

**Tutorial:**

**Using Wiimote Controls in Unity Indie**

By Meena Seralathan

## Introduction

Interested in making the next New Super Mario Bros. in Unity without shelling out the money for Unity Pro? This tutorial will show you an easy way to use your Wiimote in Unity Indie to control a character in a 2D Platformer; the general concept will also work for First-Person Shooters, 3D Platformers, and a number of other game types as well.

So let's get started!

## What You Will Need

For this tutorial we will use a program that will handle Wii input independently from Unity; if you are under Windows, this program is called WiinRemote, and if you are using MacOS it is DarwiinRemote. A quick search on either will lead you to the program you need.

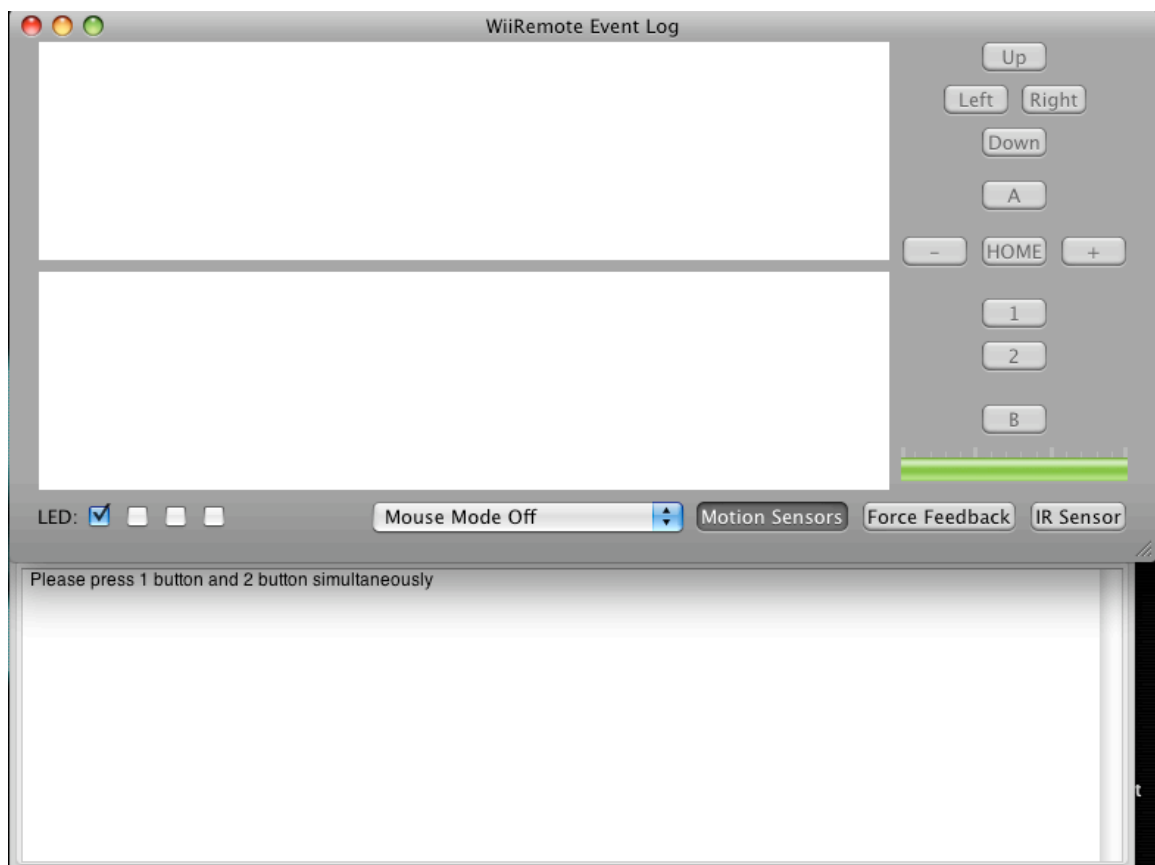
You will also need the 2D Gameplay Tutorial project. This has been provided for you, though if you already have a project based off this tutorial that you want to work with, you may use that.

