Girls' Programming Network

Extra Bop It with micro:bits!

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

This workbook and related materials were created by tutors at:

Sydney, Melbourne

Girls' Programming Network

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

Writers

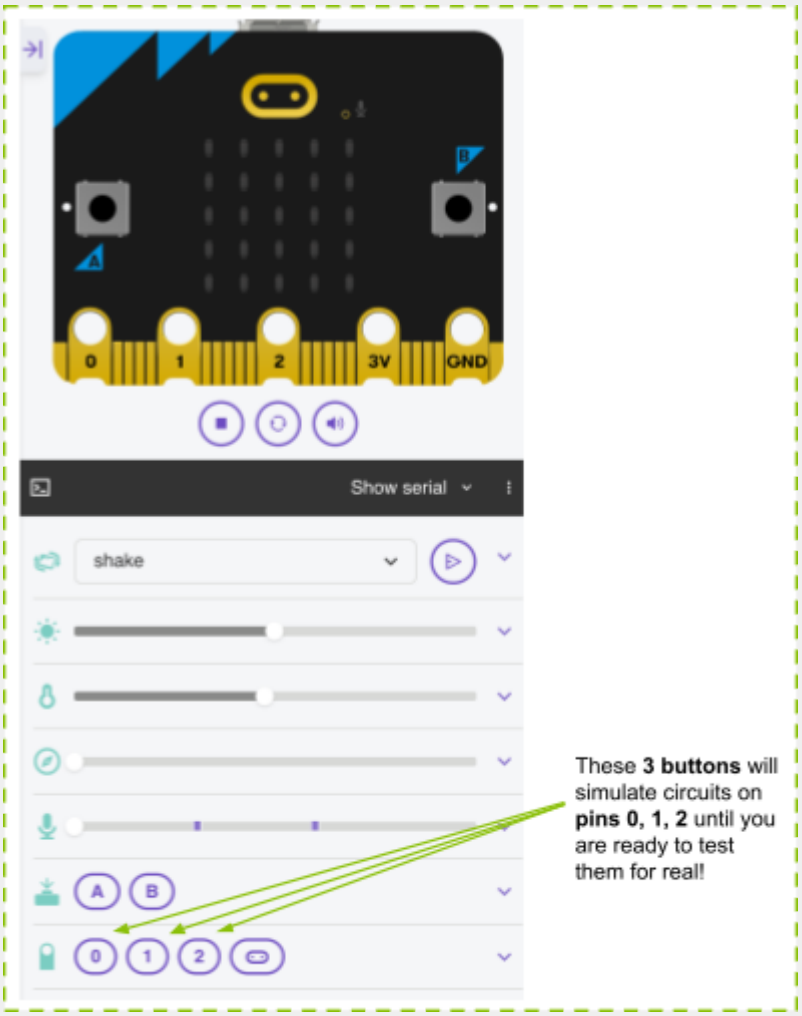
Isabella Hogan
Angela Sojan
Sophie Lee-Goh

Testers

Amanda Hogan

Extension 1: Making your own Buttons⭐

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. On a new line, after the `import` statements, prepare pin 0 for reading input with the code `pin0.read_digital()`
2. Repeat for pin 1 (and pin 2 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 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 and test it out using the first pin button!
5. We need to make sure that the player knows what pin they should be doing. Replace where you're displaying **ARROW_W** for press a and **ARROW_E** for press b with showing the numbers **0** and **1** for the corresponding pins.

Note: If you would like to test your code on the website instead of with your newly made buttons, swap out **`.read_digital()`** with **`.is_touched()`**

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 making 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 a third pin! ★

Create another action and a button on pin 2

★ Use the micro:bit buttons again! ★

With 2 buttons and 3 pins, you could have up to 5 actions!

