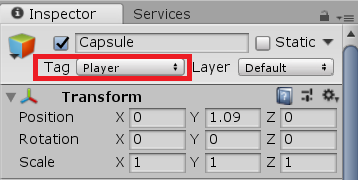
This script will demonstrate how to allow an object to both jump **and** double-jump.

**Step 1 – Creating the scene**

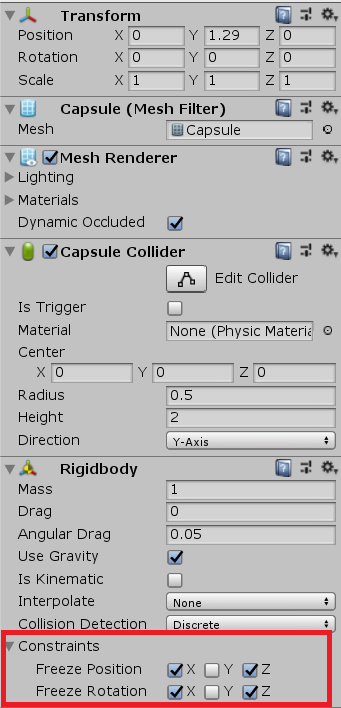
To begin, you must first construct your scene. To do this, create a scene and call it whatever you wish, preferably to the name relating to what it is you’re making, such as “double jump”. When loading this scene, you will need to navigate to the 3D Object dropdown tab from the “GameObject” tab in the top left of your screen. From this you will need one of the following:

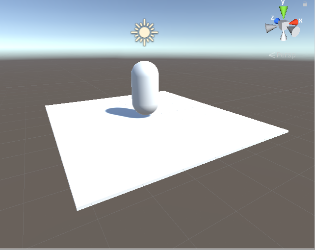
* A cube
* A capsule

You must provide the capsule with a form of physics to ensure it does not simply hover in place. To do this, give the capsule a “Rigidbody” by adding the component via; the inspector tab on the right-hand side after selecting the capsule. Our capsule will be considered as our “player” in the scenario when put into the context of a game. As such, you must set the tag of our capsule as a player, as seen below:

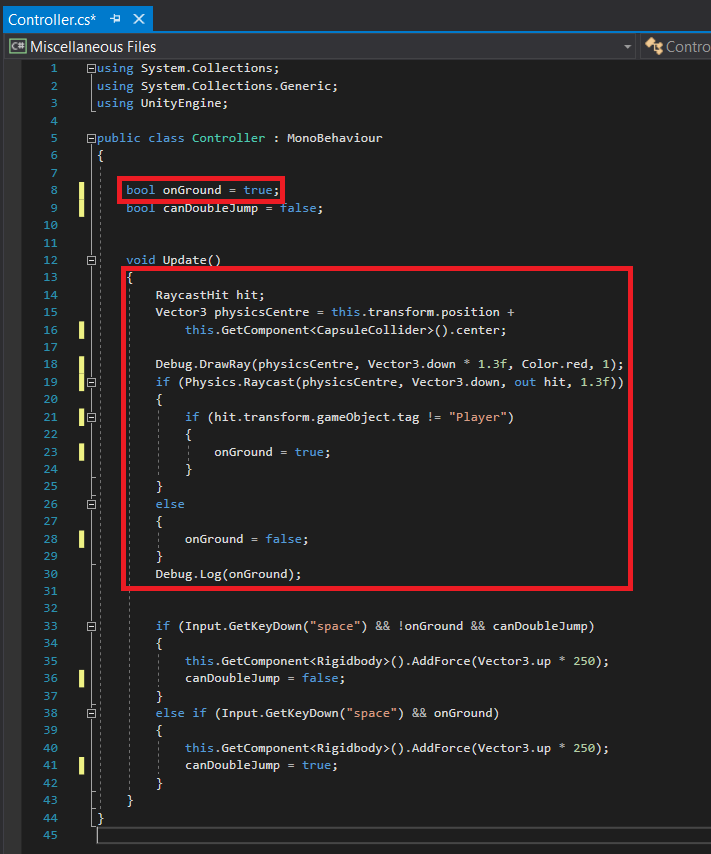


As the capsule has a rounded bottom it will be prone to tilting when landing from a jump and falling over when being moved. To prevent this, go to the Rigidbody component and select the constraints dropdown. You will be selecting the following checkboxes:



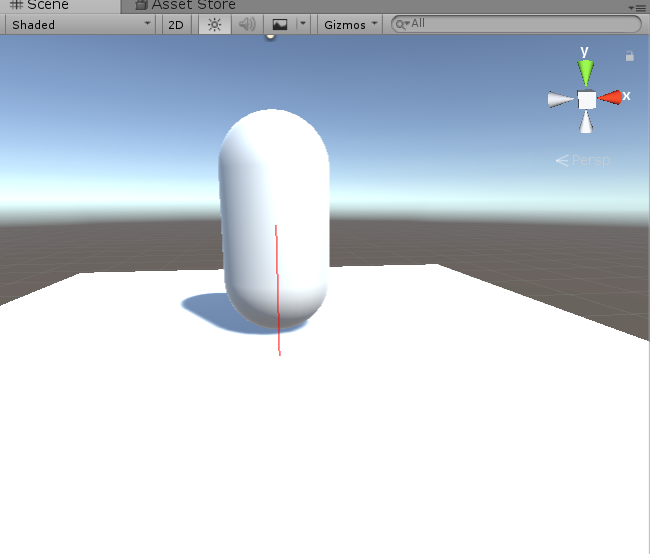
After your capsule is correctly configured, select your cube and construct it so that it looks like a base/platform for your capsule to sit on top of, your scene must look like the following:

