CODE NINJAS®

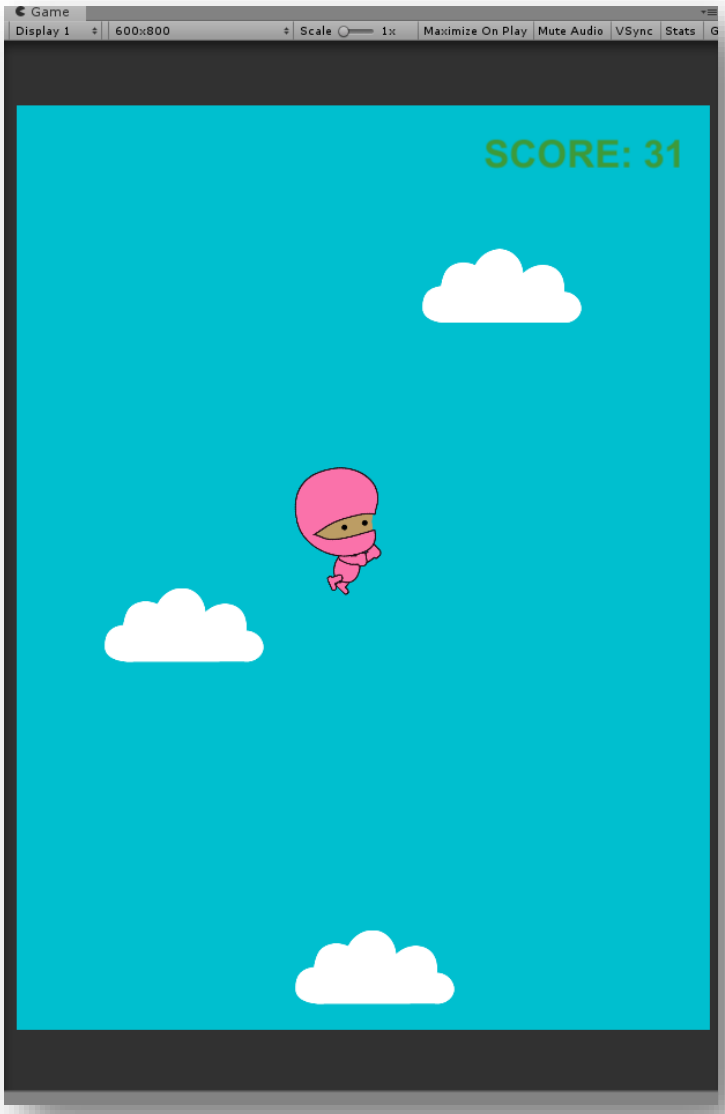
Silver Belt Ninja Guide
Activity 03: Cloud Hop

Activity 3

Cloud Hop

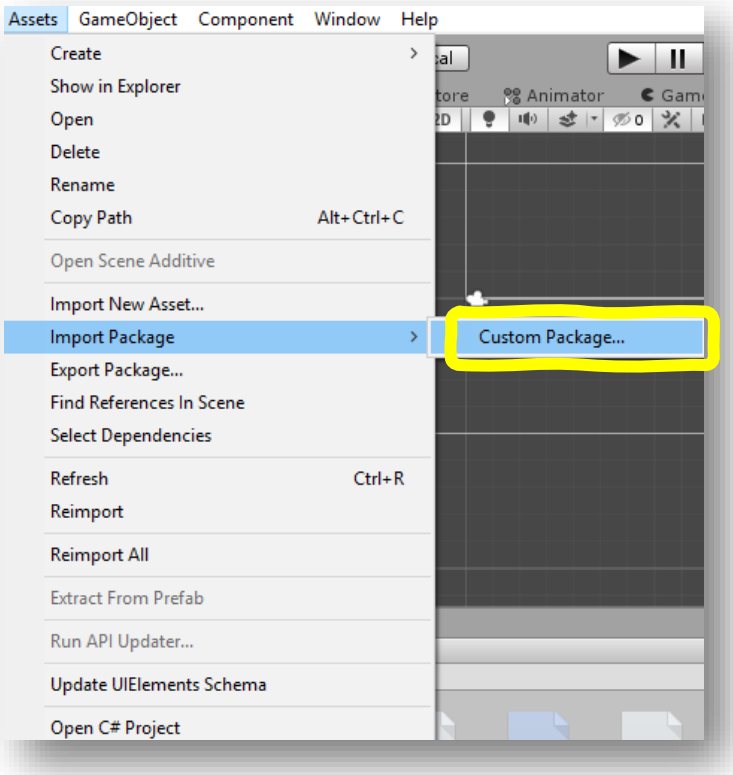
You will be able to make a player jump using `Input.GetButton` to check for input, `AddForce` to simulate the jump, and `velocity` to check if the player is grounded. You'll also learn more about using the **rigidbody** component.

