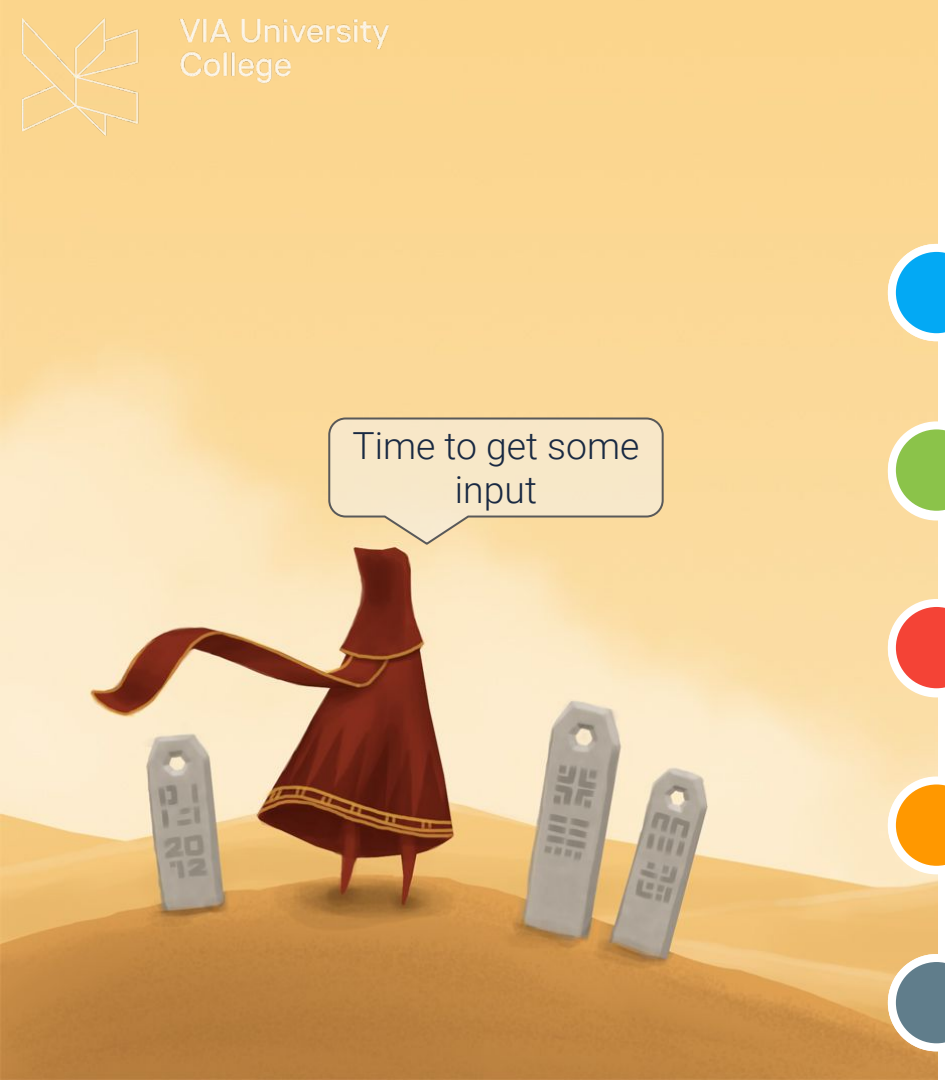
VIA University College




# Vectors & Input

Positions, directions and managing input in Unity

## Basic Vector Math

Adding, subtracting, scaling and normalizing vectors

## Working with Vectors in Unity

Methods for doing common vector operations

## Basic Input

Simple input with keyboard and mouse

## The Input Manager

Learn about Unity's Input Manager

## Exercises

Whack-a-mole and Type Attack!

# Last Week

Event functions?

Composition over inheritance?

Accessing variables?

Accessing components?

Manipulating the transform?

Time.deltaTime?

Activating and instantiating GameObjects?

Enabling/disabling components?

# Basic Vector Math



Unity uses space geometry

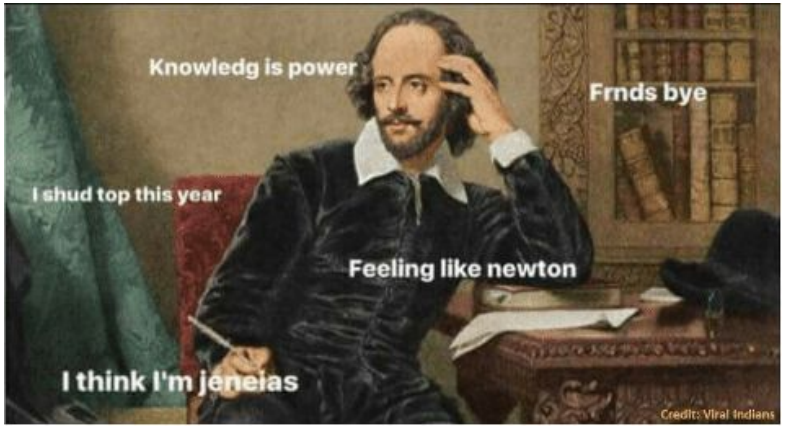
Almost everything has a position in 3D space

