XR Storytelling Framework - VR Research Group

Hannah Do[*] Kamil Sachryn[†] Sabina Bhuiyan[‡]

Abstract

XR Storytelling Project presents live animations for viewers to be fully immersed in a story with the extension of VR environment. The project includes the dynamics of different animations that involves physics of AI, waypoint and graph system. Each animation was created to depict the different types of character actions in the Storyboard. In addition, Unity MLAPI was added for the multiplayer functionality for user designed environments, characters and animations.

1 Introduction

Virtual reality provides viewers a fully immersive experience, creating a sense of being part of a story. Our project aims to implement this by engaging the viewer as a character in the story, instead of just watching the story. The framework also allows different types of stories for viewer to follow up as their own characters.

2 Prerequisites

Requirements for development are C and Unity Version 2021.1.13. And limited by Unity Game Engine requirements, project runs on both Windows (7, 8, 10) and Mac OS X (10.9+).

3 Animations

The animations with AI were created to depict the different types of character actions for the Storyboard via Unity3D.

Live animations are available here :

https://github.com/VR-Research-Group/XR-Framework/blob/main/Hannah/readme.md

3.1 Basic Animations

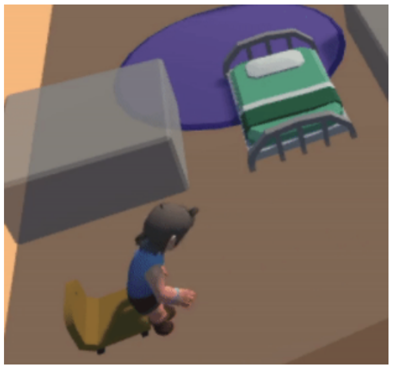
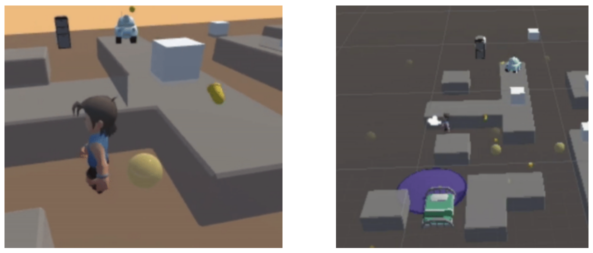


Figure 1: Basic Animations

[*]e-mail: hannah.hj.do@gmail.com
[†]e-mail:kamil.sachryn47@myhunter.cuny.edu
[‡]e-mail:sabina.bhuiyan00@myhunter.cuny.edu

Basic Animations were imported from Mixamo, character environment models were utilized from Meshtint and Distant Lands assets. Based on the proximity of the target object, the character goes through a sequence of animations (stand to sit, sitting, sit to stand) and walks away as the object actions are completed.

3.2 Physics and AI

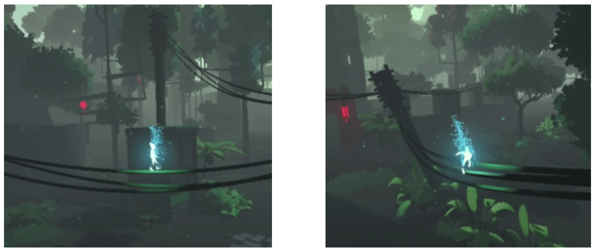


(a) Camera view (b) Bird eye view

Figure 2: Physics and AI

The tank, an example of an object in the story, is implemented as an agent with automated targeting. A script is embedded that measures Euler angles between the target (main character) and itself (main character) which determines the rotating direction. The trajectory of the shooting material (bullet) is also calculated with the gravity, speed, and preset angles. The physics and equation were referenced from Unity Learn's Physics of AI.

3.3 Waypoint Variations



(a) Walking Will-o'-the-Wisp (b) Jumping Will-o'-the-Wisp

Figure 3: Waypoint Variations

Character navigation is led by graphs of waypoints (yellow balls for visibility) with smooth rotation. This can be also applied to nonplayer character/object navigation to add movement to the background objects. Unity provides simple waypoint mechanism via Vector3 distance and Quaternion rotation.