In Cloud Hop, the sky's the limit! Jump from cloud to cloud to see how far you can go. Each cloud earns you a point but be careful - falling off restarts the game!

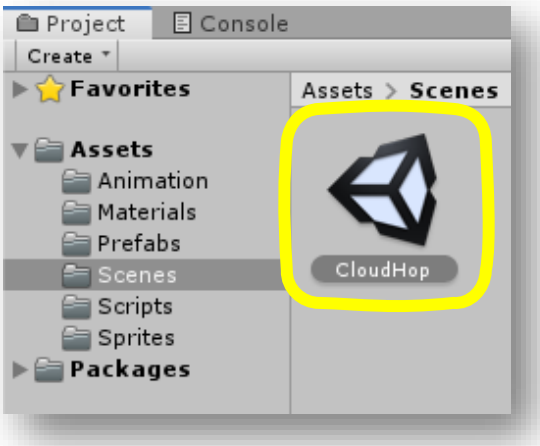


1 Start a new Unity Project and name it *YOUR INITIALS – Cloud Hop*. Be sure to select **2D core**.

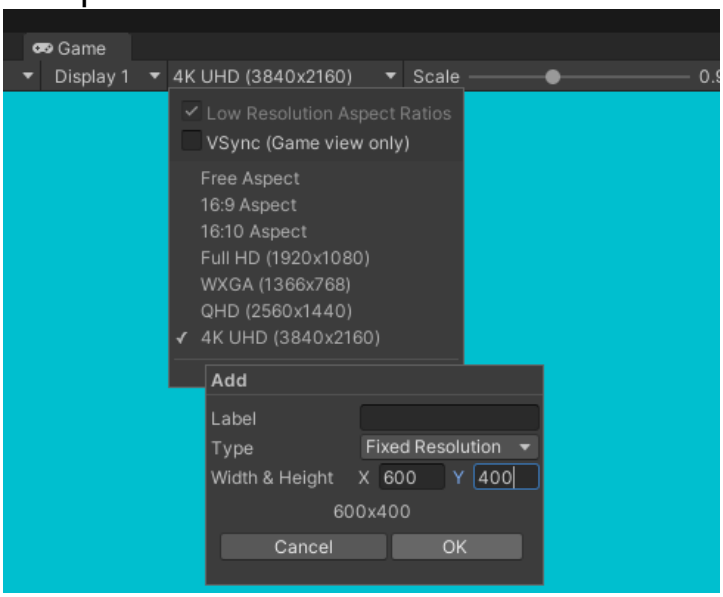
2 We've created a starter pack to give you a head start! To use it, import the **Activity 03 - CloudHop.unitypackage** by going to **Assets > Import Package > Custom Package> All > Import**.



3 To open the starter package, double-click on the **CloudHop** scene. You can find this in the **Project** tab under **Assets > Scenes**.

