LECTURE 10

Blueprints in Action 1

LECTURE GOALS AND OUTCOMES

Goals

The goals of this lecture are to

- Present transforms
- Explain world coordinates
- Demonstrate the difference between relative and world-based transforms
- Show how to use vector operations
- Show some vector functions

Outcomes

By the end of this lecture you will be able to

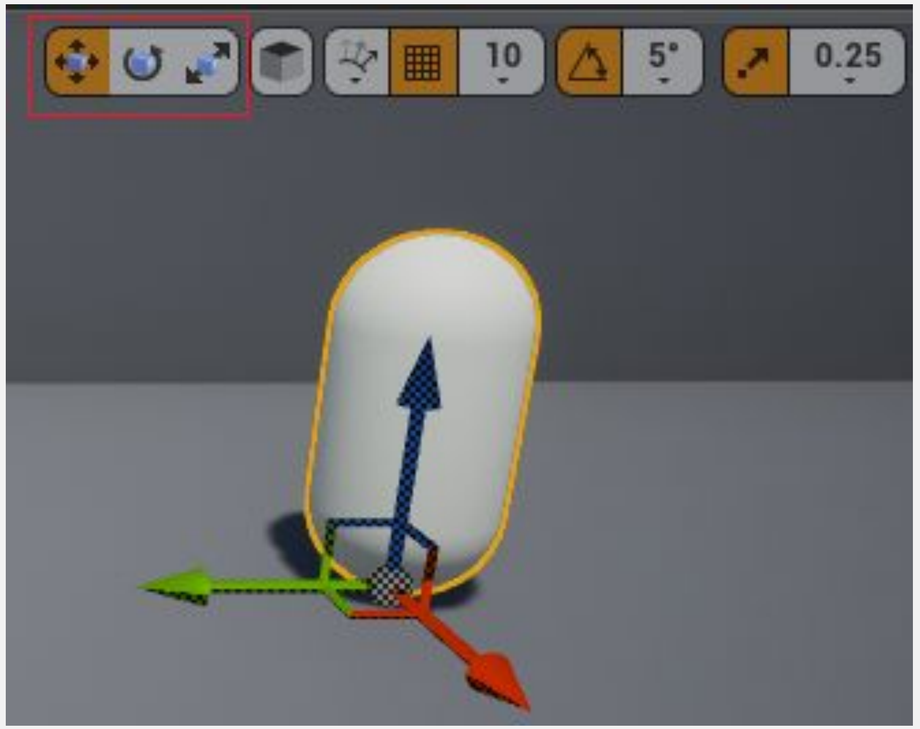
- Use vectors to represent location and movement
- Distinguish between world and local coordinates
- Use vector operations and functions

TRANSFORM

Every Actor has three **Transform** properties that represent its world location, rotation, and scale, as shown in the top image on the right.

You can modify the Actor’s transforms using the Details panel or by using gizmos in the Level Editor.

In the Level Editor, there are buttons to select the type of transformation to apply to an Actor. The bottom image on the right shows those buttons highlighted in red.

