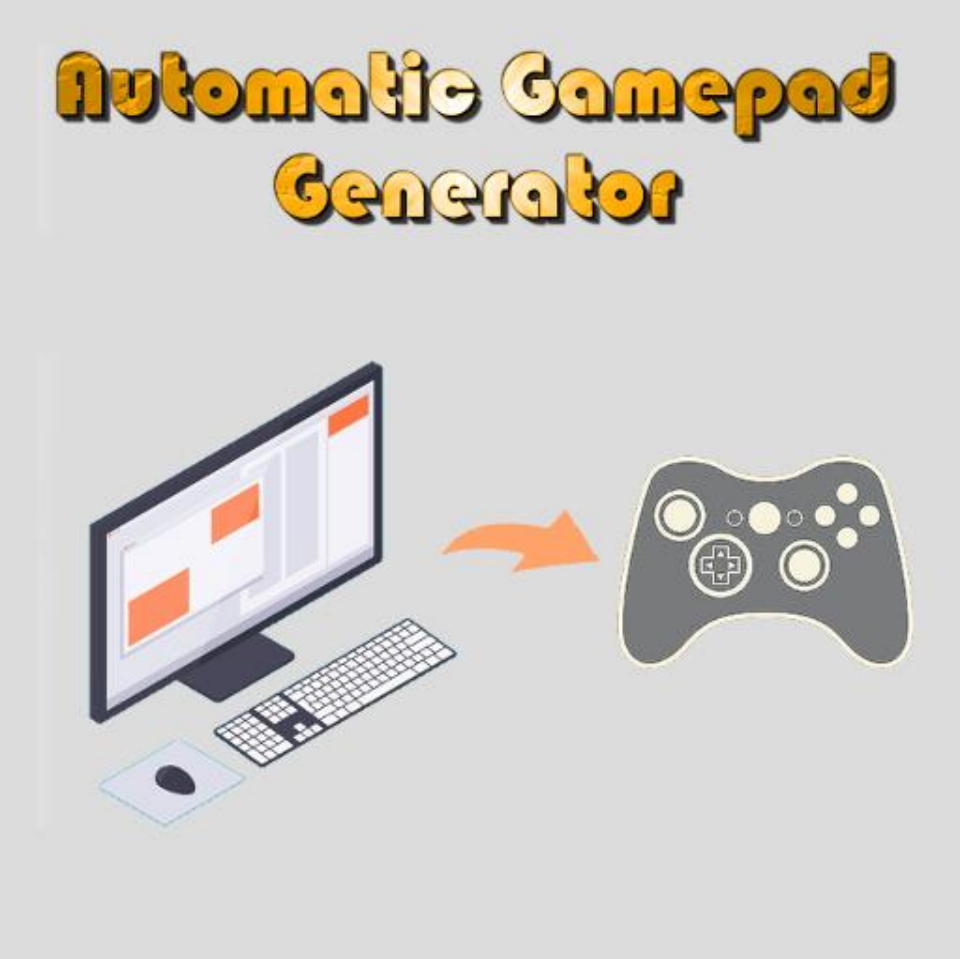
# Auto Gamepad Generator Documentation




## Introduction

Auto Gamepad Generator makes adding controller support to your existing game that has mouse/keyboard controls just a few clicks away. Takes less than 5 minutes to support controllers!

You've created your game with keyboard/mouse controls using Unity's Input API for quick iteration in the editor. Perhaps you've thought about controller support but which controllers and which platforms? Then you looked into it and have realized that there are no standards on controller mapping at all and it's a mess. Finally, something easy can be done about it!

With Automatic Gamepad Generator you can support today's most common gamepads within 5 minutes. No coding required and simply use our user friendly editor window to add the controller mappings that you desire for your game and that's it! We generate the additional Unity Input Manager virtual axes and buttons for you. Existing code will now work for gamepads!