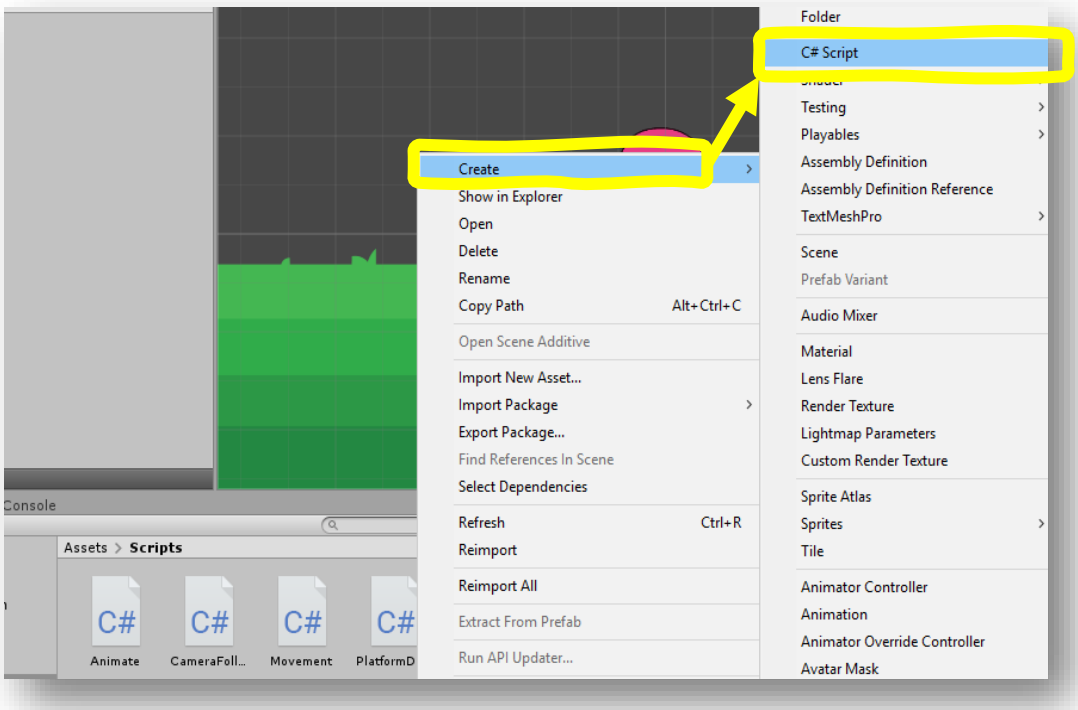


4 Go to the **Game** tab and change the Game Resolution to 600x800. After you change the Game Resolution, you'll want to select the **Scene** tab for the next steps.

