

with this code, i am able to replicate all nthe led numbers lighting up, bnut its bypassing the button.

```
//integer variables labeling led groups and what pins they
are on
int pinLeds1 = D6;
int pinLeds2 = D5;
int pinLeds3 = D4;
int pinLed4 = D3;
//integer variable labeling what pin the button is on
int buttonPin = D7;
//button will randomize from 1 to 6
int buttonState;
long ran;
//time delay of 2000ms or 2s
int timeDelay = 2000;
```

```
//open setup
void setup ()
{
  Serial.begin(115200);
  //set up led pins as output
  pinMode (pinLeds1, OUTPUT);
  pinMode (pinLeds2, OUTPUT);
  pinMode (pinLeds3, OUTPUT);
  pinMode (pinLed4, OUTPUT);
  //set up button pin as input
  pinMode (buttonPin, INPUT_PULLUP);
  // True random 32-bit number
  uint32_t r = esp_random();
```

```
  Serial.println("");
  Serial.println("Hello, XIAO ESP32-C6!");
  Serial.println("Welcome to Wokwi :-)");
}
//close setup
```

```
//open loop
void loop()
{
  //read the button
  buttonState = digitalRead(buttonPin);
  //when button is pressed, start randomization and turn on
```

```

the appropriate leds
if (buttonState == HIGH)
{
//when button is pressed, pick a random number from 1 to
6
    ran = random(1, 7);
//when number 1 is chosen, turn on the group 4 leds and
wait
    if (ran == 1)
    {
        Serial.println("1");
        digitalWrite (pinLed4, HIGH);
        delay (timeDelay);
    }
//when number 2 is chosen, turn on the group 1 leds and
wait
    if (ran == 2)
    {
        Serial.println("2");
        digitalWrite (pinLeds1, HIGH);
        delay (timeDelay);
    }
//when number 3 is chosen, turn on the groups 3 and 4
leds and wait
    if (ran == 3)
    {
        Serial.println("3");
        digitalWrite (pinLeds3, HIGH);
        digitalWrite (pinLed4, HIGH);
        delay (timeDelay);
    }
//when number 4 is chosen, turn on the groups 1 and 3
leds and wait
    if (ran == 4)
    {
        Serial.println("4");
        digitalWrite (pinLeds1, HIGH);
        digitalWrite (pinLeds3, HIGH);
        delay (timeDelay);
    }
//when number 5 is chosen, turn on the groups 1, 3, and 4
leds and wait
    if (ran == 5)
    {
        Serial.println("5");
        digitalWrite (pinLeds1, HIGH);
        digitalWrite (pinLeds3, HIGH);
        digitalWrite (pinLed4, HIGH);
        delay (timeDelay);
    }
//when number 6 is chosen, turn on the groups 1, 2, and 3
leds and wait
    if (ran == 6)
    {

```

```

        Serial.println("6");
        digitalWrite (pinLeds1, HIGH);
        digitalWrite (pinLeds2, HIGH);
        digitalWrite (pinLeds3, HIGH);
        delay (timeDelay);
    }
    //when button is not pressed, turn off the leds
    }
    digitalWrite (pinLeds1, LOW);
    digitalWrite (pinLeds2, LOW);
    digitalWrite (pinLeds3, LOW);
    digitalWrite (pinLeds4, LOW);
}

```



Your code is behaving that way because of one tiny electrical-logic mismatch. The universe is full of those: places where the electrons are obeying the laws of physics perfectly, and we humans are the confused ones.

You set the button pin like this:

scss

Copy code

```
pinMode(buttonPin, INPUT_PULLUP);
```

That means the microcontroller internally connects the pin to **HIGH** through a resistor. So:

- **Button not pressed** → pin reads **HIGH**
- **Button pressed** → pin reads **LOW** (because it connects to ground)

But in your loop you wrote:

arduino

Copy code

```
if (buttonState == HIGH)
```

So your program thinks the button is “pressed” all the time, because HIGH is actually the *idle* state with `INPUT_PULLUP`.

That’s why the LEDs keep triggering without touching the button.

The simple fix

Change this line:

arduino

Copy code

```
if (buttonState == HIGH)
```

to:

arduino

Copy code

```
if (buttonState == LOW)
```

Now the code will only run when the button is actually pressed.