We need points and vectors

**Point**: A position in space

**Vector**: A direction in space

Let's start with 2D

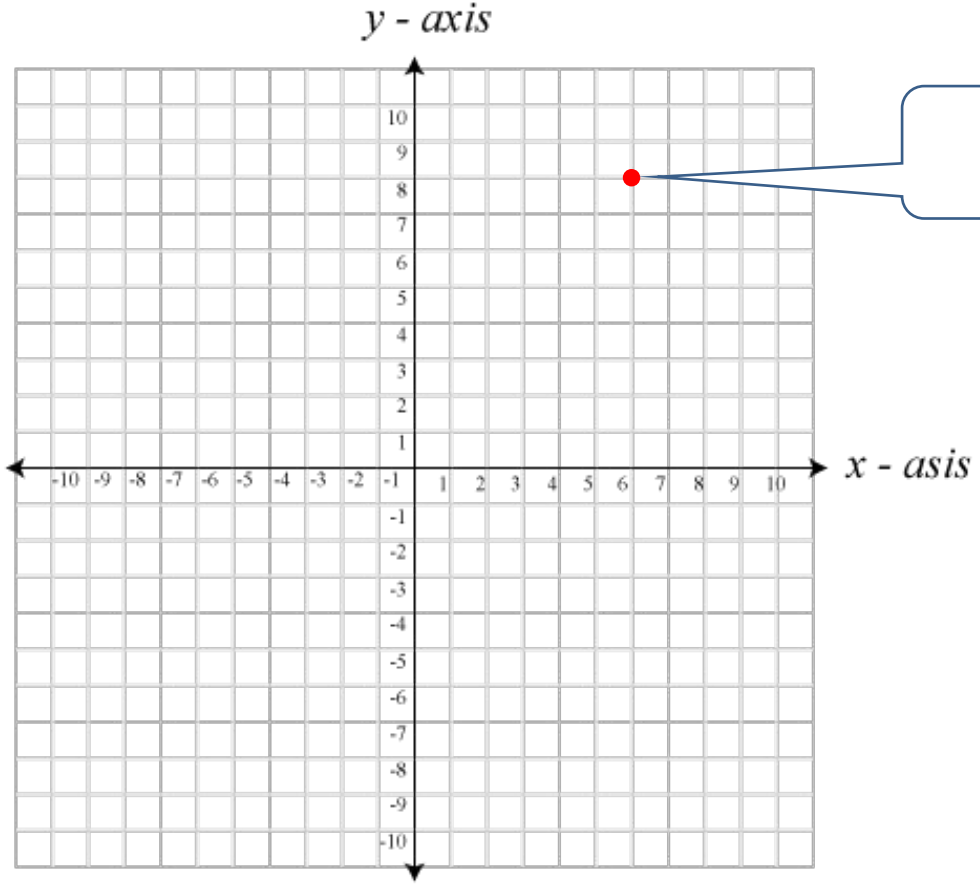



When you do Vector Math in Unity

Vector3 struct

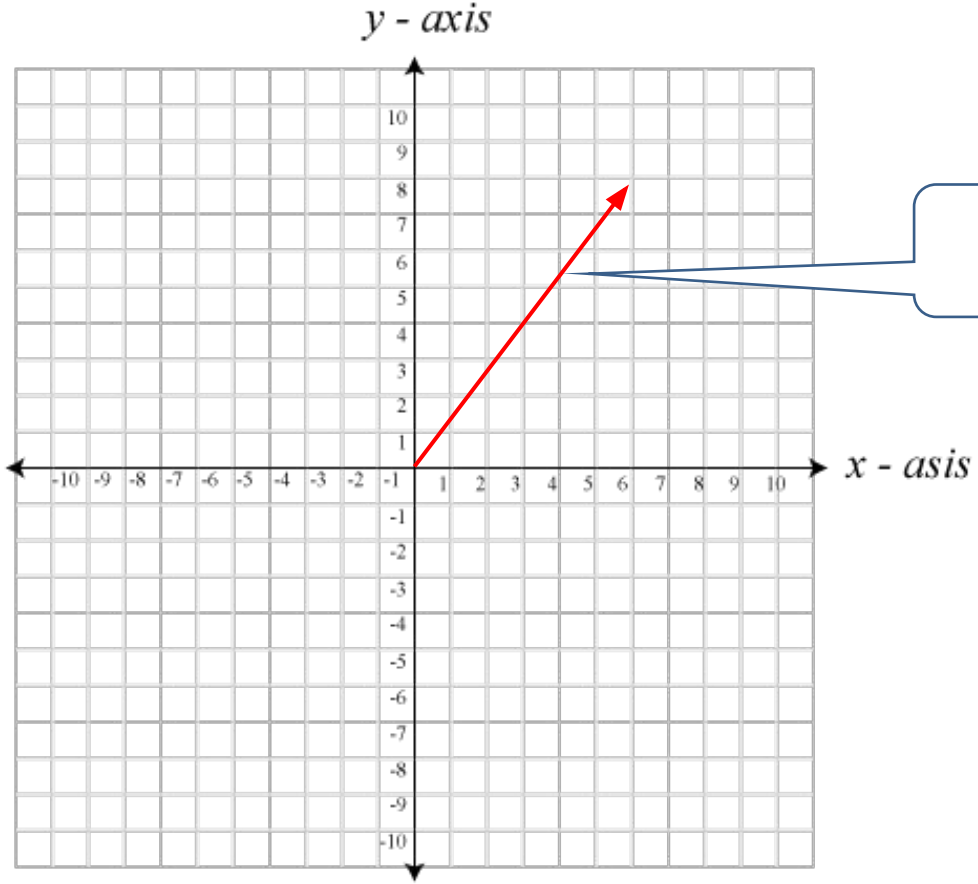
# Basic Vector Math





# Basic Vector Math

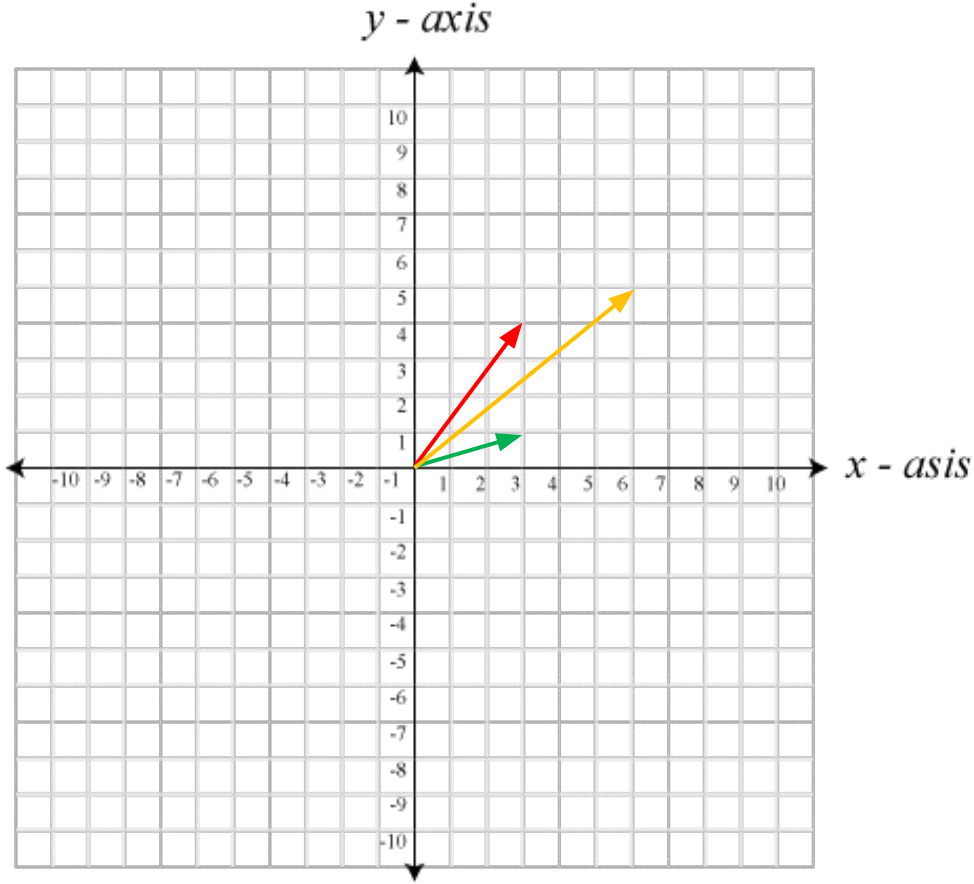





Vector: (6, 8)

We can say:
A point is a position
A vector points to a position

# Vector Addition






Adding vectors

$a = (3, 4)$

$b = (3, 1)$

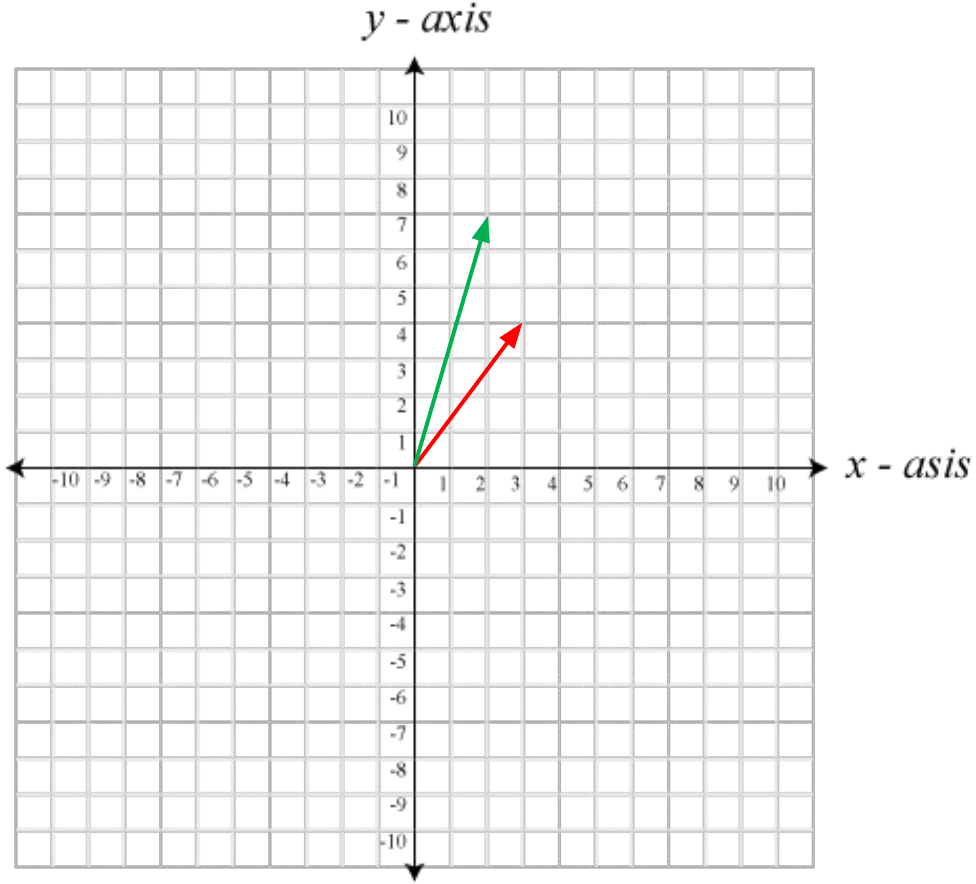
$a + b = v$

$(3, 4) + (3, 1)$

$v = (6, 5)$

# Vector Subtraction






Subtracting vectors

$a = (3, 4)$

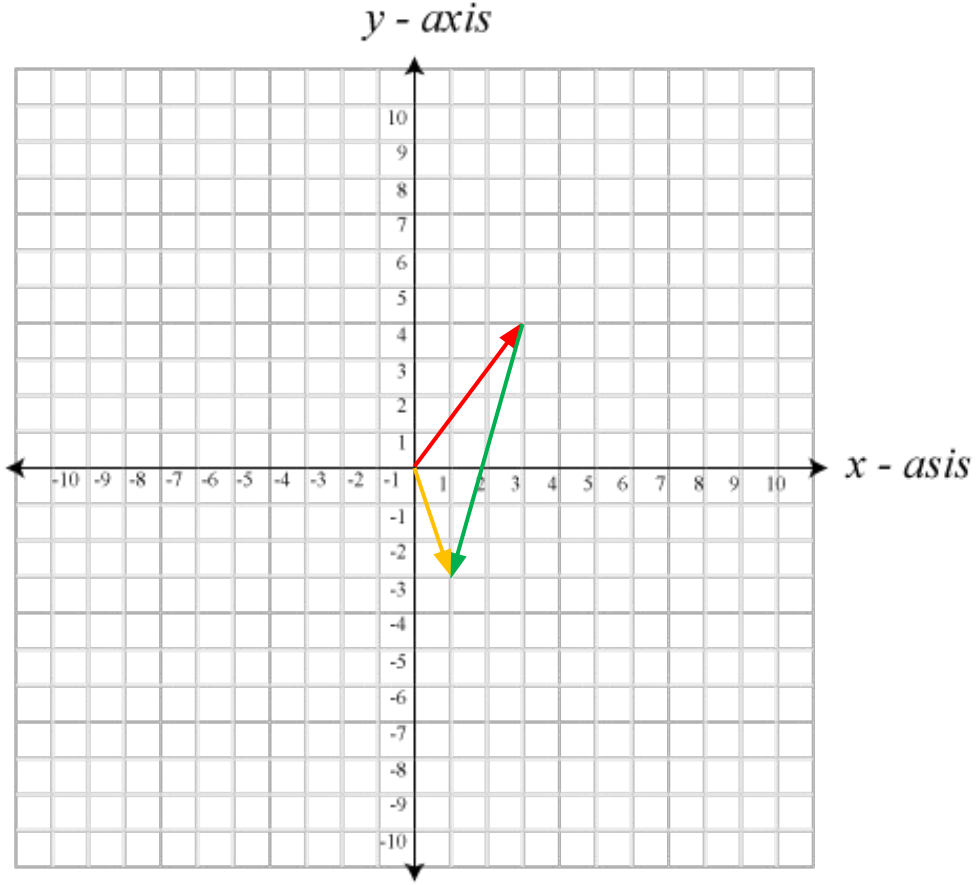
$b = (2, 7)$

$a - b = v$

$(3, 4) - (2, 7)$

$v = (1, -3)$

# Vector Subtraction






Subtracting vectors

$a$ = (3, 4)