In addition, waypoints can be incorporated with different animation sequences via different functions - patrol, idle and jump functions were implemented in this story. Throughout the game play, the incrementing waypoints are conditioned with the current time in Unity that enables the character to pause in between the animations.

Simple patrol function allows smooth animation sequence, while jump and idle functions use time to wait for certain amount of seconds before moving onto the next waypoint. In the example, the character jump animation is implemented with time pauses in between the jumps. Waypoint also works for all 3 dimensions, allowing walks on different heights and positions

The animation story line shows Will-o'-the-wisp jumping off from the wire, stepping on invisible stairs, and swimming towards the bottom ground. The assets are from Distant Land's Athazagoraphobia and Cryptid.

3.4 Character Interactions

Figure 4: Character Interactions

Unity also allows change of physics state of the character in the script - here the character's rigidbody is changed from a floating state to a state with mass pulled by gravity.

When Will-o'-the-wisp almost reaches the ground, gravity is applied to the rigidbody, allowing smooth transition from state of jumping in the air to a state of walking or patrolling over the ground.

As for the NPC characters, they are implemented with a set of animations which is iterated to create interactions with the main character until the camera leaves the scene.

Simple interaction is created between Will and Spectre with animator iterations. Will tries to talk to Spectre by approaching him and chatting out loud to wake him up. However Spectre barely responds by raising his head slightly to see where the sound is coming from, waves with his hand, and goes back to resting state. Will loses interest and leaves to look for another creature.

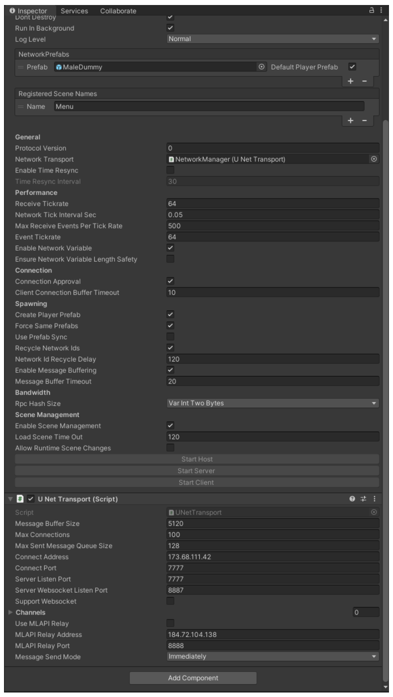
4 Networking

Networking in Unity is handled via Unity's official API, MLAPI. It has a variety of prebuilt functions which assist in creating a project where people can freely interact, with many customizable options.

We specify interactions which should be performed on each client, and how those interactions affect each connected client through MLAPI.

4.1 Networking Overview

MLAPI uses many library functions created to turn standard unity functions into networked ones, which greatly speeds up the transition from a regular Unity project into a networked one. This interaction takes place with a series of handshakes between Unity GameObjects via the transport connection created by MLAPI. One of the users serves as the "host" for the session, with many critical functions being passed specifically through the host rather than allowing a user to manage them, such as the creation and destruction of GameObjects. Once a client connects to the host, several security checks take place to make sure that the user is allowed onto the server, such as a password and hash check to discern maliciously modified files. Once connected objects are able to communicate between clients will send data to the host whenever they request a change in the server, and the host will notify objects of any changes, this guarantees synchronization across all connected clients. In order to limit communication delays there is a distinction between Networked objects which will synchronize, and non-networked objects which will only change on the client that tried to interact with them. The same system is applied to each connected person, their local variables, and interactions thereof, a communication or request to the host player and then the host will make any required changes.



