WHACK-A-MOLE WITH MICRO:BIT

In 10 steps!

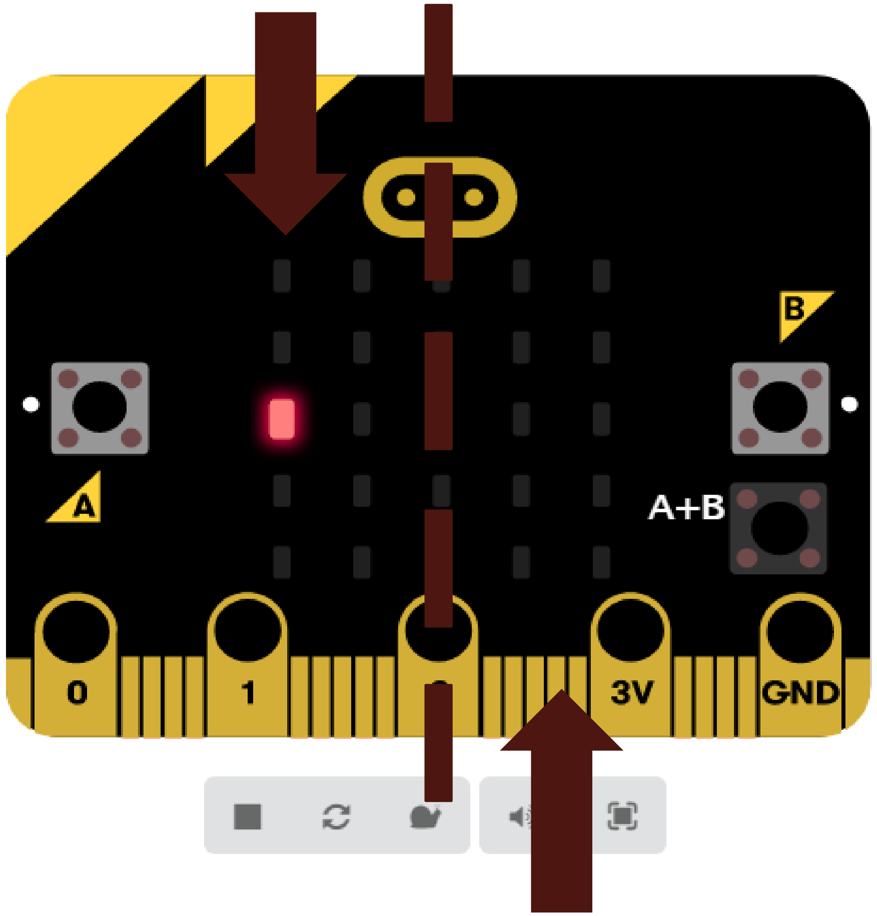
WHACK-A-MOLE

Moles in micro:bit a small, and they move very fast! This program will train up your reflexes catch them all.

Our mission is to program our micro:bits to create and randomly position a moving 'mole' – or 'sprite' (a single LED light). We'll try to 'whack' these sprites by pressing the 'A' or 'B' button when they are in the correct part of our screens. Left for 'A', right for 'B'. We'll gain points for correct presses, and lose lives if we get it wrong… these moles are sneaky!

Our program will keep track of our scores (which we'll be able to view by pressing 'A+B' together), and the number of lives we have left. It's game over if you lose three lives!

Press 'A' when the mole is on this side to score! If you press 'B' when it's here, you'll lose a life…



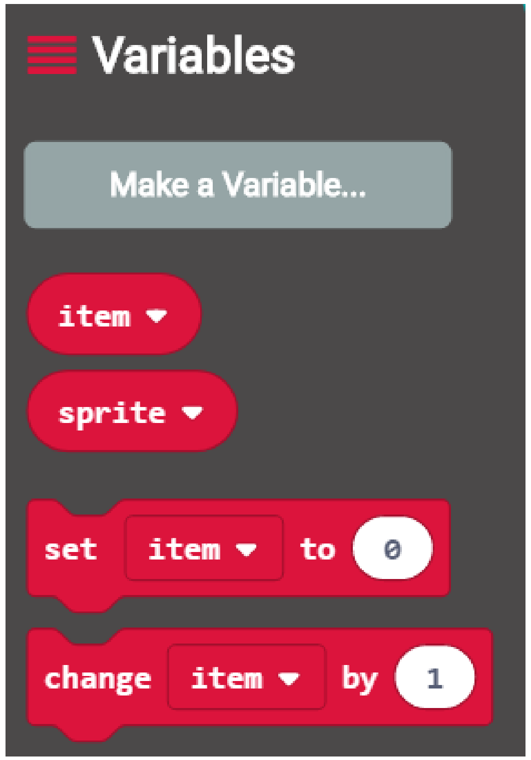
Press 'B' when the mole is on this side to score – pressing 'A' will cost you a life!

GETTING STARTED WITH MICRO:BIT

To get set up with your micro:bit, we need to head on over to https://www.microbit.org/, where you can find plenty of information about getting started, and loads of ideas for coding projects.