$b$ = (2, 7)

$a - b = v$

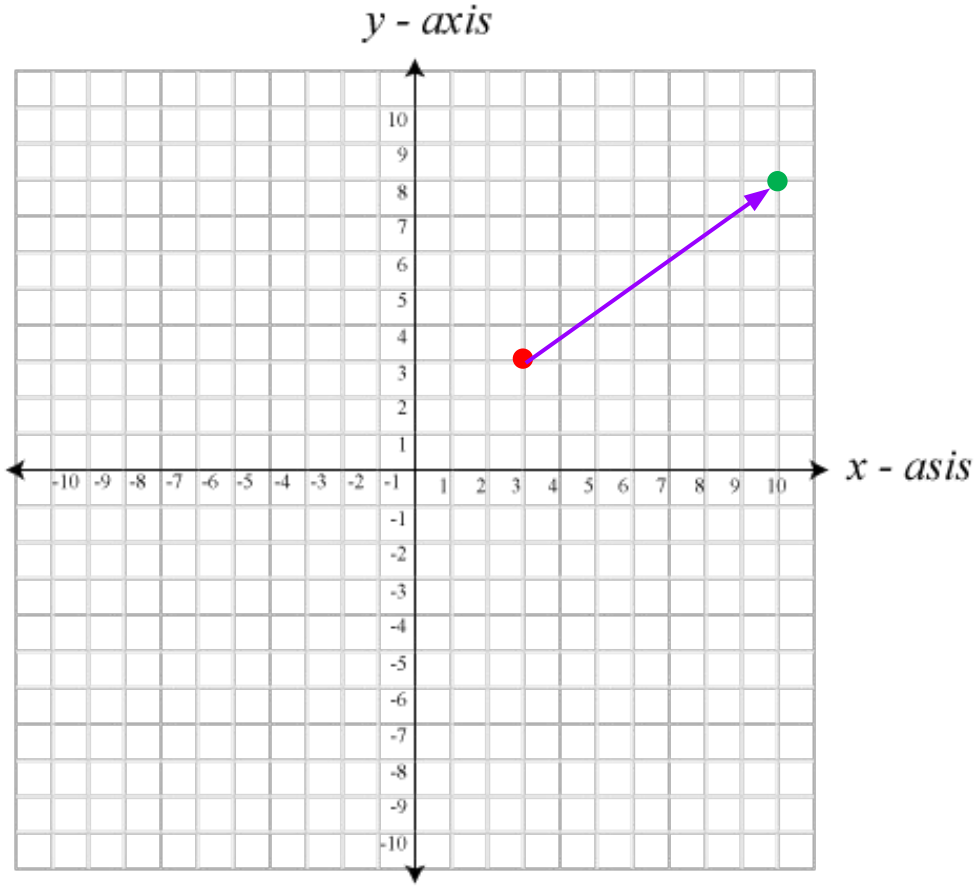
(3, 4) -
(2, 7)

$v$ = (1, -3)

$a$ + reverse of $b$ -> (-2, -7)

# Direction from Point A to Point B



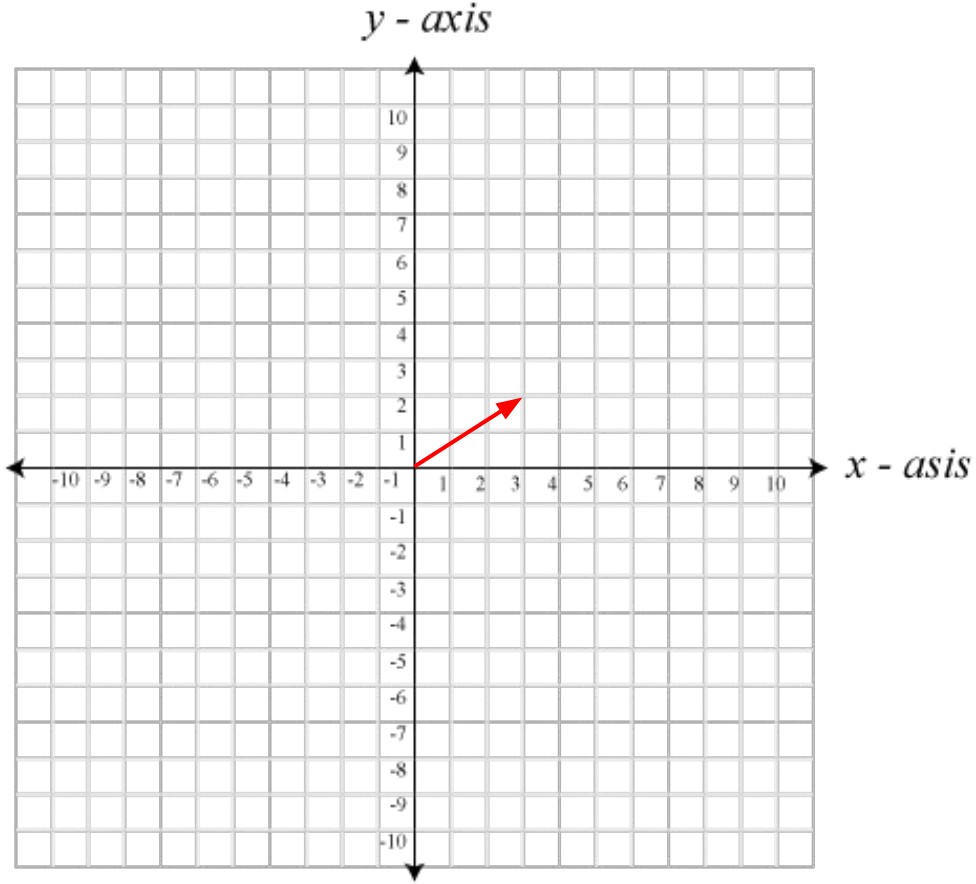



- We have two points, a and b
- How do I figure out the direction from a to b?

- “to minus from”
- The ‘to’ point is b, the ‘from’ point is a.
- I.e.: b – a = dir
- (10, 8) – (3, 3) = (7, 5)

