Senior Honours Project

Supplemental Document

University of St Andrews

Build Instructions

for Unreal Collaboration & Unreal Selector

Paul Graham(*170004605*)

Supervised by

Dr. Alan Millar

April 27, 2020

1 Introduction

In this document, we will look at how to build and run both **Unreal Collaboration** & **Unreal Selector**, and will briefly look into how to include **Unreal Collaboration** into your existing Unreal Engine project.

2 Contents

3 Prerequisites

The following section references the prerequisite software you will need to install to work on each application.

3.1 Unreal Collaboration

You will require the following to build **Unreal Collaboration** on its own, for deployment to a **Windows** environment (which we'll assume you have administrator permissions on). If you are planning on deploying to a **Linux** environment, you will require additional items, specified with a *****. Links may point to additional requirements that you will also need to download & install.

- Python **3.8** - https://www.python.org/downloads/
- Visual Studio IDE (*any version*) - https://visualstudio.microsoft.com/
- Unreal Engine **4.23.3+** - https://www.unrealengine.com/en-US/
- ***** Unreal Engine **4.23.3+** Source - https://www.unrealengine.com/en-US/ue4-on-github/
- ***** clang toolchain **-v15+** for Linux deployment - https://docs.unrealengine.com/en-US/Platforms/Linux/GettingStarted/index.html

3.2 Unreal Selector

3.2.1 Server

The following programs are used for running the flask server that powers the **Unreal Selector** server, links may point to additional requirements that you will also need to download & install.

- Python **3.8** - https://www.python.org/downloads/

3.2.2 Client

The following programs are used for running the electron client that powers the **Unreal Selector** client, links may point to additional requirements that you will also need to download & install.

- NodeJS **12.14+** (with `npm`)- https://nodejs.org/en/

4 Running

4.1 Unreal Collaboration

To run Unreal Collaboration, simply follow the steps outlined below:

1. Right click on the `.upoject` file in the included `UnrealCollaboration` folder, and select Switch Unreal Engine version
2. Select the version of Unreal Engine you installed & hit Ok
3. Open the built `.sln` file in Visual Studio IDE
4. Ensure you have selected `Win64` as architecture, and `Development Editor` as the target solution
5. Select `Local Windows Debugger`

You can also open the `.upoject` file directly, however you will be unable to edit C++.

4.2 Unreal Selector

4.2.1 Server

To run the server, follow the following steps:

1. Install `pipenv` with `pip install pipenv`
2. Install dependencies by using `pipenv update`
3. Setup the database with `pipenv run python server/setup.py`
4. Setup an admin account for yourself with `pipenv run python server/admin.py [username] [password]`
5. Run the server by executing `pipenv run python server/main.py`

Additionally, you may want to change settings within `server/config/config.json` to your liking.

4.2.2 Client

To run the client, follow the following steps:

1. Install dependencies by using `npm i`
2. Start the client by using `npm start`
3. Login to your server (if you're on the same computer as the server the hostname is `localhost`) using the admin account you made earlier
4. Build **Unreal Collaboration** projects
5. Upload projects through the **Admin** page

5 Using Unreal Collaboration for existing projects

This section is condensed from `README.md` inside of **Unreal Collaboration**, please review that document for more information including how to update **Unreal Collaboration** in an existing folder.

5.1 Importing Unreal Collaboration

1. Ensure your project is built for C++ by adding a `None` class to your project - please see https://allarsblog.com/2015/11/05/converting-bp-project-to-cpp/ for a tutorial on how to do this
2. Generate project files for your project by right clicking on the `.uproject` file and selecting Generate project files
3. Ensure you have selected `Win64` as architecture, and `Development Editor` as the target solution
4. Select `Local Windows Debugger` and ensure your project compiles & runs
5. Close both Unreal Editor & Visual Studio, copy the absolute path to your root project directory
6. Open up **Unreal Collaboration** in a terminal window & execute `python source/merger.py "[path]"` where `[path]` is the absolute path you copied in the last step
7. Rebuild visual studio project files by following step 2

5.2 Migrating our example Blueprints

You may also want to migrate our existing blueprints as examples to work off of. To do so follow the following steps:

1. Open the **Unreal Collaboration** sample project in Unreal
2. Under `Contents`, right click on the `Blueprints` folder
3. Select **Migrate**
4. Select the `Contents` folder of your new project

After migrating over the files, you need to select our Blueprinted gamemode `BPGameMode` as your gamemode under `Project settings` → `Maps & Modes`.

6 Building Unreal Collaboration for Unreal Selector

When deploying, you must consider what architecture both your clients are going to use to download the projects (either Windows or Linux) & the architecture the server is going to run on. For simplicity, the following guide over covers Windows deployment to Windows clients, for Linux options please see `README.md` within **Unreal Collaboration**.

Please refer the amazing guide provided by the UE4 community wiki for more information on deployment https://www.ue4community.wiki/Legacy/Dedicated_Server_Guide_(Windows_%26_Linux).

