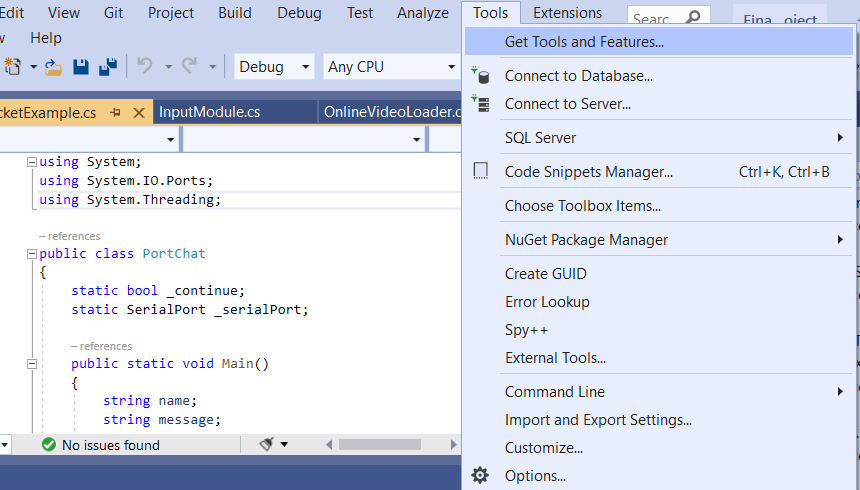
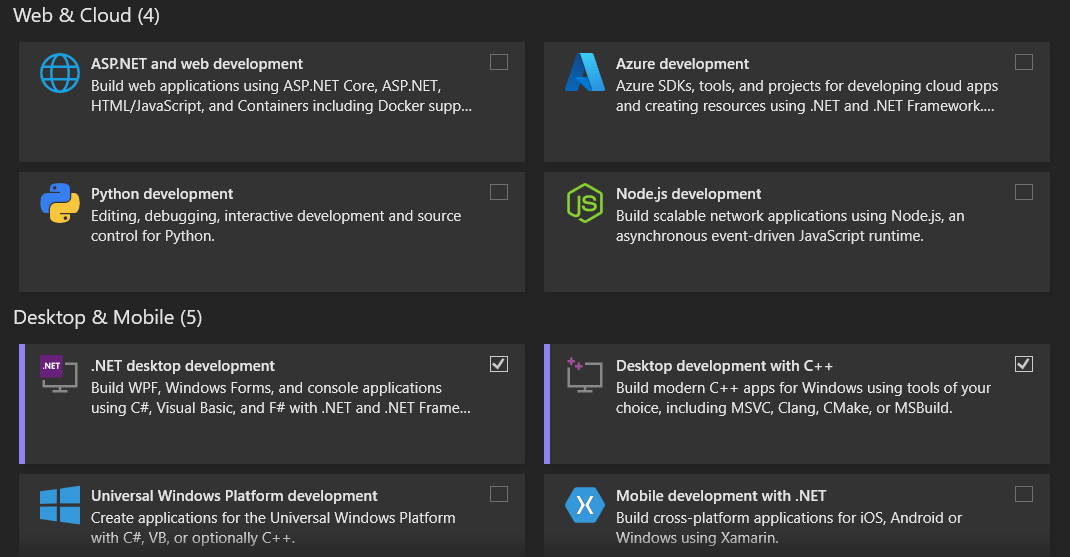
The following were methods that were explored for communication between Arduino, Raspberrypi and Unity. The two main methods considered were **serial communication** via **sockets** and **web server** communication.