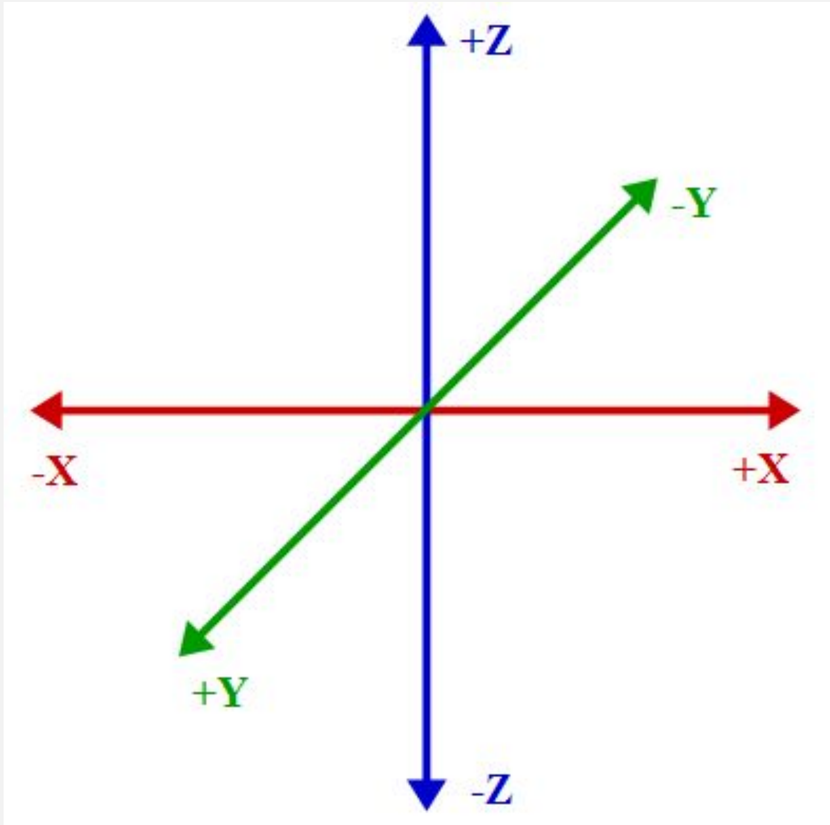


WORLD COORDINATES

3D space is represented by three axes: **X**, **Y**, and **Z**.

There are different ways to organize these axes. Unreal Engine uses the approach displayed in the figure on the right.

Any position in 3D space can be represented using a set of values for **X**, **Y**, and **Z** indicating the position on each axis. These values are stored in the **Location** variable, which is part of the transform struct, and they determine what is known as **world location**.



LOCATION FUNCTIONS

By default, an Unreal unit (uu) equals 1 centimeter. To use the values of the **Location** variable in a Blueprint, the following functions can be used:

- **GetActorLocation**: Returns the current position of the Actor.
- **SetActorLocation**: Sets a new position for the Actor.
- **AddActorWorldOffset**: Uses the values of the **Delta Location** parameter to modify the current position of the Actor.

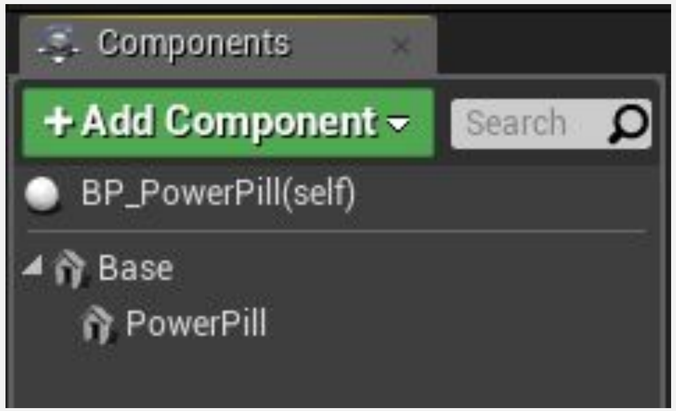
RELATIVE TRANSFORM

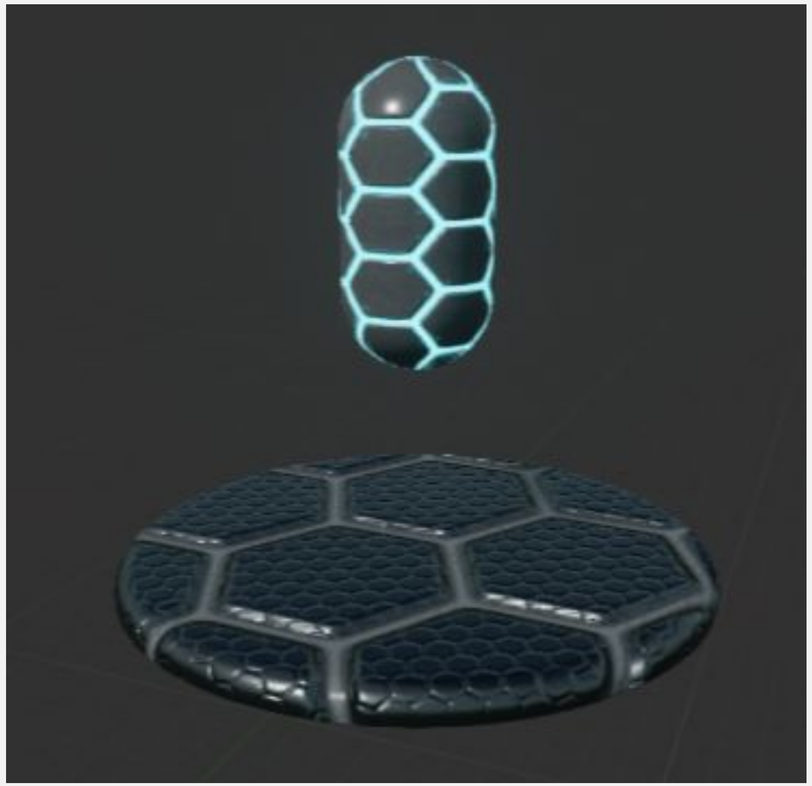
There is another concept known as **relative transform**.