Features

- Automatically detects your input manager
- Provides easy drop-downs for mapping
- No new API to learn
- No code changes required
- Remaps both keyboard and mouse
- both controls coexist seamlessly
- Works both in game and with Unity GUI menus
- Supports many controllers and platforms
- Takes just a few minutes

Platforms Supported

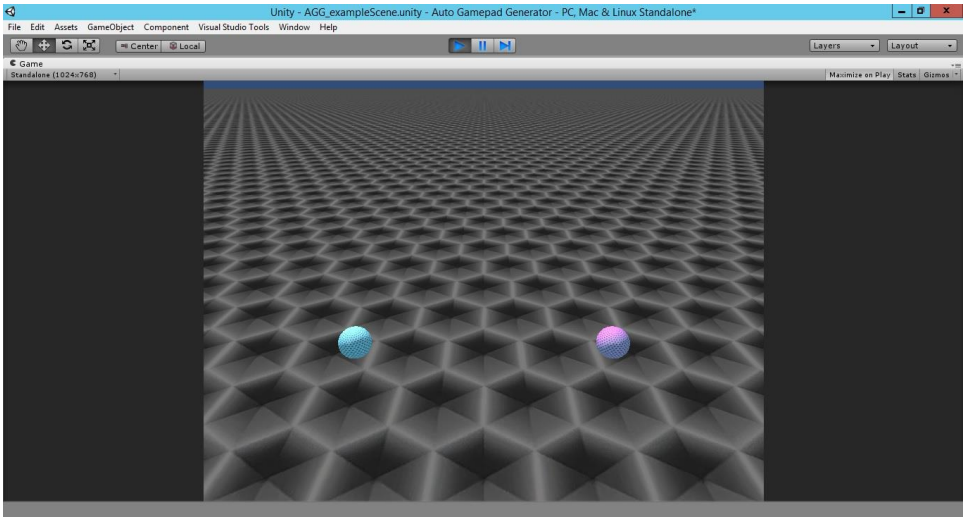
- Windows
- Mac
- Linux
- IOS
- Apple TV
- Android
- Xbox Consoles
- Playstation Consoles
- Fire TV
- Android TV
- WebGL (Chrome and Firefox)

## Controller List

List of the many supported controllers

# Example Scene

Auto Gamepad Generator comes with an example scene called AGG_exampleScene. When you open this scene you are presented with an example that is ready to show you how easy it is to add controller support. The first thing you will see is our intro explanation. It explains that the simple scene involves two spheres that are setup for keyboard/mouse input only. It also explains to open our mapping window you simply select Window -> AutoGamepadGenerator and our editor window will popup. This scene is designed for 2 players with the first player using the arrow keys and the second player using the WASD keys. Full keyboard controls for the example listed a little bit further down. If you hit play now you will see that only keyboard/mouse control is setup.


Full Controls

**Player 1**

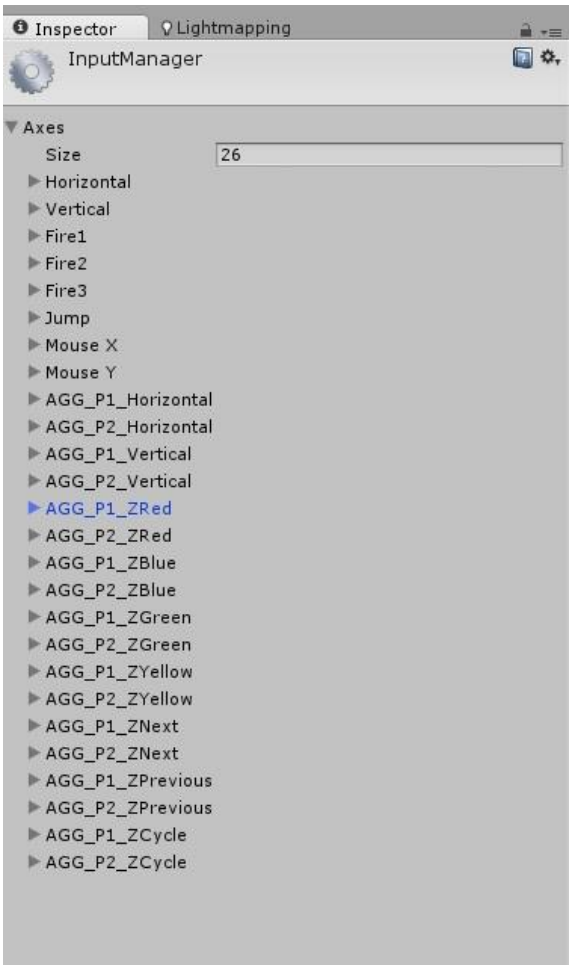
Movement- arrow keys
Turn Sphere Red- Keypad 0
Turn Sphere Blue- Keypad .
Turn Sphere Green- Keypad 1
Turn Sphere Yellow- Keypad 2
Turn Sphere Next Color- Keypad 9
Turn Sphere Previous Color- Keypad 7
Cycle Next Colors Fast - Keypad 6
Cycle Previous Colors Fast- Keypad 4