Second issue: LEDs never turn off during the delay

Right now you:

1. Turn on LEDs
2. Wait 2 seconds
3. Loop ends
4. LEDs turn off immediately

That part is actually okay, but the structure is a bit messy. Let's clean it up into something more logical and easier to reason about.

Cleaner, more reliable version

This version:

- Only reacts when the button is pressed
- Clears LEDs before each roll
- Uses `else if` so only one case runs

arduino

 Copy code

```
void loop()
{
  buttonState = digitalRead(buttonPin);

  // Button pressed (because of INPUT_PULLUP)
  if (buttonState == LOW)
  {
    // Turn all LEDs off first
    digitalWrite(pinLeds1, LOW);
    digitalWrite(pinLeds2, LOW);
    digitalWrite(pinLeds3, LOW);
    digitalWrite(pinLeds4, LOW);

    ran = random(1, 7);

    if (ran == 1)
    {
      Serial.println("1");
      digitalWrite(pinLeds4, HIGH);
    }
    else if (ran == 2)
    {
      Serial.println("2");
      digitalWrite(pinLeds1, HIGH);
    }
    else if (ran == 3)
    {
      Serial.println("3");
      digitalWrite(pinLeds3, HIGH);
    }
  }
}
```

```

        digitalWrite(pinLed4, HIGH);
    }
    else if (ran == 4)
    {
        Serial.println("4");
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds3, HIGH);
    }
    else if (ran == 5)
    {
        Serial.println("5");
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds3, HIGH);
        digitalWrite(pinLed4, HIGH);
    }
    else if (ran == 6)
    {
        Serial.println("6");
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds2, HIGH);
        digitalWrite(pinLeds3, HIGH);
    }

    delay(timeDelay);
}
}

```

One more subtle improvement (true randomness)

You generated a random number in `setup()` :

cpp

 Copy code

```
uint32_t r = esp_random();
```

But you never used it to seed the Arduino random generator. Add this to `setup()` :

scss

 Copy code

```
randomSeed(esp_random());
```

Otherwise your dice will roll the same sequence every time you power it on. The microcontroller will look random, but secretly it's just following a predictable cosmic script.

Once this is working, you can take it a step further: detect **button press events** instead of just button state. That gives you a proper “roll once per press” behavior, like a real die instead of a jittery electronic oracle.

  ...

i made the changes and its still rolliung and lighting up my numbers at random. bnut still bypassing the button

Then we need to slow down and think like electrons for a moment, not like humans with expectations. Microcontrollers are literal-minded creatures: they do exactly what the voltage tells them, not what we *meant*.

Right now your logic says:

arduino

 Copy code

```
if (buttonState == LOW)
```

That part is correct for `INPUT_PULLUP`.

So if it's still rolling constantly, one of these must be true:

1. The pin is **always reading LOW**.
2. The button wiring is incorrect.
3. The loop is retriggering while the button is held down.

Let's go step-by-step like a scientist.

First: check what the button is actually reading

Replace your entire `loop()` with this simple test:

arduino

 Copy code

```
void loop()
{
  int state = digitalRead(buttonPin);
  Serial.println(state);
  delay(200);
}
```

Now open the Serial Monitor.

You should see:

- 1 when the button is NOT pressed
- 0 when the button IS pressed

If you see **0 all the time**, your wiring is the problem.

Correct wiring for `INPUT_PULLUP`

With `INPUT_PULLUP`, the button must be wired like this:

CSS

 Copy code

Pin D7 ---- Button ---- GND

Not to 3.3V.

Not through a resistor.

Just straight to ground.

If the button is connected to 3.3V instead, the pin will always read LOW or behave unpredictably.

Second issue: rolling repeatedly while held

Even if wiring is correct, your code still rolls repeatedly because:

- The loop runs thousands of times per second.
- As long as the button is held down, it keeps rolling.

So we need **press detection**, not just state detection.