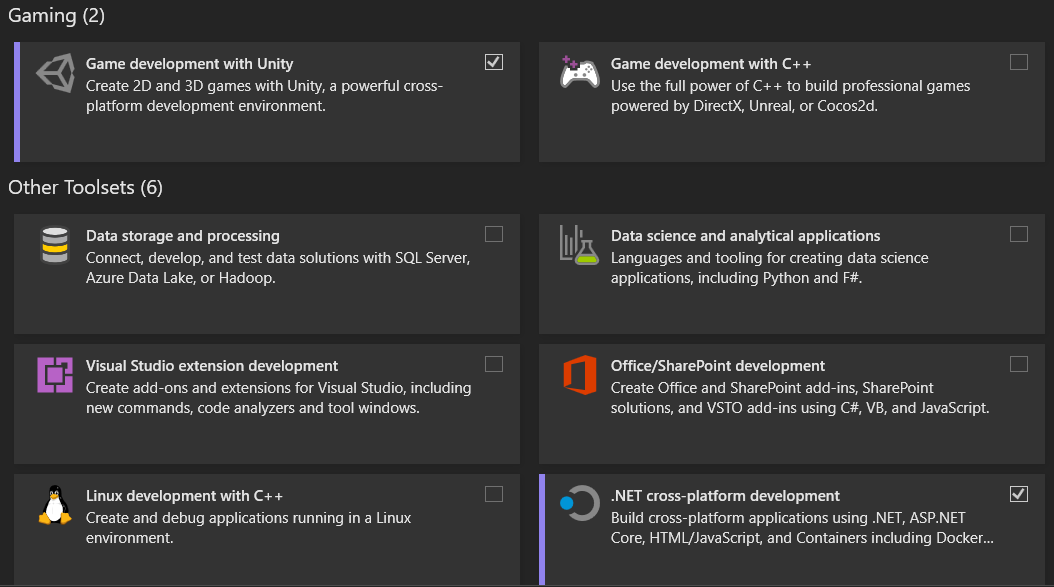
**Visual Studios Prework**

Visual Studios requires the .net package to be implemented. It’s also useful to add the Unity package to Visual Studios. I’ve used VS 2019. Here are some useful screenshot references to install.



Select the following packages listed in the pictures below.



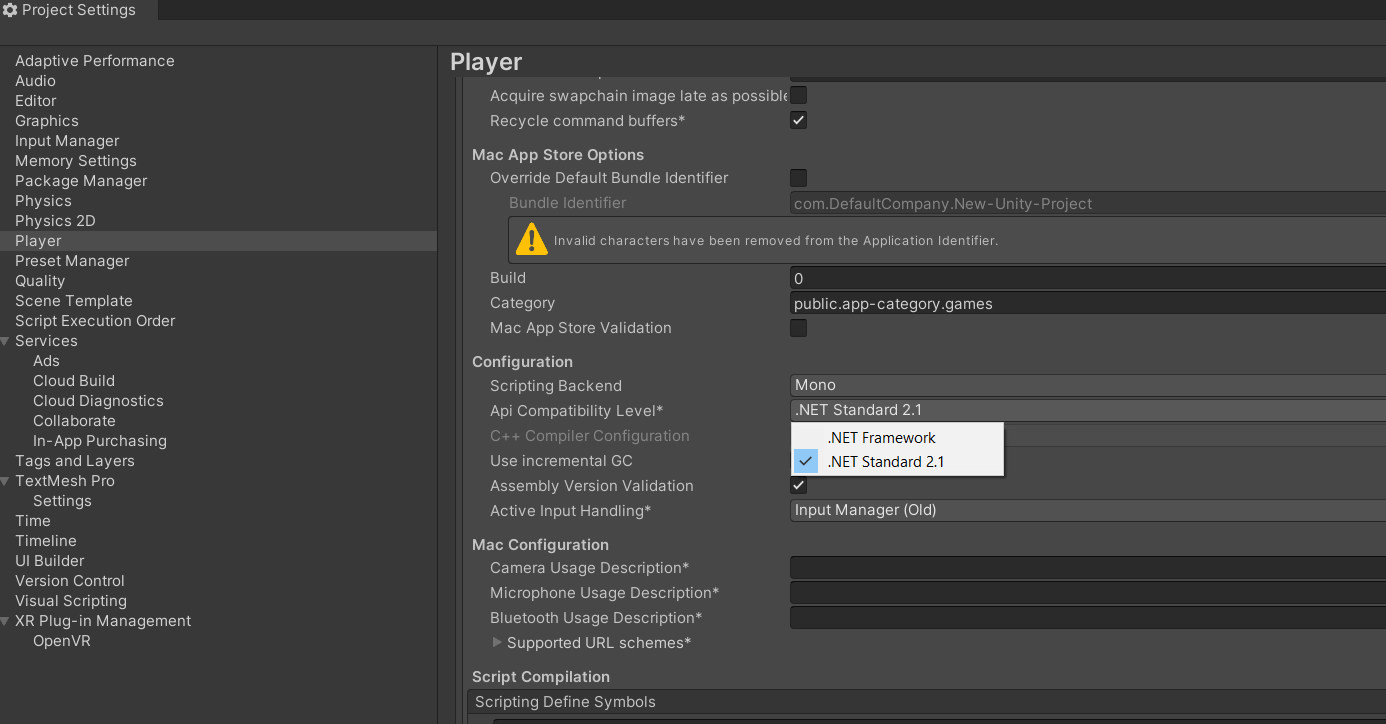


If your System.IO.Ports is highlighted red, you haven’t installed it correctly.