We can open a new workspace to create our whack-a-mole program at https://makecode.microbit.org/#editor

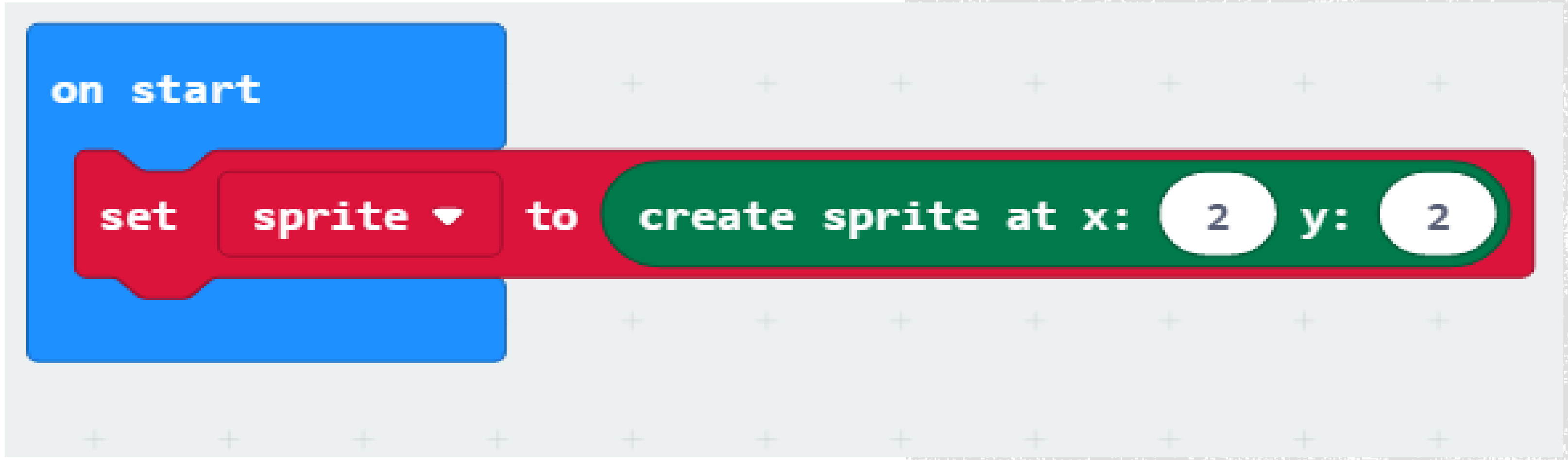
STEP 1: CREATING OUR SPRITE



On Start, the first thing we want micro:bit to do is create a sprite at a random position.

To do this, we'll first need a ***sprite* variable** to store this sprite in, so we'll start by creating a new variable called 'sprite'.

Then, within the ***on start***, we'll set this new variable to ***create a sprite*** on our micro:bit screen. By default, our sprite will be created at x:2, y:2… the middle of the micro:bit screen.



```
on start
    set sprite to create sprite at x: 2 y: 2
```

This might seem a little bit odd. If the screen has 5 LEDs across (columns), and 5 LEDs up (rows), why is 2 in the middle?

STRANGE NUMBERS

Well we have this funny thing in programming where we start counting from zero, not one. So if you've got 5 moles, you'd count them as 0-4:

number 0

number 1

number 2

number 3

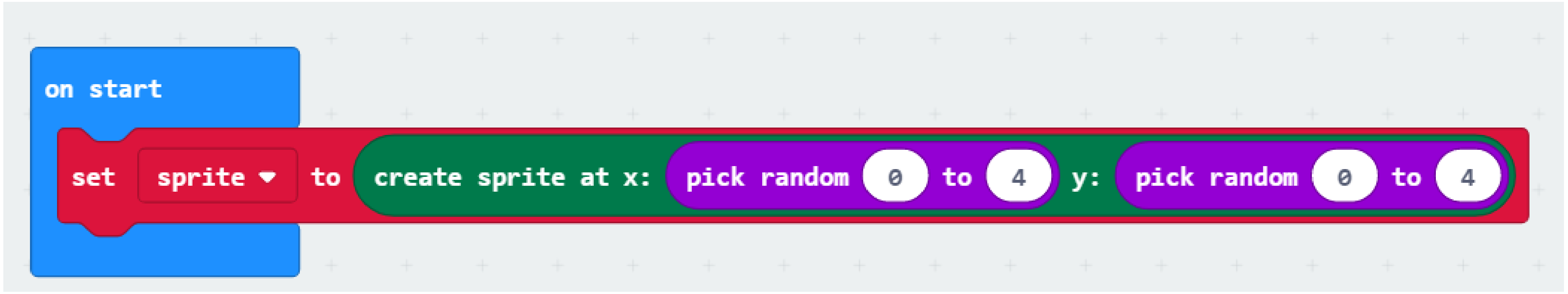
number 4

Because of this, on our micro:bits, the middle LED column and row – column/row 3 of 5 – is given the value of 2. So if our sprite starts at 2, 2 (3rd column, 3rd row), it will start in the middle.

