UNInput

Introduction

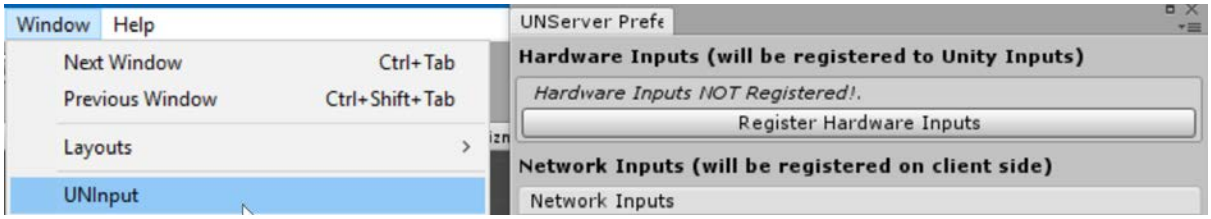
The inspiration behind UNInput's essence relies on couch gaming. Let's say you and three more friends decided to play a split-screen racing game or an arcade co-op game, but there are not enough gamepads at your disposal to play such games. Probably you would find a game that lasts only short rounds, having to switch the controls on each play, or even give-up completely on playing! In the meanwhile, you where using your cell phones and, perhaps, even used their network connection to search for other games...

Even holding such powerful devices on our bare hands, we are unaware of how many times they could link the dots in our lives, provide collective entertainment and ease most of our daily problems in fast and practical manners! And **that** is the thought that brought the idea of an Universal Network Input to life. So how do we provide a simple manner for game developers to make such things possible? And how do we simplify that without taking their customization liberty, in order to keep the aesthetics and the feeling of their games in first place? The following chapter will provide a general description of the asset's usage pipeline, and its components.

Setup

Now that there is a fair sight to what the asset must achieve. We can trace the workflow in order to understand how to use it, starting with the setup and diving into the code. The reason we called it Universal in the first place is because of an Virtualization of every connected control. The first type of control that is supported is the Hardware Control.

Hardware Controls: Unity doesn't provide default support for multiple of these devices, so you would have to manually register every one of them in the Input Manager (which takes tons of time). Therefore, UNInput does that work for you, all you have to do is to get into the asset window on Unity's toolbar (**Window/UNInput**) and press the "**Register Hardware Inputs**" button - as following:

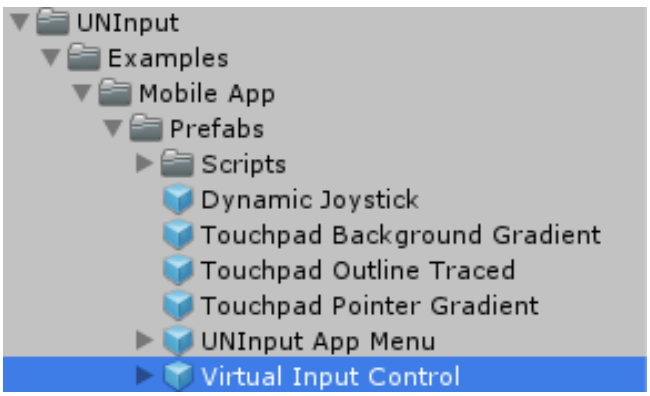


After that, you are ready to go and use **up to eight connected hardware controls** in your game! We will go into further details on programming usage and classes on the next chapter.

Network Controls: As the core difference of our asset with respect to others on the Asset Store, we provide a seamless usage of virtual controls connected through Wi-Fi. These are mobile devices running an App (a client) attached remotely to our game (which acts as a server) and are, unlike Hardware Controls, very customizable.

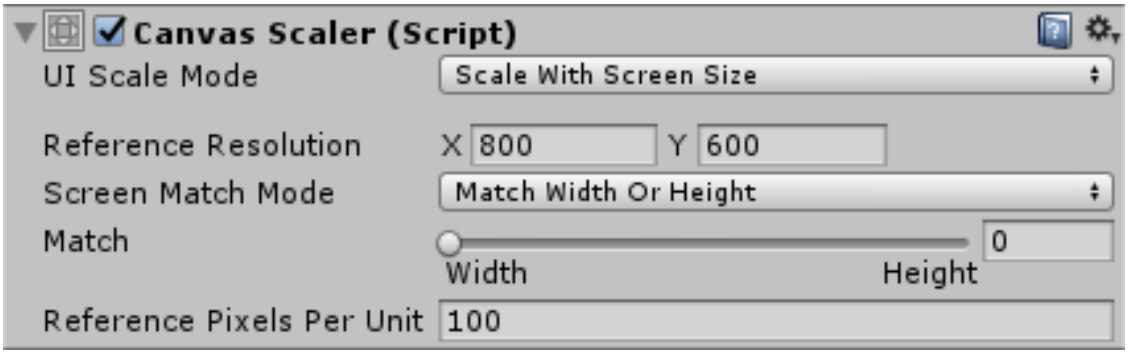
So, in order to retain the customization capability without wasting your time and money to build such App and then release it on the Play Store, we created our own App, called UNControl, which can be downloaded free of charges and used for this purpose. The way it works is that the game sends a packet containing the interface prefab to the mobile, which is then instantiated on the fly.