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 ☑

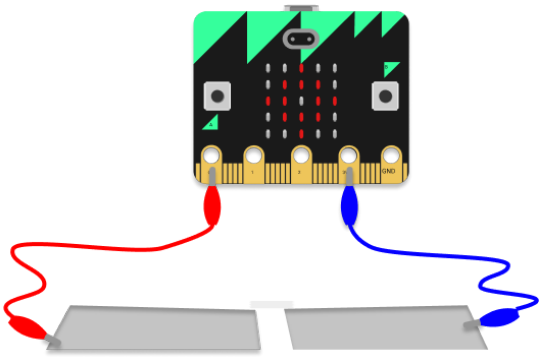
If you can tick all of these off you have finished this Extension

- ☐ Have buttons/contraptions that complete circuits for your game
- ☐ Your game completes actions based on buttons connected to pins

Build a Button

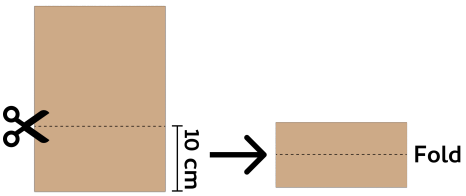
Build the broken circuit

1. Get 2 alligator clips, connect one to **Pin 0** and one to the **3V Pin**.
2. **Connect** the other ends of the alligator clips to 2 pieces of aluminium foil.
3. **Stick** those to the table so there is a gap in the middle.

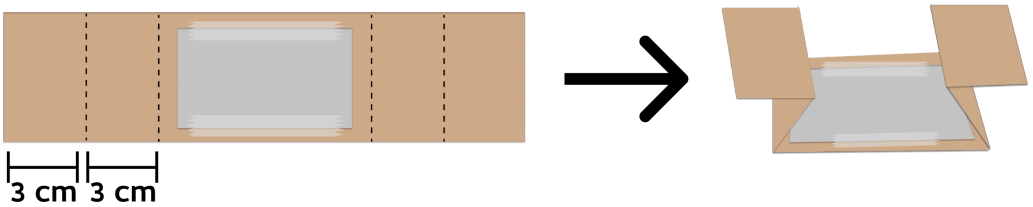


Build the button

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

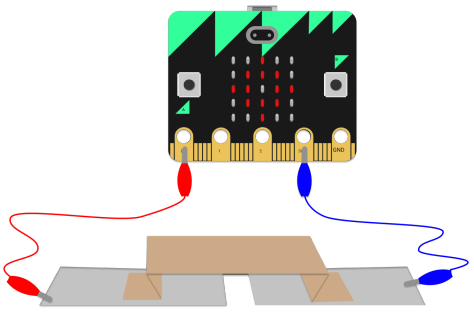


5. **Take** the folded strip.
   Stick additional aluminium foil to the centre.
   Crease the paper to make the button shape.



Add the button

6. **Place** the button on the broken circuit.
   Align it so that when you squish the button, the aluminium 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 2: Shake it Off⭐

Task 2.1: 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 2.2:

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 2.3: Shake it!

Now check to see if they shook it!

1. Add an `if` statement to check to see if the micro:bit was shaken!

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

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.

★ BONUS 2.4: 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 3: Oh No! Too Slow!!⭐

Task 3.1: 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. On a new line, after the first time we choose the **`chosen_action`**, create a new variable called **`turn_start`** and set it to the **`running_time()`**.

3. When the player moves on to the next action, we want to restart the timer. After we pick the action at the end of the `while` loop, set **`turn_start`** to be **`running_time()`** again.

Task 3.2: 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 the **`if`**s where we check 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 start a **new turn.** Set the **`turn_start`** and keep going. Set the **`turn_start`** to **`running_time()`**, choose a new random **`chosen_action`** and then add a **`continue`** to restart the loop.

Hint

When using **`display.show()`**, you can control how long an image displays.

Use the **`delay`** parameter to tell it how long to display the image for, and the **`clear`** parameter to tell it to clear the display after the delay time has passed.

```
display.show(Image.SMILE, delay=2000, clear=True)
```

Task 3.3: Speeding up each turn

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

1. After the player **successfully completes** an action and you choose a new `chosen_action` 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 3.4: 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 `chosen_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 `turn_start`.
- ☐ 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.