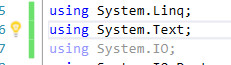
**Serial Communication Tutorial**

<https://sirwilliam.hashnode.dev/serial-communication-between-arduino-and-unity-3d>

One error that took some time to debug is calling stringbuilder directly in the library. Without the call, Visual Studios can’t find the class within System. Also note the .net 4 reference is incorrect inside Unity as the reference to change the .net type is either framework or .net 2.1 . You need to use framework which will show errors in Visual Studios, but will allow you to compile in Unity. The incorrect API selection is shown below.



Add the Using System.Text; line



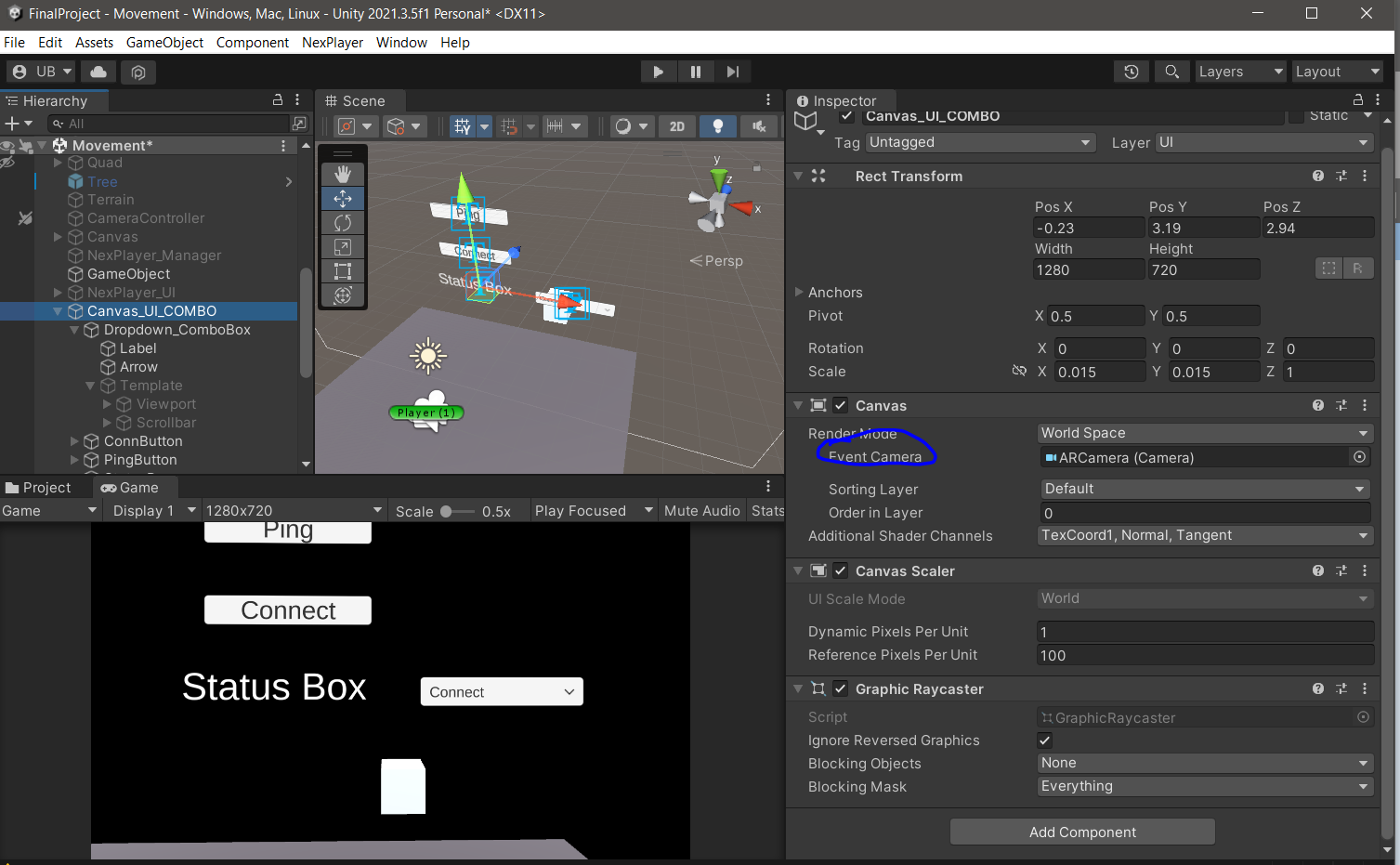
The information from the video tutorials despite not being used for video helped with understanding the canvas and button development which is needed next. Specifically, the webcam canvas generation from Jeff Komarow. I’ve included the script for Unity’s half in the submission.

Follow the Arduino Code, test the script for reading and writing to the port for verification.

Second, follow the Unity code as described in my notes. Fix the remaining errors and then move onto the testing step with both below.

For the UI steps, here are some important notes and helpful videos. They were vaguely described in the tutorial above so I’ve added some useful steps as beginners.

