Frame-rate independent FixedUpdate vs Update

Asked 5 years, 9 months ago Modified 5 years, 9 months ago

Viewed 4k times

3

I've already read this and the official documentation: fixedUpdate(), and the deep explanation.

So I've tried to separate my code. First, in `Update()`, I dont give the full code the variables are self-explanatory:

```
private void Update()
{
    if (Input.GetButton("Jump")) {
        if (groundsTouched>0) {
            _jumpRequest = true;
        } else {
            _keepOnJumping = true;
        }
    } else {
        _keepOnJumping = false;
    }
    /* Handle release button: */
    _fallRequest = true;
}
```

And now I'm doing all the calculations in the `FixedUpdate()` like this:

```
private void FixedUpdate()
{
    if (_jumpRequest) {
```

```
        if (!_jumpGravitySent) {
            _jumpGravitySent = true;
            _animator.SetBool("Jump", true);
            _jumpRequest = false;
            jumpTimeCounter = jumpTime;
            /* Cancel all force (couldn't find a
better way) */
            _rigidbody.velocity = Vector3.zero;
            _rigidbody.angularVelocity =
Vector3.zero;
            _rigidbody.AddForce(
                Vector3.up * jumpVelocity,
ForceMode.VelocityChange
            );
        }
    } else if (_keepOnJumping) {
        jumpTimeCounter -= Time.fixedDeltaTime;
        if (jumpTimeCounter >= 0) {
            _rigidbody.AddForce(
                Vector3.up * jumpVelocity *
jumpKeepMultiplier,
                ForceMode.Acceleration
            );
        }
    }
    if (groundsTouched == 0 &&
        _rigidbody.velocity.y > velocityFallMin &&
        _rigidbody.velocity.y < velocityFallMax
    ) {
        _animator.SetBool("Jump", false);
        _animator.SetBool("Fall", true);
    }
    if (_fallRequest) {
         fallRequest = false;
```

The problem I've faced was really strange: when the FPS was low, the player couldn't jump high.

Unity QA saw my problem and their answer was:

> You are adding Force dependent on fixedDeltaTime which is dependent on your available performance (or frame rate essentially).
>
> If you go to Edit->Project Settings->Time and change fixed Timestep to a larger value, you will get the expected behavior. Try a few different values of Fixed Timestep and see how the behavior changes.
>
> Another suggestion would be to rewrite the code so it's not dependent on frame rate (eg. use Velocity and not force or add a specific amount of force on jump, not dependent on Timestep).

"Another suggestion would be to rewrite the code so it's not dependent on frame rate" -> how would you do that, I thought that my code above is doing it!

What am I missing? What am I doing wrong / what could be the solution?

unity-game-engine

Share

Improve this question

Follow

edited Mar 9, 2019 at 3:50

asked Mar 8, 2019 at 15:12

Olivier Pons

One thing you can try is using `Time.deltaTime` inside of `FixedUpdate()`. It might be the same value. Another thing you can try is `Rigidbody.MovePosition()` instead of `AddForce()`. You could also try a different force mode in `AddForce()` like `ForceMode.Force` or `ForceMode.Impulse`. Is your mass of yoru object set too high? – Eliasar Mar 8, 2019 at 22:26

@Eliasar Thanks for your suggestion but I've already tried everything but `MovePosition()`. I'll try this tonight. For the other suggestions, if you watch carefully my code you will notive that I only call `Time.fixedDeltaTime` when I do `jumpTimeCounter -= Time.fixedDeltaTime`, this is the only time where I do this. So with `deltaTime` this doesn't change the problem, unfortunately. – Olivier Pons Mar 8, 2019 at 22:46

1 Answer

Sorted by: Highest score (default)

7

I wonder who's that Unity QA that gave you that answer, it's cringeworthy.

Let me explain what happens, *in reality*.

1) Let's start with `fixedDeltaTime`: **this value is NEVER dependent on frame rate**. It can be set in the Editor (in `Edit->Project Settings->Time`), and that's the value that's kept at runtime, *unless* any script changes it with an assignement. The Unity engine never changes it by itself.

2) The Physics loop: in a full engine loop, Unity will execute a number of Physics loop (0,1, or more) and then a single Rendering loop. The number of Physics loops executed per Rendering loop is based on that `fixedDeltaTime` and how much time has passed since the last one (i.e.: the `deltaTime` of the Rendering loop).

