game more challenging by detecting incorrect actions!



Task 1.1: That's not Button A!

In our code, after we check what action was chosen, we check if the player correctly chose that action or not. Now we want to do something if the player chooses *wrongly*.

Let's start by changing the `"button_a"` action.

1. Add an `elif` statement right after the `if` statement that checks whether we have pressed `button_a`.
2. Make the new `elif` statement check whether we have pressed `button_b`.
3. In the new `elif` statement, add a `break`.

   `break` will end the game by exiting the `while` loop.

Hint - Break or continue?

`break` is similar (but different) to `continue`, it will skip over the rest of the code in the loop, and exit the loop.

`continue` will skip over the rest of the code in the loop, and start again from the beginning of the loop.

Task 1.2: Do it again!

Now we need to do the same thing for `button_b`, and any other actions!

1. Complete `Task 9.1` for each of the different `actions` your program has.

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

☐ If you choose the wrong action the game ends.

☐ You have tried your game and made sure that if you choose the right action, it still works.

TUTOR TIPS

The code should look like this (no bonuses):

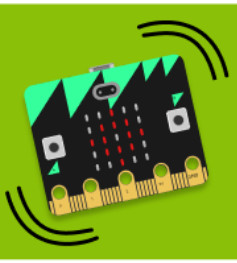
```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

#setting up variables
score = 0
actions = ["button a", "button b"]
action = random.choice(actions)

turn_length = 1000
turn_start = running_time()
start_time = running_time()
end_time = running_time() + 10000
#print(action)

while running_time() < end_time:

# Checking if ran out of time
    if running_time() - turn_start > turn_length:
        display.show(Image.SAD)
        sleep(1000)
        turn_start = running_time()
        continue

# If the action is button a
    elif action == "button a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        elif button_b.is_pressed():
            break
        else:
```

```
            continue

# If the action was button b
    elif action == "button b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
            display.show(Image.HAPPY)
            score = score + 1
        elif button_a.is_pressed():
            break
        else:
            continue

    sleep(500)
    action = random.choice(actions)
    turn_start = running_time()
    turn_length = turn_length - 100

display.scroll(str(score))
```

Extension: Shake it off

The Micro:Bit can detect shaking!

Let's make Shake an action in our game!



Task 1.1: A new action!

Let's set up our new action!

1. Add `"shake"` to the list of `actions` at the start of our program.

Task 1.2: Let's check to make sure

Tell the player to make the shake action!

1. Add a new `if` statement to check if the chosen action is our new action (you can copy one of the `button_a` or `button_b` `if` statements)

2. Pick a new image that will tell the player to do the new action (like an up or down arrow).

Task 1.3: Shake it!

Now check to see if they shook it!

1. Add an `if` statement to check and see if the Micro:Bit was shaken!

2. If it was, remember to `show` a happy face and to increase the `score` by `1`.

3. Remember to add `else: continue` after your `if` statement to make sure you stay on this action until the player gets it right!

4. You should also check if the player chose the *wrong* action. (They should lose the game.)

Hint - Shake Gesture

You can check to see if the Micro:Bit was shaken using the following code:

```
if accelerometer.was_gesture("shake"):
    # do something
```

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You have added “shake” to the list of actions at the start of your code.
- ☐ When “shake” is picked a new image appears on the display.
- ☐ When the new action is picked and you do the *right* action, a happy face appears and you get a point added to the score.
- ☐ When the new action is picked and you do the *wrong* action, you lose the game.