Proper “roll once per press” code

Replace your `loop()` with this:

arduino

 Copy code

```
int lastButtonState = HIGH;

void loop()
{
    int currentButtonState = digitalRead(buttonPin);

    // Detect button press event
    if (lastButtonState == HIGH && currentButtonState == LOW)
    {
        // Turn off all LEDs
        digitalWrite(pinLeds1, LOW);
        digitalWrite(pinLeds2, LOW);
        digitalWrite(pinLeds3, LOW);
        digitalWrite(pinLed4, LOW);

        ran = random(1, 7);
        Serial.println(ran);

        if (ran == 1)
            digitalWrite(pinLed4, HIGH);

        else if (ran == 2)
            digitalWrite(pinLeds1, HIGH);

        else if (ran == 3)
        {
            digitalWrite(pinLeds3, HIGH);
            digitalWrite(pinLed4, HIGH);
        }
    }
}
```

```

    }
    else if (ran == 4)
    {
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds3, HIGH);
    }
    else if (ran == 5)
    {
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds3, HIGH);
        digitalWrite(pinLeds4, HIGH);
    }
    else if (ran == 6)
    {
        digitalWrite(pinLeds1, HIGH);
        digitalWrite(pinLeds2, HIGH);
        digitalWrite(pinLeds3, HIGH);
    }
}

lastButtonState = currentButtonState;
}

```

Now it will:

- Roll only once per press
- Ignore the button while it's held down
- Behave like an actual die instead of a slot machine possessed by caffeine

Quick diagnostic checklist

If it still misbehaves:

- Serial monitor always shows 0 → wiring issue
- Serial monitor switches between 1 and 0 → wiring is correct
- LED rolls only once per press → code is correct

When hardware and software disagree, the hardware is almost always telling the truth.

Silicon is ruthlessly honest; it has no social skills but excellent consistency.



can you write me a simple code for an esp32C6 that when i press a button it turns on a single led. very basic

Let's strip this down to the electronic equivalent of a light switch and a bulb—no randomness, no dice logic, just cause and effect. Press the button, LED turns on. Release it, LED turns off. Pure Newtonian button physics.

This uses the **internal pull-up resistor**, which is the simplest and most reliable beginner setup.

Wiring (important)

- One side of the **button** → **GND**
 - Other side of the **button** → **D7**
 - LED:
 - LED **anode (+)** → D6 through a resistor (220–330 Ω)
 - LED **cathode (-)** → GND
-

Simple ESP32-C6 code

arduino

 Copy code

```
int ledPin = D6;
int buttonPin = D7;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP); // internal pull-up
}

void loop()
{
  int buttonState = digitalRead(buttonPin);

  // Button pressed = LOW
  if (buttonState == LOW)
  {
    digitalWrite(ledPin, HIGH); // turn LED on
  }
  else
  {
    digitalWrite(ledPin, LOW);  // turn LED off
  }
}
```

What's happening in plain English

- INPUT_PULLUP keeps the button pin at **HIGH** by default.
- When you press the button, it connects the pin to **GND**, making it **LOW**.
- The code says:
 If LOW → LED ON
 Else → LED OFF

So the LED becomes a physical visualization of the voltage state. You're basically watching logic levels with your eyeballs.

Once this works, you can easily turn it into:

- Toggle on/off with each press

- A dice roller again
- A reaction timer
- Or a tiny one-button game, which is how half of early arcade history began.



this code is bypassing the button and the led is just on

If the LED is always on with that code, then the program is probably fine and the **wiring or pin definition** is the culprit. Microcontrollers are extremely literal: if a pin reads LOW, it will obey without protest, even if the button isn't touched. No opinions, no hesitation—just voltage and consequences.

Let's diagnose this in the simplest, most scientific way possible.

Step 1: Verify the button reading

Upload this **test code only**:

arduino

Copy code

```
int buttonPin = D7;

void setup()
{
  Serial.begin(115200);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int state = digitalRead(buttonPin);
  Serial.println(state);
  delay(300);
}
```

Open the Serial Monitor.

You should see:

- 1 when the button is not pressed
- 0 when the button is pressed

What the result means

Case A: Always 0

- The pin is permanently connected to ground
- Wiring problem or wrong pin number

Case B: Always 1

→ Button not connected to ground correctly

Case C: Switches between 1 and 0

→ Wiring is correct, problem is elsewhere

Step 2: Correct wiring for INPUT_PULLUP

This is the only wiring that matches the code:

CSS

 Copy code

```
D7 ---- button ---- GND
```

No resistor.