For example, let's say `fixedDeltaTime = 0.0166667` and the time passed since the last Physics loop it's less than that, say `10`ms. Unity won't execute the Physics loop. Now suppose that even the next frame has been rendered in `10`ms - this means that since the last Physics loop, `20`ms have passed. Since this is higher than the `fixedDeltaTime`, Unity will execute a Physics loop. Sometimes it may happen that a frame is rendered very slowly (due to unforeseeable reasons), for example in `40`ms. To keep the physics simulation consistent, Unity has to run *two* Physics loop in a row, that's because `0.04/0.0166667 = 2.4`.

Keep in mind that Unity keeps track of the difference between the last Physics loop start time and the next one: if the rendering lasts `10`ms every frame, and `fixedDeltaTime` is set to `166667`ms (60Hz), as soon as you start the runtime, Unity will execute the first Physics loop, then skip one after the 1st rendering frame (because only `10`ms have passed instead of `166667`), then execute one Physics loop after the 2nd rendering frame (`20`ms passed against the `166667`). But now we have the loop desynchronized by `3.3333`ms, so Unity will keep track of that.

After the 3rd frame, another `10`ms have passed, but no Physics loop will be executed since `10+3.3333 = 13.3333` which is still lower than the `fixedDeltaTime`. Now, let's suppose that the 4th rendering frame "goes wrong", and it lasts `25`ms instead of just `10`. At the start of the next Physics loop, a total of `25+13.3333 = 38.3333` have passed since the last Physics update, and `38.3333/16.6667 = 2.3`, and Unity will execute **two** Physics loop in a row to keep up with the fixed step simulation, before going on in rendering the 5th frame.

After all this intro, let's get back to your problem, and look at what happens:

At a certain point you execute `Update()`, and set `_jumpRequest = true;` and `_fallRequest = true;`.

After this rendering frame, `FixedUpdate()` is executed for the first time, executing the `AddForce ForceMode.VelocityChange` line, and setting `_fallRequest = false;`, `_jumpGravitySent = false;` and `_keepOnJumping = false;`. After the end of this `FixedUpdate()`, Unity executes the physics simulation, adjusting the position and velocities of the rigidbody thanks to the physics engine.

Now the problem triggers: since the rendering frame was slow, the physics loop is executed at least twice in a row, but no `Update()` is executed in between, so everything in `FixedUpdate()` is skipped, but the physics simulation is

run for a 2nd time, draggin the rigidbody position down in respect to the intended, topmost position.

When `Update()` is executed again, finally your code sets `_keepOnJumping = true;`, and when it gets back to `FixedUpdate()` it will execute the `AddForce ForceMode.Acceleration`, but immediately after that, another physics simulation is executed for the 2nd time (due to low frame rate), dragging again the rigidbody down before it can be rendered on screen.

Hope this helps in understanding your problem and why it happens, so that you now have the right tools to fix it properly.

Share Improve this answer

Follow

answered Mar 9, 2019 at 1:27

Galandil

4,249 ● 1 ● 16 ● 24

The problem was there: `jumpTimeCounter -= Time.fixedDeltaTime`. I'm still having hard times to use timing properly. I've managed to see more clearly the problem using 4 buttons, each one setting a different `Time.timeScale`: 0.2, 1, 2, 4. It seems for what I want I can't rely on time: using `fixedXxx` or not, using almost all the properties here: docs.unity3d.com/ScriptReference/Time.html I didn't find a viable solution... the solution is to work on the max height of what the player can reach as long as he presses jump.
– Olivier Pons Mar 9, 2019 at 5:49

FYI I've made everything work not on a time basis, but on a *destination* in space basis. For example, when the player press "Jump", I define a "`max Y`" to reach before falling, and

as long as he keeps the "Jump" pressed, I change the velocity. If the "`max Y`" is reached, then I ignore the "Jump" and the player falls down. If the "Jump" is released in-between, I stop changing the velocity. Thus it's not a matter of time anymore but a matter of position. Same principle for monster generation: I generate a new them when a monster reached a certain position. – Olivier Pons Mar 13, 2019 at 8:18