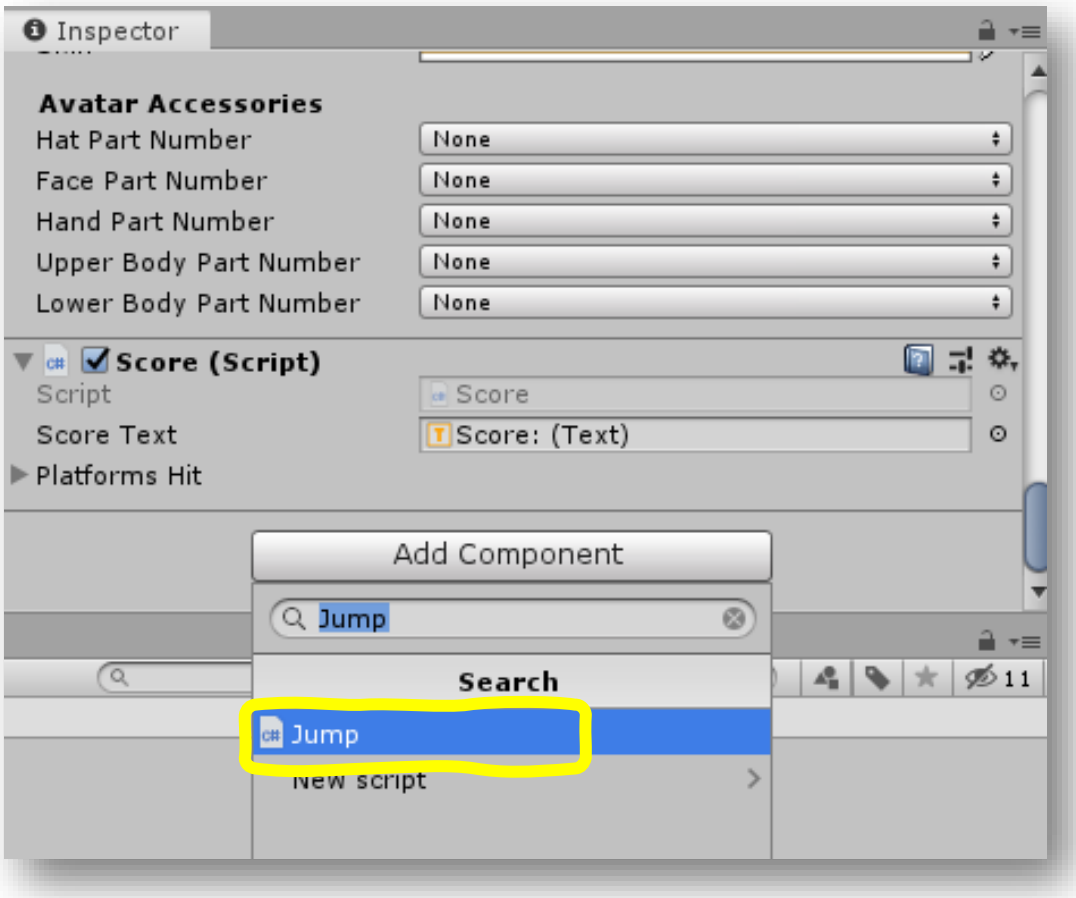


Note: If you do not have the "600x800" option in the drop-down menu, click on the plus sign at the bottom and enter 600 for width and 800 for height.

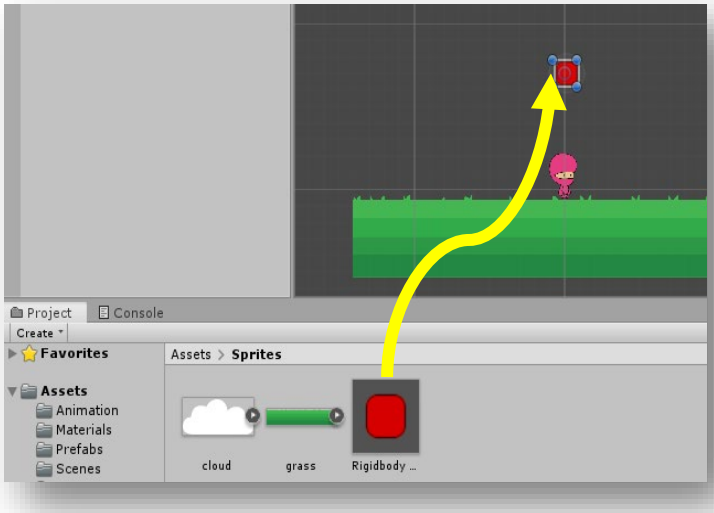
5 Create a new C# Script by right-clicking in your **Scripts** folder, then hover over **Create** and click **C# Script**. Name this script *Jump*.



6 To add the Jump script to the player character, click on **Player** in the Hierarchy. Then, in the Inspector, scroll to the bottom and click **Add Component**. Search and click on **Jump**.



7 Remember that adding a **rigidbody** component means the object will be affected by gravity. Let's see what this looks like. Go to the **Sprites** folder and drag **Rigidbody Test** into the scene.