# Vector Scaling






Scaling vectors

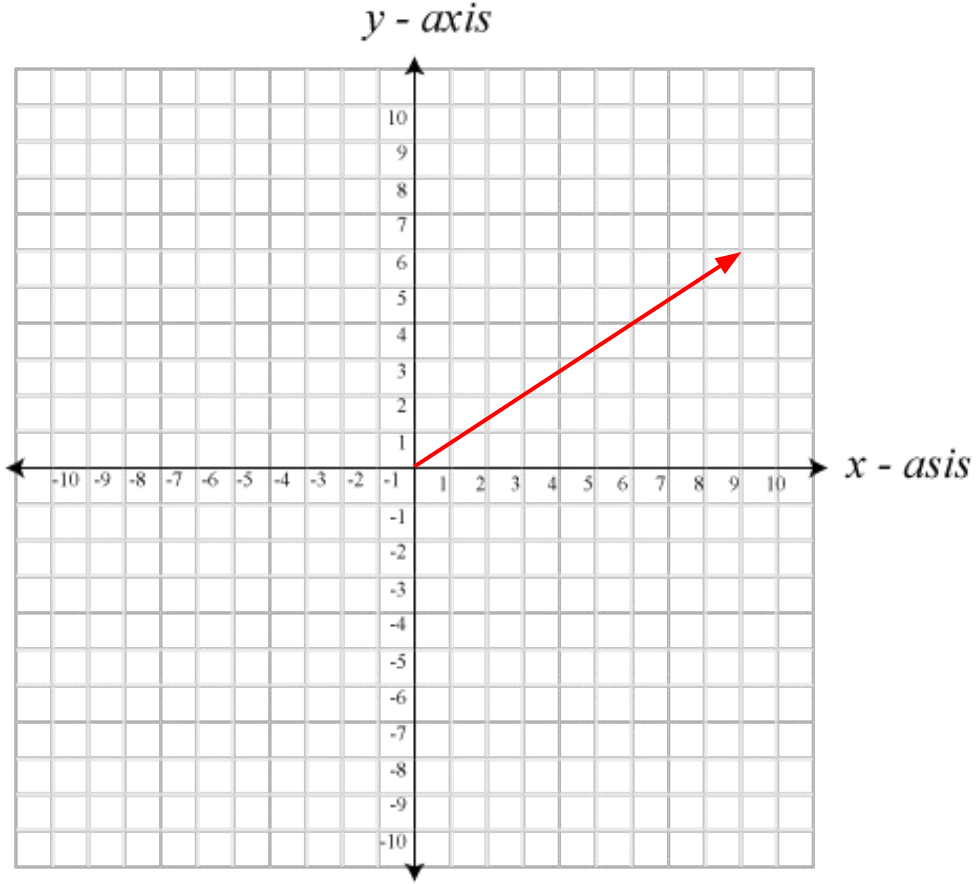
$a = (3, 2)$

$a * 3$

$a * 3 = (3, 2) * 3 = (9, 6)$

# Vector Scaling






Scaling vectors

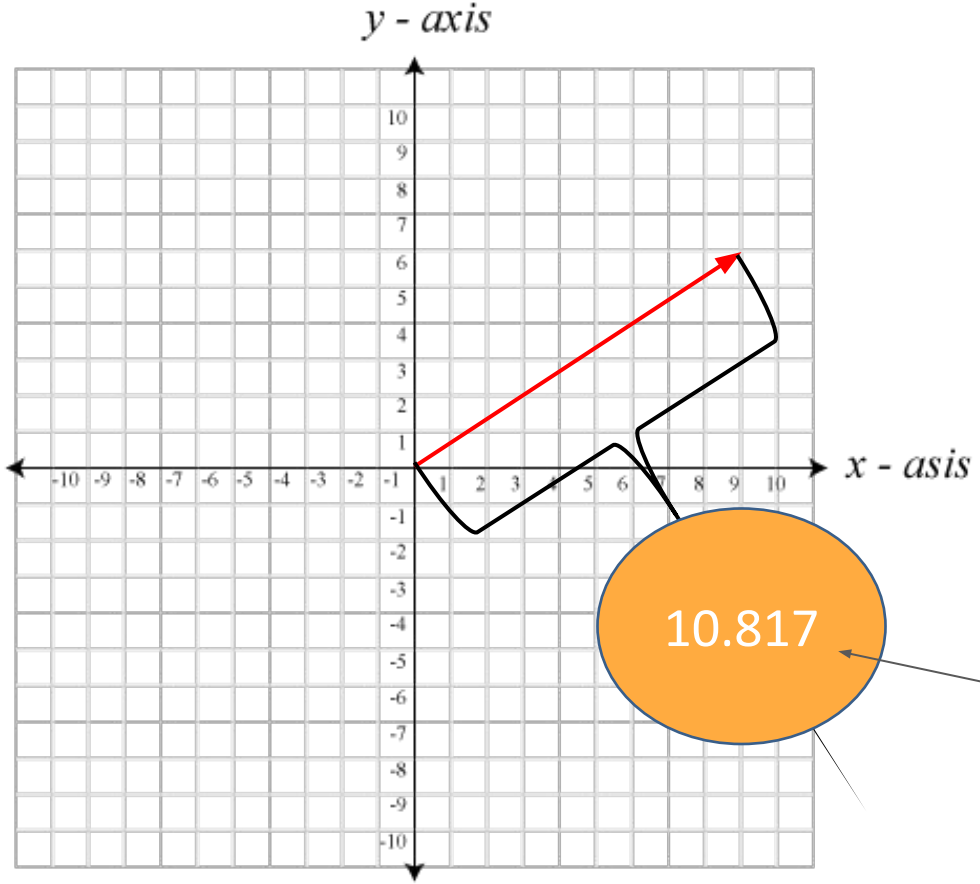
$a = (3, 2)$

$a * 3$

$a * 3 = (3, 2) * 3 = (9, 6)$

# Vector Normalization

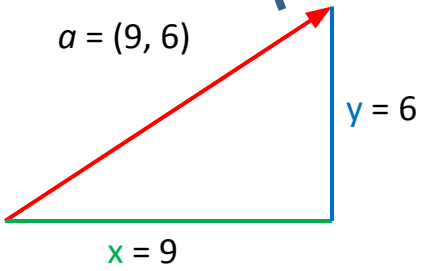





- A *unit vector* is a vector of length 1
- Our vector has length 10.817
- Directions usually needs unit vectors, how do we change a given direction (vector) to have length 1?
- -> normalization, pythagoras

# Vector Normalization






Length of *a* is given by:

$$\sqrt{(x^2 + y^2)} = c$$

Length of *a* = $\sqrt{(9^2 + 6^2)} = 10.867$

Scale *a* by its inverse length = *a* * 1/10.867

norm(*a*) = *a* * 1/10.867

= (9, 6) / 10.687

= (0.84, 0.56)

Result is a vector of length 1.
With the same *direction* as *a*

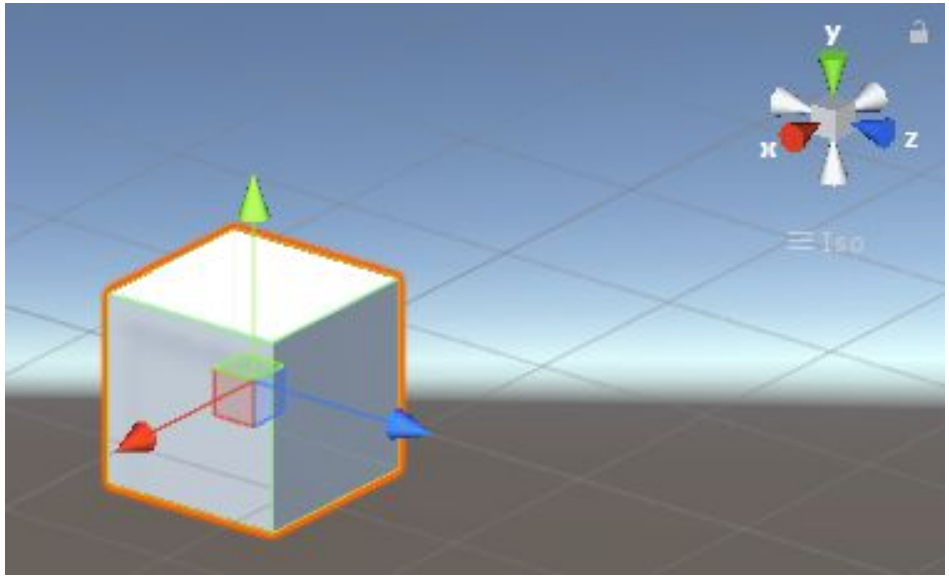
# There’s More Important Vector Math!



# Unity Vectors



- Unity often uses 3d vectors
- The same math applies

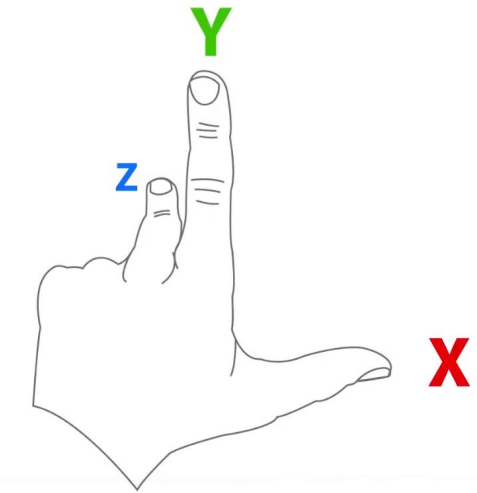


# Unity Vectors



In unity vectors and points are the same (we have no Point class):

```
Vector3 a = new Vector3 (3f, 6f, -1f);
```




The left hand rule

# Unity Vectors



- Whether it's interpreted as a point or vector depends on its use:

```
Vector3 vec = new Vector3 (3f, 6f, -1f);

transform.position = vec;

rigidBody.velocity = vec;

transform.forward = vec;
```

If we set a position, it's a point:

If we set a velocity (speed + direction), it's a vector

If we set a direction, it's a vector

# Unity Vectors

Working with Vectors in Unity

Notice Vector**3**

```
Vector3 vec = new Vector3 (3f, 6f, -1f);
```

Scaling a vector

```
vec = vec * 3f;
```

Length of a vector

```
float length = vec.magnitude;
```

Adding vectors

```
vec = vec + new Vector3 (1f, 3, 2f);
```

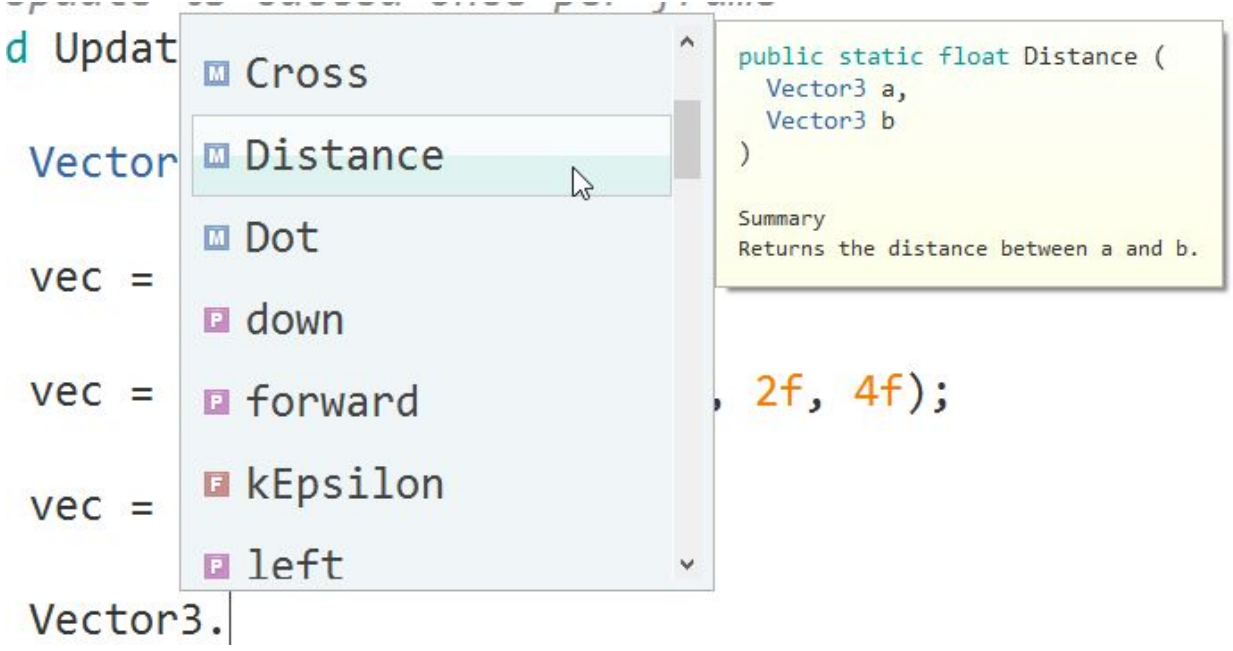
```
vec = vec.normalized;
```

Normalizing a vector. You can also you .Normalize()