TUTOR TIPS

The code should look like this (no bonuses):

```
# <the student's name>
from microbit import *
import random

#starting the game with an image
display.show(Image.TARGET)
sleep(1000)
display.clear()

#setting up variables
score = 0
actions = ["button a", "button b", "shake"]
action = random.choice(actions)

turn_length = 1000
turn_start = running_time()
start_time = running_time()
end_time = running_time() + 10000
#print(action)
```

```
while running_time() < end_time:

# Checking if ran out of time
    if running_time() - turn_start > turn_length:
        display.show(Image.SAD)
        sleep(1000)
        turn_start = running_time()
        continue

# If the action is button a
    elif action == "button a":
        display.show(Image.ARROW_W)
        if button_a.is_pressed():
          display.show(Image.HAPPY)
          score = score + 1
        elif button_b.is_pressed():
            break
        else:
            continue

# If the action was button b
    elif action == "button b":
        display.show(Image.ARROW_E)
        if button_b.is_pressed():
            display.show(Image.HAPPY)
            score = score + 1
        elif button_a.is_pressed():
            break
        else:
            continue

# If the action was shake
    elif action == "shake":
        display.show(Image.DUCK)
        if accelerometer.was_gesture("shake"):
            display.show(Image.HAPPY)
            score = score + 1
        elif button_a.is_pressed() or button_b.is_pressed():
            break
        else:
            continue

    sleep(500)
    action = random.choice(actions)
    turn_start = running_time()
```

```
    turn_length = turn_length - 100

display.scroll(str(score))
```

★ BONUS 1.4: More actions!

Using the steps above, you can keep adding more actions!

Find some inspiration for different actions at
https://bbcmicrobitmicropython.readthedocs.io/en/latest/tutorials/gestures.html

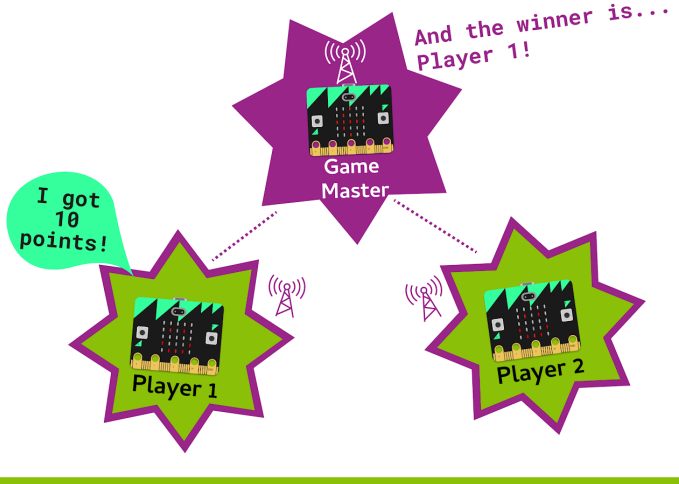
Extension: Race Your Friends

Want to play Bop It with friends?

We'll race our friend to 10 points!

The game master will get our games to start at the same time and wait to hear who gets 10 points first.

Our games will run just like before! But will talk to the Game Master at the start and finish!



Learning about Radio

We'll need to know how to use the radio for this extension. Here's some commands:

Action	Code
Set the channel, we set it to 6.	`radio.config(group=6)`
Turn the radio on	`radio.on()`
Send a message, we sent "bop"	`radio.send("bop")`
Receive a message, check if it matches "bop"	`if radio.receive() == "bop":`

Part 1

Give your game radio

Make your game start when the Game Master says to via radio. Radio back when you get 10 points!

Each players Microbit will still run its own game and generate it's own moves.

Part 2

Make a Game Master

Create a Game Master microbit. It will use radio to tell the players when to start their games.

It will wait until it hears back from the first player to get 10 points to announce the winner.

Part 1: Give your game radio

We'll make our Bop It multiplayer compatible with radio!

It will:

- wait for a start game message from the Game Master.
- Radio back to the Game Master if it gets 10 points.
- Still chose all it's own moves and keep track of it's own points.

Task 1.1: Find some friends!

For this section you will need to work in **groups!** So make a group of **at least two people!**

One person will need to be the **game master (skip to Part 2 code)** and the other people will be the **players.**

The players will not be able to play until the game master tells them to!

Hint - Downloading Code

When you plug in a Micro:Bit to the computer, and download the code, the Micro:Bit can then be unplugged and will still remember that code

(and can be run as long as it has a **battery pack**)!

Task 1.2: Configure the Radio

We need to configure the radio to start off with

1. At the top of your program, `import radio`.
2. After the target image is displayed, turn the radio on with `radio.on()`
3. Then configure the radio's group with `radio.config(group=100)`. Your room coordinator will tell you what group number to use.

TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups:

Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0 to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these “overflow” channels as backups like this:
If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren’t using all of theirs and if you can use one.

Task 1.3: Ready, Set, Go!

Make the Micro:Bit wait until it’s been told to start!