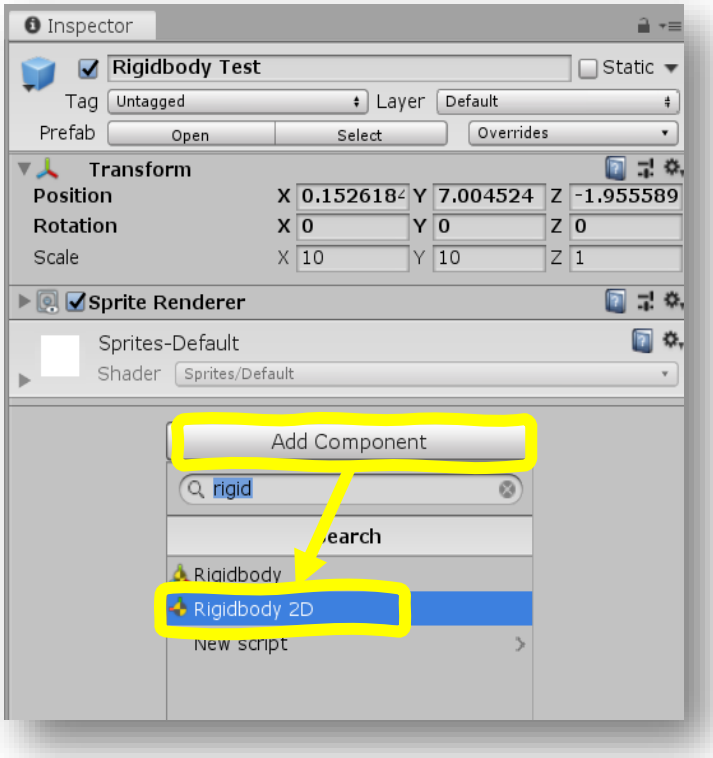


8 Press **Play**. See how the object just stays in place, not doing anything?

Go ahead and exit play mode.

9 Now try adding a **rigidbody** to the sprite to see what changes! To do this, click on the **Rigidbody Test** game object in the **Hierarchy**. In the **Inspector**, scroll to the bottom and click on the **Add Component** button, search for *Rigidbody2D*.

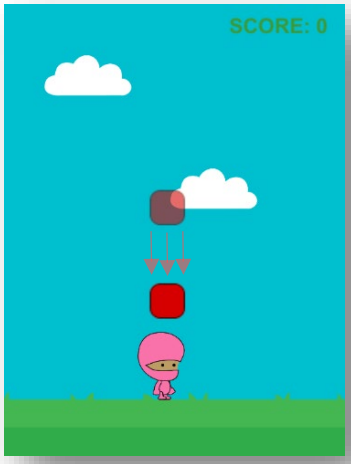
Remember, because this is a 2D game, we use Rigidbody 2D. A 3D game would just use a Rigidbody.



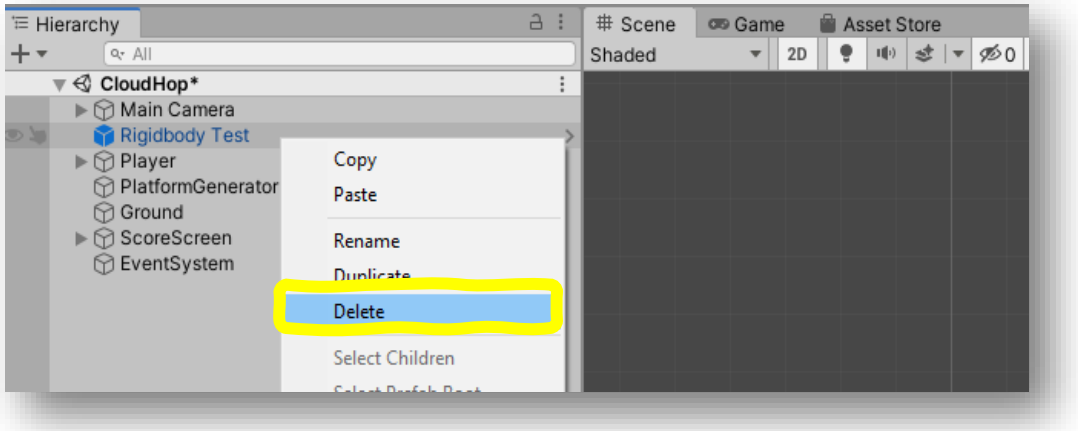
10 **Play** the game to see what happens when an object has a **rigidbody**.

It falls!

Remember, it is affected by physics, including forces like gravity.