# Unity Vectors

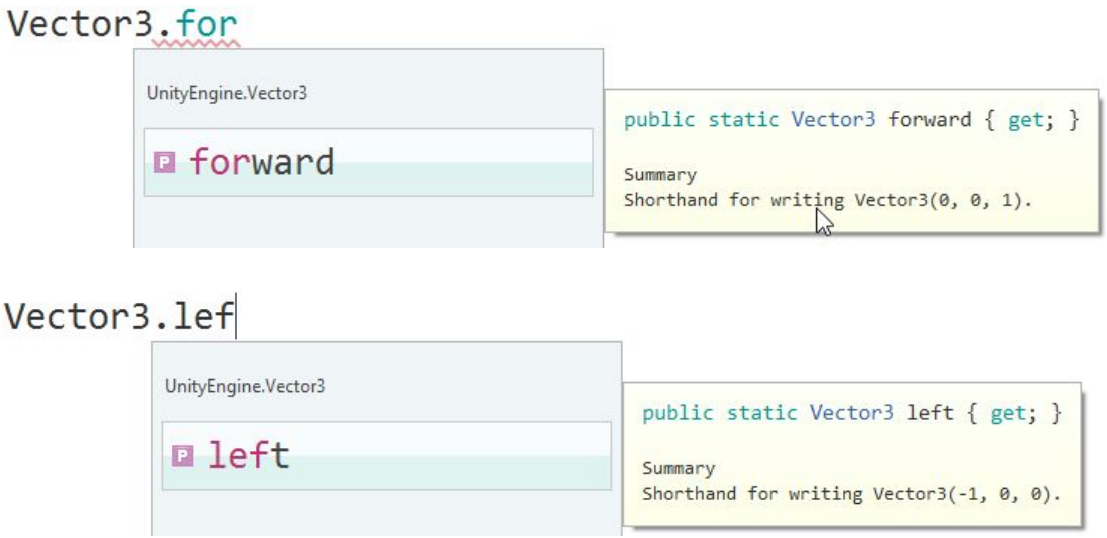


Your usual auto-complete



# Unity Vectors

Working with Vectors in Unity



# Unity Vectors

Working with Vectors in Unity

```
Vector3 forward = transform.for
```

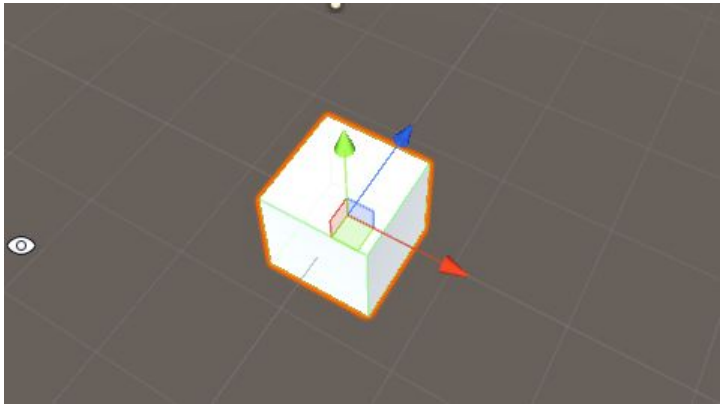
UnityEngine.Transform

forward

UnityEngine.Component

```
public Vector3 forward { get; set; }

Summary
The blue axis of the transform in world space.
```



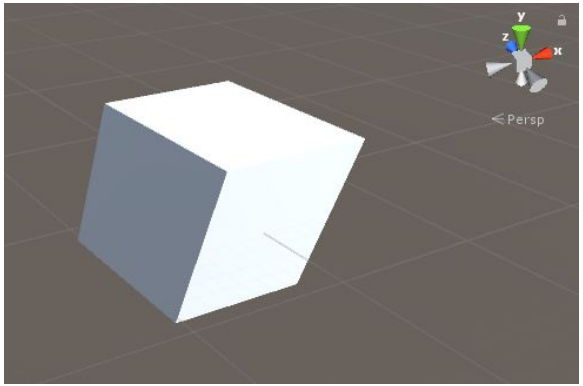
Forward

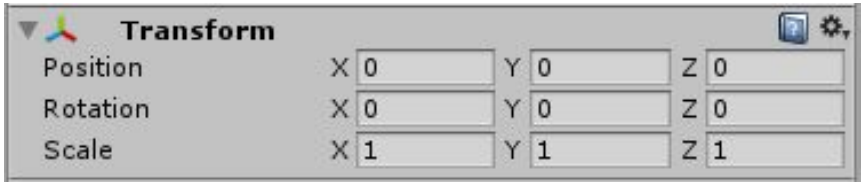
Right

Up

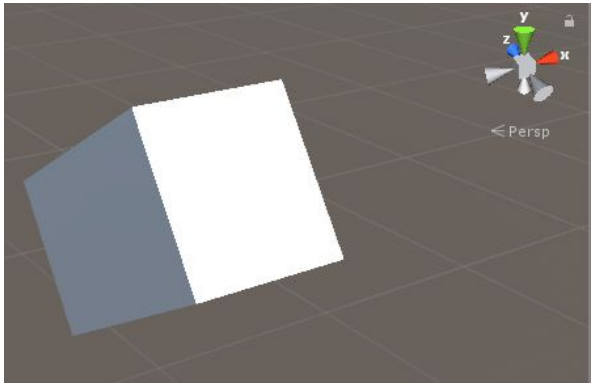
# Unity Rotations

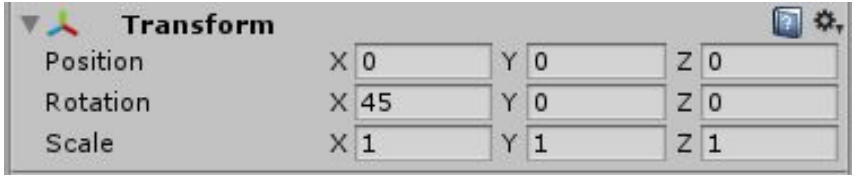





```
transform.eulerAngles = new Vector3 (45f, 0f, 0f);
```

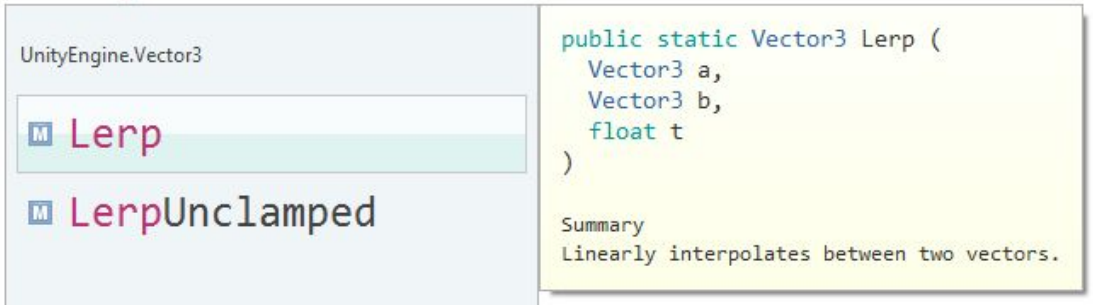


# Linear Interpolation



Vector3.Lerp



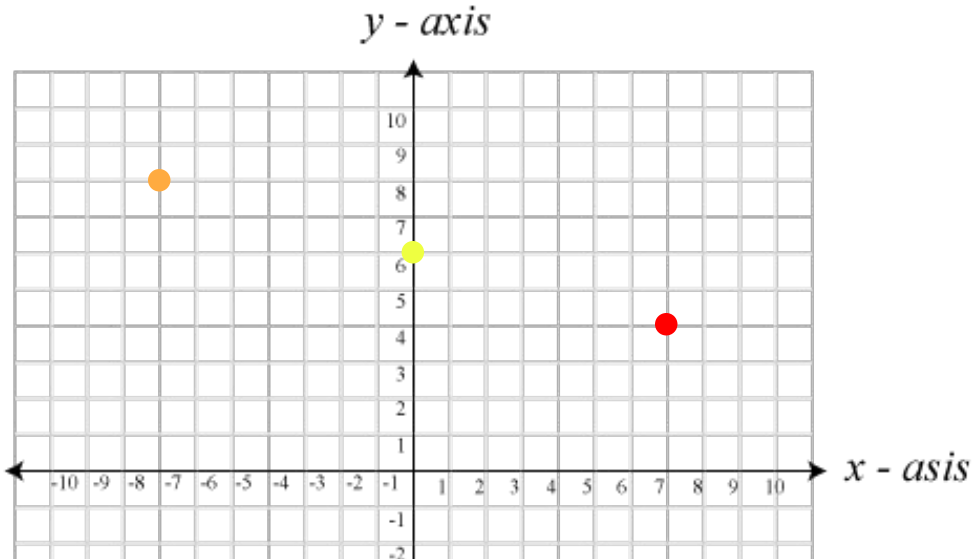

Will return a vector *t* percent between *a* and *b*

# Linear Interpolation



```
Vector2 lerp = Vector2.Lerp(new Vector2(-7f, 8f), new Vector2(7f, 4f), 0.5f);
```

The result is a vector, *lerp* = (0f, 6f), i.e. half way between the two input vectors