You will (of course) need a Wiimote, a working Bluetooth connection, and (if you plan on using IR for movement), an IR sensor bar.

## Connecting Your Wiimote to Your Computer

Make sure your computer's Bluetooth is activated, disconnect any Wiimotes currently connected to your computer (and remove them from your list of added devices — see appendix if you are unsure of how to do this), and then do the following:

**Mac:** Open the DarwiinRemote application. You will get a screen similar to this one:

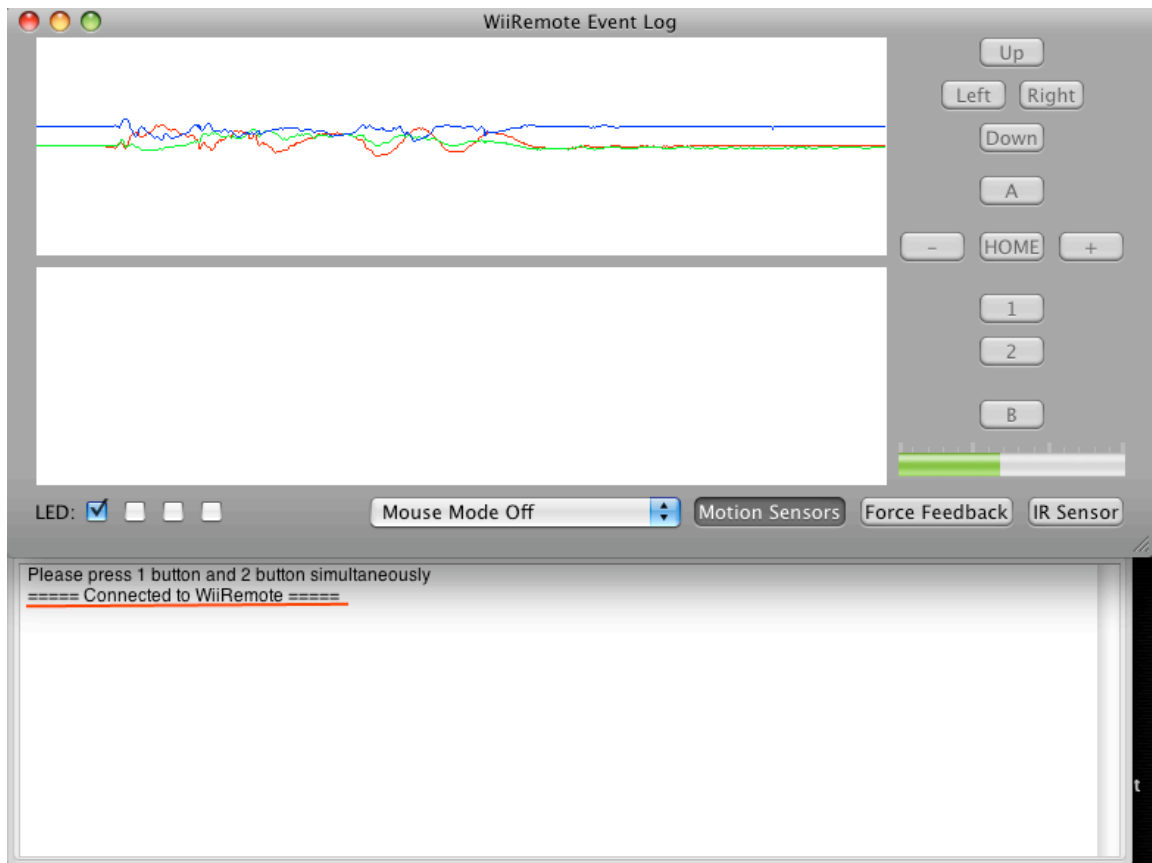


The empty white box is where all the Wiimote's motion will be visualized; to the side is a map of the Wiimote that will indicate which buttons are being pressed.

The bottom window is a debug window; when you start the application it will tell you to press 1&2 (in other words, to

connect your Wiimote to the computer). Either press 1&2 or press the little red button in the back that is used to connect the remote to a Wii; the blue lights on the bottom will flash a bit, and eventually you should see either red, blue, and green lines in the top window that oscillate whenever you move the Wiimote around, or an error message in the debug window saying the Wiimote was unable to connect.