STEP 2: LET'S GET RANDOM!

But our sprite always appearing in the middle of screen will get boring fast, so we want it to appear at random locations.

This is pretty easy: all we need to do is make sure that the row and column for our sprite isn't always 2. We'll replace the default numbers for ***create sprite*** with ***pick random*** between 0-4.

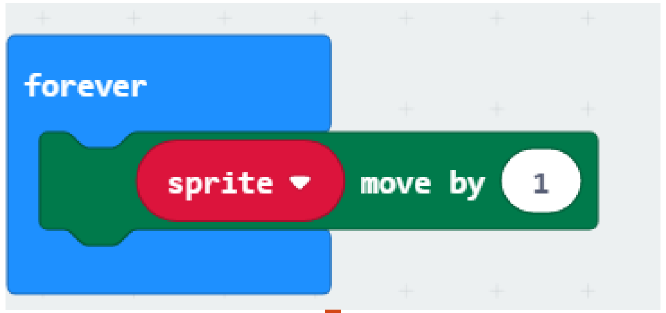


```
on start
    set sprite ▾ to create sprite at x: pick random 0 to 4 y: pick random 0 to 4
```

STEP 3: LET'S GET MOVING….

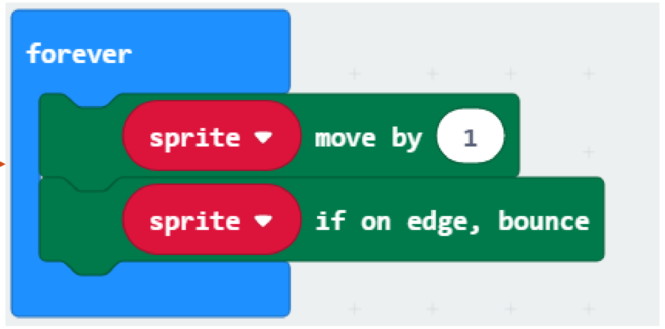
A randomly positioned sprite is good, but we won't be mole-ready until our reflexes are. We've got to train with moving targets, so let's get our sprite moving.

In our ***forever*** block, we'll start by setting our ***sprite*** variable to ***move by*** one.