1. Before your main game `while` loop, add a new `while` loop that waits for the radio incoming message `"start"`.

2. Inside the `while` loop, add a `pass` statement.

Hint - Radio Messages

You can read the message that the radio has received with the following code:

```
incoming = radio.receive()
```

Task 1.4: Game over!

Send a message to the game master when you’ve reached 10 points!

1. Update your main game `while` loop so it only runs if the score is less than 10.

2. At the end of your code, and outside of the main game while loop, send the player’s name via the radio!

Hint - Radio Send

You can send a message using the radio with the following code:

```
radio.send("Rama won!")
```

You will need your friend to set up their game master to actually test this code!

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

☐ You have configured your radio using the channel group number the room coordinator gave you.

☐ The game doesn't start until the game master says start!

☐ When you have reached 10 points, the player's name is sent to the game master.

Part 2: Make a Game Master

We'll re-write our game so it uses radio to complete actions

It will:

- **Choose actions, keep score, and loop, just like before.**
- **Listen for radio messages that match the action it is waiting for.**

Task 2.1: Configure the Radio

We'll need to start a new file for our game master!

1. Click the **Project** button to the left, and then click **Create file**.
2. Name your new "game_master" file.
3. At the top of your file, `import` the `microbit` and `radio` modules.
4. Turn the radio on with `radio.on()`.
5. Configure the radio to use the channel that the room coordinator gave you.

TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups: Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0 to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these "overflow" channels as backups like this:
If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren't using all of theirs and if you can use one.

Task 2.2: Set up the game

Let's set up the variables we need!

1. Create a variable called **`winner`**, and set it to `None`.
2. Constantly scroll a message that says `"PRESS A to Start"`.
3. Make sure your message has a wait of `False`.

Hint - Scrolling messages

To make a message scroll constantly, and have a wait of false, you can use the following code:

```
display.scroll("Welcome to GPN", wait=False, loop=True)
```

Task 2.3: Start the game!

Send a message to the players when you're ready to start the game!

1. At the end of your code, create a while loop that keeps running `while` the winner is equal to `None`.
2. Inside the `while` loop, add an `if` statement that checks to see if **`button_a`** was pressed.
3. If **`button_a`** was pressed, send a message using the radio with the message `"start"`.
4. Outside of the `if` statement, but still inside the `while` loop, set the value of **`winner`** to be the message the radio receives.

Task 2.4: Configure the Radio

Display the winner!

1. At the end of your code, **`scroll`** who the **`winner`** was continuously!

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this extension:

- ☐ You have configured your radio using the channel number the room coordinator gave you.
- ☐ Your radio sends a message of “start” when button_a is pressed.
- ☐ When there is a winner, their name is displayed!

★ BONUS 2.5: Images!

Our game master doesn’t really do anything when it’s waiting for a winner. Let’s make it display some images!

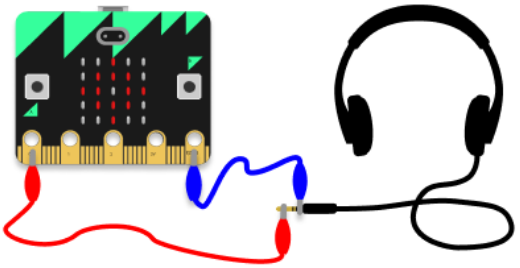
1. In your code, just before when the **`winner`** variable is created, create a new list called **`images`**. Add as many images as you want in this, such as **`Image.CHESSBOARD`** and **`Image.CHESSBOARD.invert()`**.
2. Inside your **`if`** statement before the start radio message is sent, start displaying the images on repeat by using the following code:

```
display.show(images, wait=False, loop=True)
```

Extension: Play that funky music!

Games are better with sounds!

Use headphone and the music library to make your game more exciting!



Task 1.1: Set up the headphones!

First, we'll need to connect our headphones like in the picture above:

1. Connect one alligator clip to the `GND` pin of the `Micro:Bit`. Connect the other end to the `base` of your headphone jack.
2. Connect another alligator clip to `pin 0` of the `Micro:Bit`. Connect the other end to the `tip` of your headphone jack.
3. At the top of your code, `import music`

Task 1.2: Play a sound!

Let's play the A tone when you need to press button A!