To understand this concept, take as an example a Blueprint with two components, a **Base** component, which is the root component, and a **PowerPill** component, as seen in the image on the right.

The **Location** values of the **PowerPill** component are **X** = “**0**”, **Y** = “**0**”, **Z** = “**70**”.

The position of the **PowerPill** component is **relative** to that of the root component, so when the **Base** component is moved, the **PowerPill** component will follow the root component’s movement, as it must always be located at a distance from the **Base** component of 70 cm on the Z axis.





RELATIVE TRANSFORM FUNCTIONS

The position of the **PowerPill** component can be obtained in two ways: its relative location, which in this example is (0, 0, 70); or its world location, which in this case is the world position of the parent component (the **Base** component) plus the relative position of the **PowerPill** component.

The position of the component can also be defined in two ways using the functions **SetRelativeLocation** and **SetWorldLocation**. The **SetRelativeLocation** function defines a new position of the **PowerPill** component relative to the position of the root component, and the **SetWorldLocation** function receives as an input parameter a world coordinate and defines the position of the **PowerPill** component so that the sum of the position of the **Base** component and the position of the **PowerPill** component is equal to the specified world coordinates.

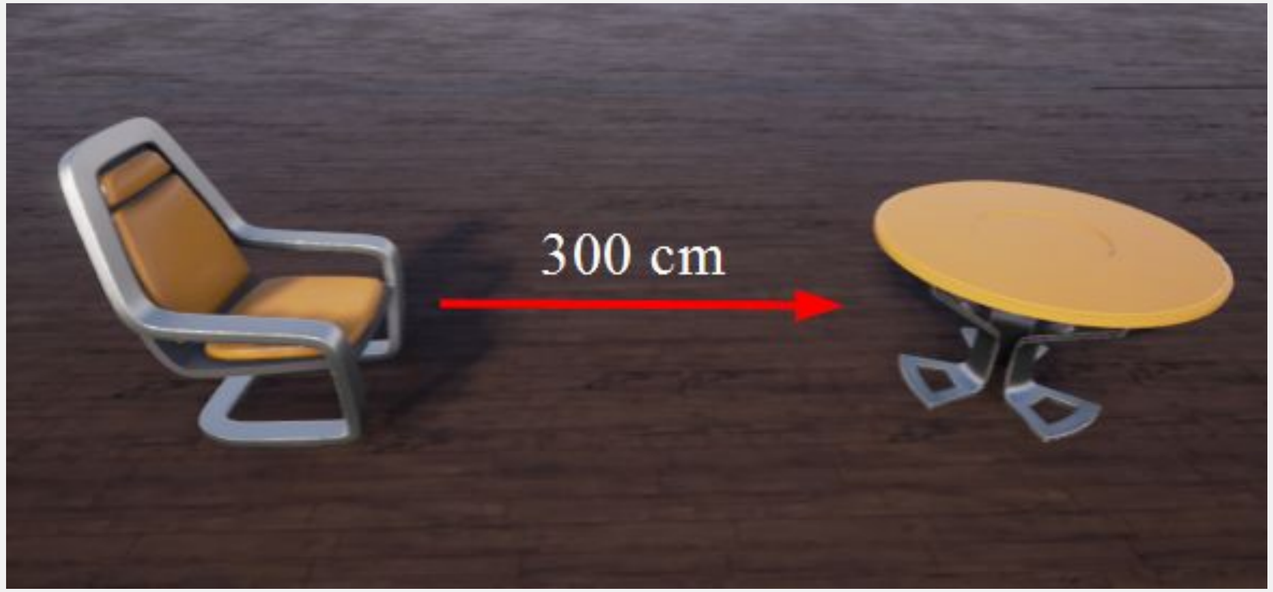
POINTS AND VECTORS

A **vector** is represented in Unreal Engine as a structure that contains the float values X, Y, and Z.