**Player 2**
Movement- WASD
Turn Sphere Red- f key
Turn Sphere Blue- g key
Turn Sphere Green- v key
Turn Sphere Yellow- b key
Turn Sphere Next Color- e key
Turn Sphere Previous Color- q key
Cycle Next Colors Fast - x key
Cycle Previous Colors Fast- z key

## AutoGamepadGenerator Editor Window

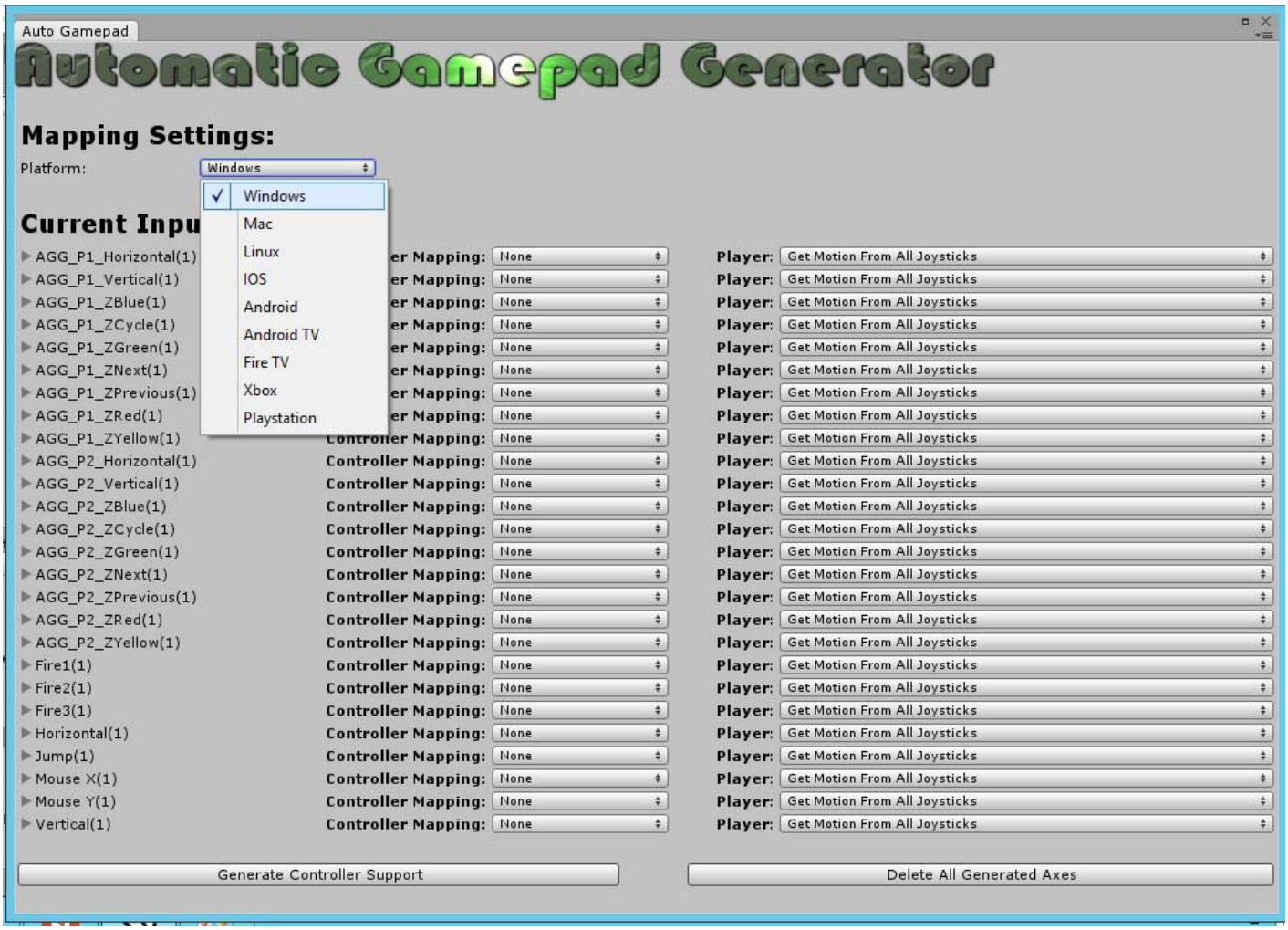
Our editor window is where all the action takes place. As you can see below you are presented with a nice GUI that represents Unity's Input Manager that can be found under Edit -> Project Settings -> Input. The example scene has in it all the input axes that are prefixed with AGG. Our scene doesn't use any of the default Unity included axes like horizontal, vertical, fire1, submit, or cancel but I've kept them in the screenshot for reference. In your game obviously these will be different and will be whatever you have setup for your game when you did your keyboard/mouse controls. Don't forget that if you are using Unity 4.6 or later that by default horizontal, vertical, submit, and cancel is the default way to navigate the new Unity GUI UI's (unless you change this in your game) and should be mapped to controllers as well.