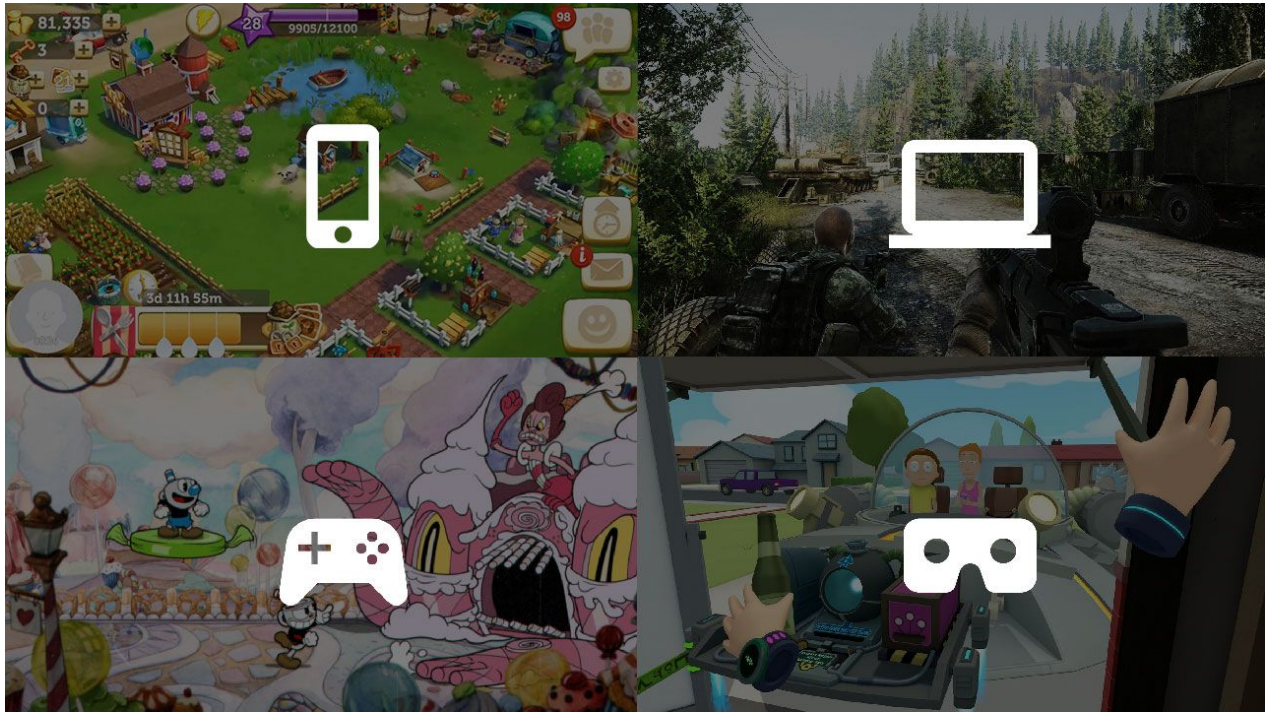


# Multiple Input Types



In most games, a user interacts with the game somehow:

- Keyboard
- Mouse
- Controller
- Something else..



# Input

Basic Input

Always check for input in Update()

VERY IMPORTANT

We get input from Unity's 'Input' class, using static methods

Input events are refreshed after each Update-call

Input are usually one of three events:

- Key is pressed down
- Key is held down
- Key is released

```
if (Input.GetKeyDown (KeyCode.Space)) {
    Debug.Log ("Space is pressed");
}
if (Input.GetKey (KeyCode.Space)) {
    Debug.Log ("Space is held down");
}
if (Input.GetKeyUp (KeyCode.Space)) {
    Debug.Log ("Space is released");
}
```

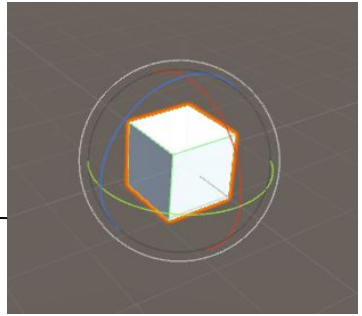
Conventional Game Input

# Simple Controls



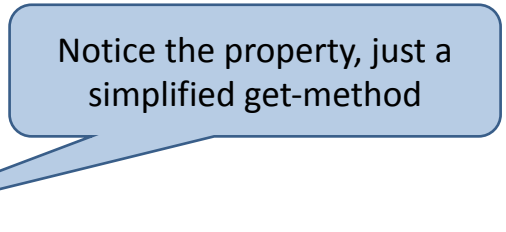
```
void Update () {
    if (Input.GetKey (KeyCode.W)) {
        transform.Translate (Vector3.forward * 5f * Time.deltaTime);
    }
    if (Input.GetKey (KeyCode.S)) {
        transform.Translate ( -Vector3.forward * 5f * Time.deltaTime);
    }
    if (Input.GetKey (KeyCode.A)) {
        transform.Rotate (Vector3.up, -50f * Time.deltaTime);
    }
    if (Input.GetKey (KeyCode.D)) {
        transform.Rotate (Vector3.up, 50f * Time.deltaTime);
    }
}
```

Rotate around an axis



# Listening for Any Input






```
void Update () {
    string input = Input.inputString;
    if(input != null && !"".Equals(input)) Debug.Log (input);
}
```

The call ‘Input.inputString’ returns whichever key was pressed this frame.

# Mouse Input



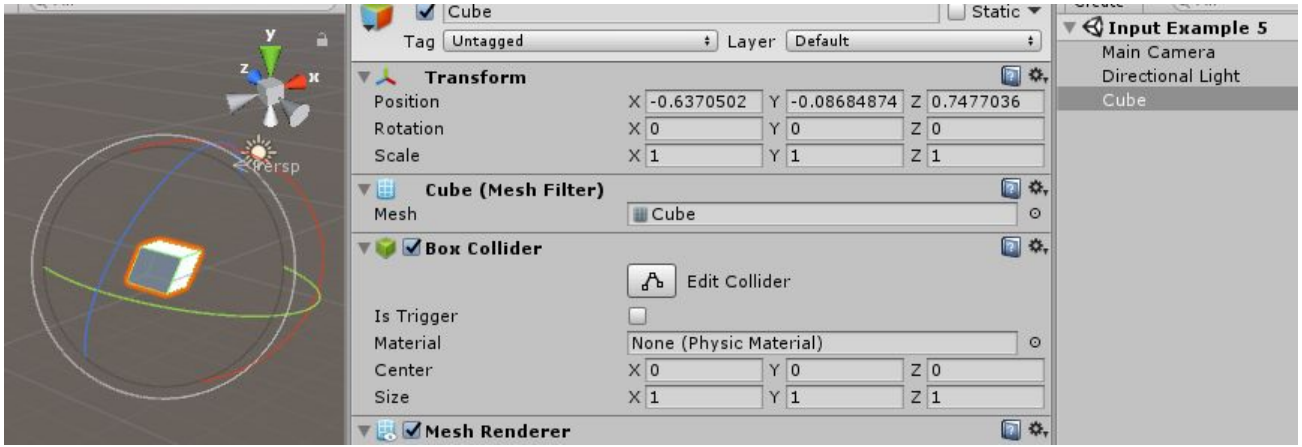
Sometimes we want to click objects

We can use MonoBehaviour event functions for this

```
void OnMouseDown() {
    Debug.Log ("Clicked");
}

void OnMouseOver() {
    Debug.Log ("Hover");
}
```

The object we're clicking must have a collider



# Mouse Input



Detecting mouse events

- OnMouseDown, OnMouseEnter, OnMouseOver, OnMouseExit
- Uses Colliders to register the hit

Detecting mouse position

- Input.mousePosition
- Vector3 with z as 0.0. The values are in pixels. (0,0) is the bottom left of your game window.
- Editor coordinates != Build coordinates

