**Reaction Game**

**Paper Version**

**What You Need:**
- 1 x Crazy Circuits Bit Board
- 1 x micro:bit V2
- 4 x LEDs
- 1/8” Maker Tape
- Paper

**Optional:**
- 1/4” Maker Tape

**How it Works:**

The LEDs will flash on and off in order (one at a time) and when the fourth LED is lit you should press the button. If you successfully press the button while the fourth LED is lit, you’ll get a point, and the score will be shown on the LED matrix on the front of the micro:bit.

If you press the button when LED one, two, or three is lit you will lose a point. (The default code will not go below zero, since there isn’t an easy way to display a negative score on the LED display.)

Each time you press the button at the correct time the speed will increase, making it more difficult to get the timing right for the next time.

The piezo speaker will play two different tones, one when you successfully press the button, and one when you press it at the wrong time. It will also play a song when you win by scoring the determined amount of points.

If you want to abandon a game and start over you can press the reset button. (Button B on the micro:bit)

**Note:** For this version of the Reaction Game we’re using a V2 micro:bit so we can take advantage of the on-board piezo speaker.

**Scoreboard:**

We’re going to use the built-in LED matrix on the front of the micro:bit to display the score.

Each time you get a point it will light up another LED, and if you lose a point it will turn off an LED to indicate your score is lower.

With the code we provide you can then set your `winScore` to a maximum value of 25 if you want to use the entire matrix.

For more educational guides and resources visit us at BrownDogGadgets.com
**Make Your Button/Switch:**

This version of the Reaction Game requires you to make your own button (or "momentary switch") that you’ll trigger to play the game.

Making an effective button is the additional challenge for this version. We’ve included two examples here from our Paper Circuits guides on switches to get you started.

You’ll want a switch that only turns “on” when you press it, and then turns “off” when you release it.

---

**Lever Button**

1. Cut out button using scissors. Fold on dotted line.
2. Place Maker Tape on top of switch. Leave enough extra tape to stick to switch down to paper and to wrap around the bottom of the switch.
3. Fold tape over long end of switch and attach securely.
4. Stick it down to the paper so it will close the gap in the circuit.
5. Press the switch down to make contact!

*For more educational guides and resources visit us at BrownDogGadgets.com*
Reaction Game

The Code:
We’ve broken our program into seven sections. We’ll start by looking at the smaller sections then move on to the forever loop, which is the most complex.

Don’t worry if you don’t understand all of the code! You can always build the circuit, load the code, try it out, and then make changes to the code to see how it affects the gameplay.

The on start section always runs first in our program, and we used it to set things that only need to happen one time at the start.

For this program we start by enabling the built-in LED matrix, then calling the resetGame function (more on that later). We also set pin 5 to up so we can use it with the button connected to it, and enable the built-in speaker.

You’ll notice our “event handler” for button B also contains the call to resetGame. This allows us to reset all our variables at any point while the game is running if we want to start over.

A note about “event handlers”
Typically if we want to check for something happening in our program we need to use a conditional statement in our forever section. We would add in the code blocks and the logic and test for a specific condition (like a button press.)

Event Handlers are special “functions” that run all the time, and continually check for a specific thing, like button B being pressed.

In this case we use the special event handler to trigger the game to reset the variables and start over.

Our lightLED function allows us to call the function with a set of parameters.
When we call our function we “pass in” four numbers. Our numbers need to be either a 0 or a 1, which corresponds to setting each pin to 0 or 1, which is off or on. This allows us to turn on (and off) all four LEDs with a single command.

If we have a block like so:

It will turn on the first LED and turn off the rest of them. If we pass 0, 1, 0, 0 it will turn on the second LED and turn off the rest of them, etc.

In our resetGame function we set the variables for the game.

If we did not have a special reset button we could probably have put all of these blocks into the on start section, but since we want to reset the game without having to power off and back on the micro:bit this method makes sense.

Here’s what each of our variables does:

- **winScore** sets the score you need to reach to win the game.
- **interval** sets the amount of time each LED remains lit for.
- **timeChange** sets the amount of time we subtract from interval after each successful button press.

The three variables that affect gameplay, winScore, interval, and timeChange, are interdependent on each other. If you lower the winScore the game will be easier, but you can then lower the interval to make it more challenging again.

If you adjust the timeChange to a much higher number, or set the interval to a much lower value it might make the game unplayable, or just more challenging! We’ve set what we think is a good balance as a starting point, but adjusting them and seeing what happens is part of the fun of game development!

For more educational guides and resources visit us at BrownDogGadgets.com
Displaying the Score:

If you built the version of the Reaction Game that uses a 7 Segment Display you may remember that the code to show the score was just a single block of code. This time we need to do a bit more work to display the score.

We have two functions, showScore which deals with looping through all the LEDs and turning them on one at a time, and clearScreen which looks through all the LEDs and turns them off one at a time.

You might also notice that the first thing our showScore function does is call the clearScreen function. This is so we can always wipe the screen completely before displaying the score. Since the score could go down if you lose a point we had to make sure we didn’t always just count up but also had the ability to count down.

If you want to make your own scorekeeping functions you can look at using the show number command in MakeCode or the show leds command.

We chose this method as a fun programming challenge! And since we created it as functions separate from our code we can easily use it in other programs that need a score keeping system.

Our low-res/low-tech scoreboard isn’t fancy, but it gets the job done!
The `forever` section of our code has a lot going on, so we’ve broken it into four sections to explain it.

If you checked out our guide titled *Blink Without Pause* we showed that while it’s common to cause an LED to blink on and off using the `pause` command if you want non-blocking code (that is, code that never “pauses” while running) you can implement a timer that checks if a certain number of milliseconds has passed and then turn on (or off) your LED.

We use that method in the first part of the `forever` section to change which LED is lit each time our interval is passed.

We also check if our `whichLED` variable has increased past 4 and set it back to 0 if it has.

In our next section of code we check which LED is specified and then use our `lightLED` function to turn on one of the four LEDs.

Our `lightLED` function call passes four values to the function which means that with a single block of code we can control all four LEDs. This is much more efficient than using four blocks of code to individually control all four LEDs within each `if/else` statement.
This section checks to see if the fourth LED is active, and also checks if the button is being pressed. If those conditions are met we increase the **score** by one, decrease the **interval** by the amount of time set for the **timeChange**, we play a tone, and we pause for half a second.

But what if the fourth LED is not active, and the button is pressed? Then we subtract one from the score, play a different tone, and again pause for half a second.

Lastly, we have a check if the score is less than one and if it is, we set it to zero. If we remove this check then our score could keep decreasing to negative numbers, which might make the gameplay even more challenging! Feel free to remove this section or adjust it in some other way.

The **call showScore** block handles calling the function to display the score on the LED matrix.

And finally, we check to see if our score is equal to the **winScore**, and if it is, you won the game! We then turn on all the LEDs, play a victory song, wait for two and a half seconds, then reset the game so we can try again.