Extension 4: Wrong Button!⭐

Task 4.1: 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 `"press 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 4.2: Do it again!

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

1. Complete `Task 1.1` for each of the different `actions` your program has.

☑ CHECKPOINT ☑

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

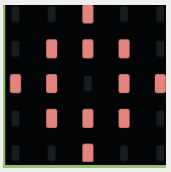
- ☐ If you choose the wrong action the game ends.
- ☐ You have tried your game and made sure that if you choose the right action, it still works.

Extension 5: Create your own Images ⭐

Task 5.1: 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 separated by a colon.



displayed and

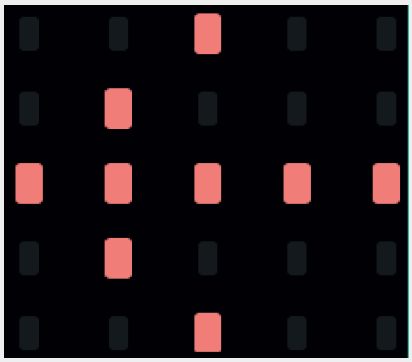
So for this target image, we can display it as

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

instead!

Task 5.2: What's the value?

1. What's this image's value?



.....

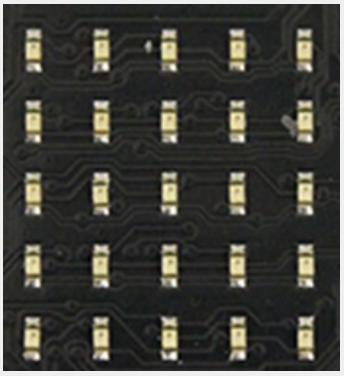
2. What about this one?

.....

Task 5.3: 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 5.4: Display it!

Let's display the image now!

1. Change the existing `display.show(Image.WHATEVER)` to something like `display.show(my_image)`, replacing the Image.WHATEVER with your custom image 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

Extension 6: Play that funky music! ⭐

Task 6.1: Set up the headphones!

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

NOTE: you can skip this step if you have a V2 micro:bit (shiny gold logo). Your micro:bit already has a speaker built-in.

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.

Task 6.2: Play a sound!

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

1. At the top of your code, `import music`.
2. Inside the `if` statement that checks to see if `"press a"` was selected, play the tone `"A"` for `two` beats.
3. 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 6.3: Play more sounds!

Let's make the other actions play sounds too!

1. Inside the `if` statement that checks to see if `"press 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 6.4: 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 `"press a"` is the selected action, the A tone plays for 2 beats.
- ☐ When `"press 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 either out loud (V2) or through your headphones (V1)!

★ BONUS 6.5: 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 7: One More Time ⭐

Task 7.1 Starting over

Let's add a restart button to our game. When the pin logo is touched, the game should restart.

On a new line directly under `while running_time() < end_time:` let's add an `if` statement.

Check `if` the `pin_logo` has been touched.

Inside this `if` statement, you will need to

- Show `Image.TARGET` for 1,000 milliseconds, then clear the display
- Reset the `score` to `0`
- Reset the `lives` to `3`
- Choose a random action from the list of `actions` and store it in `chosen_action`
- Set `start_time` to `running_time()`
- Also set `end_time` to `start_time` + 10,000 milliseconds

Task 7.2 Keep it going

We can now restart the game while we are in the middle, but what if we want to play again once the game is over?

Go to the top of your code, underneath `import random`. Let's add `while True:`, so that our game infinitely loops. You will then need to indent the rest of your code so that it is inside this while loop.

Task 7.3 Reflection (Optional)

Try moving the restart code so that it is after the `if chosen_action == "press a":` section, instead of at the top of the while loop.

Does the game still restart when you touch the logo?
If not, why do you think this happens?

Hint

Think about what the `continue` keyword does.

Hint for tutors: The game will likely stop responding to the logo because the `else: continue` skips the rest of the current loop. When continue runs, Python immediately goes back to the start of the `while` loop, so any logic that is placed below will never get reached.