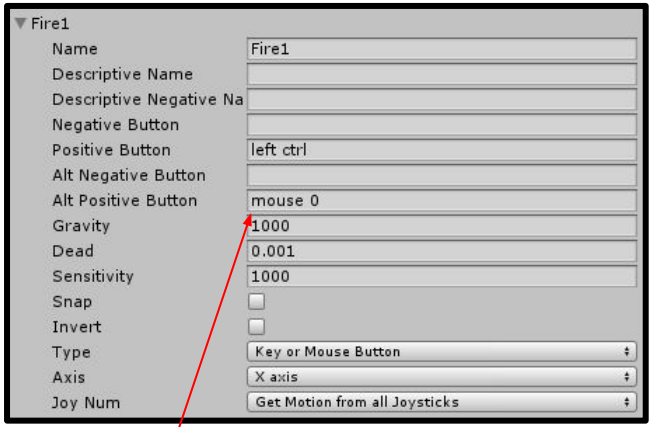
Mouse input as axis:

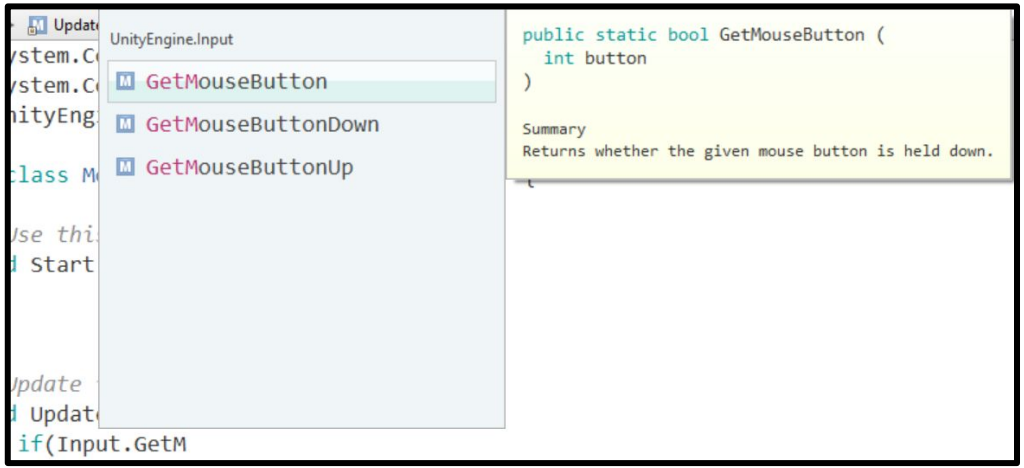
- Input.GetAxis(“Mouse X”)
- Input.GetAxis(“Mouse Y”)

OnMouseDown

# Mouse Input

Basic Input




```
if (Input.GetMouseButtonDown (0)) {
    Debug.Log ("Say hello");
}
if (Input.GetButtonDown ("Fire1")) {
    Debug.Log ("Other hello");
}
```

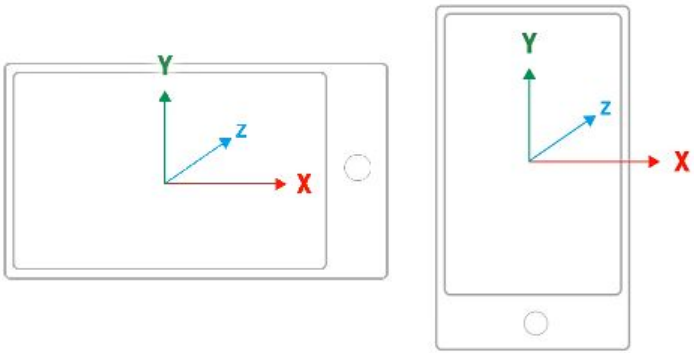
# Mobile Input



Possibilities with a mobile device: Multi-touch, accelerometer, GPS, camera, microphone, compas, virtual keyboard, etc…

**ALL MOUSE FUNCTIONALITY WORKS FOR TOUCH! :)**

But if you want multi-touch functionality you have to use Input.Touch(int), that stores all touches in an array.




Accelerometer:

transform.Translate(**Input.acceleration.x**, 0, **-Input.acceleration.z**);

# GetKey vs GetButton



- GetKey uses hardcoded key codes `Input.GetKey (KeyCode.W)`
- GetButton uses the Input Manager to handle inputs

We don't want to hardcode every button press through scripts. That's why it's good practice to use GetButton instead of GetKey.

```
void Update () {
    if (Input.GetKeyDown("space")) {
        Debug.Log("Pressing space!");
    }

    if (Input.GetButtonDown("Jump")) {
        Debug.Log("Also pressing space!");
    }
}
```

Better!!

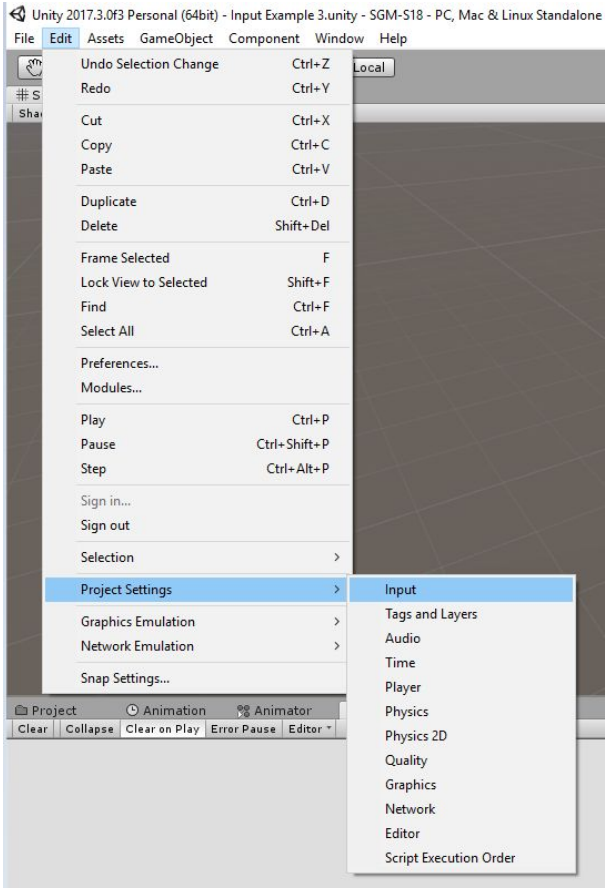
The input manager has two types:

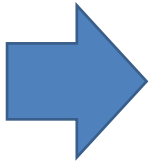
- Buttons -> [true, false]
- Axes -> [-1, 1]

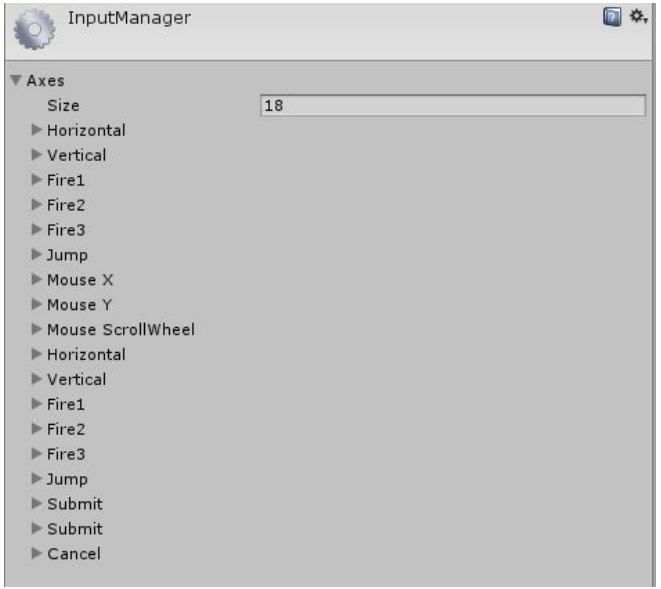
GetButton and GetKey

# Input Manager

The Input Manager





# Input Manager



- Movement example from before
- Now “Vertical” references the “Vertical” axis from the Input Manager

```
void Update () {
    float vertical = Input.GetAxis ("Vertical");
    if (vertical != 0f ) {
        transform.Translate (vertical * Vector3.forward * 5f * Time.deltaTime);
    }

    float horizontal = Input.GetAxis ("Horizontal");
    if (horizontal != 0f) {
        transform.Rotate (Vector3.up, 50f * Time.deltaTime * horizontal);
    }

    if (Input.GetButtonDown ("Jump")) { // teleport
        transform.Translate (Vector3.forward * 3f);
    }
}
```

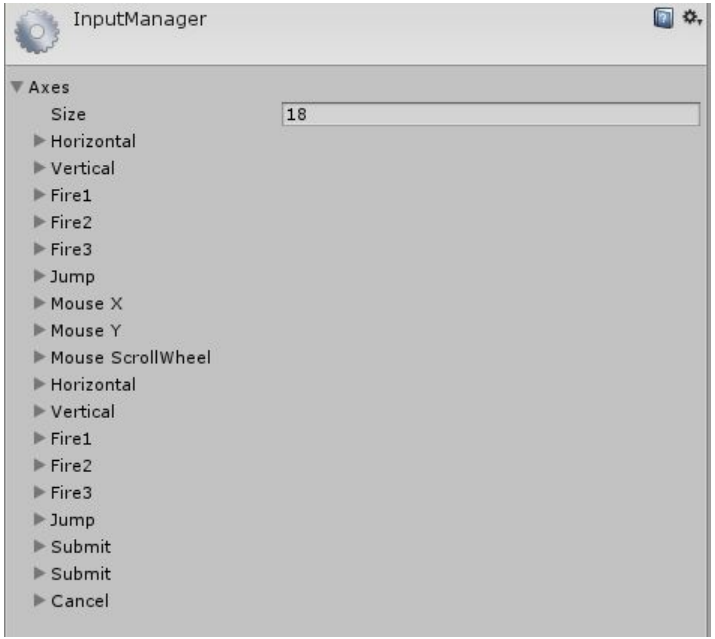
GetAxis returns a float [-1, 1]

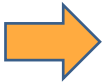
What’s going on here?