Video on dropdown (combo boxes) <https://www.youtube.com/watch?v=5onggHOiZaw>



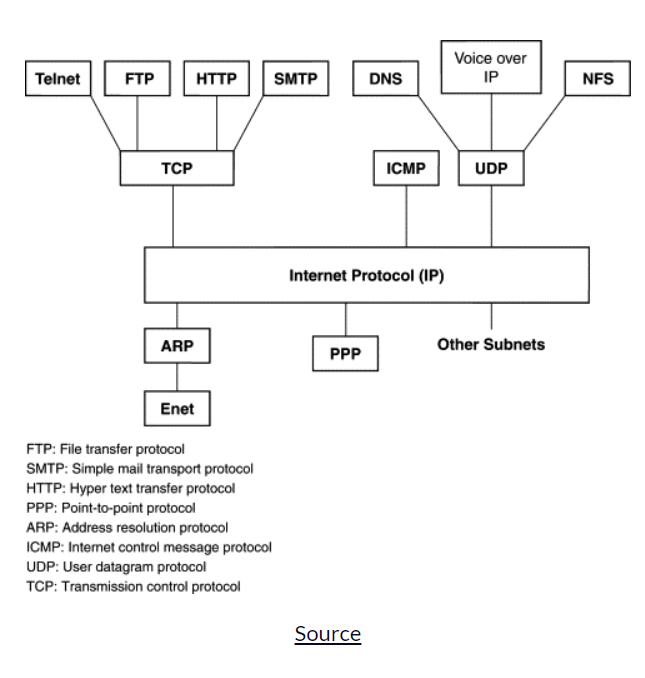
In the picture above, the combo box wasn’t working when clicking from the view. You need to link the camera to the canvas event camera for select-ability. Drag the ar camera into the canvas event camera box as shown.

For adding the script, they are best added by creating empty objects into the asset list of the scene and using the \*add component\* to link the script to an object. This object is then linked to other objects and allows multiple instantiations.

Another aspect you will need to do is make sure the buttons are linked properly in the dropdown(combobox) to the script. If you look at the empty object with the script attached you will notice you can add references to the scripts class in the window. Instead of linking the scripts to the buttons.onclick() method, you will link the reference text and button to the script. This was the first time I witnessed the reversal in Unity as most tutorials reference linking to the onclick method.

**Web Server Communication**

In order to try and develop web server communication for the VR system, I first looked into available types of communication and what Unity’s standard rendering tools take. I found Unity looks for HTTP pages and this falls under the TCP file types. Audio which wasn’t included, falls under UDP, and could be explored later. It’s unclear whether the HTTP requires a prerecorded video but has appeared so during testing and forums exploration. The notes on Unity’s webpage refer to it as a “File URL” though I wasn’t sure if file is also considered a packet in transmission. The CS department was about as knowledgeable if not less than I was when searching for more discussion on this. The forums show various references that the HTTP site requires a .mp4 file. <https://forum.unity.com/threads/play-video-from-url.512813/>



<https://www.freecodecamp.org/news/tcp-vs-udp/>

<https://docs.unity3d.com/ScriptReference/Video.VideoPlayer-url.html>

<https://forum.unity.com/threads/play-live-video-stream.567379/>

<https://www.youtube.com/watch?v=sFNxpjJtYTg>

This German ran into the same issues as myself and ended up using the same tool Vuforia which I had also used as a last resort.

After failing with 6+ tutorials I have not found any other method than Vuforia’s built in backend asset to fetch video.

My comments for further research in this area is to start backwards with preexisting tools from other programming languages and projects. I started cursory research into Youtube webscrapping with python and this may be an alternative to generate data for the C# Unity importing. Tutorials for webscrapping and grabbing data using python may be a good alternative to find starting material for video decoding/encoding and file transfers.

<https://medium.com/swlh/scraping-live-stream-video-with-python-b7154b1fedde>

From this tutorial the best note’s I’ve found describe “ *The stream of .ts (transport stream) files being fetched are the raw video stream data and can be downloaded and played, they are the same as .mpeg files. These files are what we want to automatically download and put together.*“

This method uses selenium for automating the initialization of the stream and keeping it up. It is a front end method for selecting video elements on screen. They go further by pulling .ts file links from network logs and write the data to a file for any .ts links. The data is appended to one file and continuously downloaded.

There’s also another article on this issue which researchers explain in more detail the challenges of bringing video to Unity and adaptions over the years. <https://users.informatik.haw-hamburg.de/~ubicomp/projekte/master2018-proj/melles.pdf>

Another potential method could be forcing data to the native webcam socket. Since webcams work for Unity and can be registered, maybe one could overwrite the packets with HTTP data and audio.