Because we have not yet added controller support all the input Axes have (1) after them to indicate that there is only one current mapping corresponding to the keyboard controls. Below is how the same information is displayed in the normal Unity Input Manager and it is important to keep in mind these are just different representations of the same data.

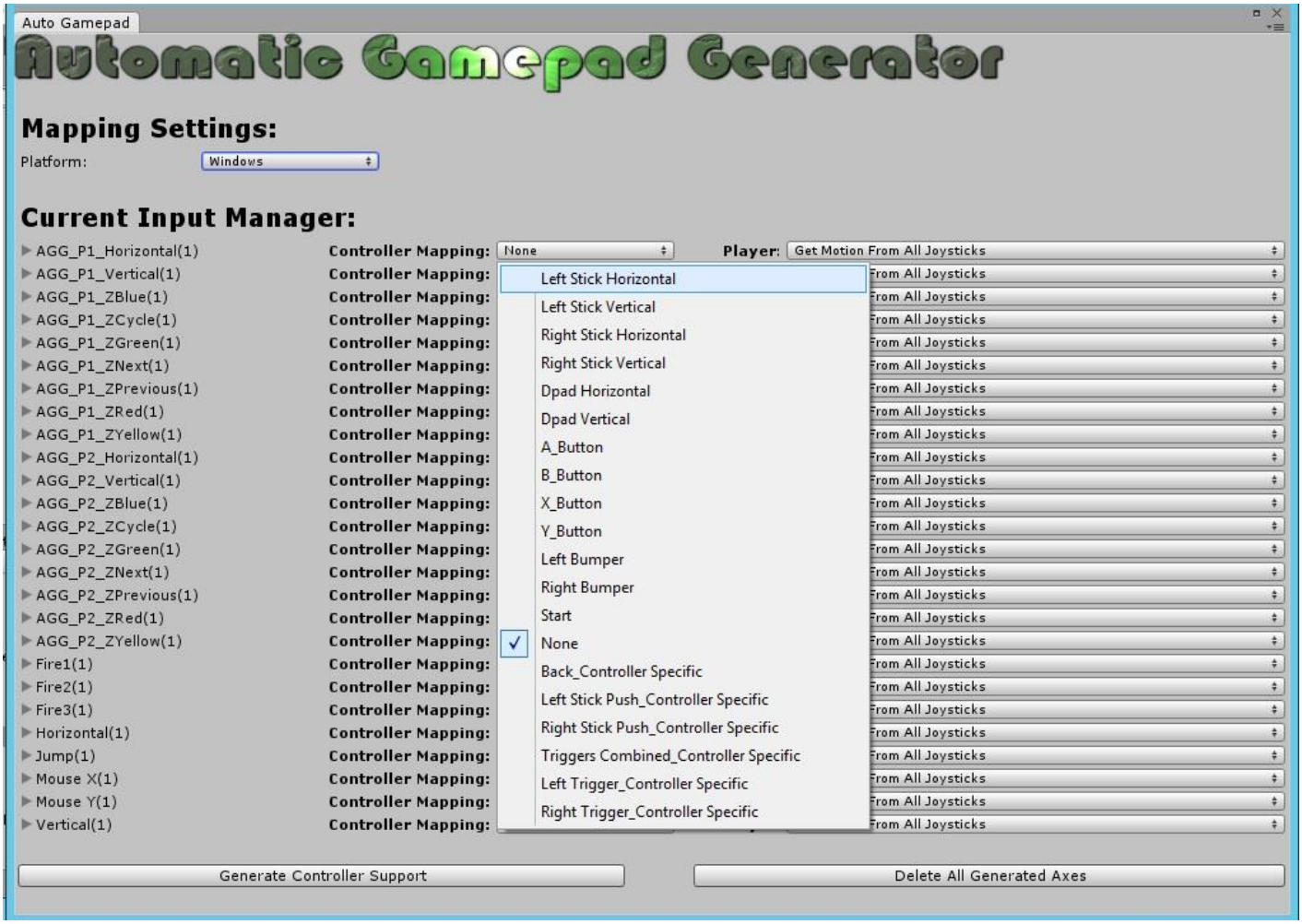



## Adding Controller Support

We have seen the basics of our existing scene and know it only supports keyboard/mouse currently and what is in our Unity Input Manager. Now I will show you how easy it is to add controller support. The first thing to do is to select our platform. The axis that will be automatically added for you are different depending on which platform your game is on so select what platform you wish to deploy to currently. If your game is multiplatform select the platform you are currently doing a build (you can regenerate easily later when you do a build for a different platform). If using play mode in editor for testing select the platform Unity is currently running on.

