NET REWINDER

Table Of Contents

Introduction

Net Rewinder is a server side hitbox rewinder / lag compensator for use with any networking library. It focuses on doing only one thing, and doing that one thing simply and well. One function call rewinds hitboxes (your selected transforms) to exactly where they were at the time you specify. You can use any representation of hitboxes you want, any networking library, and any hit detection routines you want. You can run whatever tests you want against them, and then restore them to their original positions with another call.

Features:

- One-button setup for most use cases
- One-call hitbox rewind
- One-call hitbox restore
- Full source code included
- Use any hitbox system
- Zero update allocations
- Supports 2D and 3D - anything with a transform can be rewound
- Preview demo scene with commented code
- Works with any network library (**UNet**,**Photon**, **Bolt**, **Forge**, etc)
- Works with **UFPS** and other FPS frameworks.

Links:

- Webplayer demo

Usage Instructions

Using Net Rewinder is fairly simple, and falls into two basic stages:

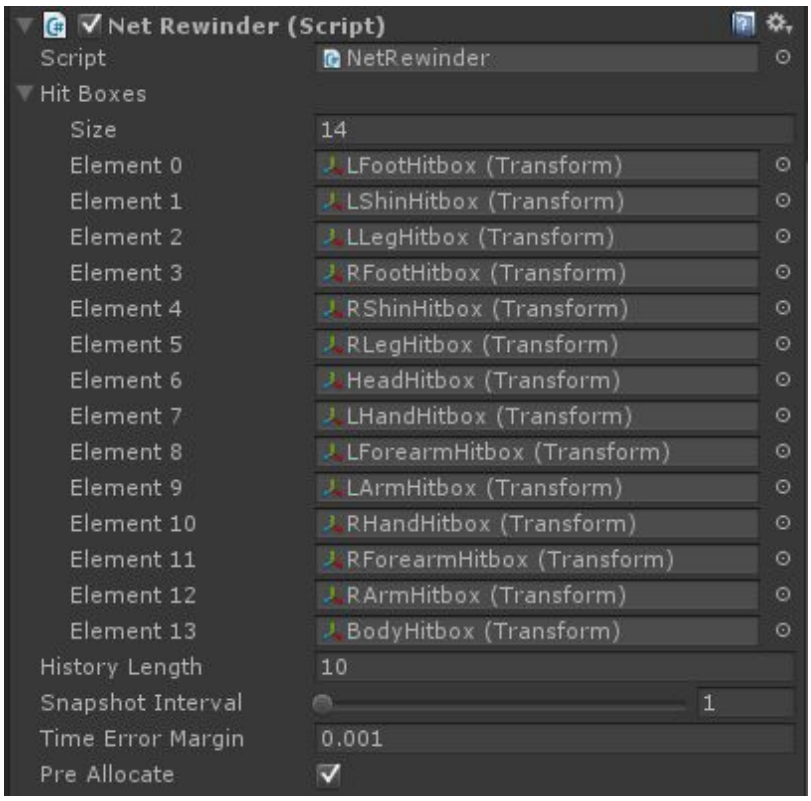
- Preparing the character gameobject.
- Writing code to use Net Rewinder's rewind feature.

Preparing Your Character GameObject

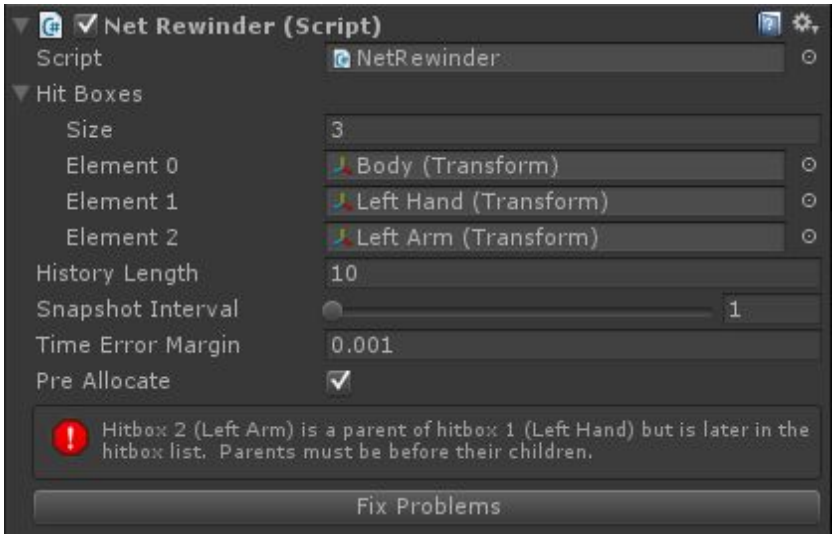
Preparing a gameobject for use with Net Rewinder consists of the following steps:

Add a Net Rewinder component to your gameobject.