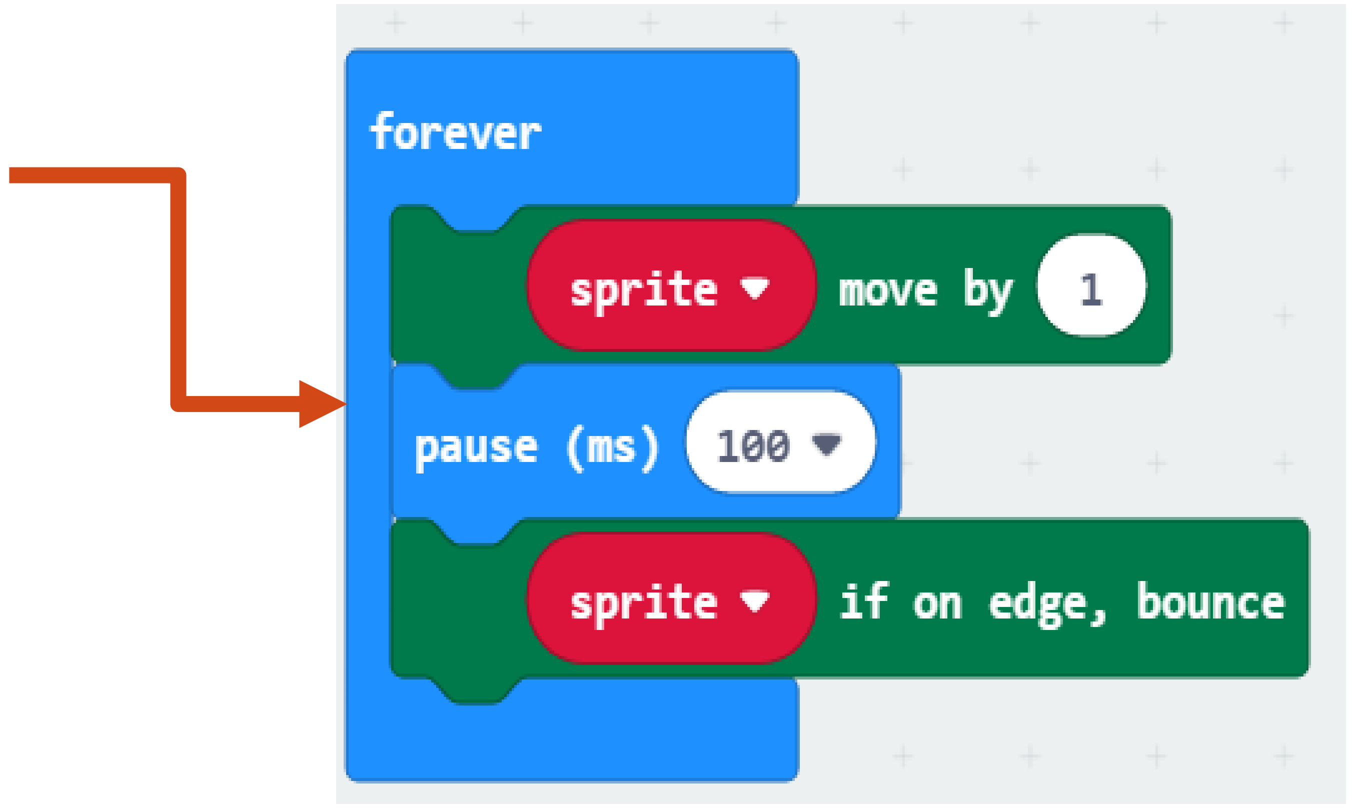


```
forever
  sprite move by 1
```

You'll probably notice that our sprites get stuck against the edge of our micro:bits pretty fast. So ***if*** our ***sprite*** is ***on edge***, we'll need it to ***bounce***.



```
forever
  sprite move by 1
  sprite if on edge, bounce
```



```
forever
    sprite ▾ move by 1
    pause (ms) 100 ▾
    sprite ▾ if on edge, bounce
```

Our sprite is on the move, but that movement speed would challenge even the most expert reflexes. Let's slow down our sprite's movement by having it ***pause*** for a few milliseconds after each move.

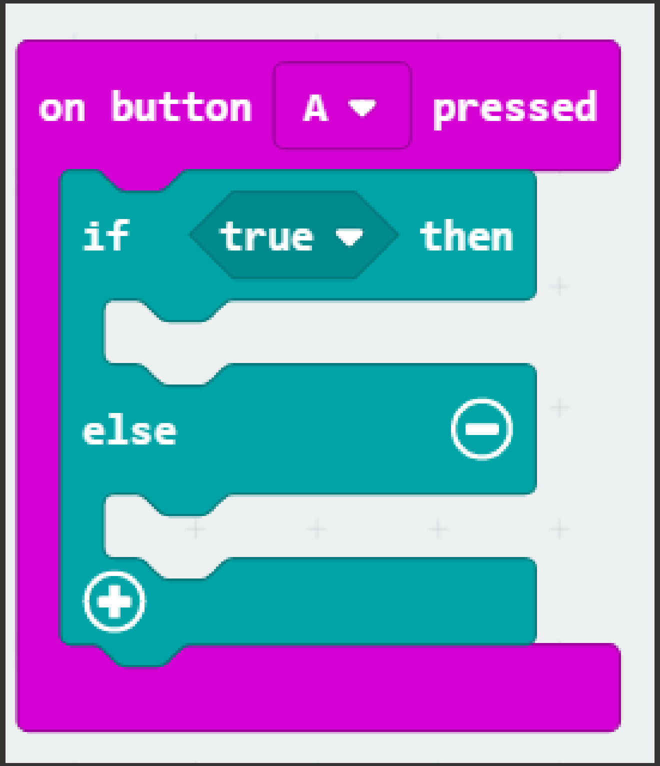
We can start with an 100ms – but if you want to really test your reflexes, you could decrease this pause to 75ms, or even 50ms!