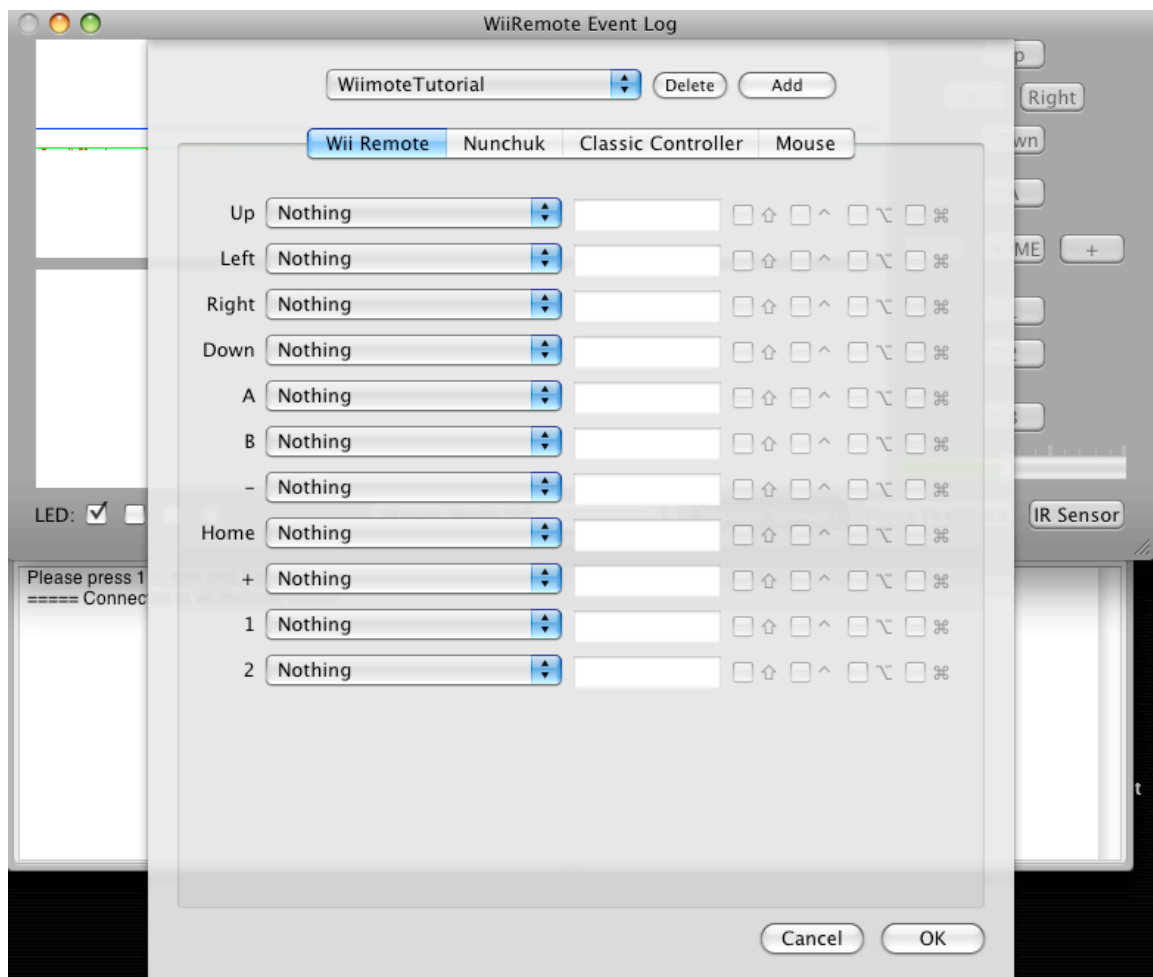
If you see the former you are good to go; if you see the latter repeat the process until you see the motion visualization (if necessary, try disconnecting the device from the Bluetooth screen and trying again).



each of the colored lines corresponds to a particular axis in space

Once you have the Wiimote connected, go to the program menu and click DarwinRemote -> Preferences...