Dont Destroy	✓
Run In Background	✓
Log Level	Normal
Prefab	MaleDummy
Default Player Prefab	✓
Name	Menu
General	
Protocol Version	0
Network Transport	NetworkManager (U Net Transport)
Enable Time Resync	
Time Resync Interval	30
Performance	
Receive Tickrate	64
Network Tick Interval Sec	0.05
Max Receive Events Per Tick Rate	500
Event Tickrate	64
Enable Network Variable	✓
Ensure Network Variable Length Safety	
Connection	
Connection Approval	✓
Client Connection Buffer Timeout	10
Spawning	
Create Player Prefab	✓
Force Same Prefabs	✓
Use Prefab Sync	
Recycle Network Ids	✓
Network Id Recycle Delay	120
Enable Message Buffering	✓
Message Buffer Timeout	20
Bandwidth	
Rpc Hash Size	Var Int Two Bytes
Scene Management	
Enable Scene Management	✓
Load Scene Time Out	120
Allow Runtime Scene Changes	
Script	UNetTransport
Message Buffer Size	5120
Max Connections	100
Max Sent Message Queue Size	128
Connect Address	173.68.111.42
Connect Port	7777
Server Listen Port	7777
Server Websocket Listen Port	8887
Support Websocket	
Channels	0
Use MLAPI Relay	
MLAPI Relay Address	184.72.104.138
MLAPI Relay Port	8888
Message Send Mode	Immediately

Figure 5: Network Settings

4.2 Scenes

We use three scenes to show the flexibility and powers of MLAPI. The first is a menu scene which allows for both hosting and joining games, while the second and third scenes are both two different environments where objects can be manipulated at will, and players can initiate moving between scenes. Certain objects in the scene were marked with "movable" tags allowing for players to change their positions, while others such as walls and the floor are static and immovable. Scenes control what is loaded into the environment, and can be changed at will, however synchronization across loads must be a priority. We use a strict order of loading in order to make sure that every client is synchronized, with the host having to approve and commence scene changes. Having each connected client manage it's own scene leads to differences across objects when scenes load separately. Data must flow through the Host and have them load each scene into memory, sharing networked objects and variables with each client. Using MLAPI's scene manager we send load signals to each client, and have a guest be able to initiate a change with the correct permissions. Scenes are dynamic, allowing each guest to add or change any amount of objects within it, and have those newly added objects be intractable by any other connected client. Each time a scene change request is made we move all connected clients to the new location, and load all network variables again for all connected guests, leaving the host to do the actual scene load.

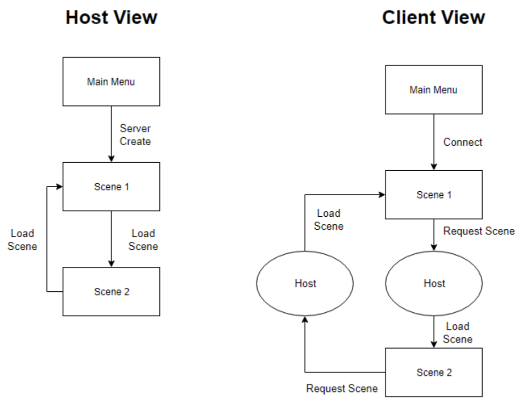


Figure 6: Example of Scene Changes

4.3 Objects and Movement

Objects in unity represent almost everything that interacts with the player, and are the basis for all Unity projects. Normally, objects will not synchronize across connected clients as they require two components in order to become a networked object. First, we attach NetworkObject which will allow each client to synchronize specific variables about the object, and makes it so that prefab objects are loaded quickly and efficiently, secondly we attach a NetworkTransform which governs the position, movement, and how quickly the object is updated across clients. These two components are vital to any object which will be interacted with across clients, as without them a change in any one client will not be shared to the rest. This same idea is applied to player characters, the NetworkTransform is able to take input from a clients and share it to all other clients, applying animations and functions as required by each client. This means that the host only transfers data about the NetworkTransform when a change is made rather than having to do all the computation itself for each client.

