Game: Reaction Time Test

By Koldo Olaskoaga

Pictures by Koldo Olaskoaga

This article will discuss how to create a simple program that measures the user's reaction time. The program will be created using Open Lab Roberta.

Challenge

Create a program that measures the speed of reaction to a light stimulus.

To achieve this, I am going to use the same electronic device that I used for the LEGO MINDSTORMS article (HispaBrick #15), that which I later completed and published with more detail in Issuu [1]: 3 different colored buttons corresponding to three different colored lights all connected to NXT. However, I have updated it to EV3 as can be seen in the picture and in this case using the EV3 light.

As a stimulant, I will use EV3's very own light. After pressing the corresponding key the achieved time will be shown on the screen.

Program

The final objective will be to create a program that has the following characteristics:

- A green,orange,or red light will be turned on and one will have to click on the corresponding key.
- The light be turned on for a margin of time between 2 and 5 seconds.
- 3 attempts will be given and after each the achieved time will be shown as well as the best time thus far.
- The necessary instructions will be shown on the screen.

Starting everything at once isn't recommended, it is always best to divide the challenges into smaller challenges that can be combined later little by little to create something more complex.Therefore we will start by looking at just one corresponding button, covering the other objectives step by step.

As always, before creating the program in the desired language, it is best to write the algorithm in your natural language, meaning, in your first language.

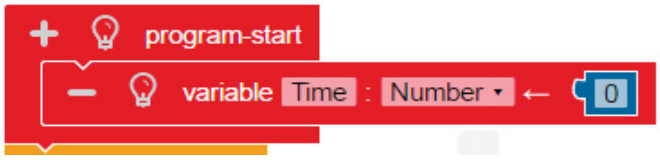
Step 1

Objective: When the green light turns on, the green button must be hit. Afterward the elapsed amount of time will be shown on the screen.

Let's look at the following steps,that is, the algorithm:
1. Create a variable that allows the amount of time elapsed before the button has been clicked to be stored.
2. Turn on the green light (EV3).
3. Start the stopwatch (the EV3 timer and stopwatch are continuously running, so what you need to do is reset them to zero when you want to begin counting).
4. Wait until the green button is pressed (EV3's port 1).
5. Store the result in a variable (the timers can't be stopped, but the result can be assigned to a variable when desired, so you can record the value at any given time).
6. Show the result on the screen.

Once we have the algorithm, we can then convert the program into the desired language. In this case I am going to use Open Roberta Lab (NEPO from here on),a program which we talked about in the previous issue of HispaBrick Magazine, although using EV3-G would be much the same. You have to select the expert mode NEPO-blocks as there are some blocks not present in the beginner mode.

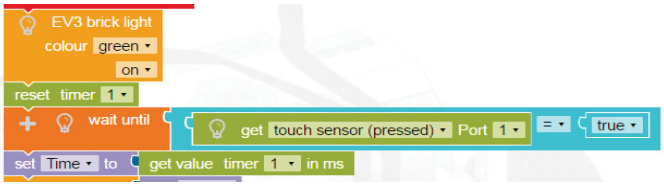
The Program: in NEPO the variables are defined in the program-start block,all that is needed to create one is to press on the plus sign.This way we can give each their own name, indicate what kind of variable they deal with (there are 9 different kinds) and assign it an initial value.



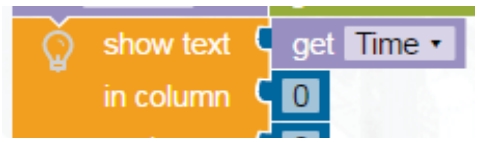
After turning on the EV3 green light, you have to reset the timer to zero, there are 5 different timers available, in this case we will use number 1.

The process of creating the instruction that will pause the program until the green key has been pressed (above the touch sensor connected to port 1) is very similar to that used in Scratch. Here we use the **Wait Until** block, that can be found in the control menu **Control** > **Wait** to which we need to connect a logic operator. In this case it is to see if the touch sensor connected to port 1 has been pressed.

Once the program can continue the value of timer 1 is assigned to the variable **Time**. This value will be expressed in milliseconds.



All that is left is to show on the screen is the achieved time.



Now hit execute and observe the following happening:

- The green light turns on as soon as the program starts
- It does not show the reading on the screen as the program exits as soon as it finishes.

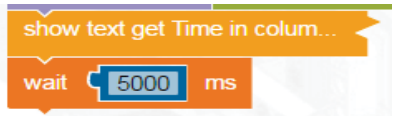
The reason is that we haven't marked the runtimes that we are interest in. We haven't asked it to wait a bit to turn on the green light after the execution. We haven't given it proper time to show the reading on the screen,proper time, because in fact, the result was shown, but once shown, the program is ended right away and we haven't been able to take it in. These are two things that we will fix by introducing several changes.