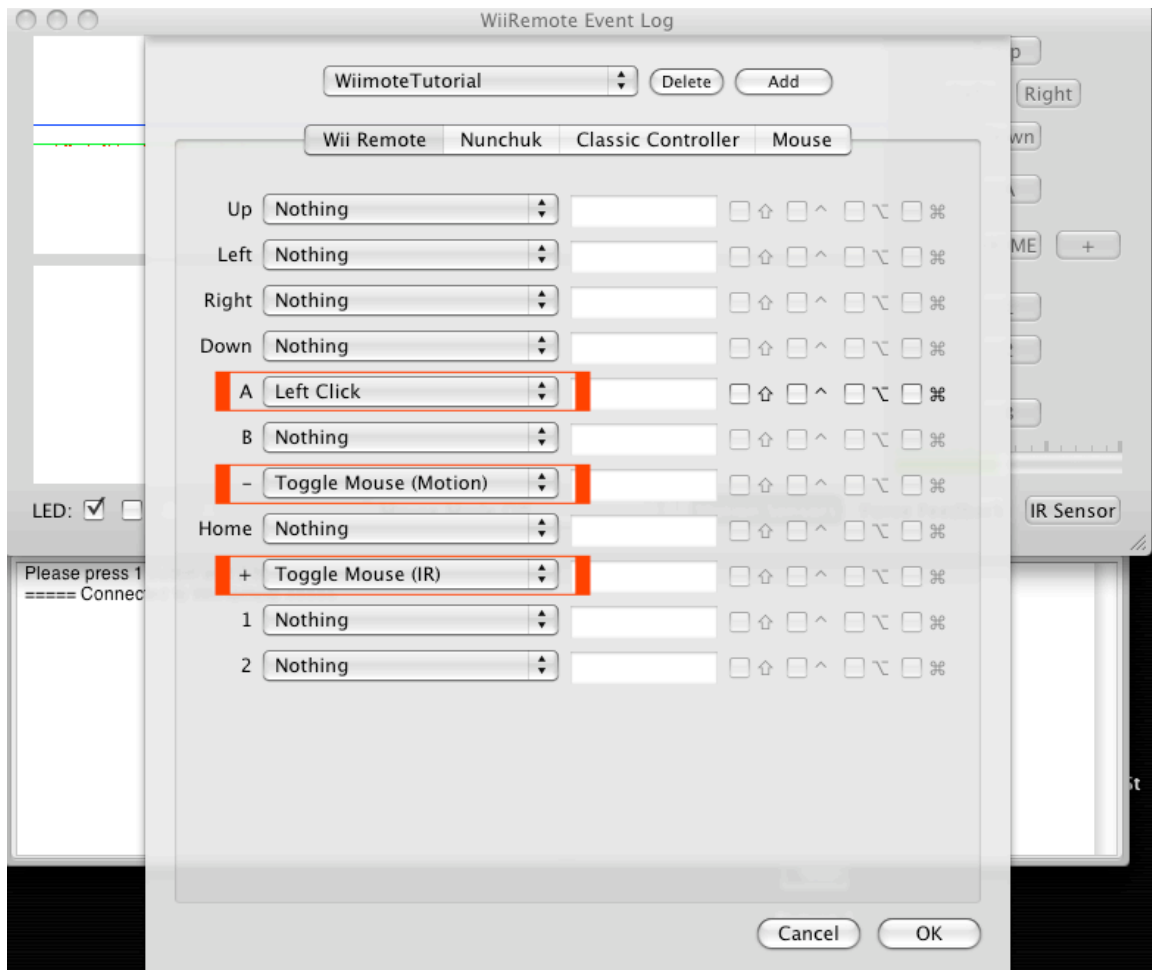
You should get this screen:



Click on the Wii Remote tab if it is not already selected. You'll see a form with all the buttons on the Wiimote, along with drop-down lists where you can choose what to map each button to. When you first open the program it will already have

most of these pre-assigned; for our purposes, click the Add button at the top of the window, name the new mapping “Wiimote Tutorial”, and hit OK. Now everything should be empty.