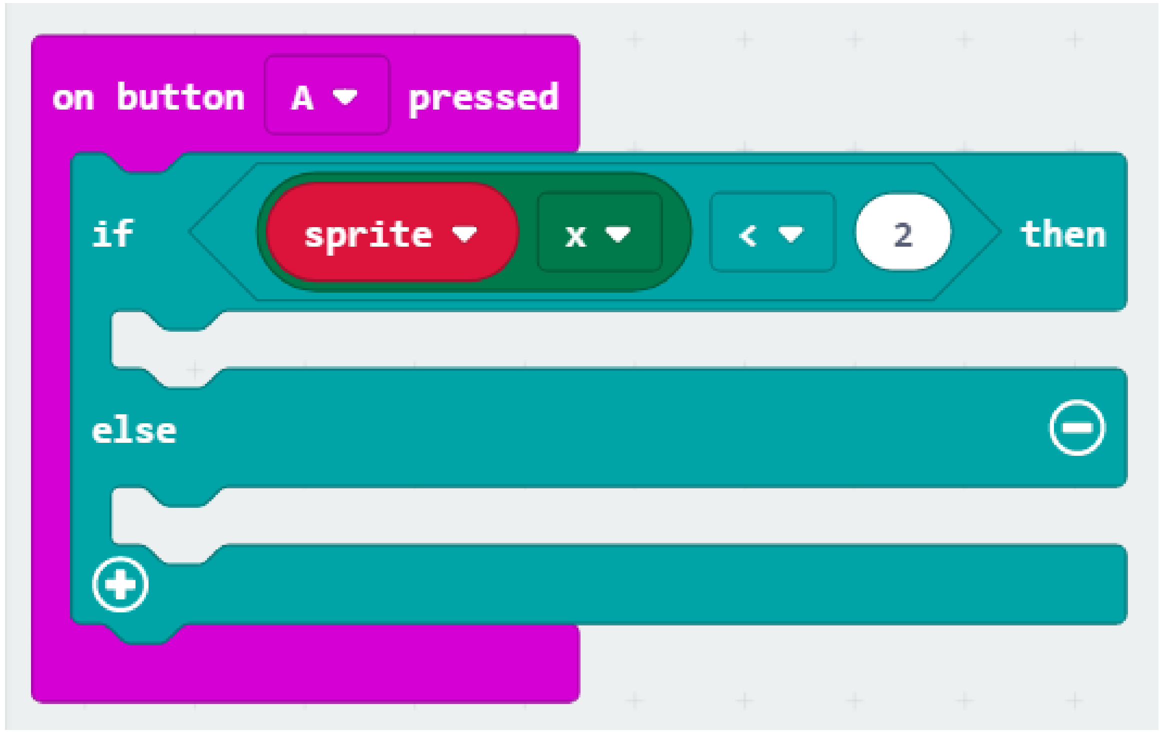
STEP 4: BUTTON MASHING

We've got our sprite up and moving. Now we're going to make our program responsive to button pressing.

On button A being ***pressed***, we'll want our program to check the location of our sprite, increasing our score ***if*** the sprite is on the correct half of the screen – or ***else*** we will lose a life.

To dot this, we'll need an ***if/else*** loop inside our button 'A' press input.





```
on button A pressed
    if sprite x < 2 then

    else

```

STEP 5: WHICH SIDE?

The x coordinate represents the column of our sprite. So, to check that our sprite is on the left side of our micro:bit screen (in column 0 or column 1) , we'll need to set our ***sprite's x*** coordinate to be ***less than 2***.

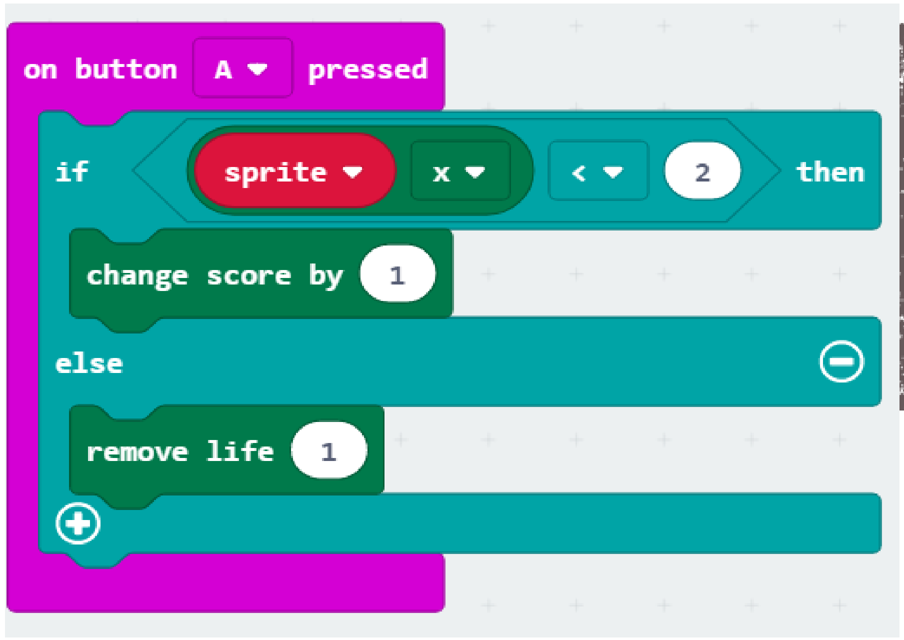
STEP 6: SCORING

In this game, we score if button 'A' is pressed when the sprite is on the left of the micro:bit screen, but we'll lose a life if 'A' is pressed at the wrong time.

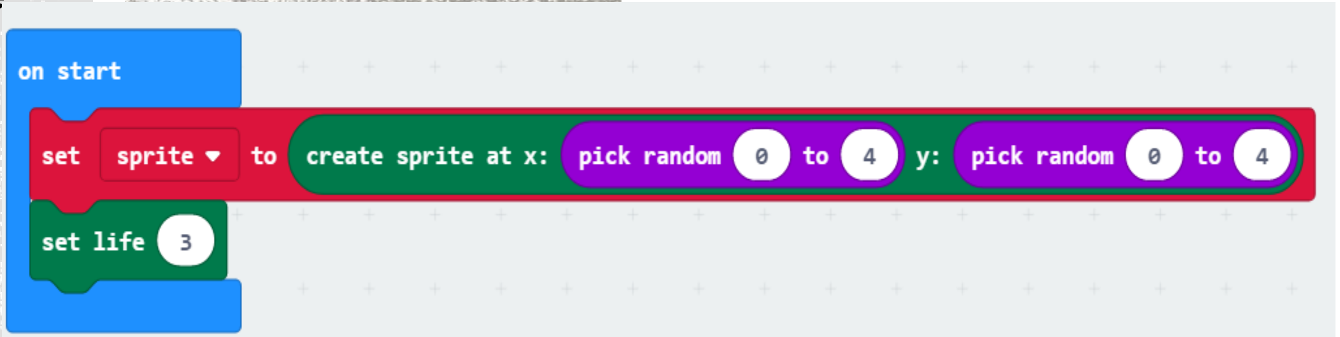
To make this happen, we'll need to ***change score by 1 if*** our sprite is in the correct place, or ***else remove a life***.