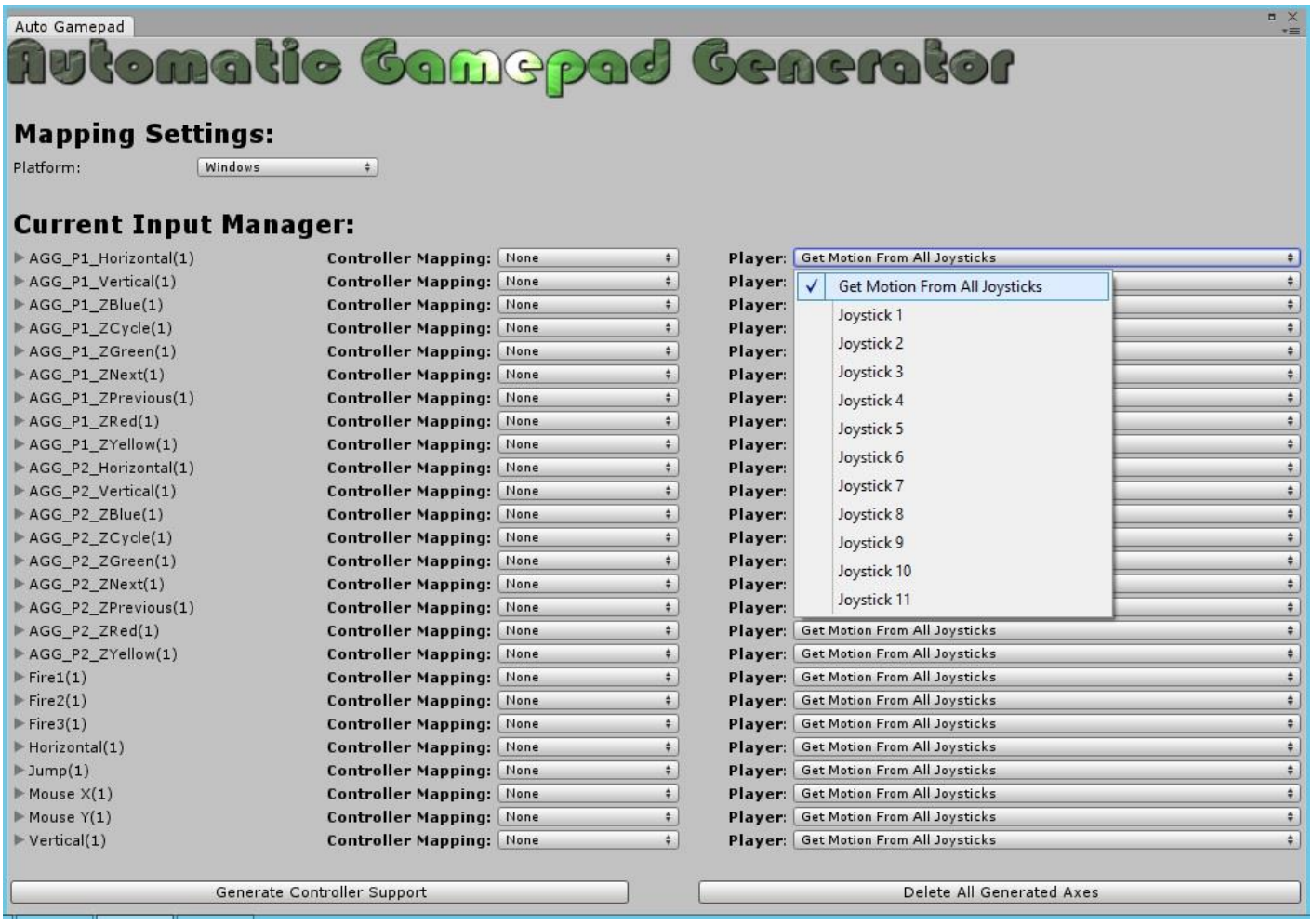



After selecting the platform we will simply add the mappings desired. Each Axis has two dropdowns to accomplish the mapping. The first is called Controller Mapping and is how we select what we want to control this input via the controller. As you can see in the screenshot below we can select any controller analog stick, trigger, or button.




As you can see all the mappings possible are listed and you simply select the one desired. You will notice the last 6 have a controller specific suffix. This is because not all controllers or platforms have these. For example, IOS mFi controllers don't have left or right triggers, clickable sticks, or a back button. If you know the platform you are targeting has those feel free to use them but it's important to know not all do. Also, you will see left or right trigger listed separately and also a triggers combined. On most platforms the triggers are treated as separate axis but on others it is one axis. This is fundamental to the platform so it is important to know which you are targeting. Windows and Xbox treat them combined (unless on windows the user installs a different non standard driver) but every other platform has separate axis. You're safest mappings are the non controller specific ones, but if you know the platform well you can certainly use the others.

The next dropdown pertains to what player you want to be mapped to. This dropdown is self explanatory if you want it to correspond to player 1 select joystick 1. Player 2 select joystick 2. If you are creating a single player game and any plugged in controller is fine you can select Get motion from all joysticks.




When done mapping all the controllers for your game simply click the Generate Controller support button and there you go controllers now work! Below are the recommended mappings for our example project. If you are on a Mac and not windows replace triggers_combined with left or right trigger.