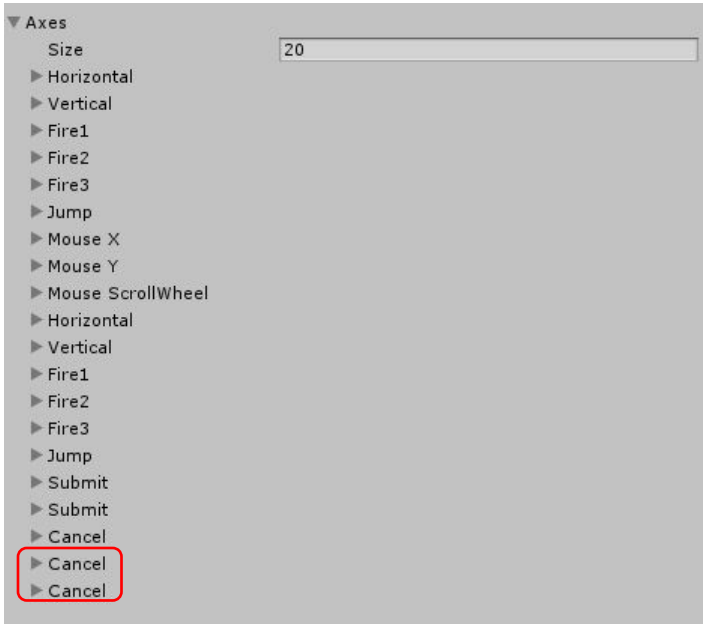
# Input Manager



- To modify the InputManager, change the ‘18’ to a larger size, this will create more buttons for you to edit:

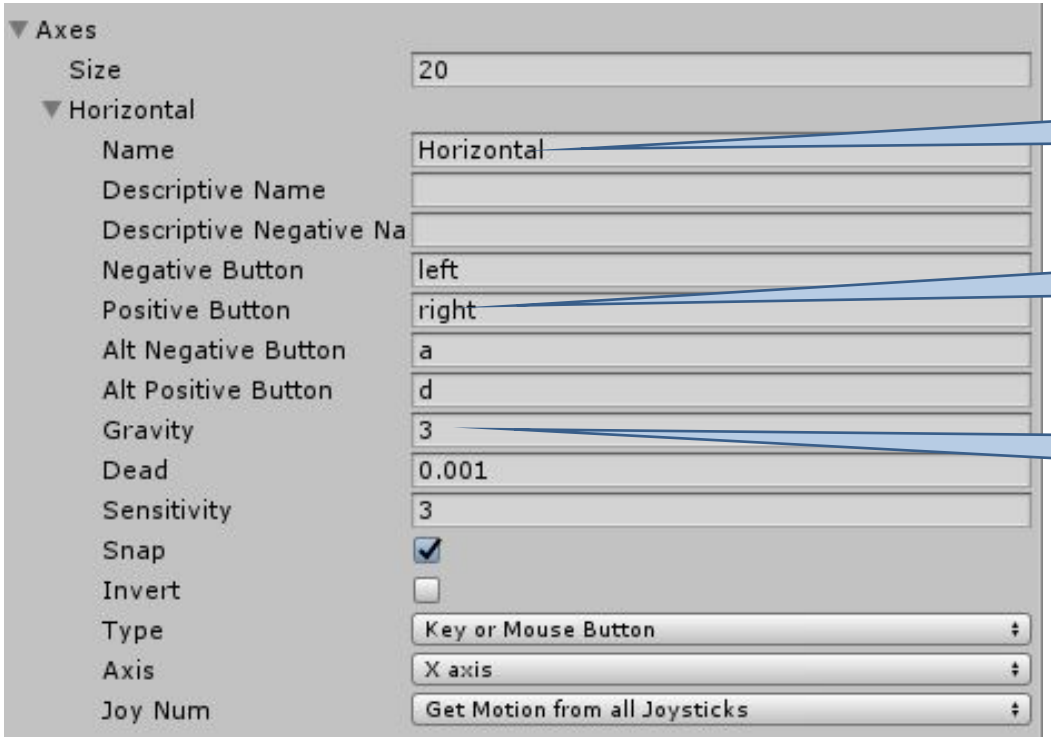




# Input Manager, Axis Example






The reference name

Positive (and negative) buttons

When released, how fast do we go back to 0?

```
float horizontal = Input.GetAxis ("Horizontal");
float horizontal = Input.GetAxisRaw ("Horizontal");
```

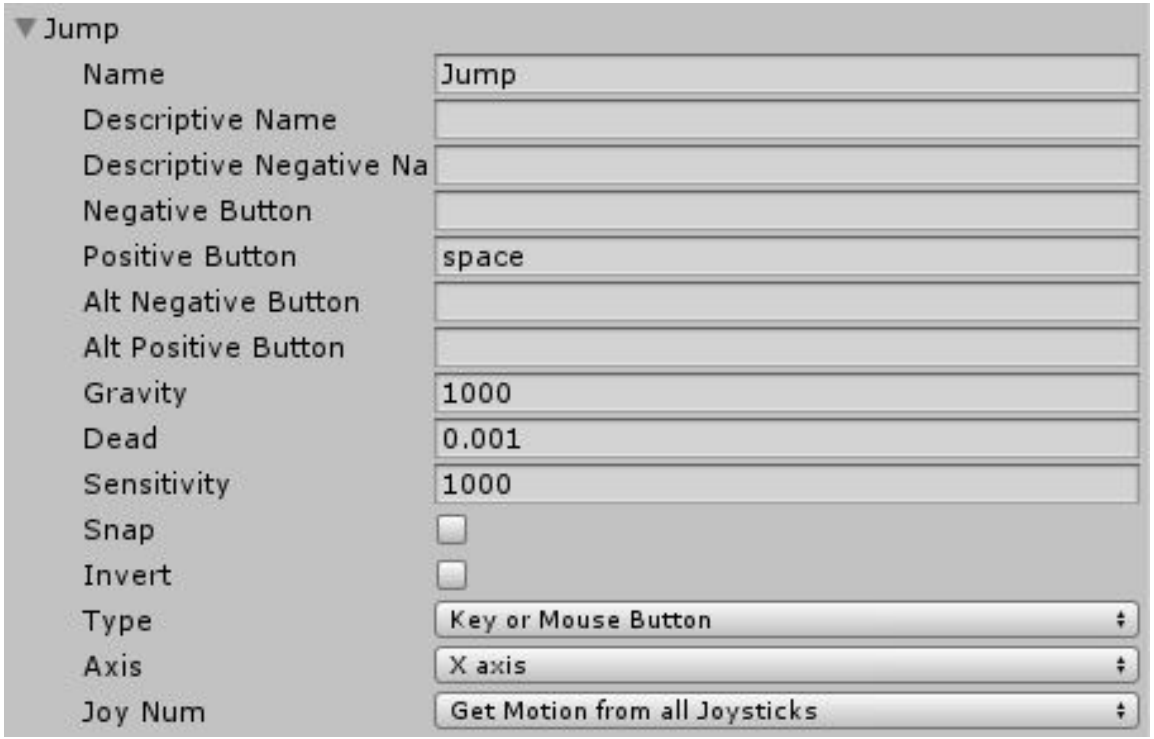
[-1, 1]

-1, 0, 1

GetAxis

# Input Manager, Button Example



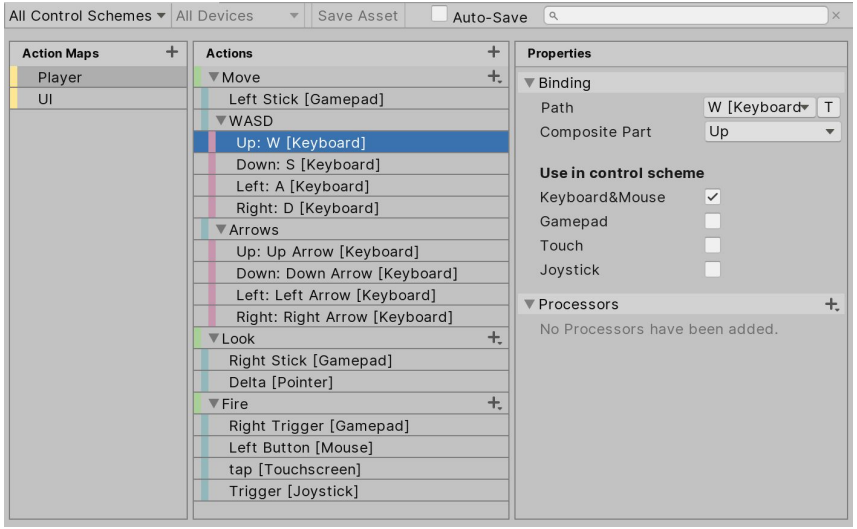



GetButton(“Jump”)
GetButtonDown(“Jump”)
GetButtonUp(“Jump”)

```
Input.GetButtonDown ("Jump")
```

# The New Input System

## The Input Manager

# Exercises



Exercises are available on itslearning.

Remember to import the example unitypackage.


Today is also a good day to start on your course project!

[Input Scripting Reference]