What we will change: Once the program begins, it will wait a period of time between 2 and 5 seconds before turning on the green light and at the end of the program we will include a wait of 5 seconds so that we are given time to see the achieved result (the changes appear in bold text in the algorithm).

1. Create a variable to be able to store the time
2. Wait between 2 and 5 seconds
3. Turn on the green light
4. Start the stopwatch
5. Wait until the green button is pressed (port 1)
6. Store the stopwatch reading in a variable
7. Show the result on the screen
8. Wait 5 seconds

A random start up time between 2 and 5 seconds is like throwing a die and seeing how it falls, except in this case the number of results are much higher.Given that the control block **wait** asks that the time be given in milliseconds, you would have to calculate a number between 2000 and 5000, even though you can also calculate a number between 2 and 5 later multiply it by 1000.

Toward the end of the program a **wait** block is added to give time to read the result.



As you can see, the blocks that were in the first program have been made so they occupy less space. This can be done by right clicking on the block that you wish to collapse.

Upon executing the new program we can see that the two previous problems have been solved and that we have successfully responded to that which was asked of us for step 1. However, in the challenge it asked that 3 rounds be played, with that said, let's go to step 2.

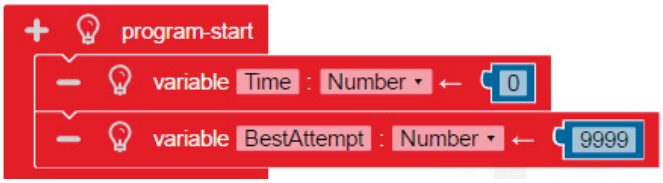
Step 2

Objective: when the program begins,the name of the game will appear on the screen along with a message that asks that the user press on the green key to continue. After a 2 to 5 second wait, the green light will turn on and once the green key has been hit, the reaction time will be shown on the screen as well as the best time thus far. After 2 seconds the program will continue. This will be done 3 times before coming to an end.

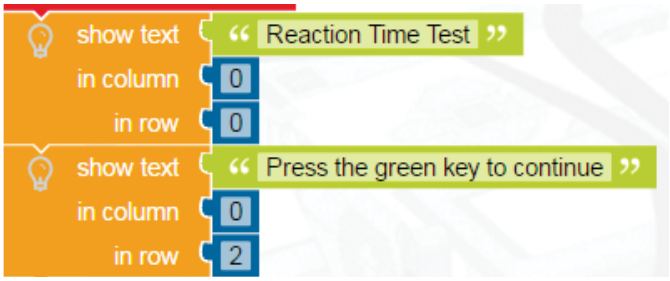
The new algorithm includes the following:

1. Create time and best result variables
2. Show on the screen the name of the game and a text that asks the user to press the green key to continue.
3. Repeat the following steps
    a. Wait for the green key to be hit
    b. Wait 2 to 5 seconds for startup
    c. Turn on the green light
    d. Start the stopwatch
    e. Wait until the green key has been hit
    f. Store the stopwatch reading in a variable
    g. Compare the new result with the one stored in the best result variable
        i. If the value of the Time variable is lower than the Best result variable, store in the best result variable the new value
    h. Show the reading on the screen, along with the best result thus far.
    i. Wait 2 seconds

Even though in the prior step we saw how to create a variable and initiate it, we are going to see a peculiarity with the new variable "Best Result". This variable has to save the best result that has been achieved up that point so that it can be compared with the current result (step 3g of the algorithm). To see if there has been an improvement, compare the new result with the best achieved value,and if it is higher, the old best result will be replaced. ver si ha habido mejora comparará el nuevo resultado con su valor, y si es menor lo sustituirá. So the starting value needs to be big enough so that after the first attempt, when the comparison is executed, the compared value is less than the initial value. This value is then updated with the new reaction time. A safe value that will ensure it is bigger than the expected reaction time is 9999 milliseconds, but apart from this we could have chosen any other high enough value.

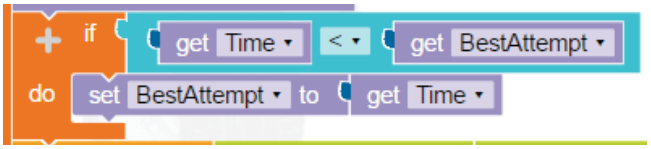


Once the program begins the name of the of the game as well as instructions in two different rows.



