When I was a child, I used to play the game console of whack-a-mole. The hamster would randomly run out of the hole. When the hamster came out, he would hit it with a mallet. It was a small game to test his reaction. I choose to use LEDs and buttons to simulate the game of whack-a-mole. First, the LED will light up randomly, indicating that the hamster appears. Press the corresponding button to turn off the LED. If successful, the score will increase by 1. This program can also be extended to reaction training lights, which are tools used to train athletes

**Connection:**

**电子游戏截图

中度可信度描述已自动生成**

The first thing to do is to achieve random blinking of the LEDs. I now have 5 LEDs, and I want to be able to light up three randomly, so I start by randomizing three numbers from 1 to 5 that without repeat. After setting the LED pins, etc., use **millis()** to generate a set of random numbers at a fixed time, and light up the ledPin generated by the random numbers.

It should be noted here that random(5) will only randomly generate numbers from 0 to 4.

**Use millis() instead of delay()**

The well-known delay() in Arduino takes the program pause as an argument in milliseconds. millis(), on the other hand, is a function that returns the number of milliseconds that have passed since the program started. At first glance, you might doubt the usefulness of this feature. The truth is that it is very useful in many cases, often &quot;replacing&quot; delay() entirely. First we see how to use millis() almost exactly like delay().

Why use millis() instead of delay()?

Now, we compare millis() with two advantages compared to delay().

Accurate Timing

The first advantage we will discuss is accurate timing. In terms of code, we covered it in the previous chapter. With millis() we can ensure that the loop executes as many times as needed, regardless of the execution time (obviously, as long as the execution time is shorter than the required time, it can be executed). With delay() this is not possible because we can not know how long the loop will take to execute. Such precise timing is useful when sampling or running filters at a certain frequency.

Not blocking

Another benefit of millis() is that it does not prevent us from running code while “waiting”.

Suppose we want to print “Hello” every second while doing other things. It is not possible to delay() because it pauses the entire code. Here is one way we can do this:

This block of code is very similar to the first, except that it does not block the rest of the program when not printing over serial.

In this project, we press the button to turn off the corresponding LED and accumulate points

From the above program, we can know that there will be 3 random numbers stored in the array n[5]. In fact, 3 random numbers will be randomly selected from 0~4, which represents the position in the array.

ledPin[]、ButtonPin[]，The table corresponding to the location is as follows:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| ledPin[] | 12 | 11 | 10 | 9 | 8 |
| ButtonPin[] | 7 | 6 | 5 | 4 | 3 |
| Index position | [0] | [1] | [2] | [3] | [4] |

Suppose when the random number is selected, I get n[0]=3, n[1]=1, n[2]=4, I just need to check ButtonPin[3], ButtonPin[1], ButtonPin[4 ] Check if these three buttons are pressed. If pressed, the corresponding LED will be turned off and the score will be +1.

Finally, an LCD is added to display the score.