11 Delete **Rigibody Test** from the **Hierarchy**.

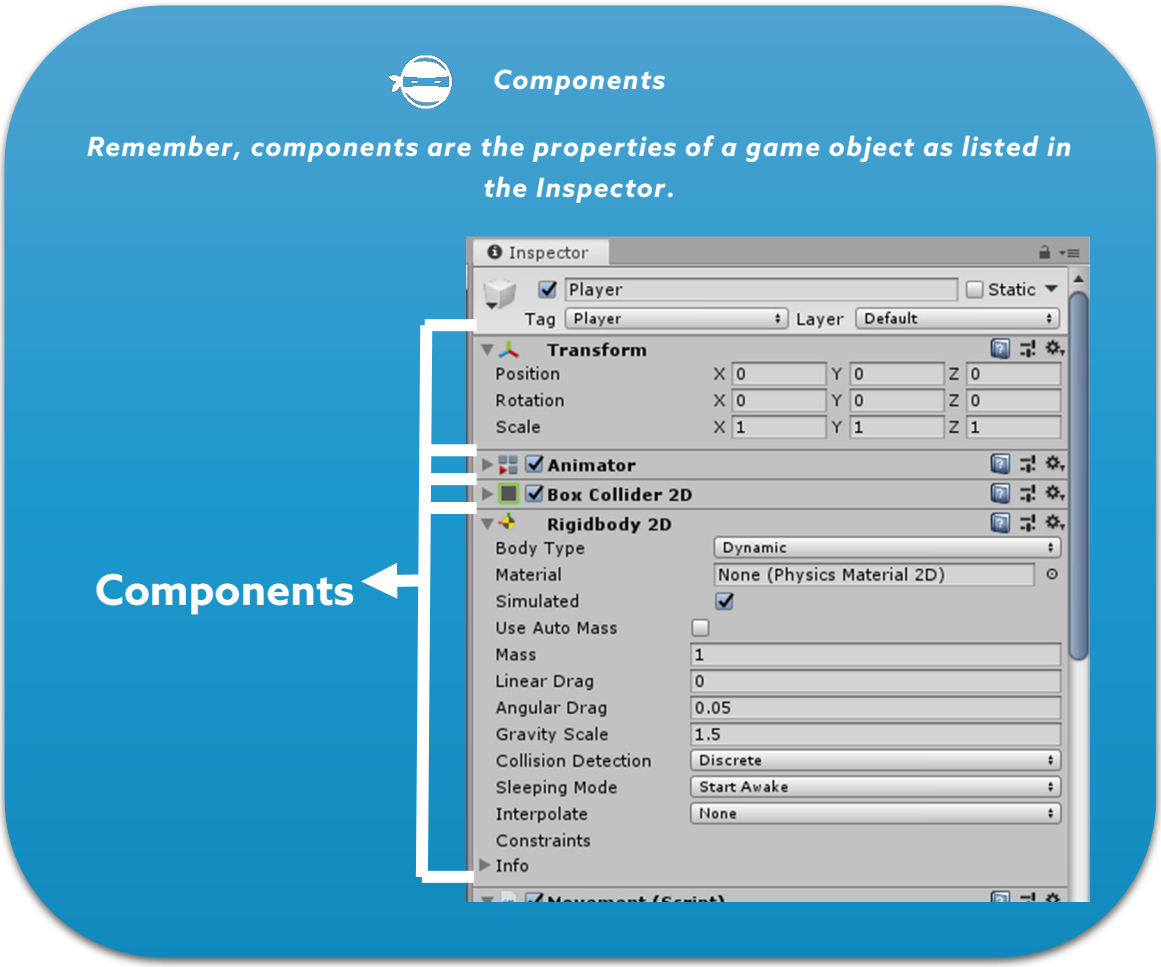


12 Let's use **rigidbodies** in the Cloud Hop game to make the player fall. You'll want to open your **Jump** script.

13 First, declare `private Rigidbody2D rb`. We do this so when we use the name "rb," Unity will know we are referencing a Rigidbody2D.

```
public class Jump : MonoBehaviour
{
    private Rigidbody2D rb;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
```

14 Notice we did not declare `Rigidbody2D rigidbody` as a public variable. Previously, we would use a public variable and then drag the object into the slot in the Inspector. Instead, we will use a new function called `GetComponent<>()`.



We can assign and reference a component in our code using `GetComponent<ComponentName>()`. Get the rigidbody component by adding the following line to your Jump script's `Start()` function:

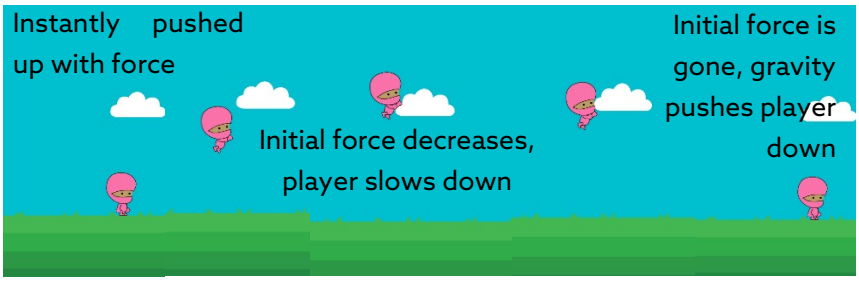
```
public class Jump : MonoBehaviour
{
    private Rigidbody2D rb;

    // Start is called before the first frame update
    Unity Message | 0 references
    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
    }
```

