**RESEARCH PAPER**

**Scene Manager for Real-Time**

**Concurrent Scene Manipulation**

**2018**

**By James Schmidt**

**Version History**

**Version 1.1**

Purpose

The purpose of this research is to aid programmers by giving them the ability to concurrently manipulate a scene in real-time. Doing so will make working on a scene easier and more efficient by allowing for fast iterations, near zero merge conflicts, and promoting collaboration amongst developers/designers.

Introduction

With the ever-growing demand for gaming studios to publish their games on time, developers and designers have been left to work harder to get changes submitted before their deadline. Is this due to a miscalculation of time on the project managers side? Or is the process in which developers and designers submit their changes outdated? Say a developer/designer wants to make changes to a scene in their game. They have a few choices on how to proceed. They could checkout the scene and assets from the build to make changes and submit. Doing so, however, locks anyone else out from updating that scene until it’s checked back in. They could go the naïve way and change the scene in their editor and submit their changes in small chunks to make sure the master is always up to date. This however, will lead to a ton of merge conflicts between other developers/designers who are also working on the same scene. Lastly, they could create a new branch on their source control and make changes to the scene there. Then submit their changes to the server and create a pull request to have it merged in with the master.

So far, every one of these choices is very time consuming. Every change submitted requires approval before it can be accepted to the master branch. Those who don’t take the time to check a change before approval risk having to revert to a previous build which could potentially destroy weeks of work. To combat a lot of this, many game companies utilize collaboration tools to keep their developers/designers on the same page. Collaborating saves time when multiple people can complete their tasks side by side without worry of merge issues or checking individual work for approval. So, if collaboration can save developers/designers time, then why not add collaboration into an editor/engine?

The Plan

Thus, with that question in mind, I embarked on a journey to create an editor/engine than can utilize proper collaboration. My journey hit an abrupt stop however after I realized that I would first need an existing engine to make a collaborative editor. At this point my choices were: Unity, Unreal, Lumberyard, or make a custom engine. Luckily, I had some prior experience with Unity doing a project for AI, so I decided to go with it. With the engine picked, I started to make plans on how to implement the collaboration.

It was at this time that I remembered the “Shared World Shooter” (1) talk from GDC 2015 given by Justin Truman at Bungie. His talk mentioned how they managed to utilize a hybrid of dedicated servers and host swapping peer-to-peer architecture to make their server footprint small while giving their game states low latency and seamless updates. Though the peer-to-peer seemed a little overkill for my project, I did however take away how they utilized their server-based architecture to handle managing states. Coming to the idea of making a host to store the master scene state while other clients can send manipulations to that state lead me to my next issue, what happens when two people want to manipulate the same object.

That one ended up being a little easier after relating it to mutex locking from multithreading. All I’d have to do is store a locked list generated by the objects being manipulated and the client that’s using them. Objects that are locked can’t by manipulated by other clients, thus relinquishing the issue. Also, since there’s only ever one master state, the client that gets their request to lock an object in first becomes the owner of that object. Giving only one client access to update an object at a time successfully allows the master scene state to update without worry of collisions between object ownership. With those ideas in hand I agreeably pushed forward and drafted some requirements and constraints for the project.

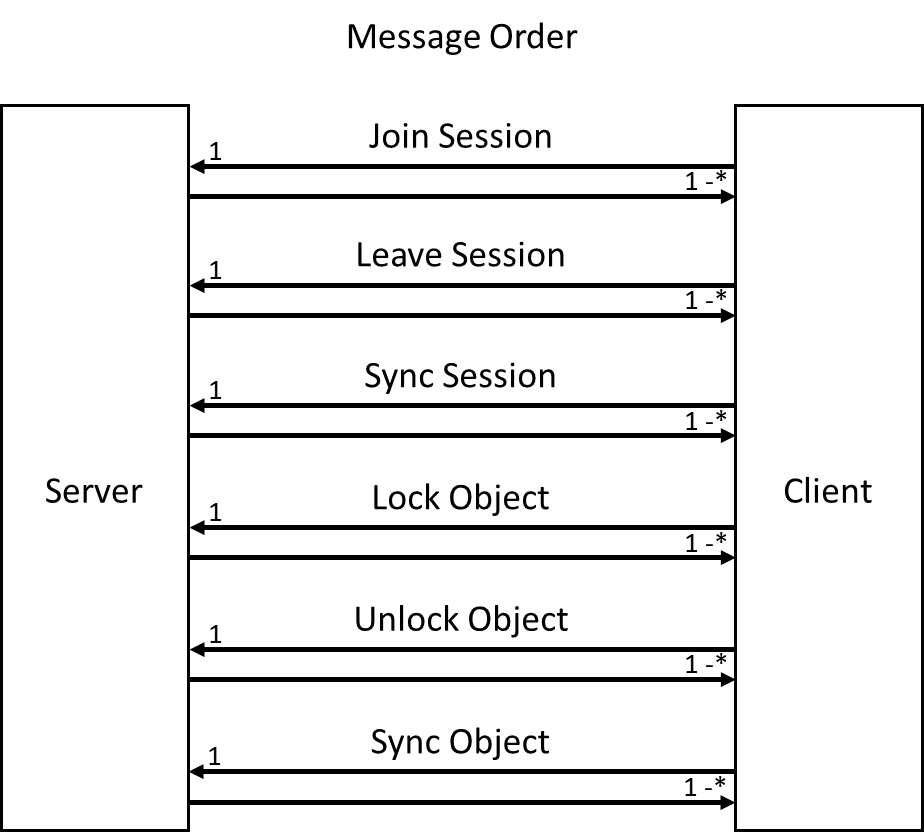
Requirements and Constraints

* The project should allow for clients of the same Unity project to join.
* The project should show real-time updating for game objects manipulated by other clients.
* The project should utilize a locking/unlocking mechanism for game objects per client owner.
* The project should utilize a client/server architecture model.
* The project should be easy for programmers to understand/use.
* The project should be developed using Unity.
* The project shouldn’t account for asset management.
* The project must work Unity version 2017.3.1f1 or higher.
* The project must compile properly and work with Visual Studio 2017.

Functionality

Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

Technical Diagrams

 Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

Where the Fun Began

Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

The Headache of Properly Syncing Objects

Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

Networking for Noobs

Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

Conclusion

Bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh bleh

Resources

1. **Shared World Shooter:** Justin Truman from Bungie (GDC 2015) - <https://www.gdcvault.com/play/1022247/Shared-World-Shooter-Destiny-s>