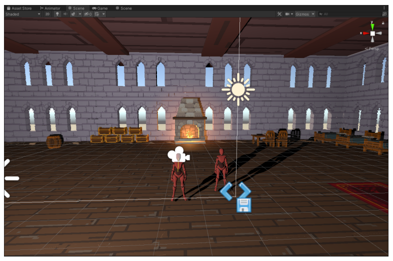


Figure 7: Player Character Objects

4.4 Hosting

The host of the server has control over everything that happens. They can restrict actions by connected guests, interact with everything, and choose when to change scenes or shut the program off. When a server is created by the host, they set the password and use their IP address as the location for the server, which the clients then have to connect to with the correct credentials. Once a client connects to the host, all of the network objects share their variables and position with the connected client, including all previously connected clients, and then continue this behavior through the host at specified intervals. Currently all guests have the same permissions as the host, however even still they must all be executed through the host via specialized functions, a guest cannot move the object without having the host move it and then synchronize it back to the guest as if the guest had moved it in real time. When the Host's IP is entered we do not require a password for simplicity although the code is easily editable to require one.

4.5 Compatibility

Both animations and loading in remotely added objects is easily compatible with network in MLAPI. Animations are all played from every client's own files and memory, with only the trigger for them handled by MLAPI, meaning we can very easily add and remove new animations so long as the client and host files are identical. With this we can also choose which animation to play based on what the client initiating the animation does. For adding new objects we can dynamically attach components onto them via scripting and have any compatible object begin communicating with the server. However, they must have specific functions and components given to them based on what they do, and as such any added objects have to be categorized by the user who uploads them.

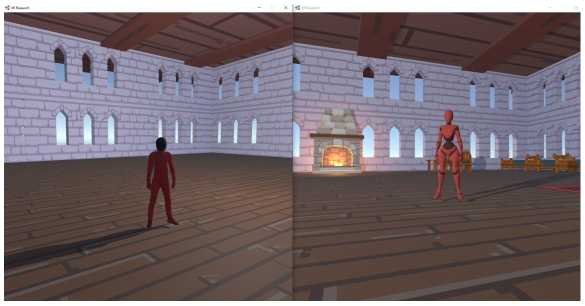


Figure 8: Two Avatars in one Scene

5 Environment Hosting and Cloud Integration with AWS and Unity 3D

One major aspect of this project was utilizing the cloud to load environment objects dynamically. To do so, we used Amazon Web Service (AWS). AWS includes a broad spectrum of services, such as AWS Mobile SDK for Unity. This platform can be used to quickly create efficient apps while connecting it to storage and data services. The simple objective of this platform is to have access to AWS while using Unity. The services provided allow users to save player and game data, save objects in the cloud, etc.

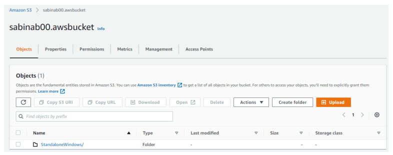
5.1 Sample Game Environment

Figure 9: Sample Game Environment

The purpose of the sample game environment was to demonstrate the workings of AWS cloud services. In this game, several packages and assets were downloaded from the Asset store, and a simple 2D game was set up with straightforward game objects and sprites. Using AWS will allow the game and all its assets and environment variables to be uploaded to the cloud.

5.2 Setting up an Amazon S3 Bucket

The first step of connecting AWS with Unity 3D was creating an AWS account. After doing so, an AWS bucket was set up, with the addition of an object. The purpose of doing this was to obtain the object URL to make the connection to Unity easier. The figure below illustrates a sample S3 bucket.



Objects are the fundamental entities stored in Amazon S3. You can use Amazon S3 inventory to get a list of all objects in your bucket. For others to access your objects, you'll need to explicitly grant them permissions. Learn more

Name	Type	Last modified	Size	Storage class
StandaloneWindows/	Folder	-	-	-

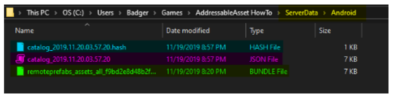
Figure 10: Creating an Amazon S3 Bucket

The next step is to drop any object into the bucket that was just created. This will be deleted later, so it does not matter what the file is. After the URL is saved, the object can now be deleted.