But wait a second, how can our program remove a life if we doesn't know how many we started with? Let's fix this by going back to ***on start*** and adding a ***set life*** to 3.

When our life counter get's to zero, it'll be a game over!



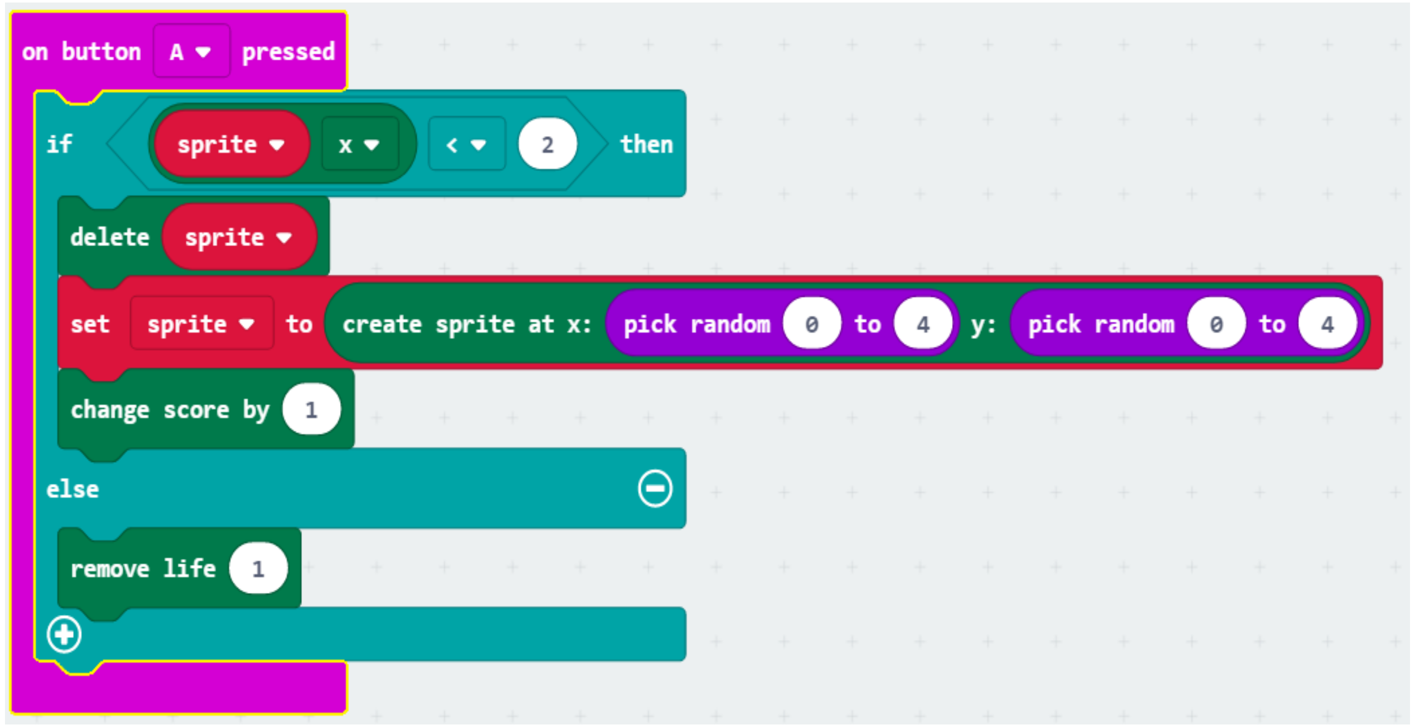
```
on button A pressed
    if sprite x < 2 then
        change score by 1
    else
        remove life 1
```



```
on start
    set sprite to create sprite at x: pick random 0 to 4 y: pick random 0 to 4
    set life 3
```

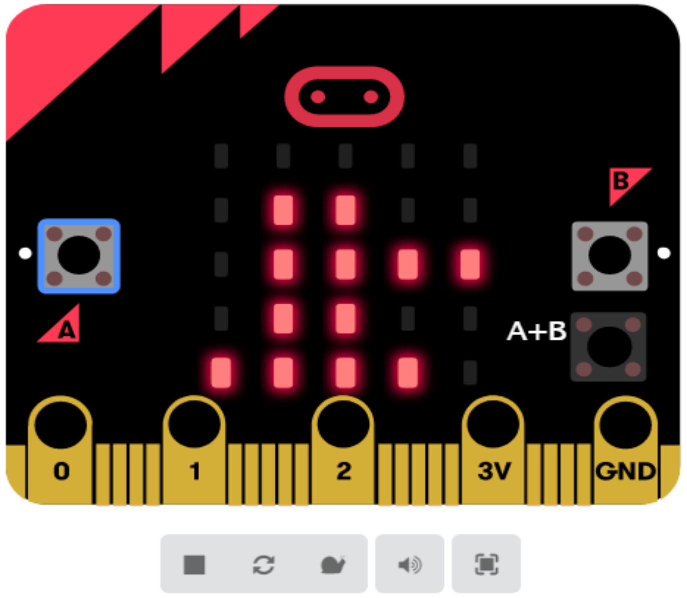
STEP 7: DELETE-CREATE

We're now able to score in our training program, but it's not really testing us if it's always the same sprite following the same path. This sprite should be just one of many – not our arch-nemesis!



```
on button A ▾ pressed
    if sprite ▾ x ▾ < ▾ 2 then
        delete sprite ▾
        set sprite ▾ to create sprite at x: pick random 0 to 4 y: pick random 0 to 4
        change score by 1
    else
        remove life 1
```