6.1 Prerequisites

Before building for **Unreal Selector** deployment, ensure the following has been setup within your project:

- You are using a Blueprinted version of `NetworkBaseGameMode`, with the HUD Class set to a Blueprinted extension of `NetworkHUD`
- You have setup the required characters & player controllers within the gamemode to be Blueprinted versions of their respective parent classes
- Setup an empty level called `Transition`, and build it by selecting `Build`
- Within `Project Settings` → `Maps & Mode`, ensure you have the gamemode set to the Blueprinted gamemode above, and set the `Transition` level to point to the empty transition level
- Within `Project Settings` → `Packaging`, edit the `List of maps to include in a packaged build` setting by including the path to both the server map & the transition map (maps should be in `/Game/[Map Folder]/[Map Name]`) See Figure 1
- Every asset should be set to allow for cooking (for example, one should not refer to `UBlueprint` within code as these are not explicitly packaged)

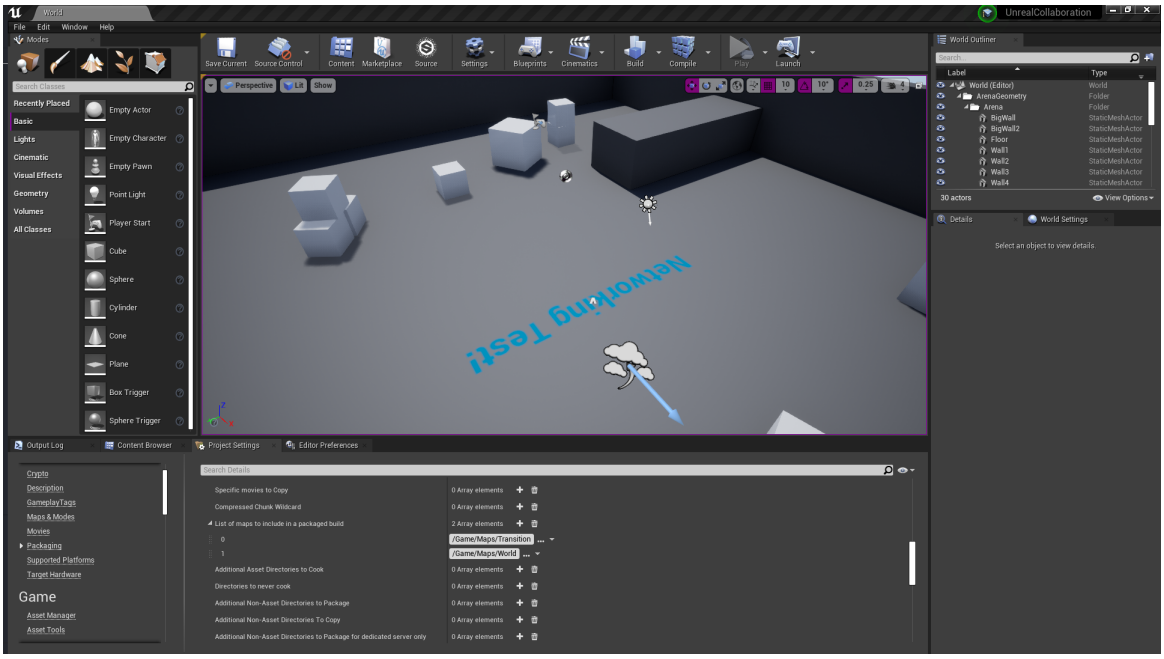

Figure 1: A example of the packaging settings one should use

6.2 Client

When building the client to deploy to **Unreal Selector**, follow the steps outlined:

1. Compile & run the project using `Development Editor` as above
2. Package the project for your desired **client** architecture using `File` → `Package project`. Save the project to a known location, e.g. `Build`
3. When this is complete, move the **Contents** of `Build/[OSName]NoEditor` into its own folder
4. ZIP the folder, ensure the zip file contains the contents of the folder at its top level & not the folder itself (Figure 2)
5. Rename the zip file to the publicly visible name of your project

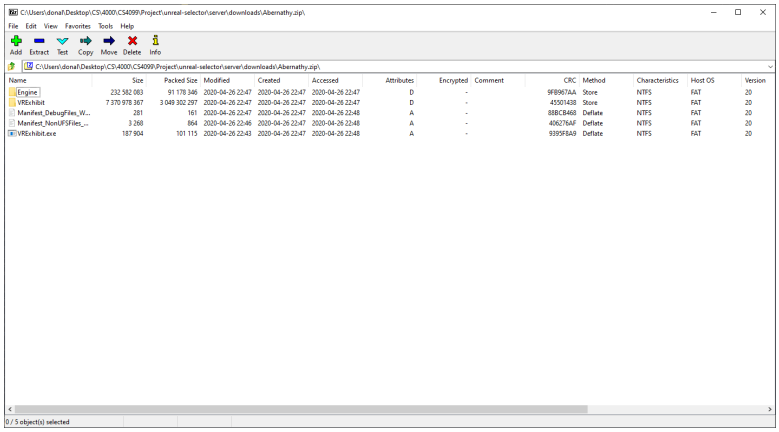

Figure 2: The top-level contents of a zipped build

6.3 Server

When building the server, steps 1-2 are identical to building the client, however **please only skips if your client & server architectures are the same**.