Assign hitboxes. You can do this manually by dragging the desired transforms into the Hit Boxes list. Alternately, if you have colliders on all your hitboxes, you can click Find Hitboxes, and the Net Rewinder should find all your hitboxes for you.



Note that if any of your hitboxes are children of each other, the parents will need to be added to the list first to ensure accurate rewinding. Don't worry about this too much though, since Net Rewinder will automatically detect problems and offer to fix them for you.



Set the History Length, Snapshot Interval, and Time Error Margin as desired.

Here's what these settings do:

History Length:
This determines the length of history in seconds that Net Rewinder will remember. The higher this value is, the more memory Net Rewinder will use. You'll want to make sure this value is high enough to allow you to rewind to the oldest snapshot you'll ever need. The largest value you'll need here is approximately 2x the highest connection latency you plan to support. The default (10s) is very generous, and can safely be lowered in most games.

Snapshot Interval:
Net Rewinder times snapshots using the physics timescale. This is how many intervals will pass between snapshots. A snapshot interval of 1 will take a snapshot every physics frame (.02s by default). 2 will take a snapshot every second frame, and so on. The lower this number, the more accurate (and memory intensive) Net Rewinder will be.

Time Error Margin:

Net Rewinder can rewind to interpolated time points between snapshots. Any requested rewind point that is closer to a snapshot than the Time Error Margin will be rewound to the closest recorded snapshot rather than an interpolated value. The default value (.001s) is generally fine here.

Using Net Rewinder In Your Game Loop

In general, use of Net Rewinder consists of the following server-side steps:

- Rewind hitboxes to the desired time
- Perform hit calculations
- Restore hitboxes to original state

Rewind Hitboxes

Rewinding the hitboxes on your character requires a single call:

```
// Rewind hitboxes to the desired time
if (rewinder.Rewind(rewindTargetTime)) {
    // NetRewinder.Rewind will return true on success.
```

NetRewinder will return true if successful. If you request an invalid time (in the future, or too far in the past), NetRewinder will return false. This lets you decide how to handle invalid rewind requests.

Perform Hit Calculations

Once rewound, you can perform whatever calculations your game requires to validate a hit check. If you're converting a single-player game to multiplayer, or adding lag compensation to an existing multiplayer game, you'll want to move your existing hit calculations to occur right after the rewind call. Here's the code the demo uses to display a preview of the rewound hitboxes

```
        // Update preview elements
        // In a real scenario, you'd be doing your hitbox raycasts etc here
        SetPositionsAndRotations(hitboxPreviews, hitboxes);
```

Restore Hitboxes

Once you've performed your hit tests, you need to restore the hitboxes to their original location. This is as simple as calling the Restore method on the attached NetRewinder component:

```
// Restore hitboxes to their original positions
// It's important to do this during the same frame as the rewind
rewinder.Restore();
// NetRewinder.Restore also returns true on success, though it's not typically necessary to test for this.
// Attempting to restore when not in a rewound state will return false, and add a warning to the debug log.
// Note that you can perform multiple rewinds of the same object to different time points, then a single restore afterwards.
// This can reduce performance impact if you receive multiple shot notifications from clients in the same frame
```

Some Things To Note

There are a few things to be aware of when using Net Rewinder:

Update vs LateUpdate

If rewinding objects on very low-latency connections (eg LAN), it's possible that you'll be rewinding to a time that is more recent than the last snapshot. Net Rewinder handles these situations cleanly by interpolating between the most recent snapshot and the current gameobject's state. Due to the way Unity handles animations, however, in some situations Transform positions can be inaccurately reported during Update. For this reason, it is recommended that you perform hit tests in LateUpdate rather than Update.