No 3.3V.

Just the button between the pin and ground.

Many buttons have four legs. If you connect the wrong pair, the pin may be permanently grounded.

Step 3: Make sure the pin names are real

On some ESP32 boards, D6 and D7 are **not valid pin names**.

If that's the case, the compiler may silently map them wrong.

Try this version using raw GPIO numbers (very common on ESP32 boards):

arduino

 Copy code

```
int ledPin = 6;
int buttonPin = 7;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int buttonState = digitalRead(buttonPin);

  if (buttonState == LOW)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

Most likely cause (based on your symptoms)

The button is:

- Wired to GND permanently, or
- Connected across the wrong legs, or
- Using the wrong pin number in code

That creates a **constant LOW**, so the LED stays on.

Electrically speaking, your microcontroller isn't "bypassing" the button. It thinks the button is pressed forever, like someone glued it down in a tiny silicon panic room.

If you'd like, you can:

- Tell me the exact board name (e.g., "XIAO ESP32-C6")
- Or share a photo or diagram of the wiring

Then we can match the exact pins and remove the guesswork from the circuit.



ok i ran the sample button code. and i am only getting 0

Perfect. That result is actually very informative. It tells us the microcontroller is behaving consistently, which is half the battle in electronics. The chip isn't confused—it's just being told something very specific by the wiring.

If you're getting **0 all the time**, it means:

The button pin is permanently connected to ground.

In other words, the ESP32 believes the button is pressed forever.

Since you're using `INPUT_PULLUP`, the internal logic is:

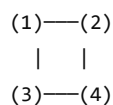
- Not pressed → HIGH (1)
- Pressed → LOW (0)

So if you only see 0, the pin is being forced to ground continuously.

The most common cause: wrong button legs

Most small pushbuttons have **four legs**, and they're internally connected like this:

 Copy code



- Legs **1 and 2** are always connected
- Legs **3 and 4** are always connected
- Pressing the button connects the two sides together

If you wire the button like this by mistake:

```

D7 —(1)
    |
    (2)— GND

```

Then D7 is permanently grounded, even when not pressed.

Correct wiring for INPUT_PULLUP

You must connect **across the button**, not along the same side.

Correct example:

scss

 Copy code