These values can be interpreted in various ways. One way a vector is used is to represent a **point** (or position) in 3D space. For example, every Actor has a variable called “**Location**” that returns a vector value.

Vectors are also used to represent movement. In the example on the right, in order to guide a robot to go from the chair to the table, two pieces of information are needed: the direction in which the robot should move and the distance.

Both the direction and the distance can be gathered into the vector: “300, 0, 0”.



VECTOR ADDITION

The sum of two vectors is determined by adding each of its elements.

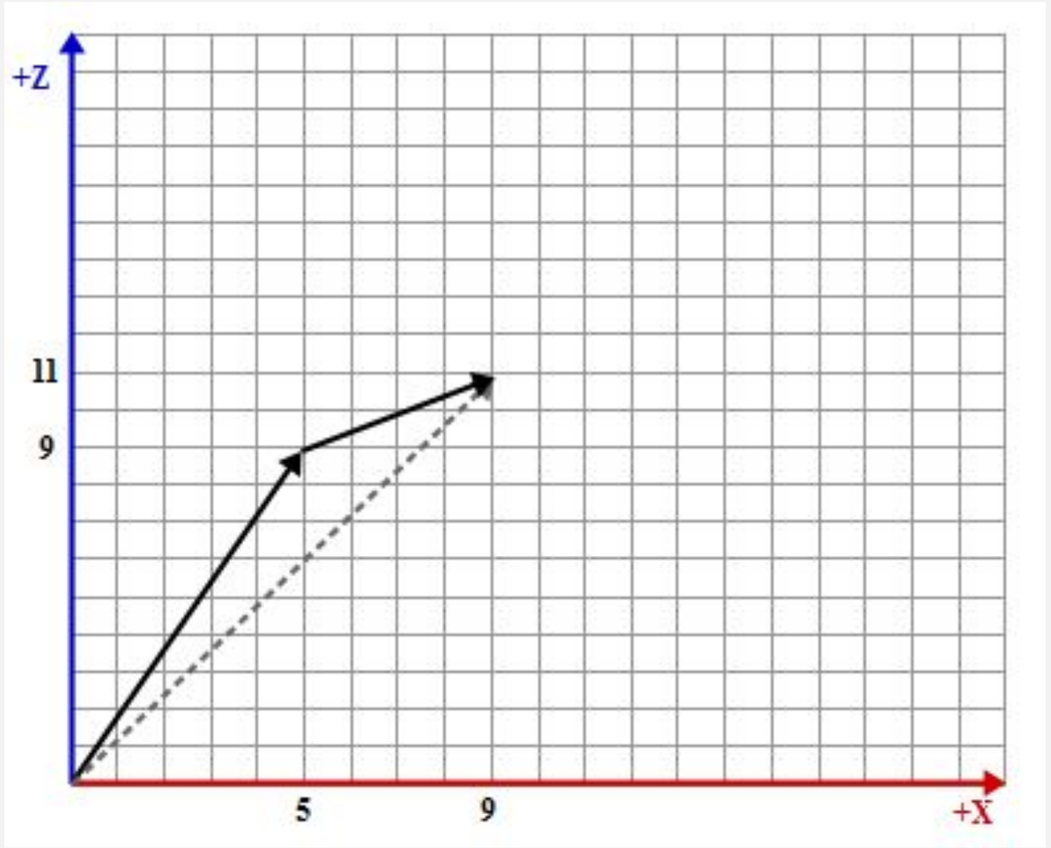
An example:

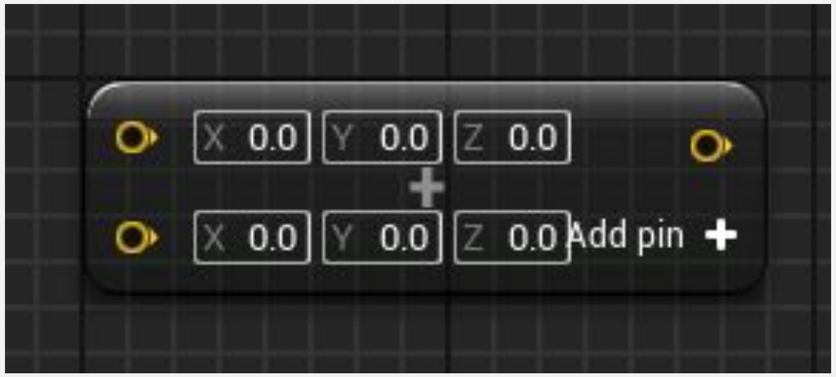
V1 = (5, 0, 9) and V2 = (4, 0, 2)

V1 + V2 = (5 + 4, 0 + 0, 9 + 2)

V1 + V2 = (9, 0, 11)

The bottom image on the right shows the Blueprint operator for the sum of vectors.