Then, a **Loop** begins that will be repeated 3 times, this block can be found in the **Control** menu. The first instruction that we see inside is the “wait until the green touch sensor is pressed” (touch sensor port 1) before a new attempt begins.

What comes next is the same as what we had in step 1 until we got to the comparing a new result with the best result. In this case we will need to use a conditional control structure, that is, one of the blocks that appears under **Control** > **Decisions**. In this case, if the result has a value that is less than what is stored in the variable **BestResult**, it will substitute the value for the new one. If not, the block will exit without making any changes.



Now all that is left is to show the results. It can be done in the same way as the previous step, we are going to see how to combine the result with the text we want. In the Text menu we have several blocks that allow you to manipulate text chains. One of them can be seen in the following image that allows you to create. One of them can be seen in the next image and allows you generate a sentence, combining text strings with the result we just generated. We will do the same for the best result, but on a different line.

So far the program is going the way we wanted, but there are still some small things left to be corrected:

- the green light stays on all the time: we need to turn it off when we no longer need it
- the width of the screen is limited, so if we want to show a text that is longer than the width we need to cut the text and display each fragment on a separate line.
- the text chains are superimposed: when we tell the program to display a new text chain we only delete the area that is overwritten, so we need to clear the screen before showing new text..

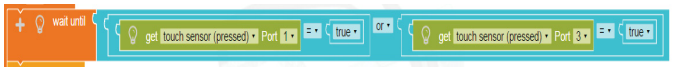
These improvements are easy enough that you can work them out by yourself..

Step 3

Objetive: incorporate a second key. Now you won’t know which of the two will light up and you will have to press the right key for the program to advance.

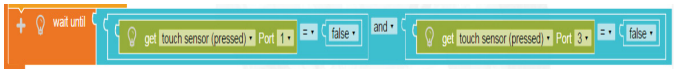
1. Create the variables
2. Show the name of the game on the screen and a text that asks you to press any key to continue
3. Repeat the following steps
    a. Wait until a key is pressed
    b. Clear the screen
    c. Wait a random time between 2 and 5 seconds
    d. Randomly select which light to turn on:
        i. If it is the first one: turn on the green light, start the stopwatch and wait for the green button to be pressed
        ii. If not: turn on the red light, start the stopwatch and wait for the red button to be pressed
    e. Store the value of the reading of the stopwatch in a variable
    f. Switch off the light
    g. Compare the new value to the value stored in BestResult
        i. If it is less, store the new value in the variable BestResult
    h. Show the reading on the screen as well as the minimum value and a message asking to press a key to continue
    i. Wait until no key is pressed
4. Wait for 2 seconds

The program starts in the same way. However, now any of the keys that are used can be pressed, in this case both red and green. This requires a modification of the block that waits for the key press before starting the sequence. Now it will have to wait for one or the other key to be pressed, so the logic operator OR is needed.



After clearing the screen and waiting for the light to turn on it is time to decide which light will turn on and then wait for the corresponding key to be pressed. To take this decision we are going to generate a random number between 1 and 2. If it is 1 we will turn on the green light, if it is 2 the red light. After turning on the light, the stopwatch is restarted and the program waits for the correct key to be pressed. If the wrong key is pressed nothing will happen - the program will simply continue waiting for the right key to be pressed. When the right key is pressed the program will continue and assign the result to the variable **Time**.

After turning off the light it checks if the time is better than the previous one or not, using the same method as before. After presenting the result on the screen the program shows text indicating that in order to continue you need to press one of the keys. No 2 second wait is necessary. This is where the algorithm "**wait until no key is pressed**" comes in. Bear in mind that instructions in a program are executed at lightning speed, so it could happen that when you remove the 2 second wait, the program continues with the next loop without pausing to show the result. This is due to the fact that the program may execute in less time than it takes to release any of the keys. EV3-G has an option for the touch sensor which is "wait for press or release", which would solve this issue, but this option is not (yet) available in NEPO. In this case, the difference with the previous logic operator is that now both keys need to be released so we will use the logic operator AND.



How to continue

The examples shown above are not the only two possible algorithms for this challenge and can surely be improved. The following are some ideas to optimise them:

- Modify the program for a third key
- Instead of using a light to identify the right key, use a sound or tones that can be associated to the keys.
- In addition to showing the best result, show the average response time.
- If you press a key too early an error sound is reproduced or you lose the game.
- Convert milliseconds in seconds before showing the result on the screen
- Create the program in a different programming environment
- Have the program ask the user for a name and store both the name and the results in a file. For now this cannot be done in NEPO, but it is possible in other programming environments that allow the use of files

[1] https://issuu.com/koldo_lrobotikas/docs/memorygame
#

About the autor

About.me/koldo_olaskoaga