Now let’s make a simple mapping that will let you use the Wiimote as the mouse; you will need to be able to enable and disable motion or IR sensors at will, and to left-click. For now let’s set A to be our left-click, let’s toggle our use of motion sensor movement with the – button, and let’s toggle IR sensor movement with the + button. Click on the drop down menus next to each of these buttons and scroll until you find these input options; ultimately your new window should look like this:



Click OK to exit out the preferences, and make sure both the Motion Sensor button and the IR Sensor button are pressed in the top window.

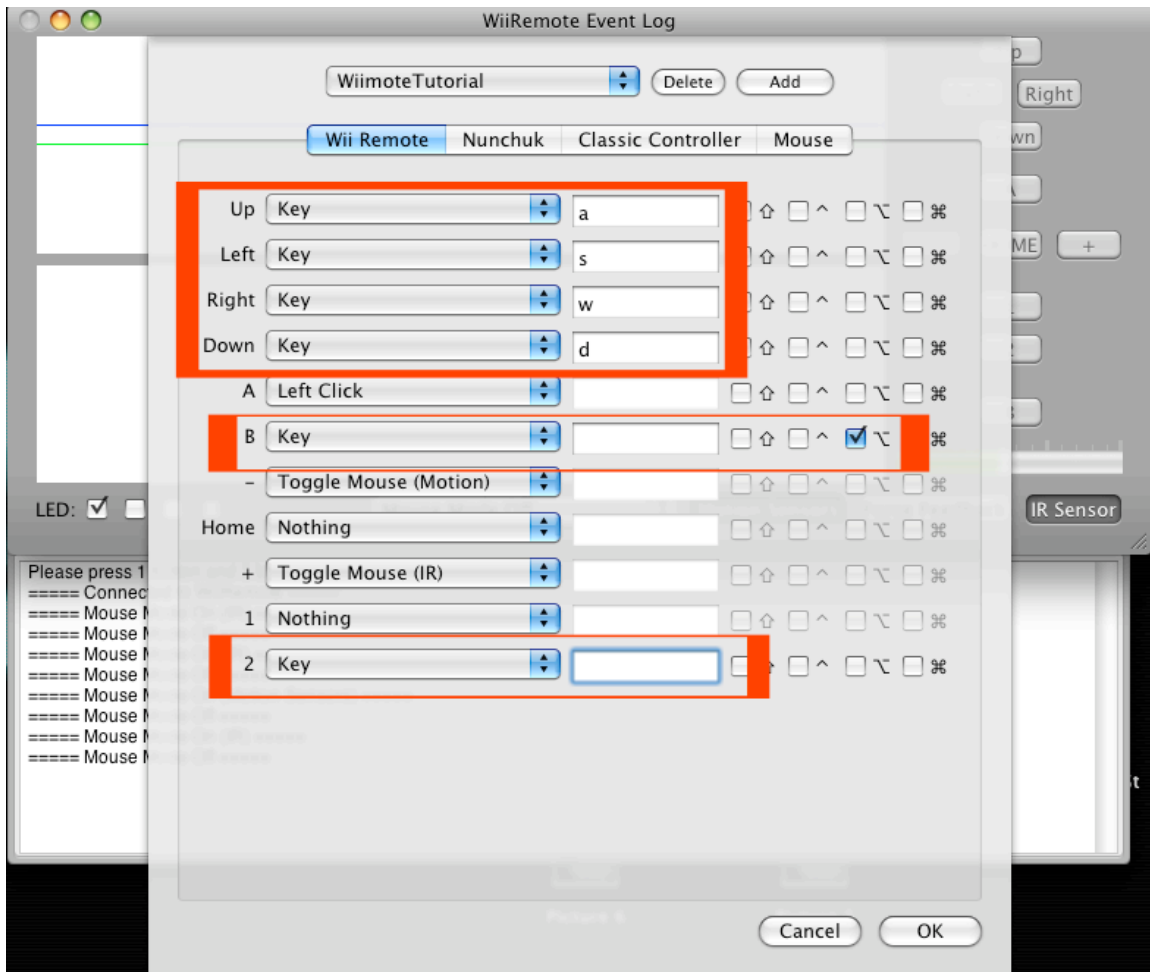
Now try using the Wiimote! Press the - button and tilt the Wiimote around, clicking things with A, or press the + button and aim the Wiimote as you would with the Wii. The sensitivity of both can be tinkered with in the Preferences window under "Mouse." If you don't like left-clicking with A, try mapping other buttons to the left-click and testing it out.