If you hit play now you will see that your controller now controls the spheres.

Please note: If you are testing in the editor make sure you have plugged in the controller before you started the editor. If not, please restart unity to ensure the controllers work.

## Advanced Considerations

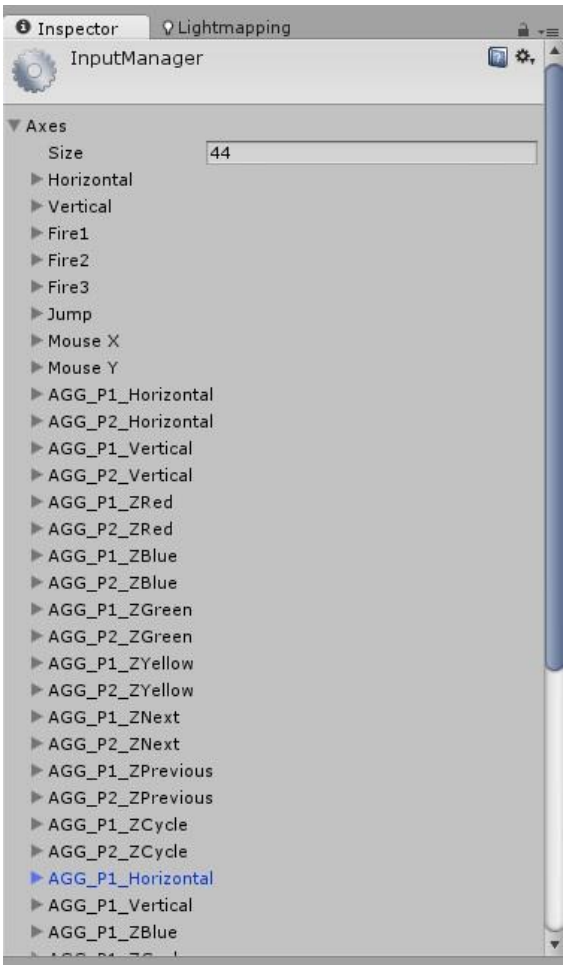
When you generate the controller support you'll notice the a generating dialog and then a success message briefly. If you don't have any further mappings you can close the window. If you leave the window open, you will notice a slight difference in the window pictured below.


Override defaults

As you can see all the items that we generated controller support for now have a (2) next to the name. This is because there are now two mappings under that name one for the keyboard controls and the other for the controller controls. We can expand the first one and we will see the details.


You can see above that corresponding to that entry the two mappings. The first one is the one we just generated that corresponds to "left stick horizontal" of "Joystick 1" and the second is the original keyboard controls of the left and right arrow. A couple things to note here if we care about the advanced controls. First, when we expand any entry there are override settings for gravity, dead zone, sensitivity, snap, and invert. Auto Gamepad Generator will always generate nice behaving values whether it is an analog stick or digital button so you generally don't need to touch these. However, if you want to you can just modify these values before you hit the generate button and they will be honored. Otherwise, we use our default values. Second, you will notice any generated axis has a descriptive name of AGG_Generated. This is so if we want to delete all our generations later we can do so easily if we made a mistake or if we want to deploy to a different platform. The whole process takes only about a minute so feel free to experiment if you want do a build and if you don't like it simply delete your old mappings and generate new ones.

Like before this is just a different view on the same old data. Below is what the Input Manager now looks like. You'll notice that AGG_P1_Horizontal and the others now have two virtual axes instead of before where there was only one.




More Than One Mapping

It's common for some games to have more than one mapping desired. An example of this is many games allow the user to use either the left stick or dpad for movement. This is easily accomplished in Auto Gamepad Generator by mapping left stick horizontal generating and then mapping dpad horizontal and generating again on the same entry. This brings up an important point you can generate as many times as you wish and it will create multiple mappings. The old sonic games if you hit any button you would jump if you wanted to recreate this you would just map the same entry to the each button and hit generate after each mapping.

