**RayCasting**

**Debug.Draw:**

To Cast a debug draw line straight down type:

* Debug.DrawRay (transform.position, Vector3.down \* 5, Color.red);

This draws a line through the World Origin 5 units down, depending on angle of player

* *Debug*.*DrawLine*(*transform*.*position*, Vector3.down \* 5, *Color*.*red*);

DrawLine relies on world coordinates and a specific start point and end point. If you leave the end point as just Vector3.down, the function will assume you mean world space. You can create a new vector3 when factoring in your player’s transform.position like:

* *Vector3* down = *transform*.*position* + new *Vector3*(0, -5, 0); *Debug*.*DrawLine*(*transform*.*position*, down, *Color*.*red*);
* This will now act the same as Debug.DrawRay

**Gizmos:**

Gizmos are like using Debug Draws except they show up in game view and are a separate function.

void OnDrawGizmos()

{

*Gizmos*.*color* = *Color*.*white*;

*Gizmos*.*DrawLine*(*transform*.*position*, *transform*.*position* + *Vector3*.*down* \* 5.6f);

}

**Physics.Raycast :**

(Use a Ray Cast to determine if touching the ground)

if ((*Input*.*GetButtonDown*("Fire1") || *Input*.*GetButton*("Fire1")) && *Physics*.*Raycast*(*transform*.*position*, *Vector3*.*down*, .55f))

{

rb.*AddForce*(new *Vector3*(0, jumpPower, 0), *ForceMode*.*Impulse*);

}

**Using Ray casts to ignore certain layers :**

(Allow players to jump when colliding with ground and not when standing on top of other players)

* You need to create a new layer in your layers settings
* Keep track of the layer number that you want the player to be able to jump off of (ground layer, like say layer 9)
* Then, in your script add an int parameter using a bitwise operation:

int layerMask = 1 << 9; // << Shift left bitwise operator to exclude all other flags of layers

if ((*Input*.*GetButtonDown*("Fire1") || *Input*.*GetButton*("Fire1")) && *Physics*.*Raycast*(*transform*.*position*, *Vector3*.*down*, .55f, layerMask)) // add a 4th parameter to the raycast function to account for layers.

**Physics Layers:**

For example – Say you wanted to have a player not be able to jump off a wall parallel to the player’s vertical position (you don’t want wall jumping). You need to have those walls be ignored by your raycast.

Physics Layers settings Matrix for the entire project are in:

* Edit > Project Settings > Physics

To Change the layer of a specific object:

* To change the layers of an object, you can click on the drop down right below the objects name in the inspector window.
* Edit > Project Settings > Tag & Layers

To Add a new physics layer to the project:

* To add a new layer, click on the drop down below the objects name in the inspector window, and select add layer (the same you do a tag)

**Alternatives to using Ray Casting (use additional colliders):**

Attach a child object with a collider on the parent to extend the range of the collider slightly

* Use a bool value that is flipped depending if the collider is colliding with the ground.
* Tag the ground with a tag like “Ground”
* The if statement in the fixed update would look something like this:

If (bIsGrounded && Input.GetButtonDown("Fire1"))

{ rb.AddForce(Vector3.up \* jumpPower, ForceMode.Impulse); }

Then create two void OnCollision functions that flips the bool value like:

void OnCollisionEnter(Collision other)

{

if (other.gameObject.CompareTag ("Ground"))

{

Debug.Log("Hit");

bIsGrounded = true;

}

}

void OnCollisionExit(Collision other)

{

if(other.gameObject.CompareTag("G"))

{

Debug.Log("InAir");

bIsGrounded = false;

}

}