If motion control's not your thing and you would rather learn how to just use the buttons to control your character, head to the next section; otherwise head to "Changing Input in Unity," where we will take a closer look on Unity's default input interface and figure out how to change its settings for our purposes.

## **Using Wiimote to Control Character Without Motion/IR**

By default, Unity uses the W, A, S, D keys to walk around, the Ctrl button to run while moving, and the Spacebar to jump. So if we go back to the Preferences screen and map the remote to these keys, we can immediately control the movement of our character without the need to alter anything in Unity.

So let's set the buttons to the following:



(Note '2' is set to a space).

Exit out of the window, open your Unity project, run the 2D Platformer scene, and try moving Lerpz around with the sideways Wiimote configuration we just made. It should work, minus some possible issues with running.

### *What is going on?*

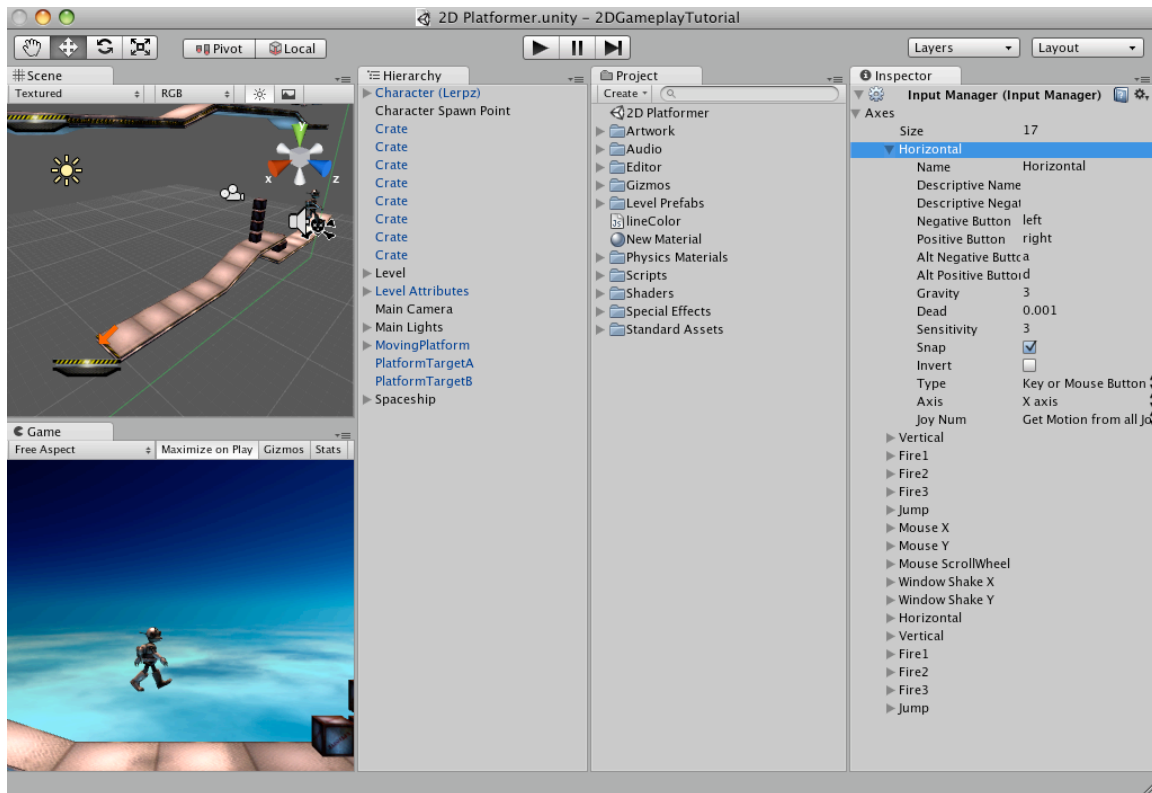
Basically, rather than directly sending Wiimote input to Unity, we use the program WiinRemote/DarwiinRemote to collect it for us, and then we use it to connect the Wiimote to the computer as an input device that can then interact with Unity Indie without the use of a plugin or other complicated workarounds.