You can map your controls all at once like the example we went through or do them individually if you want. Each time when you generate anything mapped as "none" will be ignored and only those populated will be generated.

Deleting Generated Axes

At any point if we want to start over whether it was a mistake or we're building for a new platform simply click the Delete all Generated Axes button. Then we're back to the beginning like below.


You'll notice that the original keyboard mappings are there just fine only the generated axis have been deleted so that we can remap without issues.

## Platform Considerations

Controller consistency across platforms and even within some platforms doesn't exist. Some controllers might be button 0 as the "A button" and others button 16. You don't need to worry about that we have done the hard work of giving you maximum coverage for the most popular controllers. However, there are times where conflicts are unavoidable so there will be a few controllers that don't work. We have made sure to support the controllers with the largest market share. Across different platforms the consistency is even worse so if you generate for one platform and then want to build for a second make sure to delete your generated axes change the platform dropdown and regenerate them. You would think there would be some sort of standardization across manufacturers so this wouldn't need to be done but that is not the case. The process is so quick though that if your game is multiplatform it is not that big of a deal.

Most platforms we have 100% controller coverage (IOS, android, Fire TV, android TV, Xbox, and Playstation). Even on PC, MAC, Linux we support over 75% of the controller market share on those platforms.

# Unity Input API Usage

Just about all of Unity's Keyboard/Mouse API is fully reusable between keyboard/mouse and controllers. After all, that is the point of virtual axes.

**if (Input.GetButtonDown("AGG_P1_ZRed")**
**{ //..do something }**

Or

**Float horizontal = Input.GetAxis("AGG_P1_Horizontal");**

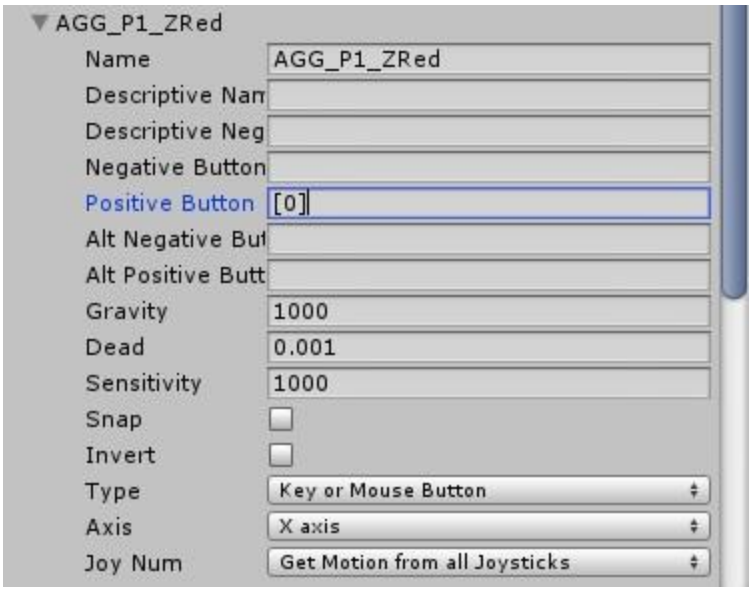
These will work regardless of whether its keyboard, mouse, controller. There is one exception when it comes to generating controller support though which is if you used GetKey and a keycode instead of creating a virtual axes. Luckily though, this is easy to remedy just make a virtual axis that corresponds to that call and now it works the same and is easy to generate controller support. To summarize if you had the following:

**if (Input.GetKeyDown(KeyCode.Keypad0)**
**{ //..do something }**

Change to

**if (Input.GetButtonDown("AGG_P1_ZRed")**
**{ //..do something }**

In the input manager you create the following and it works the same and can generate as much controller support as you want easily from then on!




## Summary

That's all there is to it! Finally it's possible to support controllers easily with no code changes, no new API to learn and it only takes a few minutes! If you didn't have support for controllers before, now you have no excuse not to support them!