Failed Rewinds

In the event of a failed rewind (requested time too old, or requested time in the future), you'll want to take appropriate action. This might be something as simple as skipping the hit checks and ignoring the event that prompted the rewind, though you may want to take other action. In the demo scene, we just ignore the invalid request:

```
} else {
    // NetRewinder.Rewind returns false on failure.  This will only happen if the rewind time is older
    // than the oldest time in the buffer (buffer length is set in seconds, default 10 s), or if the
    // rewind time is in the future.  In either case, NetRewinder.Rewind will show a detailed warning
    // in the log to help you figure out what went wrong.

    // You would insert any special logic here that you want to execute if the rewind is somehow invalid.
    return;
}
```

Timing

It's important to note that you need to always perform a Restore() in the same frame you perform Rewind(), before any other scripts attempt to move the gameobject or its hitboxes. You can perform multiple Rewind() calls on the same object in the same frame if you need to (unlikely, but you can do it if you want), and follow them with a single Restore(). However, to avoid error conditions, that Restore() call must happen before NetRewinder attempts to take its next snapshot during FixedUpdate(), or any other scripts attempt to move the gameobject's transforms.

API Reference

NetRewinder

Inspector Properties

NetRewinder's inspector properties determine its operating parameters. It is important that these values don't change after NetRewinder is initialized.

hitboxes *Transform[]*

The transforms that NetRewinder will track and rewind.

historyLength *float*

Length of recorded history in seconds. 10 seconds by default, this value needs to be high enough that it will always include your requested rewind time. A good value for this is somewhere above double the maximum connection latency you intend to support.

snapshotInterval *int*

This is the number of physics frames between snapshots. A value of 1 will result in a snapshot being taken every frame, 2 every second frame, etc. This combined with historyLength and the number of hitboxes being tracked determines how much memory is used for each gameobject. Lower values for snapshotInterval result in more accurate tracking, and consume more memory.

timeErrorMargin *float*

Any requested rewind time within this value of a saved snapshot will result in a rewind to that snapshot rather than to an interpolated value. The default value is generally fine.

Update Loop

LateUpdate()

During LateUpdate, every *snapshotInterval* frames, NetRewinder takes a new snapshot and adds it to the end of the snapshot list, overwriting the oldest snapshot if the list is already full. This is done in a zero-allocation manner, to avoid garbage collection. If the list isn't full (true for the first several snapshots if *preAllocate* is false), a new snapshot will be allocated, but it will remain in use until the gameobject is destroyed, so no garbage collection will be triggered until then.
If NetRewinder is in a rewound state, no snapshot will be taken, and an error will be added to the debug log.

Methods

bool Rewind(*float targetTime*)

Assuming a valid *targetTime* is provided (not earlier than the oldest snapshot, not later than Time.time), Rewind will take a temporary snapshot of the hitbox transforms, followed by rewinding their positions to their values as of *targetTime* as follows:

- If the gameobject is already in a rewound state, no temporary snapshot will be taken, and the rewind will proceed as detailed below.
- If *targetTime* is within *timeErrorMargin* of a saved snapshot, that snapshot's positions and rotations will be used directly.
- If *targetTime* is later than the most recent snapshot, a snapshot interpolating between the most recent snapshot and Time.time will be used for the rewind.
- Otherwise, Rewind will interpolate positions and rotations between the two closest snapshots.
- On success, Rewind will return *true*.

This is done without performing any allocations, to avoid garbage collection.
If the requested time is invalid, no action will be performed, and Rewind will return *false*.

bool Restore()

Assuming the gameobject is in a rewound state, its hitboxes will all be returned to the positions they were in before Rewind, and Restore will return True.
If the gameobject is not in a rewound state, Restore will return false, and an error will be added to the debug log.

Runtime Setup Methods

The methods below are not required during normal operation. You only need to use them if you want to change the setup of a NetRewinder component at runtime.

void Init()

Reinitialize the NetRewinder component after making configuraiton changes.

void GetHitboxes()

Collect all colliders under the NetRewinder component and use them as hitboxes. Use this if you want to set up the hitboxes at runtime. Calls Init() afterwards to ensure setup is correct.

void SetHitboxes(*List*<*Transform*> boxes)

Use the provided list of Transforms as hitboxes. Use this if you want to change the hitboxes at runtime. Calls Init() afterwards to ensure setup is correct.