This makes using the Wiimote very simple to use for a Unity game, but as you may notice when trying to run, trying to use certain button combinations with the Wiimote may not work as well as they did on the keyboard (namely, pressing B+<direction> may not work as well as Ctrl+<W, A, S, D> did on the keyboard). It's not a flaw of the Wiimote, so much as a difference in how we are used to pressing buttons on each device.

If you don't really care about running or have minimal problems running in your game, then congratulations, you are done with this tutorial! If you're still waiting to figure out how to use motion/IR-dependent controls, or if you want to fix that pesky running problem, we will discuss how to change Unity input in the next section.

## Changing Input in Unity

Now let's turn back to Unity for a moment. Open up the 2D Gameplay Tutorial and double-click the 2D Platformer scene. Make sure you are in a 2 by 3 layout. Go to Edit-> Project Settings -> Input. In the Inspector Pane you will notice it says Input Manager, and has an Axes property in it. Expand it and you will find a list of different default actions that can be made, and if you expand each one you will find the buttons/keys mapped to each action by default.



The Horizontal part of the Axes property, expanded

In this tutorial's project, since there are no weapons, they use "Fire1" as the input for running; if we expand Fire1 and change the use of the Ctrl button to be something that will make running on the Wiimote even easier (say, 'p'), the input Unity uses will change such that users will have to hold 'p' to run. We can then go back to WiinRemote/DarwiinRemote and change B to respond to 'p,' and when we try controlling the character with the Wiimote again running will work better.

We can also change the Horizontal control to move when someone left-clicks, but if you try this out you notice the character just walks in the direction it's facing in, regardless of where the mouse moves. In order to get the character to follow the mouse we will finally be doing some coding, which will change the character controller code to change the direction of the player based on the position of the mouse. We will also get the character to jump if the Wiimote is flicked up.

## Looking at the PlatformerController Code

In this project the character controller is called the PlatformerController, and we will be altering this file for our purposes; however, the same basic premises apply to controllers in other games.

Basically, we want to change the controller code such that the character rotates to the right if the mouse is on its right, and the same for the left. This requires getting the mouse input, figuring out where the character is on the screen, and updating the direction the character faces.

First things first. In the Project Pane, go to Scripts -> 2D and double-click the PlatformerController file. Find the Update function, and at the beginning add the following code:

```
if (Input.mousePosition.x > Screen.width/2)
    movement.direction = Vector3(1, 0, 0);
else if (Input.mousePosition.x < Screen.width/2)
    movement.direction = Vector3(-1, 0, 0);
```

Now, if the mouse moves to the left side of the screen, the character turns, and the same is true of the right. But this is not very intuitive; we would rather have the turning depend on where the character is (roughly) on the screen, rather than just where the center of the screen is. So if we replace that code to this:

```
var turn = 500 - Input.mousePosition.x;
if (turn > 50)
    movement.direction = Vector3(-1, 0, 0);
else if (turn < -50)
    movement.direction = Vector3(1, 0, 0);
```