For that to happen, we must generate an Asset Bundle, which is a file that compresses any non-scriptable object, such as meshes, materials, etc. What this means is that you cannot put any custom script that isn't on the App by itself (this is the down side of it all), but you can customize every aesthetical aspect of the prefab like background, buttons and joystick graphics. In order to build such prefab you must use our provided button and joystick scripts (unless you are building the App we described before - our App's source-code is included on the examples), but feel free to change every graphical concept of them, as we stated before. **Start a new Unity project**, and we will be explain why this is recommend later on.

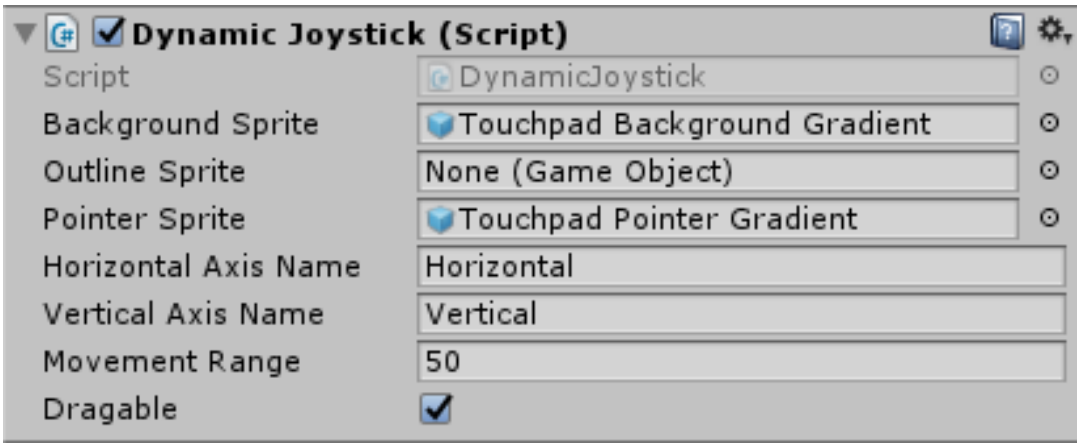


We have provided you a sample prefab with three buttons and one dynamic joystick, which goes by the name of "**Virtual Input Control**". Drag and drop it in any scene and take a closer look at it's objects, they are of utmost importance for our setup, as they are what dictates the input values that are going to our game in first place. You can find it over here (on the right):

If you pay attention, we used an canvas scaler that uses the screen size of **800x600** and matches **100% Width** as a basis for proper content scaling, because our app only runs in landscape mode:



In this first release of the Asset, we provided what we considered the best joystick for a virtual gamepad (considering you won't be looking to the mobile device's screen when you are playing), that is the Dynamic Joystick. It instantiates where you touch and can be either fixed on that point or dragged along with your fingertip so that it keeps in the joystick's edge. Play around with the parameters and see how it behaves:

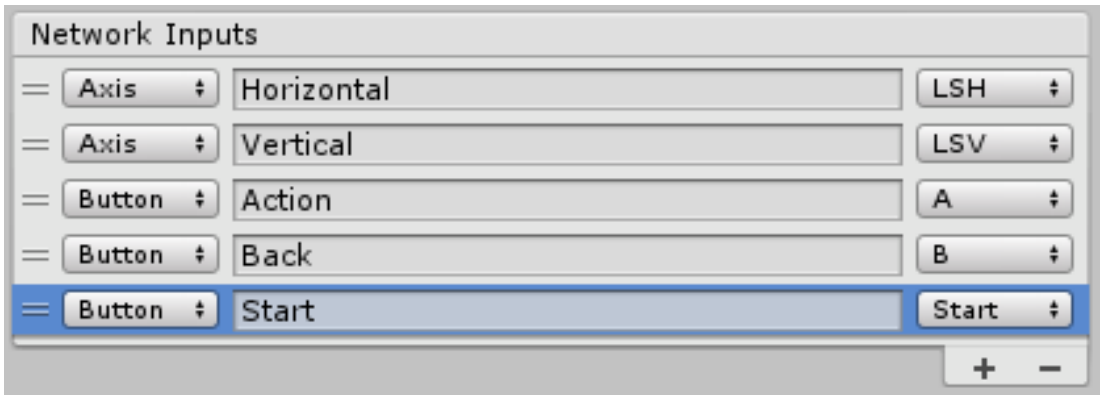


Now take a closer look at his properties. Notice how we have put some public GameObject prefabs that are going to be instantiated when you click. And also notice that there are **Axis Names** for the **Horizontal** and **Vertical** axis. These are quite important as they are the values we are going to use on the UNInput window:

The reason we do this is because our game (or server) will take such information and transmit it to the mobile device. This last will process every input in the Dynamic Joystick class and transmit it back to the server, so this values are sent along with the registration packet whenever a device is connected and used for the Input's Update messages.