1. Inside the `if` statement that checks to see if `"button a"` was selected, play the tone `"A"` for `two` beats.
2. Make sure that you set `wait` to `False` so the game keeps running while the music is playing!

Hint - Playing sounds

To play a G tone for 5 beats, you can use the following code:

```
music.play("G:5")
```

Task 1.3: Play more sounds!

Let's make the other actions play sounds too!

1. Inside the `if` statement that checks to see if `"button b"` was selected, play the tone `"B"` for `two` beats. Make sure that `wait` is set to `False`.
2. Do the same thing for any other actions you have, making sure that they each have a unique tone!

Task 1.4: Let's listen

Test your code!

1. Can you hear all the different sounds? Make sure you test every action!

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ When button a is the selected action, the A tone plays for 2 beats.
- ☐ When button b is the selected action, the B tone plays for 2 beats.
- ☐ For all the other actions you have, a unique sound is played for 2 beats.
- ☐ You can hear the sounds through your headphones!

★ BONUS 12.5: Make it talk ★

What if our Bop It could talk!

Speech is a lot like music, but we can tell it to say words!

Once you import the speech library you can start telling it things to say:

```
import speech
speech.say("BOP!")
```

Challenge: Can you make it announce the move each turn?

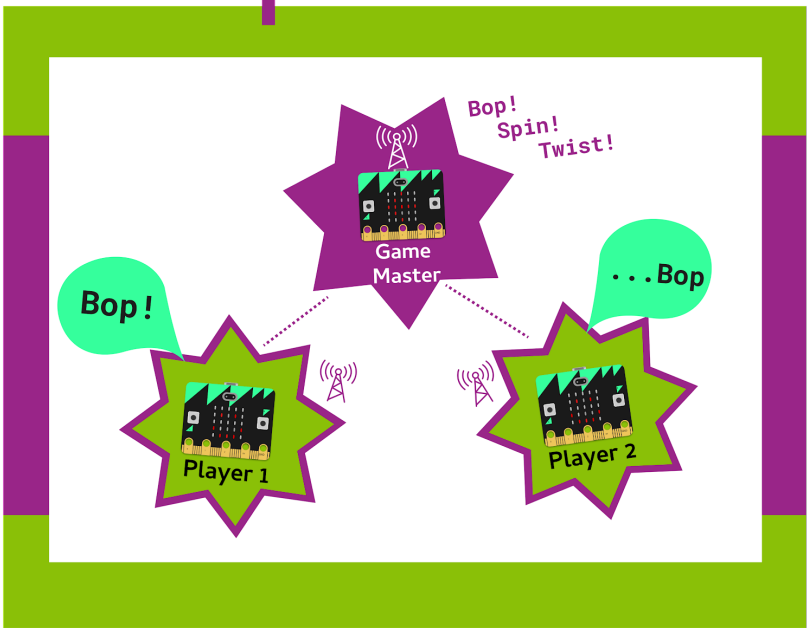
Extension: Bop Battle

Battle your friends for every Bop It point!

Be the first to do the action to get the point!

The Game Master will decide the moves and tell them to each player via radio.

Race to be the first to do the action and radio message the Game Master to score!



Learning about Radio

We'll need to know how to use the radio for this extension. Here's some commands

Action	Code
Set the channel, we set it to 6.	`radio.config(channel=6)`
Turn the radio on	`radio.on()`
Send a message, we sent "bop"	`radio.send("bop")`
Receive a message, check if it matches "bop"	`if radio.receive() == "bop":`

Part 1

Getting actions via radio

Create a player Microbit to compete in the multiplayer game!

It receives actions from the Game Master via radio and sends your name back to score when it's completed actions!

Part 2

All powerful Game Master

Create a Game Master microbit. It will use radio to tell the players an action to complete.

It will wait until it hears back from the first player to get complete the action to announce the winner!

Part 1: Getting actions via radio

Make our game multiplayer by adding radio to get actions to compete over!

It will:

- **wait to receive an action message from the Game Master.**
- **Radio back to the Game Master to complete the action and win the point!**

Task 1.1: Find some friends!

For this section you will need to work in **groups!** So make a group of **at least two people!**

One person will need to be the **game master (skip to Part 2 code)** and the other people will be the **players.**

The players will not be able to play until the game master tells them to!

