2023

Deterministic Replay: Performance Report

COMPLEX GAME SYSTEMS

CONNOR MILLS

Table of Contents

Overview

The modular system that I have created is a Deterministic Replay System. A deterministic replay is a technique that provides the user with deterministic executions of computer programs while nondeterministic factors are still present. That is to say that it provides the user with the exact same execution of computer programs irrelevant of any factors.

My modular system contains a variety of scripts that the player can use to create recordings of their game or application. The user can choose to add a recorder script onto a game object to capture that specific object's movement or they can add a tag to a group of game objects and have a recording manager handle recording all the tagged objects.

Demo Application

I have used my replay system to create a demo racing game where the player can race against ghosts of their previous runs. I used a recorder scripts to capture; the car's main body's position and rotation, each of the wheel's position and rotation, the two skid mark activation and deactivation state, and the drift smoke activation. This allows the game to record the players run through the tack and play it back to them and make the players feel like they are versing an actual player.

Issues Encountered

A number of issues were encountered during the creation and integration of the Modular Replay System. Such as when I was creating the modular system, I was using a Unity project that was using Universal Render Pipeline and when I tested the system in a brand new Unity project I got errors from the materials. The issues was that since I was using URP the materials were converted but since the new project was not using URP it broke the materials. This was an easy fix as I just converted the materials back to using the unity default render pipeline and left it for the user to decide if they want to convert them to URP.

During the creation of the replay system, I encountered difficulty in making the system as modular as possible and as simple for the user to use. To allow the user to be able to use my system in most cases it needed to be able to record a wide range of variables and states. I found a number of solutions to this problem.

The first solution was to create a simple one trick pony, a recorder that just recorded the position, rotation and scale of a game object then a replay object that moved the attached game object in the recorded motion. This simplified the steps the user needed to go through to get what they wanted.

The second solution I found was to make a recording manager that took a tag and then looked through all the game object in the scene and if they were tagged with the same tag they would be recorded. This was even more simple than the first solution as the user didn't need to put the recorder script on the game objects or create a replay object. The only downside to this was that the user would have very little control over how the replay objects would look.

The third solution I came up with was to create base classes for all the different situation I could think of. This allowed the user to create their own recorder and replay object to fit any situation that they had. I tried to make it as easy and intuitive to make the scripts by using abstract classes and abstract functions that would force the user to use the correct functions. The downside of this method was that it needed the user to have an understanding of how the code.

Required Changes

The only change that I needed to implement into my modular complex game system for it meet my outlined objective was to add a tag recording manager. Without the tag recording manager added to my system it was not keeping with making the system as modular and easy as possible to use. The tag recording manager added that extra method that filled all if not most of the solutions the system would be used for.

Performance

The performance of my system comes in two parts, the smoothness of the replay and the memory storage needs of the recordings. Since I didn't do anything to impact the smoothness of the replay and that the system records every frame. The performance of the system in respects to smoothness was at the benchmark and it didn't over or under perform.

The performance of the system in relations to the memory storage needs was a bit more complicated to test. I created a benchmark for my system by running it without any optimizations to the memory use. I then run my system with the optimizations and then compered it to the benchmark. It had made a massive improvement to downsizing the memory usage.

Future Improvements

There are several areas that I would like to make improvements to. For instance, I would like to add a resource retriever where the user could create materials for all the ghost objects. So that when any replay object is created, the material of the original object would be read and then the corresponding ghost material would be retrieved and used. This would allow the user complete control of how the ghost look in any of the system even the tag recording manager. It would also simplify the steps that the user needs to follow by just removing having to create a replay object.

Another area I would like to improve on is allowing the user to set the interval in-between each recorded frame. This would allow the user to shrink the space that the recordings would take up in their computer's memory. I could then when playing the recording, lerp between the two recorded frame for the non-recorded frame. This would stop the ghost from looking like it is flickering and jumping forward. These two improvements would have a major impact of the needed memory space while having very little impact on the smoothness of the replay.