5.3 Connecting Unity 3D to AWS Cloud

Setting up a bucket ensures that Unity will be able to host and load the Addressables package from within the cloud, which is responsible for utilizing AWS cloud services for Unity. After installing the Addressables package via Package Manager, a new profile will need to be created and activated in Unity 3D. This makes sure that the Addressable settings are updated and connected to cloud services. After activating, take the link from the object created within the S3 bucket and place it in 'RemoteLoadPath.'

After creating a new script from the Addressables Groups window, a folder called ServerData can be found within Assets. This folder should contain three files: a JSON, a HASH, and a BUNDLE file.


This PC > OS (C:) > Users > Badger > Games > AddressableAsset HowTo > ServerData > Android

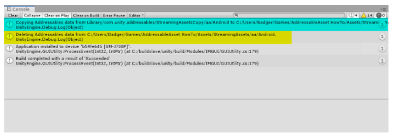
Name	Date modified	Type	Size
catalog_2019.11.20.03.57.20.hash	11/19/2019 8:57 PM	HASH File	1 KB
catalog_2019.11.20.03.57.20	11/19/2019 8:57 PM	JSON File	7 KB
remoteprefabs_assets_all_f9bd2e8d48b2f...	11/19/2019 8:20 PM	BUNDLE File	7 KB

Figure 11: ServerData Folder Contents

5.4 Building Addressable Assets with AWS

The final step of cloud integration is utilizing Addressable assets in AWS. The three files created should now be uploaded in the AWS S3 bucket. The next step is to create a new class, 'LoadAssetsFromRemote,' and add it to a game object within the hierarchy. Adding code to the class and building it establishes the set up.

While building, Unity copies the built addressable data into the project. It then removes the assets that were built and loaded remotely. In doing so, AWS aids in significantly reducing the games build size.



```
Copying Addressables data from Library/com.unity.addressables/StreamingAssetsCopy/aa/Android to C:/Users/Badger/Games/AddressableAsset HowTo/Assets/Streami
UnityEngine.Debug:Log(Object)
Deleting Addressables data from C:/Users/Badger/Games/AddressableAsset HowTo/Assets/StreamingAssets/aa/Android.
UnityEngine.Debug:Log(Object)
Application installed to device "b59fe645 [SM-J700P]".
UnityEngine.GUIUtility:ProcessEvent(Int32, IntPtr) (at C:/buildslave/unity/build/Modules/IMGUI/GUIUtility.cs:179)
Build completed with a result of 'Succeeded'
UnityEngine.GUIUtility:ProcessEvent(Int32, IntPtr) (at C:/buildslave/unity/build/Modules/IMGUI/GUIUtility.cs:179)
```

Figure 12: AWS Integration Operations

Unity now builds the assets remotely.

5.5 Breakthroughs and Setbacks

Utilizing cloud services to load assets from Unity assures that build size is greatly decreased. There is also no need to rely on the games build to host assets. This proves to be useful as more assets can be utilizing in game building without worrying about memory usage, thus paving the way for better game development.

AWS services also ensure a smooth workflow with other members. All assets can be uploaded to the cloud via S3 buckets created, and so despite the large number of Unity 3D assets, there is a seamless transition between the two platforms.

Although AWS is relatively simple to use, several drawbacks remain. For example, to obtain access to AWS management console, permission is needed by a root user of greater power. The root user has the authority to make other users part of their account, and thus grant them the ability to use all of AWS's applications. Without this access, users will need to pay to use AWS cloud services.

6 GITHUB

https://github.com/VR-Research-Group/XR-Framework

ACKNOWLEDGEMENTS

This project was supervised by Professor Oyewole Oyekoya, as a master's project at Hunter College, Summer 2021.