Task 1.2: Configure the Radio

We need to configure the radio to start off with

1. At the top of your program, `import radio`.
2. After the target image is displayed, turn the radio on with `radio.on()`
3. Then configure the radio's group with `radio.config(group=100)`. Your room coordinator will tell you what group number to use.

TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups: Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0 to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these "overflow" channels as backups like this:

If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren't using all of theirs and if you can use one.

Task 1.3: Ready, Set, Go!

Now, we're going to receive the action from the game master!

1. Find where you first set the `action` randomly. It should be above your `while` loop. `Comment` out this line!
2. Inside the game loop, change the `action` variable so it has the value of the incoming radio message.

Hint - Receiving messages

You can receive messages via radio using:

```
incoming = radio.receive()
```

Task 1.4: Run only once!

We're only competing for each individual point. So when we have a score of 1, the game should end.

Update the `while` loop so it only runs while the `score` is equal to `0`.

Task 1.5: Send the winner!

Now, tell the game master you've won!

Outside the while loop, at the end of the program, send a message to the game master saying your name!

Hint - Sending messages

You can send messages via radio using:

```
radio.send("My message")
```

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You have the radio configured
- ☐ You receive the action from the game master
- ☐ You send your name to the game master when you have won a point

★ CHALLENGE 1.5: More than one move ★

We've shown you how to do this for 1 move! Now it's your turn to do the rest!

Figure out how to make this work so the game keeps playing for many turns.

Feel free to change the whole structure of the code, there are so many ways to create your own solution!

Part 2: All powerful Game Master

Create a Game Master where the moves for the multiplayer game are decided!

It will:

- **Let someone chose a move the player must do**
- **Radio message the players the move**
- **Wait to here who completes the move first to announce the winner**

Task 2.1: Configure the Radio

We'll need to start a new file for our game master!

1. Click the **Project** button to the left, and then click **Create file**.
2. Name your new "game_master" file.
3. At the top of your file, `import` the **`microbit`** and **`radio`** modules.
4. Turn the radio on with **`radio.on()`**.
5. Configure the radio to use the channel that the room coordinator gave you.

TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups: Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0 to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these "overflow" channels as backups like this:
If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren't using all of theirs and if you can use one.

Task 2.2: Ready, Set, Go!

Let's set up the variables we need!

1. Create a variable called `winner`, and set it to `None`.
2. Constantly scroll a message that says `"CHOOSE ACTION TO START"`.
3. Make sure your message has a wait of `False`.

Task 13.2.3: Game loop!

Now, let's set up the game loop!

1. Create a `while` loop that continually loops until `winner` is not equal to `None`.
2. Inside the `while` loop, set `winner` to be the incoming radio message.
3. Outside the `while` loop, at the end of your code, `scroll` who won the game continuously!

Task 13.2.4: Choose your move!

Now, we need to choose our move and send it to the players!

1. Inside the `while` loop, check to see `if` `button_a` was pressed.
2. If it was, show a left arrow, and send a radio message saying `"button a"`.
3. Create another if statement that checks `if` `button_b` was pressed.
4. If it was, show a right arrow and send a radio message saying `"button b"`.

Task 13.2.5: Testing time!

Try playing a game with your game master!

1. Test your Game Master! Which player won?

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

☐ You have configured your radio using the channel number the room coordinator gave you.

- [ ] Your radio sends a message of “button a” when button_a was pressed.
- [ ] Your radio sends a message of “button b” when button_b was pressed.
- [ ] When there is a winner, their name is displayed!

★ CHALLENGE 2.6: More than one move ★

We’ve shown you how to do this for 1 move! Now it’s your turn to do the rest!

Figure out how to make this work so the game keeps playing for many turns.

Feel free to change the whole structure of the code, there are so many ways to create your own solution!

Extension: Co-op Bop!