Rigidbodies

Why are rigidbodies particularly useful for setting up a jump? Not only will gravity do the work of pulling our character back to the ground, but it lets us use functions specific only to rigidbodies that will do the work to push our character up in the air.

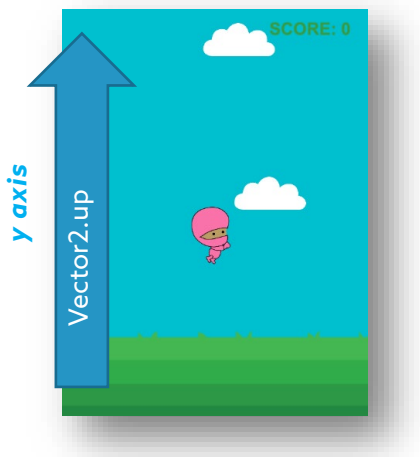
15 We will use `rb.AddForce()`. Adding a force creates movement similar to pushing something, it is pushed away at a set force or speed and then slows down over time.

AddForce takes the parameters: (float x, float y, float z, ForceMode)



ForceMode2D.Impulse tells Unity to apply the force instantly.

Instead of telling Unity the (x, y, z) coordinates of the player, we are going to use *Vector2.up* which is a shorter way of saying (0, 1). In other words, Vector2.up only changes the y direction, which is up.



Input.GetAxisRaw()

Remember, Input.GetAxisRaw()receives keyboard input for the arrow keys or WASD and returns either a 1, 0, or -1.

16 To adjust how high the player jumps, multiply it by a **jumpForce**. In your Jump script, declare `float jumpForce = 15`:

```
private Rigidbody2D rb;
private float jumpForce = 15;

// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
```

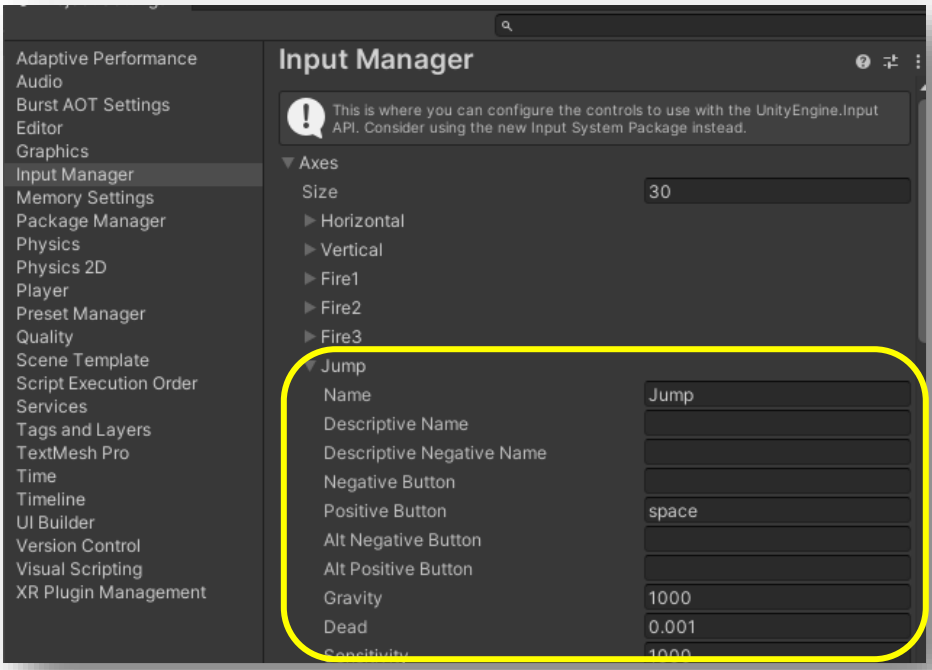
17 Let's see how the `rb.AddForce` function works by adding the following line of code in your `Start()` function. This will push the character up at a force of 15 as soon as the game is started:

```
// Start is called before the first frame update
Unity Message | 0 references
void Start()
{
    rb = GetComponent<Rigidbody2D>();

    rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
```

18 Press **Play** and test it out! In your test, your player should jump, but only at the start.

19 Let's change the code so the player jumps when you press the space bar. We will use an Input function called `Input.GetButtonDown()`.