☑ CHECKPOINT ☑

If you can tick all of these off you can go to the next extension:
☐ You can touch the pin logo in the middle of a game to restart it
☐ Your game starts immediately after the score is shown

Extension 8: Survival Mode ⭐

Task 8.1 Adding a lives variable

Let's make the game a little more challenging. If the player makes 3 mistakes, the game is over. To keep track of this, we will create a new variable called `lives`.

Create a variable called `lives` and set its starting value to 3.

Task 8.2 Updating lives

Now we need to reduce the player's lives when they press the wrong button.

In the `if chosen_action == "press a":` section,

1. Add an `elif` after `if button_a.is_pressed():`
2. Check whether button b has been pressed
3. Reduce `lives` by 1
4. Display a sad face, so the player knows they have made a mistake

Repeat the same idea in the `if chosen_action == "press b":` section.

Task 8.3 Game over

Finally, we need to make sure that the game ends when the player has run out of lives.

Find `while running_time() < end_time:`.

Update the condition so that the game only continues if there is still time remaining and the player has at least 1 life left.

☑ CHECKPOINT ☑

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

☐ You have a lives variable
☐ If the player presses the wrong button, they lose a life
☐ If a player runs out of lives, the game is over

Extension 9: How Did I Go⭐

Task 9.1 Highest score ever

Let's try and make the game remember our highest score, so we can see if we got better at the game!

Under the line where **`score`** is set to 0, create a new variable called **`high_score`** and set it to 0. (Not inside the 2nd while loop)

Check **`if`** the **`pin_logo`** has been touched.

Hint

Be careful not to put **`high_score = 0`** inside the while loop that checks time. It would reset every time the game runs again, so the game would forget the previous high score.

Task 9.2 What is my high score

Now we need to update the highscore. Each time the player earns a point, check if their score is higher than the current high score. If it is, update the high score.

Under the line where we add to the **`score`**, write an **`if`** statement to check if the **`score`** is more than the **`high_score`**. If it is, add 1 point to the **`high_score`**. Remember to check for both button A and button B

Now display the **`high_score`** at the end after you display the score.

BONUS: Who was playing

If we want to know who got the high score, we need to ask for the player's name. To do this, use **`input`** to get the player's name before the game starts.

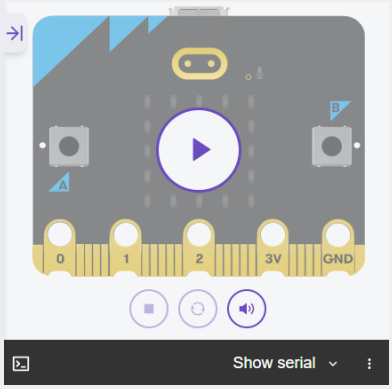
Then display the name before the score and high score is displayed at the end.

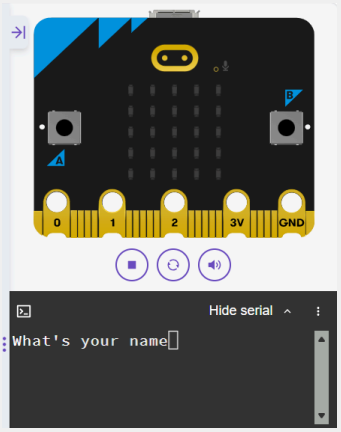
Hint

This is how we use the input function

```
player_name = input("Enter name: ")
```

When using microbits, you can input the name in the serial console on the micro:bit Python Editor website. It will look like this, click on “show serial” to expand and you can type in it.





☑ CHECKPOINT ☑

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

☐ You have made the high score variable
☐ You have verified and updated the high score
☐ You have repeated this for both buttons
☐ You have displayed the high score

Extension 10: Who's the winner⭐⭐

Task 10.1 Who was playing

We can build a leaderboard to challenge our friends!