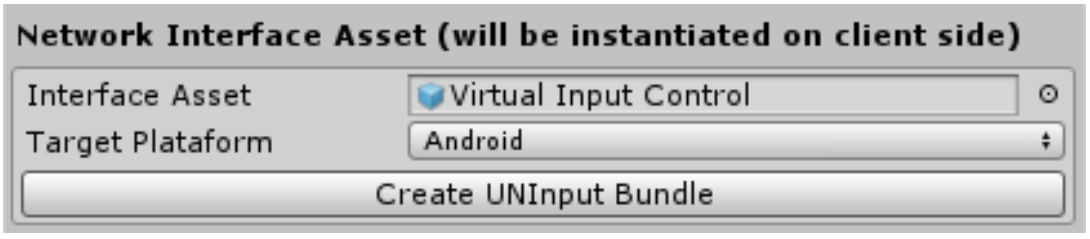
Take a look at the Button Handler script now, it works in the same way, but with less customization parameters of course:

For each button we will fulfill the string in the same way we did with the axis. Pay attention to the capital letters, **it is Case Sensitive**:



You might have realised we have also changed the **Input Type Enum** on the **left side**. And the correspondent **Hardware Input Enum** on the **right side**. The first one's existence is pretty self-explanatory. The last one's is an **easy linking** we provided for games that want to **support both Hardware and Network Inputs**. So you can use this enum whenever you want to get an axis or button and it will work right away for both of device types (without you having to memorise any string of the kind). Please notice that LSH stands for Left Stick Horizontal, and both values are the same - the first one is just an easy-to-write shortcut.

Okay, now that we have defined our prefab and applied it's changes, let's build the UNInput Bundle that is going to be sent to the device. First, **select the target platform**. We will be selecting Android; so far, UNControl has only been released in the Play Store, but Apple Store is on the roadmap. Then just click on the "**Create UNInput Bundle**" button:



What is now happening is that we are finding every bundle dependencies and tagging them with the **uninputbundle property**. Then, Unity changes the projects building target, reimport all assets and builds the Asset Bundle. And that is why it's nice to create a separated project, let's say you have a very large project and Unity reimports every textures, sound, etc. It's going to take tons of time…
This is an Unity's limitation, that doesn't provide an selective building pipeline for the Asset Bundles (used to, but that is now deprecated). Once this process is done, you must have a brand new **"uninputbundle.bytes"** file on the **Resources/Bundles** folder. If you did as we

recommended, and created a new project, now it's a good time to **copy the Resources folder** to your main game project (all you will need is in that folder). Just don't forget to import the UNInput's asset first.

It's totally recommended, if you want to build your own App, to take a look at the **UNApp.cs** file located under the **UNInput/Examples/Mobile App/Prefabs/Scripts** folder, and it's usage of **UNClient's class** (which is the network class that handles the client application network).
Be aware that, in order to use Network Classes, you must include the proper namespace:

```
using UniversalNetworkInput.Network;
using UniversalNetworkInput.Network.Internal;
```

Usage

Now that we have taken you through every setup steep, given that you have done everything exactly as we did, let's dive into the game (or server) sample. Which can be found in the **Server example folder**, right bellow **Client App**'s. Make sure that your device and your computer are in the **same Network**, it's recommended running a **Hotspot** in either one of your devices to ensure proper connection. Play it and check if it's working properly by **opening the UNControl** in your mobile device and connecting to the game.

Let's open the **PlayerMovement.cs** script. There is a lot going on, just like in your own player's script. First, check what includes we are using here (this is the main include for UNInput's general usage):

```
using UniversalNetworkInput;
```

Now, take a look at **line 132**, where we use the Horizontal and Vertical Axis to define the player's movement direction:

```
new Vector3(UNInput.GetAxis(i, AxisCode.LSH),
```

Realized how we used the function pretty much like the Unity's own **Input.GetAxis()**? The reason we must pass the "**i**" is because we must give a control ID, and as said before, we are using the **AxisCode** instead of the axis name - which is also possible - is to be able to play this game with **Hardware Controls**. So you can **plug in your control** (without connecting the mobile device) and **move the Player**. Feel free to take a look at this script and link the dots of everything that was said before with their respective usage now.

And, last but not least, the Network connection is only working because of we are initializing the server. Take a closer look at the **ServerInitializer.cs** script. And see how easy it is to start the **UNServer class** and start using remote controls:

```
//Initialize Server
UNServer.Start(4, 25565, UNNetwork.GetLocalIPAddress());
```

The parameters are optional (these are, in fact, the default parameters), but they go as following: Max Players - 4, Port - 25565, IP - PC's first TCP/IPv4 connection address.

Thank You!

Hopefully, this guide fulfill all your questions about how to setup your game. If you got any question or bugs report feel free to send us an email, please rate our asset at the Asset Store. Any kind of feedback is of utmost importance for us to develop even better tools!

Twin Ravens - twinravensdev@gmail.com
Gabriel Lanzer Kannenberg - gabriellanzerlive@hotmail.com
Luiz Augusto Wendt - luizaugustow_1@msn.com
Leonardo Gomes Santana - leonardogsantana@hotmail.com