**Step 2 – Making the Raycast**

****To begin, create a C# script named “Controller” and write the following scripts highlighted in the red boxes below:

The “onGround” bool is simply a true or false statement to determine whether our capsule is on the ground, or airborne.

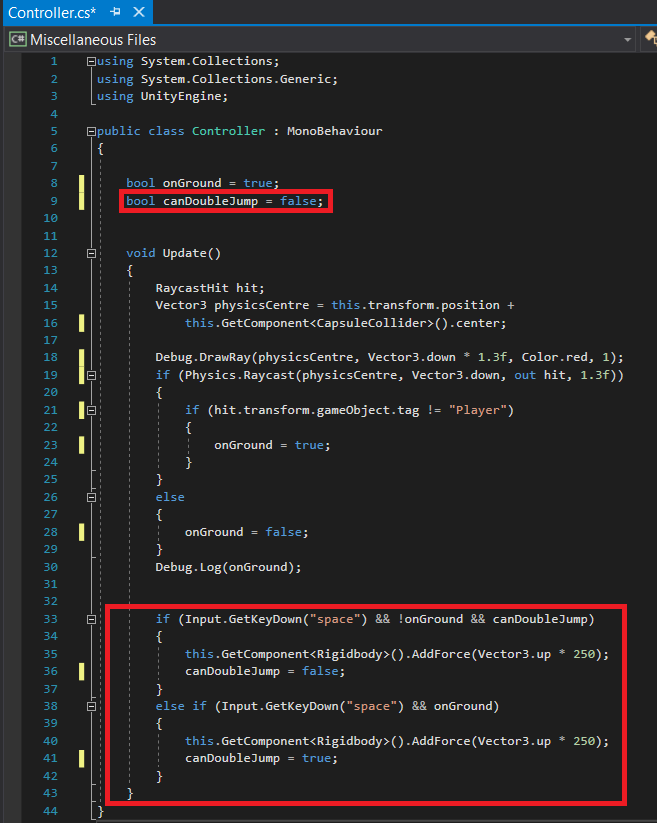
The second highlighted string of code you see is to create a Raycast, which will help our script detect whether our capsule is grounded or not, think of it as an invisible laser. From the “this.GetComponent<CapsuleCollider>().center;” line, we are specifying that the Raycast will be emitted from the center point of a CapsuleCollider, something of which our Capsule already has implemented onto it via; the inspector. From this, it will draw a Ray from the center of the Capsule downwards (Vector3.down) by a certain amount (\* 1.3f). You can choose to make your Raycast a certain colour to aid in spotting it within the scene screen as well. For the sake of easy detection, we will choose red as the colour of choice. Once this Raycast is created, it should look like the following (when applied to the Capsule):



Next, an “if” statement is made to determine whether our Raycast is hitting a “gameObject” that does not have the tag of “Player”. If this is the case, then that must mean that the player is close enough to a surface to be deemed as grounded, otherwise known as “onGround = true;”. If this is **not** the case, then an “else” statement will be created to set the “onGround” bool to false, as this would mean the Capsule is not close enough to a surface and is therefore airborne.

**Step 3 – Adding jump / double-jump conditions**

Our next step will be to allow the capsule to jump, as well as double-jump (with the help of the previously defined “onGround” bool variable) as seen by the highlighted script strings below.

****

First, create an undefined bool named “canDoubleJump” and set it to a default value of false. After this, you will be writing an “If” statement specifying what key will be pressed to allow the action in question (Input.GetKeyDown(“space”)), as well as the two other variables that must be either true/false for the action to take place. In this case, onGround must be true and canDoubleJump must be false. Once these conditions are met, this will allow your capsule (“this.GetComponent<Rigidbody>()) to jump when space is pressed, as seen by the “.AddForce(Vector3.up \* 250);” line. This is telling the capsule to “jump” at a max height of 250 units when space is pressed. This action being performed will mean that the canDoubleJump variable was set to false.

Next, we will be writing an “else if” statement to define what is to happen if the previous jump conditions are **not** met. However, these will follow the conditions “&& onGround”, which means the onGround variable is false (meaning the capsule is airborne). When the capsule is airborne, the player will be able to jump once more of an equal height that of the first jump. Once this action is done, the process will be updated and canDoubleJump will be set back to its original value; “false”.

**Step 4 – Applying the script to the capsule**

Once the Scene, Raycast and Jump conditions are created, apply the “Controller” script to the capsule by dragging the script directly onto the Capsule name in the hierarchy located on the left side of your screen. You will then see the script within the Capsules’ components bar.