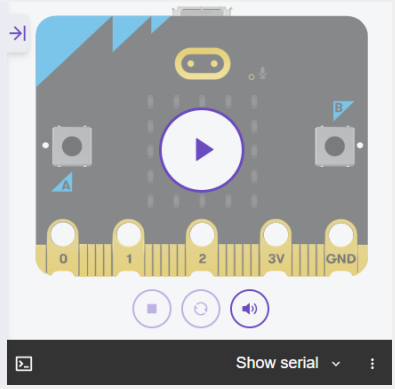
To decide who's the winner, we need to ask for the player's name to know who was playing. To do this, use **`input`** to get the player's name before the game starts.

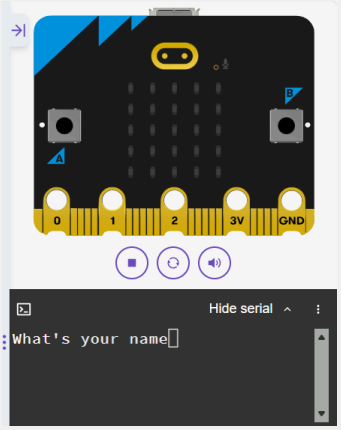
Hint

This is how we use the input function

```
player_name = input("Enter name: ")
```

When using microbits, you can input the name in the serial console on the micro:bit Python Editor website. It will look like this, click on "show serial" to expand and you can type in it.





Task 10.2 Let's build a dictionary

We also need a place to store all the players and their scores. Create a dictionary called **`player_scores`** under the line where you choose a random action.

After the game ends and before you print out the score, store the player's score in the dictionary.

Hint

This is how we use a dictionary.

```
shopping_list = {}
```

This is how we store in a dictionary

```
shopping_list[item] = price
```

Task 10.3 Time to pick the winner

Loop through the dictionary to find who has the best score.

After the line where you store the player's score in the dictionary.

- Create a variables to store the **`winners_name`** and set it to `""`
- Create a variables to store the **`winners_score`** and set it to `0`
- Use a **`for`** loop to check if the **`player_score`** is greater than the **`winners_score`**
- Update the **`winners_name`** and the **`winners_score`**

Hint

This is how we use a **`for`** loop and an if statement.

```
for item in shopping_list:
    if shopping_list[item] >= most_expensive_price:
         most_expensive_price = shopping_list[item]
         most_expensive_item = item
```

Task 10.4 Print the winner

Display the **`winners_score`** and the **`winners_name`** along players name and score at the end

☑ CHECKPOINT ☑

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

☐ You have stored the players name using input
☐ You have made a dictionary and stored the players score in it
☐ You have checked which player has the highest score
☐ You have displayed the winner and their score

Extension 11: Time to level up!⭐⭐

Task 11.1 Let's make the levels

To add different levels with different difficulties, we first need to let the player choose which level they want to play. Then, we can add code that changes the game to make each level easier or harder.

To add the levels, at the start of the game, before we display the **`TARGET`**

- Display the letter L to let the player know they are choosing a level
- Create a **`list`** containing the levels the player can choose from. In this extension, we will add a level 2
- Create a variable called **`current_position`** and set it to 0 so the game starts by showing the first level. This variable stores the current index of the level we are seeing.
- Create a **`while True`** loop so the level selection stays open until the player chooses a level.
- Display the level stored at the **`current_position`** in the **`levels`** list.

Hint

`display.show()` can only display text, so we use **`str()`** to change the number into a string before showing it on the screen.

This is how we get the value stored at a position in a list

```
list_name[position] = value
```

Task 11.2 Let's pick a level

Now we want to allow the player to pick a level, and get outside the infinite loop

- **`If`** button A is pressed, move to the next level in the list. We can do this by adding 1 to the **`current_position.`**
- We only have 2 levels, so we don't want to keep increasing forever. Instead, use the code below to go back to the first level after reaching the last one.

  ```
  current_position = (current_position + 1) % len(levels)
  ```
- Pause briefly using **`sleep`**, so one button press only moves one level
- Let the player confirm a level by pressing button B. **`If`** B is pressed, save the level at **`current_position`** into a variable called **`selected_level`**
- Use **`break`** to leave the loop and start the game.
  Outside the **`while`** loop you made in this extension, pause briefly using **`sleep`**, then display which level the player chose before the game begins

Hint

`%` is called the modulo operator.