VECTOR SUBTRACTION

The subtraction of one vector from another is determined by subtracting each of its elements.

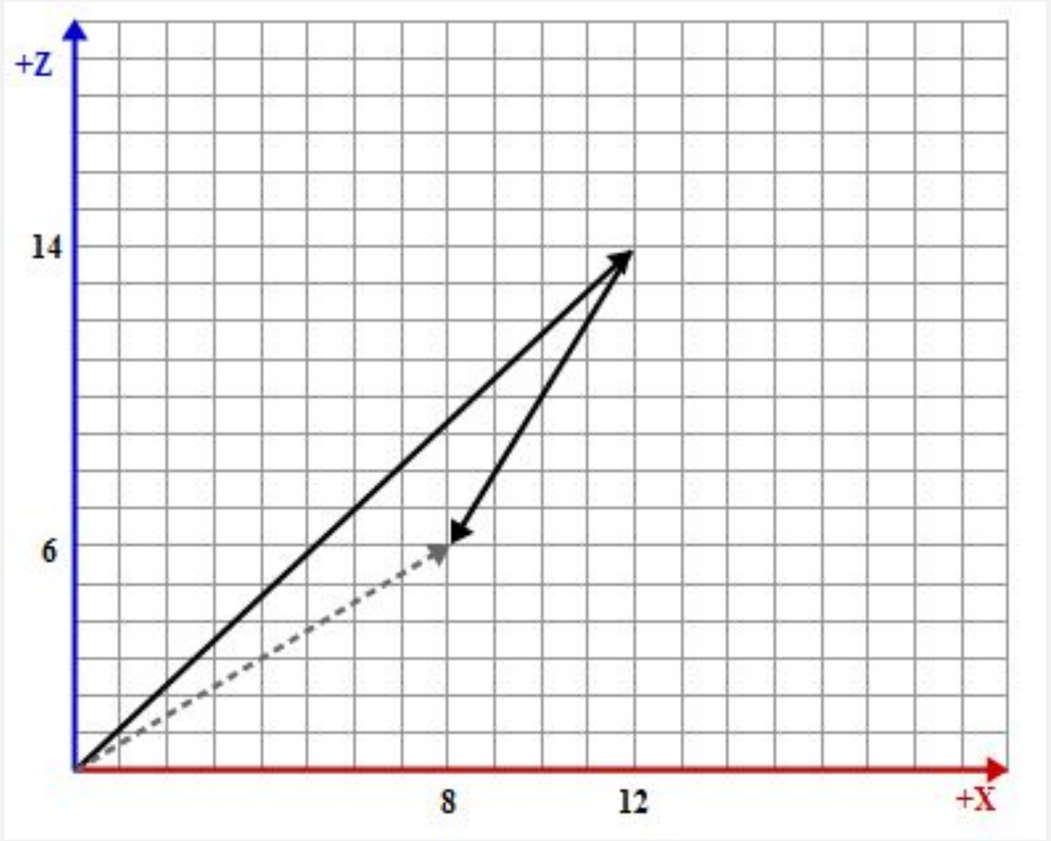
An example:

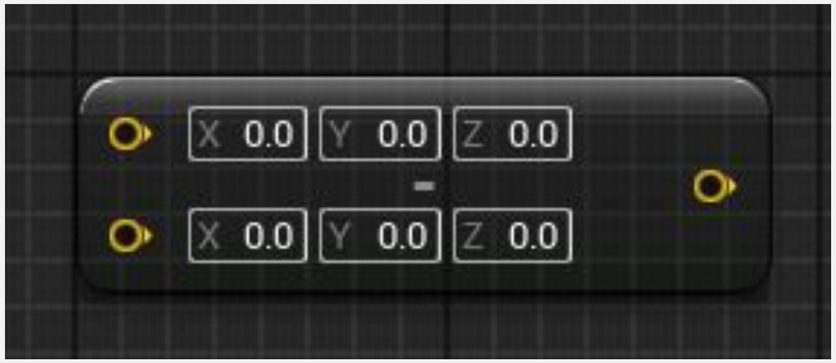
V1 = (12, 0, 14) and V2 = (4, 0, 8)

V1 – V2 = (12 – 4, 0 – 0, 14 – 8)

V1 – V2 = (8, 0, 6)

The image on the right shows the Blueprint operator for the subtraction of vectors.