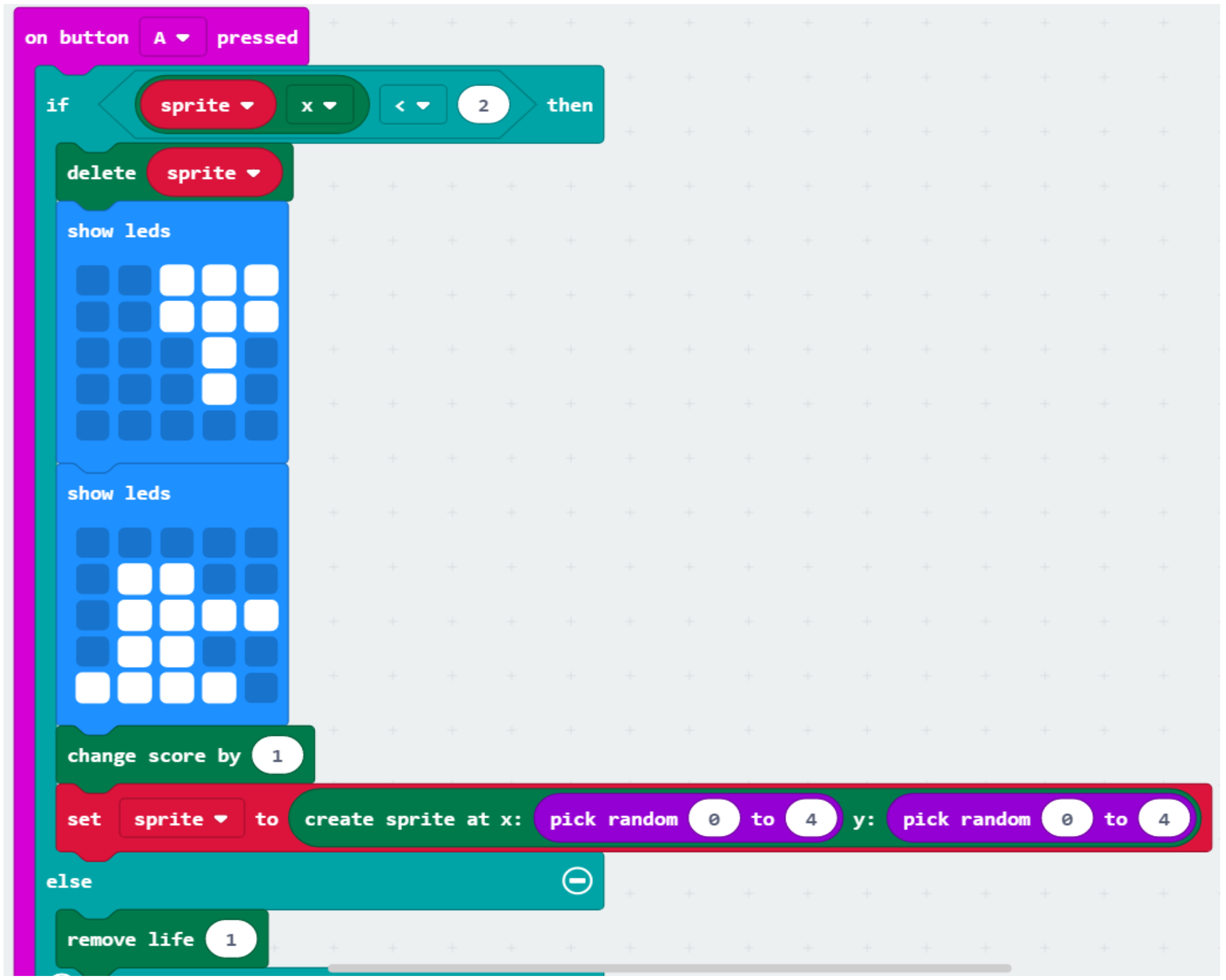
So, if we successfully catch a catch, we'll want our program to ***delete*** it, and ***create*** a new sprite at another random location before we score. You can duplicate our '***set sprite***' code from our ***on start*** block to do this.



STEP 8: ADDING SOME FLARE

What are we catching these moles with? In classic whack-a-mole, we would use a hammer. Let’s add a hammer ‘animation’ to our scoring screen, so that successfully catching a sprite will ***show*** us some hammer-time ***LEDs***.