It's used in maths! It gives you the remainder after dividing one number by another. For example, 3 % 2 = 1, 2 % 2 = 0

Task 11.3 Level 2!

In level 2 to make it more challenging, if we make a mistake, we will lose 1 point from our score. Let's write the code for level 2

After the section where we check if the action is "press A", we already add 1 point when the player presses the correct button (A).

- Now we want to add an **`elif`** statement to check if the level is 2 **`and`** if we press button B (which is wrong)
- Display a sad face
- Subtract 1 from the score
- Repeat these steps when the action is press B, and we press button A

Hint

This is how you use **`and`** and **`elif`**

```
if temperature > 20 and sunny:
      print("Let's go outside!")
elif temperature > 20 and sunny == False:
      print("It's warm, but take an umbrella!")
else:
      print("Let's stay inside.")
```

BONUS: More levels

You can create more levels, just add the number to the list of levels, and write the code for what your level will do. Some ideas are

- Add booster points if you get several answers correct in a row
- Add an opposite round, when you have to press B if the arrow point to A and vice versa

☑ CHECKPOINT ☑

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

☐ You have made the list of levels, and display the current_level
☐ The player can search through the levels with button A and confirm with button B
☐ Display the level
☐ You lose a point if the player presses the wrong button, when the action is press A and when the action is press B

Extension 12.0: Race your friends

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

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.

Extension 12.1: Give me your Radio⭐⭐

Task 12.1.1: 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 channel with `radio.config(channel=100)`. Your room coordinator will tell you what number to use.

Task 12.1.2: 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 12.1.3: 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("I 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.

☐ 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. (There should be a master microbit in the room to test on)

Extension 12.2: Make a Game Master

Task 12.2.1: Configure the Radio

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

1. At the top of your file, `import` the `micro:bit` and `radio` modules.
2. Turn the radio on with `radio.on()`.
3. Configure the radio to use the channel that the room coordinator gave you.

Task 12.2.2: 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 12.2.3: 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 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 12.2.4: 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 12.2.5: 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 13.0: Bop Battle

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!

Extension 13.1: Getting actions via radio

⭐⭐

Task 13.1.1: 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 channel with `radio.config(channel=100)`. Your room coordinator will tell you what number to use.

Task 13.1.2: Ready, Set, Go!

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

1. Find where you first set the `chosen_action` randomly. It should be above your `while` loop. `Comment` out this line!
2. Inside the game loop, change the `chosen_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 13.1.3: Run only once!

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

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

Task 13.1.4: Send the winner!

Now, tell the game master you've won!

1. 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 12.1.5: 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!

Extension 13.2: All powerful Game Master ⭐⭐⭐

Task 13.2.1: Configure the Radio

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

1. At the top of your file, `import` the `micro:bit` and `radio` modules.
2. Turn the radio on with `radio.on()`.
3. Configure the radio to use the channel that the room coordinator gave you.

Task 13.2.2: 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 13.2.6: 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 14.0: Co-op Bop

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.