LENGTH OF A VECTOR

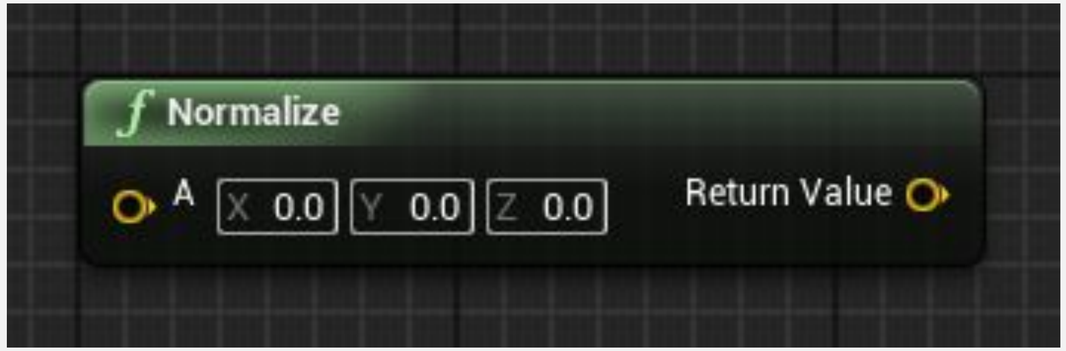
The length or magnitude of a vector can be calculated using the Blueprint function seen in the image on the right.

The value can be used to represent the distance between two points.

NORMALIZING VECTORS

Vector normalization is used to find a **unit vector**. The unit vector has a length equal to “1”.

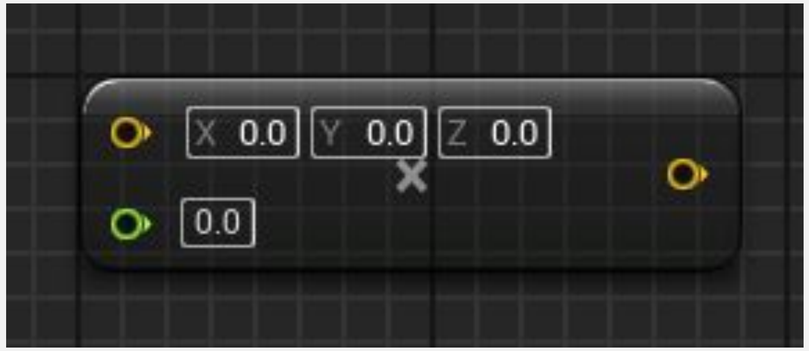
A normalized vector is often used when only the direction needs to be indicated. Some calculations should use only a normalized vector.



SCALAR VECTOR MULTIPLICATION

Multiplication of a vector by a scalar value is done by multiplying each of its elements by the scalar value.

This operation changes the length of the vector.