This returns true if the user is pressing whichever key you include in the parameters. Using `Input.GetButtonDown("Jump")` checks if spacebar was just pressed down.



Note: There are some other types of Input methods, such as GetButtonUp, which returns if the player released a button, or GetButton, which tells you if the player is holding down the button.

20 In Jump script, delete the `rb.AddForce()` function in `Start()`.

```
void Start()
{
    rb = GetComponent<Rigidbody2D>();

    rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
}
```

21 Next, in `Update()`, add a conditional statement. Use `Input.GetButtonDown("Jump")` as the condition and `rigidbody.AddForce()` as the consequence like this:

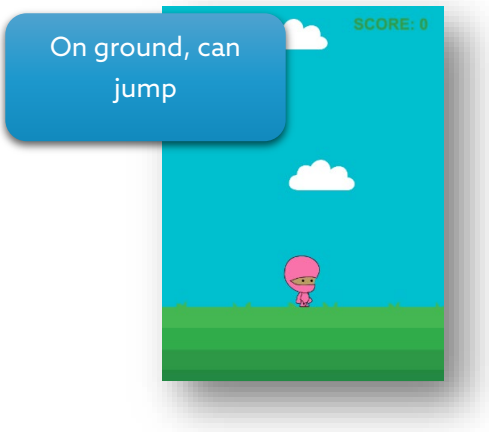
```
void Update()
{
    if (Input.GetButtonDown("Jump"))
    {
        rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
    }
}
```

22 Press **play** to test out your code! The player should now jump when you press the spacebar.

23 There is a problem with the jump; did you find it? Our code says if the spacebar is pressed, add an upward force. What happens if we just keep pressing space? Try repeatedly pressing spacebar. The player just keeps jumping mid-air, which makes the game too easy! Let's fix this.

24 First, we need a way to check if the player is on a surface (either the grass or cloud platforms).

If they are not on a surface, they are not allowed to jump again.

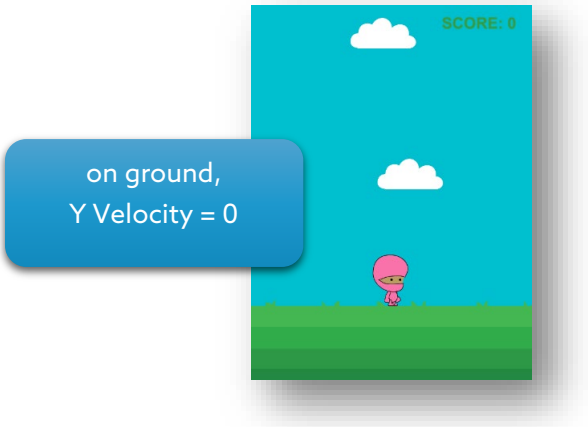


25

To check if the player is in the air, we will use `rb.velocity`.

> ***Velocity***
>
> ***The velocity of the character is the rate at which it changes position.***

When the player is staying still, their velocity is 0. In other words,
if player velocity is 0,
then the player is standing still on a surface.



26

Jumping only deals with the vertical, or y axis. Therefore, we can use `rb.velocity.y` to check if the player is jumping or not.

Let's add an if/else statement that checks if the player is jumping or not to our if statement. Add this line of code in your Jump's `Update()` function:

```
void Update()
{
    if (Input.GetButtonDown("Jump") && rb.velocity.y == 0)
    {
        rb.AddForce(Vector2.up * jumpForce, ForceMode2D.Impulse);
    }
}
```

27

Great job, ninja! Play your game to test it out. You should be able to jump from cloud to cloud. But the player should only be able to jump from a surface and not while in mid-air.