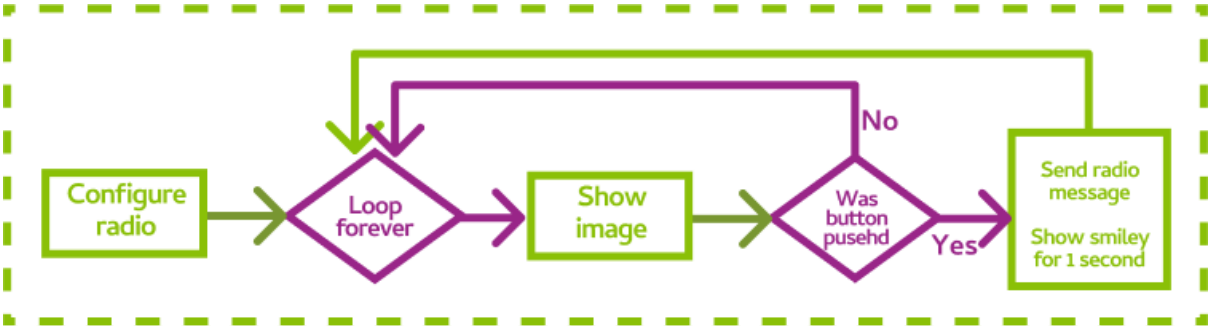
Use the Micro:Bit buttons or craft your own!

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.

Extension 14.1: Making Radio Button



Task 14.1.1: Configure the Radio

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

1. At the top of your file, `import` the `micro:bit` and `radio` modules.
2. **Turn the radio on** with `radio.on()`.
3. **Configure the radio to use the channel** that your tutors gave you using this code: `radio.config(channel=10)`

Task 14.1.2: 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 its identifier.

Task 14.1.3: 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 14.1.4: Smiley faces for pressing!

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

2. Download your code to a MicroBit, you'll need to set up the other MicroBit before they will work together.

Joining a game

Now you have a working button you can join a game. You'll need to make sure your buttons 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 buttons image when it is chosen.

Make sure your button name and image are unique for the game!

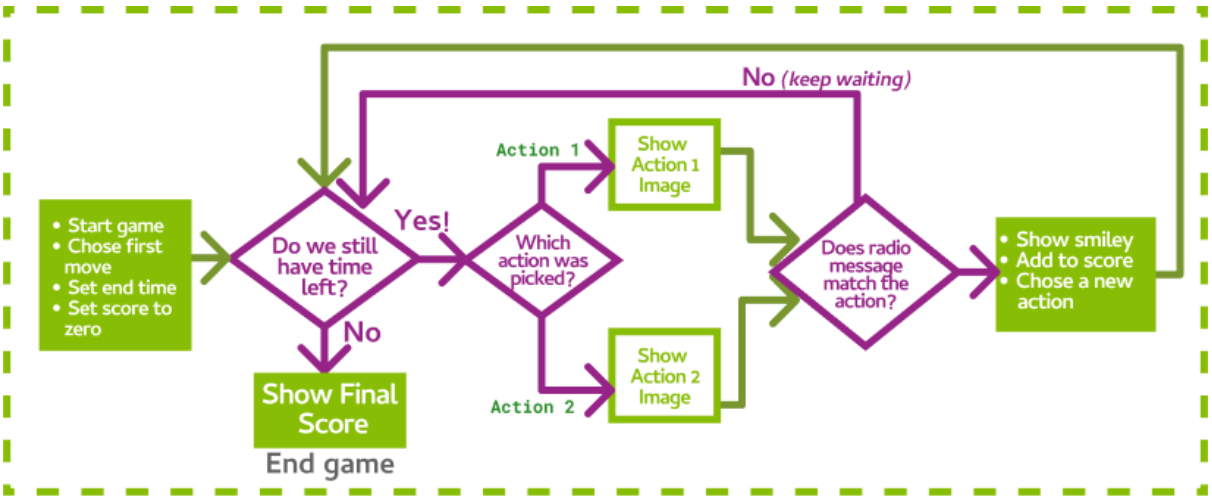
☑ CHECKPOINT ☑

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!

Extension 14.2: Listening for Radio Buttons ⭐⭐⭐

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



Task 14.2.1: Configure the Radio

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

1. At the top of your file, `import` the `micro:bit`, `random` and `radio` modules.
2. Turn the radio on with `radio.on()`.
3. Configure the radio to use the channel that the room coordinator gave you.

Task 14.2.2: 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 in 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 `chosen_action`

Task 14.2.3: 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 14.2.4: 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 14.2.5: 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 15: Testing our memory!