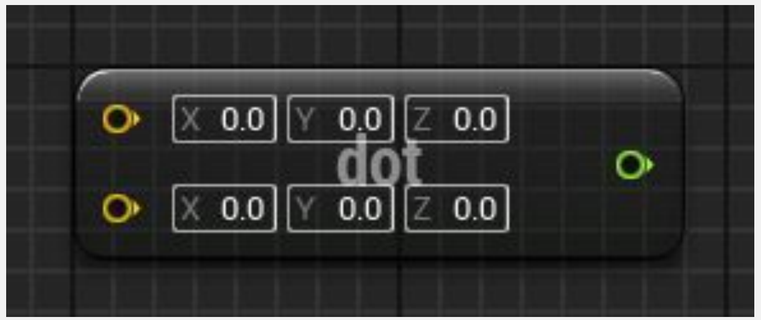


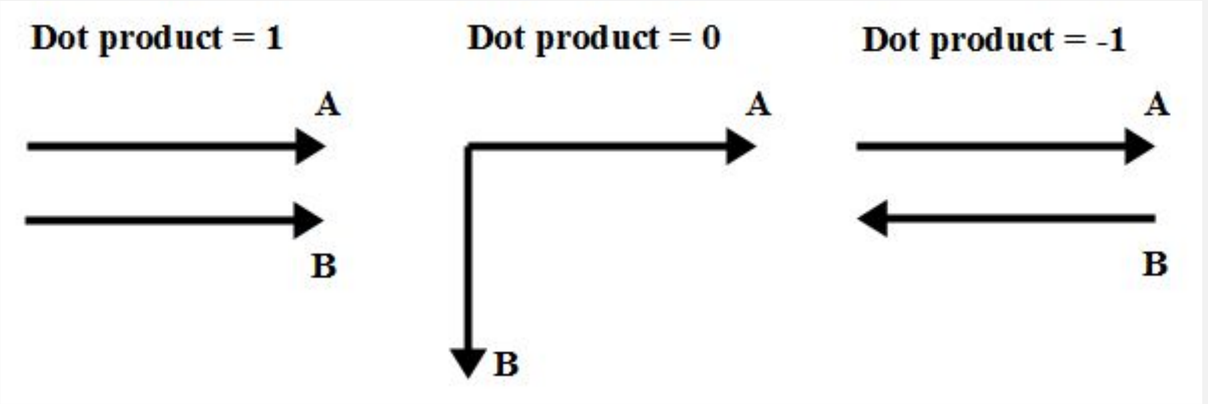
DOT PRODUCT

A **dot product** can be used to verify the relationship between two vectors, such as whether they are perpendicular or parallel.

If the two vectors are normalized, the dot product is equal to the cosine of the angle formed between the vectors and can range from –1 to 1.

The image on the right shows some examples of a dot product between two vectors.





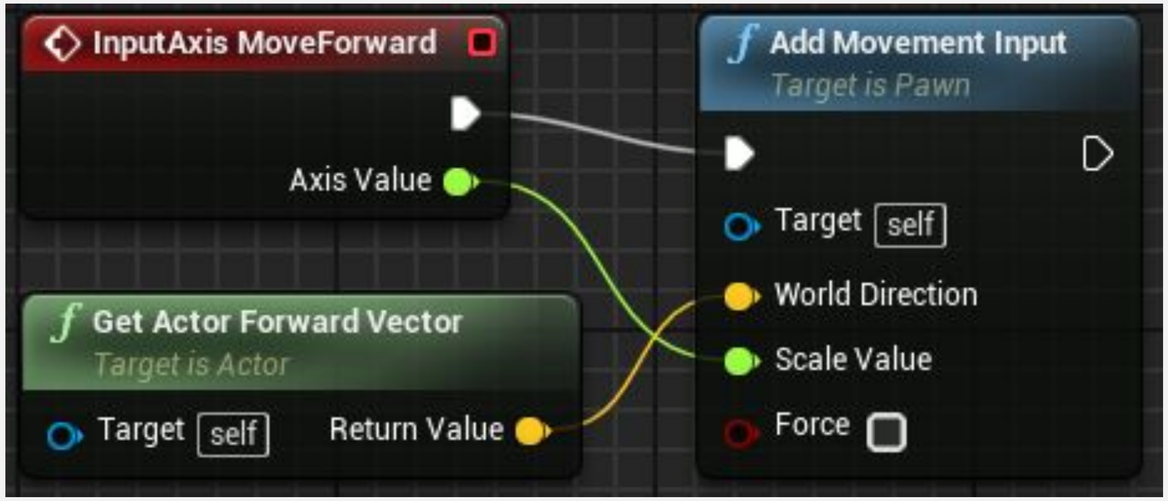
GET ACTOR FORWARD VECTOR

The **Get Actor Forward Vector** function returns a normalized vector (length = 1) that represents the direction an Actor is pointing in.

In the top image on the right, the **InputAxis MoveForward** event uses the **Get Actor Forward Vector** function to move the player forward or backward based on the value of the **Axis Value** parameter.

For example, using the standard “**WASD**” keys for movement, pressing the “**W**” key sets the value of the **Axis Value** parameter of the **InputAxis MoveForward** event to “**1.0**”, while pressing the “**S**” key sets the value to “**–1.0**” to reverse the direction.

There are two similar functions that provide vectors representing other directions: the **Get Actor Right Vector** function and the **Get Actor Up Vector** function, as seen in the bottom image.



SUMMARY

This lecture explained transforms, vectors, and world coordinates.

It also showed how to use vector operations and functions.