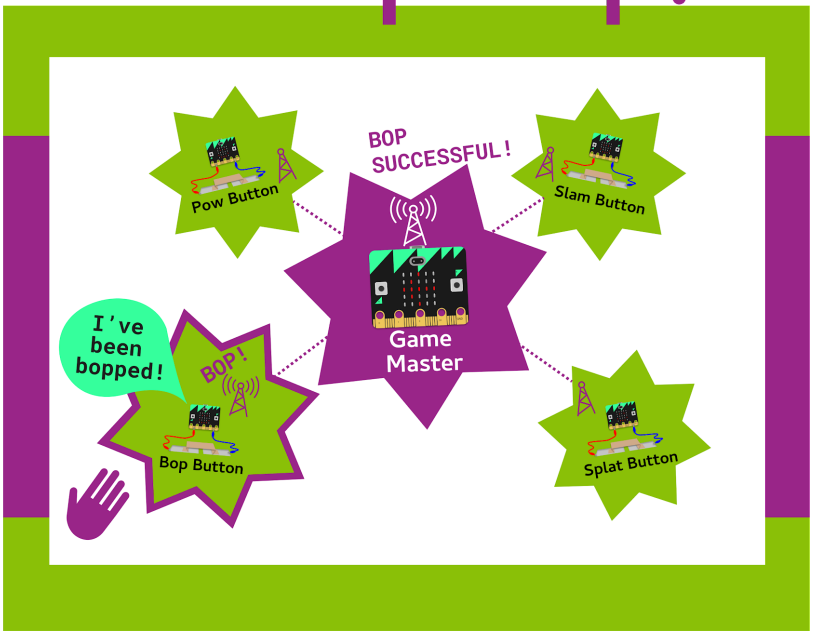
Make a Bigger, Better Bop It, and play as a team!

Spread out your game by making buttons that talk to your game with radio!

This extension has 2 parts.

- Making radio buttons.
- Changing your game to listen for the radio buttons.

You can work in groups!



Learning about Radio

We'll need to know how to use the radio for this extension. Here's some commands

Action	Code
Set the channel, we set it to 6.	`radio.config(channel=6)`
Turn the radio on	`radio.on()`
Send a message, we sent "bop"	`radio.send("bop")`
Receive a message, check if it matches 'bop'	`if radio.receive() == "bop":`

Part 1
Making Radio Buttons

A new file that sends a radio message every time you press a button.

Use the Micro:Bit buttons or craft your own!

Part 2
Listening for Radio Buttons

Adapt your game to use radio messages to complete actions.

You can connect lots of buttons, so work as a team to make more.

Part 1: Making Radio Buttons

We'll make a new program for our radio buttons.

It will:

- Always show a picture so we can identify which button it is.
- Send a radio message when we press the button.

Task 1.1: Configure the Radio

We'll need to start a new file for our game master!

1. Click the **Project** button to the left, and then click **Create file**.
2. Name your new "race_friends" file.
3. At the top of your file, `import` the `microbit` and `radio` modules.
4. Turn the radio on with `radio.on()`.
5. Then configure the radio's channel with `radio.config(channel=100)`. Your room coordinator will tell you what number to use.

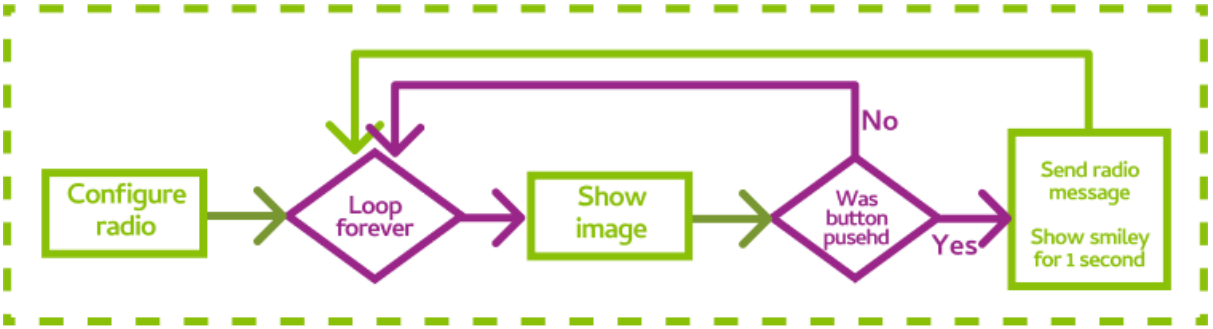
TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups: Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0 to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these "overflow" channels as backups like this:
If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren't using all of theirs and if you can use one.