Within this extension, we're going to add to the number of instructions we have to follow at a time. We will make it so that the micro:bit will give us multiple instructions at a time that we have to remember and do in the correct order, with it adding to the number of instructions every time.

Task 15.1: No time like the present

The first thing we need to do to refactor our code to work for this extension is to change it so that the game doesn't end at a certain time. This will make it more fun as we'll be able to keep playing forever

1. First, find where you set up **`start_time`** and **`end_time`** and comment it out (You can use ctrl + / to do those in one go)
2. Now we'll need another way to know that the game is over. Let's make a variable called **`lost`** and set it to **`False`**
3. Finally we need to adjust our while loop so that it continues while **`lost`** is **`False`**

Task 15.2: Multiple actions

Now lets change some things around so that we assign multiple actions each round

1. Make a variable to store the number of actions we want to generate called **`num_actions`**. For now lets set it to 2
2. You will also need to change your **`chosen_action`** variable so that instead of automatically generating an action it is an empty list (You might want to rename it to **`chosen_actions`** for more clarity) as this will eventually store all the actions chosen per round
3. Make another variable called **`current_action`** and set it to 0. This is going to store the action the user should currently be up to
4. Inside our while not lost loop, make a variable called **`correct`** and set it to **`False`**. This will track whether there has been a correct button press each round.

Task 15.3: Choosing our actions

We currently have a bunch of empty variables. Let's make it so we generate actions and store them.

1. Under where you just created your variables, make a for loop that loops **`num_actions`** times
2. In the loop, make a temporary variable (called temp or something similar) and use it to generate a random action from actions
3. Now, append temp to our **`chosen_actions`** list
4. We also want to make sure our player knows what actions they have to do. For this, write an **`if`** statement to check if temp is "press a". If it is, display **`ARROW_W`**, otherwise, display **`ARROW_E`**
5. Finally, we need to do some sleeps and clears so that it's obvious when we change to another action (especially in case we do the same action twice). You will need to sleep for **500** milliseconds, then clear the display, then sleep again for **250** milliseconds.

Task 15.4: Checking the current action

We now want to edit the if statements in your while loop so that they are checking whether the player has clicked the button for the action they are up to, and if they have moving on to check for the next action. We are doing it this way so that we wait until a button has been pressed and then make sure it was the correct button.

1. Go to the first **`if`** statement. You will need to change it slightly so that instead of checking **`chosen_action`**, it checks **`chosen_actions`** at position **`current_action`**.
2. Inside that **`if`** statement, it should check if button a is pressed, if it is, we want to make **`correct`** equal **`True`** and increase score. **`Elif`** button b is pressed, we need to set **`lost`** to **`True`**. Finally, if neither were pressed, we need to **`continue`**.
3. Now we need to copy all of that logic across for the second **`if`** statement, switching any time it says a to b and vice versa
4. After both **`if`** statements, we need to check if **`correct`** is **`True`**. If it is, increase **`current_action`**.
5. Finally, at the very bottom of the while, outside all if statements, sleep for **500** milliseconds.

Task 15.5: Resetting to the first action

Right now if you run the code you'll eventually read a "list index is out of range" error once you've done both actions, this is because we're never going back to the start of the list, or generating any new actions to remember. Let's fix that.

1. At the top of your `while`, right under where you make correct, check if `current_action` is the length of chosen_actions. This means the player got to the end of the pattern and should be reset to the start.
2. Show a happy face to tell the player that they got it correct. Then, reset `current_action` to 0
3. Now, we want to add two more actions onto the same pattern. To do that, append a random choice from actions into `chosen_actions` twice.
4. Finally, we also want to make it so the user can see the whole pattern from the start again. To do this, loop through everything in `chosen_actions`, if its "press a" show `ARROW_W` if not show `ARROW_E` then sleep and clear and sleep (like we did the first time).

☑ CHECKPOINT ☑

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

- ☐ Your game generates a long pattern of actions that gets longer the more you get it correct
- ☐ The game doesn't end until you get the pattern wrong