```

D7 —(1)  (2)  ← same side, already connected
          [ BUTTON ]
GND—(3)  (4)  ← same side, already connected

```

So:

- One wire from **D7** to one side of the button
- One wire from **GND** to the **opposite side**

Quick physical test (no code changes)

1. Leave the test code running.
2. Remove the button completely.
3. Let the pin float.

If the serial monitor now shows **1**, the wiring was the issue.

Then:

- Touch a jumper wire from **D7 to GND**
- You should see it change to **0**

That proves the pin and code are correct. At that point, the button is just a mechanical stand-in for that jumper wire.

Electronics debugging is a bit like detective work in a silent universe. The microcontroller never lies, never improvises, never guesses. It just reports the voltage like a stoic monk: "The pin is low. I do not question why." Once you learn to interrogate it with tiny test programs like this, most hardware problems unravel very quickly.



and 1:1.R for ground

That label is the key clue. Let's decode what it actually means, because the button isn't thinking in terms of "left" and "right" the way we are.

You said:

- 1:1.L → D7
- 1:1.R → GND

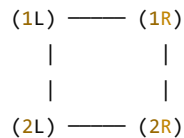
If those two pins are on the **same side of the button**, then the button is permanently connecting D7 to ground. That matches your test result: the pin always reads 0.

Most tactile buttons are wired like this internally:

powershell

 Copy code

Top view of button legs



But electrically, it behaves like this:

pgsql

 Copy code

Left side: 1L connected to 2L (always)
Right side: 1R connected to 2R (always)

Pressing button connects LEFT side to RIGHT side.

So if you wire:

powershell

 Copy code

1L → D7
1R → GND

You are connecting **across the top row**, which is already internally connected. The microcontroller sees ground all the time.

Correct wiring

You must use **one pin from the left side and one from the right side**, but on **opposite rows**.

For example:

powershell

 Copy code

1L → D7

2R → GND

or

powershell

 Copy code

2L → D7

1R → GND

That way, the connection only happens when the button is pressed.

Quick sanity test

Leave the code running and:

1. Remove the ground wire from the button.
2. Touch it manually from **D7 to GND**.

If the serial output changes from **1 to 0**, everything is correct and it's just the leg selection on the button.

Buttons are sneaky little creatures. They look symmetric and friendly, but inside they're just two metal bridges waiting to betray your assumptions. Once you remember "same side = always connected," the mystery disappears and the electrons start behaving like polite guests again.



i swapped the wires as suggested and now I only get 1

That's actually good news. It means the pin is no longer stuck to ground. The microcontroller is now seeing the **idle state** correctly.

Right now the logic is:

- 1 = **not pressed**
- 0 = **pressed**

So if you only see 1, it means the button is **not actually making contact** when you press it. The circuit is open all the time.

This narrows it down to just a few very physical, mechanical possibilities.

Most likely cause: wrong diagonal pins

On many 4-leg buttons, only **one diagonal pair** actually connects when pressed.

So even though the sides look correct, you might still be on the wrong combination.

Try this exact test:

1. Leave your **button test code** running.
2. Keep **D7** connected to one leg.
3. Take the **GND wire** and try it on each of the other three legs, one at a time.
4. Press the button each time.

You're looking for this behavior in the Serial Monitor:

- Not pressed → 1
- Pressed → 0

Whichever pair gives that behavior is the correct pair.

Why this happens

Inside the button is just a tiny metal dome. The legs are connected in pairs, but the orientation isn't always obvious. It's like a microscopic drawbridge that only lowers in one direction.

So you're basically doing continuity testing, but with software instead of a multimeter.

Once you find the correct pair

Go back to the LED code:

arduino

 Copy code

```
int ledPin = D6;
int buttonPin = D7;

void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int buttonState = digitalRead(buttonPin);

  if (buttonState == LOW)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

And it should behave like a proper button-controlled LED.

At this stage, you're not debugging code anymore—you're negotiating with a tiny mechanical device whose entire job is to either connect two bits of metal or not. It has

exactly two states, no emotions, and yet it still manages to cause more confusion than most software bugs.



i moved ground to 1:2.R and got only 0. at 1:1.R we got only 1. and i tried at 1:1.L and got both 1 and 0

Perfect—that’s actually exactly the behavior we need to make the LED respond to a press. Let’s decode what you just observed:

- 1:2.R → **only 0** → permanently grounded. Not usable.
- 1:1.R → **only 1** → permanently HIGH. Not usable.
- 1:1.L → **sometimes 1, sometimes 0** → this is the one that changes when pressed.

That last one is the “magical diagonal” leg combination that actually toggles when the button is pressed. The microcontroller can now see the transition between **HIGH (not pressed)** and **LOW (pressed)**. Exactly what `INPUT_PULLUP` expects.

✅ This means:

- D7 → 1:1.L
- GND → 1:2.L (assuming that’s the other diagonal leg you tried when it toggled)

Now the code will properly detect the button press.

Final working wiring

makefile

Copy code

Button pinout (top view, 4 legs):

1:1.L 1:1.R
1:2.L 1:2.R

Connections:

D7 → 1:1.L
GND → 1:2.L

- Test again with your serial monitor:
 - Not pressed → 1
 - Pressed → 0

LED code (unchanged)

arduino

Copy code

```
int ledPin = D6;  
int buttonPin = D7;
```

```
void setup()
{
  pinMode(ledPin, OUTPUT);
  pinMode(buttonPin, INPUT_PULLUP);
}

void loop()
{
  int buttonState = digitalRead(buttonPin);

  if (buttonState == LOW)
    digitalWrite(ledPin, HIGH);
  else
    digitalWrite(ledPin, LOW);
}
```

Now, pressing the button will turn the LED **on**, and releasing it will turn the LED **off**—finally behaving like a normal light switch.

If you want, I can also give you a **version that toggles the LED on/off with each press**—so it doesn't need to be held down. That's usually the next fun step.