Next we'll make a loop that sends a message every time the button is pressed!



Task 1.2: Loopy for radio

1. **Create an infinite loop** after you have set up the radio

2. Inside the loop make the Micro:Bit display an image that will be it's identifier.

Task 1.3: Checking presses & sending messages

1. **Inside the loop create an if statement that checks for an action.**

   Actions: *We have the Micro:Bit buttons, shake and other gestures, and buttons you craft yourself! Remember to use the correct code, e.g.* `is_pressed()` or `read_digital()`.

2. **In the if statement, send your action to the game master**: *Remember to use your own action name:* `radio.send("bop")`

Task 1.4: Smiley faces for pressing!

1. After you send a message **show a smiley face** for 1 second

2. **Run your code** in the Grok simulator to see it communicate to the other MicroBit!

Joining a game

Now you have a working button you can join a game. You'll need to make sure your button's message is included in the Game Master code.

You can write or radio game code in part 2, or join a friend's or tutor's game!

In the game master code you'll need to:

- Include your button name in the list of actions
- Add an if statement that displays your button's image when it is chosen.
  Make sure your button name and image are unique for the game!

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You have configured your radio with the channel your tutor gave you
- ☐ When you press your button it sends a radio message to Game Master.
- ☐ You show a smiley face every time you press your button
- ☐ You added your button to an existing Co-op game!

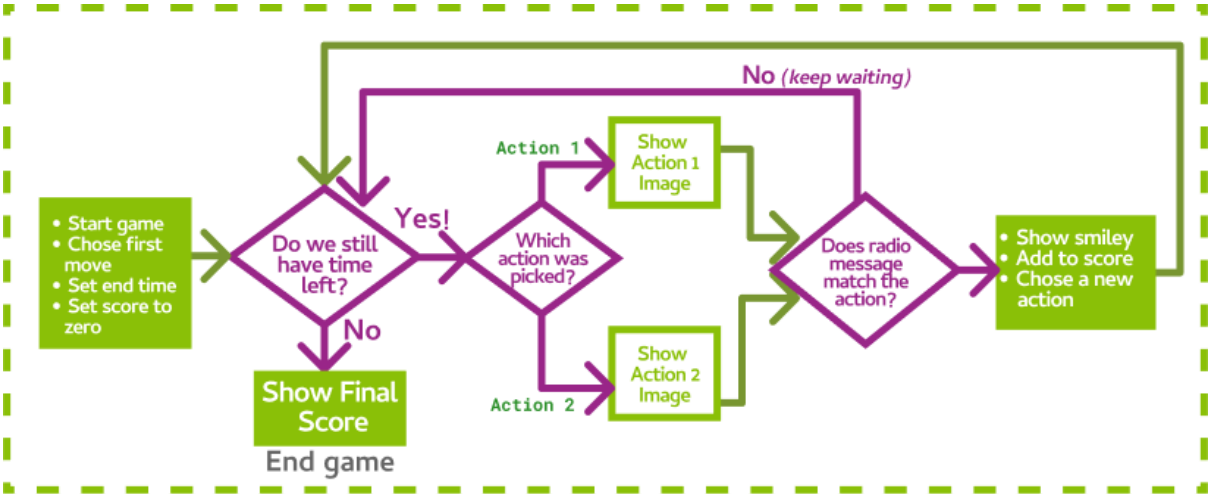
Part 2: Listening for Radio Buttons

We'll re-write our game so it uses radio to complete actions

It will:

- **Choose actions, keep score, and loop, just like before.**
- **Listen for radio messages that match the action it is waiting for.**

This new code will look a lot like your original game code!



Task 2.1: Configure the Radio

We'll need to start a new file for our game master!

1. Click the **Project** button to the left, and then click **Create file**.
2. Name your new "game_master" file.
3. At the top of your file, `import` the `microbit`, `random` and `radio` modules.
4. Turn the radio on with `radio.on()`.
5. Configure the radio to use the channel that the room coordinator gave you.