1. Compile & run the project using `Development Editor` as above
2. Package the project for your desired **server** architecture using `File` → `Package project`. Save the project to a known location, e.g. `Build`
3. Back in the project solution, select `Developer Server`
4. Right click on the solution itself, select build
5. When complete, navigate to `Binaries/[OSName]` within your project directory. Ensure `[ProjectName]Server` has the correct permissions to execute
6. Move this executable to `Build/[OSName]NoEditor/Binaries/[ProjectName]/Binaries/[OSName]`
7. Move the **Contents** of `Build/[OSName]NoEditor` into its own folder
8. Rename the folder to the publicly visible name of your project

6.4 Deploying to Unreal Selector

1. Login to unreal selector with an account of `ADD_PROJECT_RANK` or above permissions
2. Lock the server using the `Admin` page
3. Move the zip file from Section 6.2 to `server/downloads`
4. Move the folder from Section 6.3 to `server/unreal`
5. Add the project by using the `Publish Game` form on the `Admin` page. Ensure the project you are adding has the same file as the files from steps 3 & 4.
6. Unlock the server by using the `Admin` page

7 FAQ

7.1 How do I make a Blueprinted version of your C++ class?

Within the editor, go to `C++ Classes` and right click on which class you'd like to make a Blueprint of, select create Blueprint of this class & save the Blueprint with your other Blueprints. Additionally, if you already have a Blueprint which you want to change the parent class of, you can do so within `Class Defaults`.

7.2 How do I debug with multiplayer?

There are several ways to debug with multiplayer. The default approch is the following: `Use Single Process` is off (mode is on `Play as Client`), `Use Dedicated Server` is on, `Number of players is 2+`.

- **Breakpoint support** - If you are triggering a code breakpoint, ensure that `Use Single Process` is on so that Visual Studio can attach itself to the server process. When the breakpoint is hit, you can get the value of the `Role` variable within `Actor.h` to see which player the breakpoint landed on. Use continue to when it next breaks if the role is wrong.
- **Replication support** - To get a view of what is being replicated, turn `Use Dedicated Server` off. You may look at the difference between what the listen server sees and what the client sees to guide debugging.

- **Unreal Selector** - Use the default mode to emulate the conditions inside of **Unreal Selector**.
- **Virtual Reality** - See Section 7.3

7.3 Why can't I get virtual reality & mutliplayer to work locally?

Due to an issue with SteamVR, VR crashes when a **Unreal Collaboration** server is also running in the background, regardless of if the game is running within the editor or standalone. We found *workarounds* to this issue, but they aren't guaranteed to function:

1. Turn `Use Single Process` off and launch the VR Preview
2. Turn `Use Single Process` back on and launch the VR Preview again (sometimes there may be an issue with not getting a display within the VR headset. If this occurs, relaunch the preview).

Additionally, we found that our play-in-editor VR setup failed after switching to the new switching gamemode since the new process got spawned with VRMode enabled, and the editor process (which is actully in VR) got spawned with VR Mode disabled. We have included a `VRGameMode` you can use for testing within VR using the above method.

7.4 How do I replicate dynamic variables, such as skeletal meshes?

This can be done through both Blueprint or C++, we have implemented an example of this within the `VRExhibit` project. In essence, then the server calls `UponInfoChanged()` within Blueprint, it sets several replicated variables, which replicate with a **custom sync function**. When the server replicates it to the client, that function is called and the asset on the client is updated with that variable.

7.5 Why don't models show up in the editor when they show up in game?

Several models are spawned, and attached to components of actors at runtime so facilitate network support. Most notably the text actor for characters & signs. A box component exists so you can adjust where this actor spawns and attaches to.

7.6 How do I deploy the Unreal Selector Bot to my Discord server?

For this, you will need to setup a Discord Bot user on the Discord Developer portal - https://discordapp.com/developers. You will then need to invite the bot to your server with the bots `client ID` - https://discordapp.com/oauth2/authorize?client_id=[client_id]&scope=bot&permissions=8. Finally, attach your bot token to the `tokens.json` file within the **Unreal Selector Bot** project, and run the bot. For more information read the `README.md` file within the **Unreal Selector Bot** project.

7.7 The Unreal Selector client reports strange errors when I do too many actions?

This happens because you are being rate-limited. Try to wait for a few minutes & try again or restart the server to reset the rate limiter.

7.8 How do I learn more?

You can learn more by reading the `README.md` files for each of the projects as they provide comprehensive documentation into these projects. Additionally, you can join the offical **Unreal Collaboration** discord server - https://discord.gg/5weFaXF.