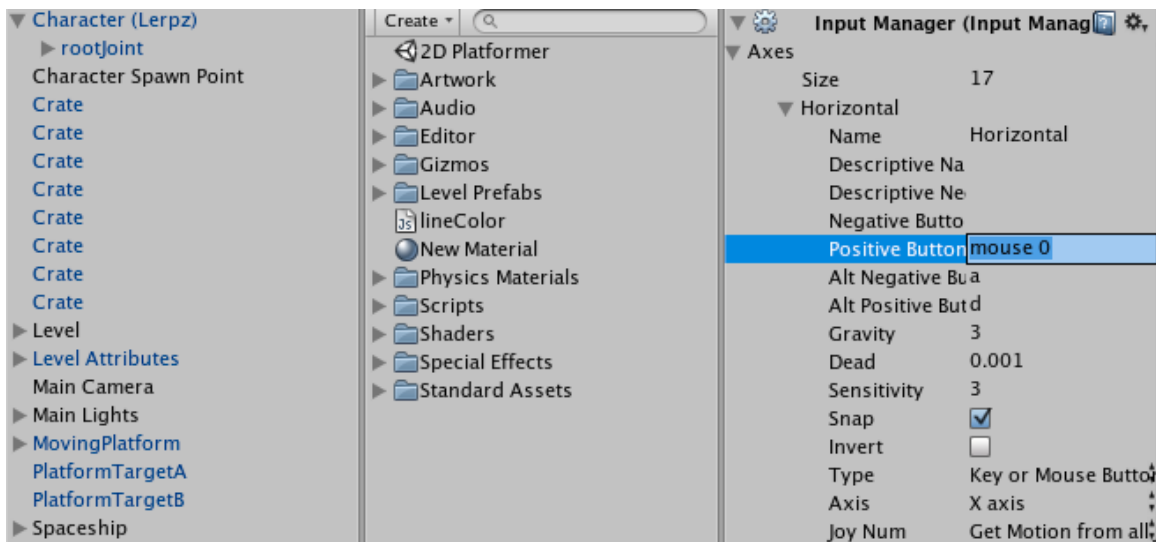
The pivoting works a bit better. It isn't actually tracking the character model, but it acts like it is, making the controls more intuitive. Try adding and subtracting constants from the position of the character in order to make the turning more or less sensitive.

Now let's change the controls so that the mouse will control movement. First, you will have to find the `UpdateSmoothedMovementDirection` function and delete/comment out these lines:

```
if (movement.isMoving)
    movement.direction = Vector3 (h, 0, 0);
```

Leaving these lines in the code will cause your character to walk in one direction (even if you turn the character), and are unnecessary.

Now go back into the Input settings for the project, and change the Horizontal controls so that they react to a left-click.



Try it out; you should be able to hold down the left-click button and see your character move, and you should be able to turn your character and have it walk in that direction as well.

Now let's work on jumping. We are going to make it so that the user flicks the Wiimote up to get the character to jump. The simplest way to do this is to trigger a jump every time the pointer is a certain height above the character. We will have to get the height of the mouse, and change all references to a "jump" button in the code (as we will not be using a button for jumping).

We are going to introduce a new variable that will use the position of the mouse to determine whether it is above the character or not. Add the following line after the declaration of the `turn` variable we just created:

```
var jumped = (400 - Input.mousePosition.y) < 0;
```

Now in the `Update` function (probably just below where you have been adding all of this code), there is an if statement that checks to see whether the jump button has been pressed and whether the player can jump. Edit it so it looks like this:

```
if (jumped && canControl)
{
    jump.lastButtonTime = Time.time;
}
```

Now find the `UpdateGravity` function and change the first line (which is looking for jump button input) to the following:

```
var jumpButton = (400 - Input.mousePosition.y) < 0;
```

Now run your project. By keeping the mouse above the character, you can get the character to jump! Assuming the player actually flicks the Wiimote, the fact that it jumps repeatedly will not be an issue; however, if you want to ensure the character will only jump again after a second flick, you can introduce some more variables to keep track of when the mouse has moved back down towards the character, and prohibit the character from jumping until this up-down movement has been completed. Try it out!

### *Extra Credit*

Now that you have the movement working, try giving your character the ability to shoot projectiles (use the FPS tutorial provided by Unity for guidance), and map the fire button to the B button on the Wiimote.

And there you have it! By altering some of the controller code provided by Unity and working with some of its built-in functionality, you have successfully used the Wiimote to control a game character without the need for plugins or other packages that would require Unity Pro! You can use the same concepts to change a First-Person controller to respond to the Wiimote, as well as controllers for almost any other type of game. Have fun testing it out!