TUTOR TIPS

The group number should be unique for the whole workshop, it can be an integer value from 0 to 83. We recommend using this formula to assign channels to student groups: Channel = XY where X is the room number (e.g. first beginners room is room 0, then second beginner is room 1, intermediate is room 2 etc.), and Y is a group number from 0

to 9. If you have more than 10 groups in your room and cannot combine the groups, you may use the backup formula:

If there are less than 8 rooms at the workshop then some channels will not be used by the above formula, we can use these “overflow” channels as backups like this:
If a workshop has 6 rooms, then we assume channels 0-59 will be taken, but we can use the channels 60-83 like this:
Take one of the leftover channels + your room number (e.g. if you are in room number 3 at a 6 room workshop you can use channels 63, 73 and 83. If you are in room number 5 you can use channels 65 and 75)

If you still need more channels, check with the other rooms to see if they aren’t using all of theirs and if you can use one.

Task 2.1: Setting up the game

1. **Create a list of all the actions** you want to connect to your game.
   You can add more buttons from your friends here later!
2. Show the starting image
3. Create a score variable and set it to 0
4. Like in your first game, calculate the ending_time for the game using `running_time()`
5. Chose the first random action from the list, assign it to a variable called `action`

Task 2.2: Play the game!

1. Like your first game, create a `while` loop that keeps running until the finish time.
2. Create if statements to show the actions.
   - The if statement should check what the action is
   - Inside the if statement show the image you associate with that button
   - Make sure you have an if statement for every action in the list

Task 2.2: Checking for messages

We need to check if we’ve received a message about button presses!

1. Go to the place in the code after your if statements show the image.
2. Create a new if statement to check on incoming messages
   - The if statement should check if the current action is equal to `radio.receive()`
   - If it is, add to the score, show a smiley and choose a new action randomly.
3. Add an else statement to deal with the case that it is not a match.
   - If it doesn’t match, use `continue` to loop again

Task 2.3: Game over!

1. After your loop add the code to scroll the final score. Time to give your game a go!

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

☐ You have configured your radio channel with the number the tutor gave you

☐ Your game chooses actions which can be completed by radio button presses

Extension: Create your own images!

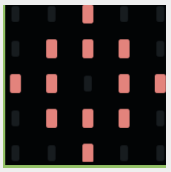
Are there images you want to use in your game that don't exist?

Learn how to create images on the Micro:Bit yourself!!

Task 1.1: How does the LED screen work?

The LED screen is actually 5 rows of 5 individual lights, and we can set them all individually!

A value of 0 is off, and a value of 9 is at its brightest. Each row is displayed and separated by a colon.

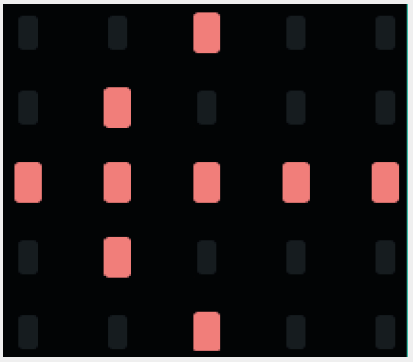


So for this target image, we can display it as

```
Image("00900:09990:99099:09990:00900")
```
instead!

Task 1.2: What's the value?

1. What's this image's value?



.....

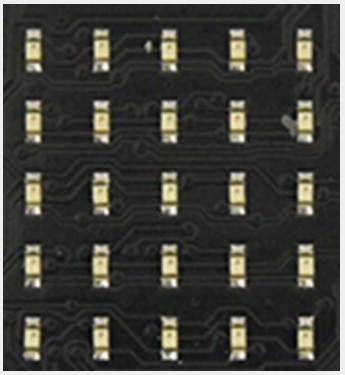
2. What about this one?

.....

Task 1.3: Create your own!

Let's create our own images now!

1. **Colour in the micro:bit with the image you want:**



2. **Write down the value of the image:**
   Remember 0 is off, and 9 is the brightest.

   _____:_____:_____:_____:_____

3. **Store the image you've created in a variable!**

Hint - Creating your own image

The following code creates an image with the first row lit up:

```
myImage = Image("99999:00000:00000:00000:00000")
```

Task 1.4: Display it!

Let's display the image now!

1. `Display` the image you created!
   Use `display.show(my_image)` like you did before. But now your variable name

☑ CHECKPOINT ☑

If you can tick all of these off you have finished this Extension:

- ☐ You have created your own image
- ☐ You have displayed the image you created