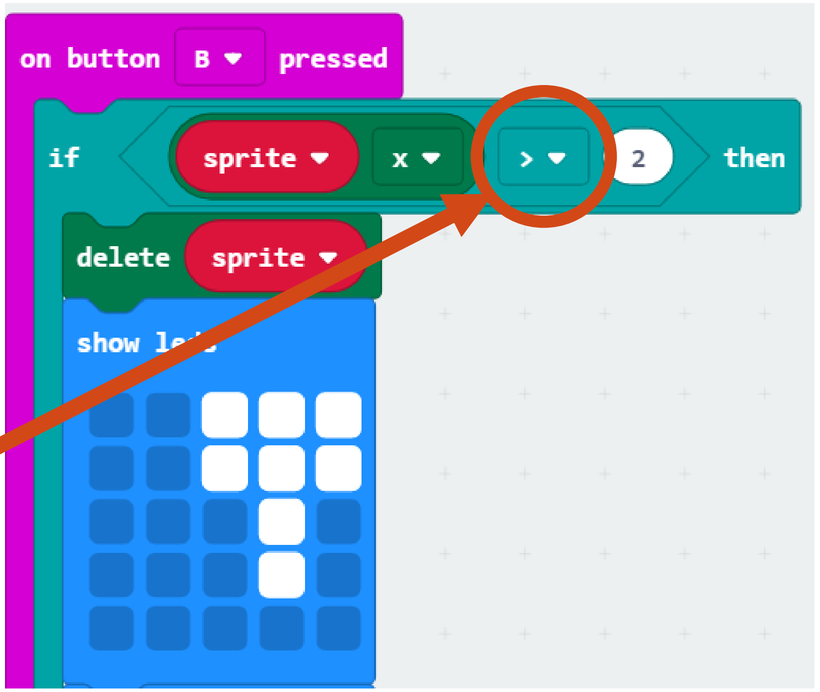
We need to put these LEDs to show *before* our score is changed, or the animation will be interrupted.



```
on button A pressed
    if sprite x < 2 then
        delete sprite
        show leds
            . . # # #
            . . # # #
            . . . # .
            . . . # .
            . . . . .
        show leds
            . . . . .
            . # # . .
            . # # # #
            . # # . .
            # # # # .
        change score by 1
        set sprite to create sprite at x: pick random 0 to 4 y: pick random 0 to 4
    else
        remove life 1
```

Pressing button 'A' at the right time (when the sprite is on the left) should now successfully increase our score, show us a hammer 'animation', and both delete and re-spawn a sprite.

Let's ***duplicate*** our code so that we'll continue playing when button 'B' is pressed. Don't forget to change the ***x*** position of our ***if*** sprite so that it's got to be on the right-hand side of our micro:bit screen to score (column 3 or column 4). So x has got to be bigger than 2.



```
on button B pressed
  if sprite x > 2 then
    delete sprite
    show leds
```

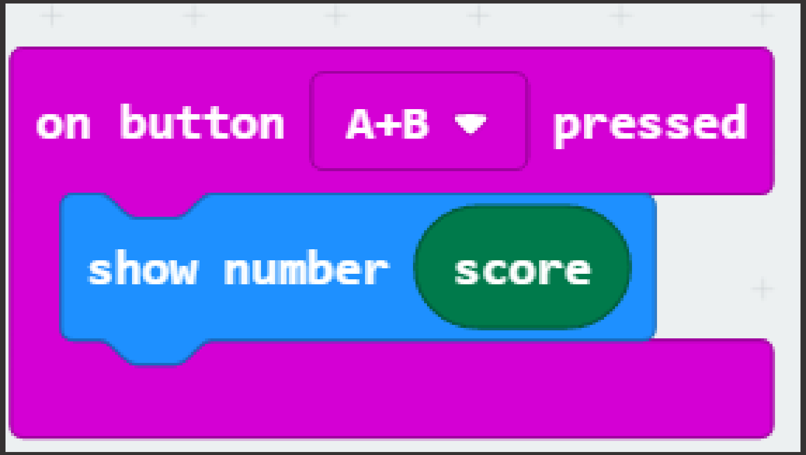
STEP 9: RINSE AND REPEAT

STEP 10: CHECKING OUR SCORES

The last bit we'll add to our program is function to check our scores, so that we now just how many moles we've caught.

So when ***button 'A+B' are pressed*** together, we'll ***show number*** our ***score***.

Play with your friends, and see who can score highest before game over! Make it more challenging by getting the sprite to move faster (reduce ***pause***), or by starting with fewer lives.



```
on button A+B pressed
    show number score
```

CONGRATULATIONS

And there you have it! You now have a master training tool to prepare for future mole attacks, which:

- Creates moles (sprites) at random locations
- Scores you for correct input, with added hammer flare
- Removes lives for incorrect input
- Shows your real-time score on ‘A+B’

micro:bit

YOU'RE JUST GETTING STARTED!
HEAD ON OVER TO MICRO:BIT'S WEBSITE FOR LOADS MORE PROGRAMMING GAMES AND CHALLENGES!