COMP2007 - Game Development

Week 3 - Code session

Animated Movement

Move the transform towards a target using an Animation Curve.
Use a float to evaluate a value from the animation curve over time and apply to movement towards a target

1. In the update method, we calculate a start time from the delta time value
2. The AnimationCurve can Evaluate a position on the curve, we store it in graphValue
3. We apply the final position by multiplying the target position by the graph value

```
void Update()
{
    startTime += Time.deltaTime;

    graphValue = curve.Evaluate(startTime);
    transform.position = finalPosition * graphValue;
```

Animated Scale

Scale the transform to a target over time using an Animation Curve.
Use a float to evaluate a value from the animation curve over time and apply to scale towards an end scale

1. In the update method, we calculate a start time from the delta time value
2. The AnimationCurve can Evaluate a position on the curve, we store it in graphValue
3. We apply the final position by multiplying the target position by the graph value

```
void Update()
{
    startTime += Time.deltaTime;

    graphValue = curve.Evaluate(startTime);
    transform.localScale = finalScale * graphValue;
```

Vector basics

All the methods of combining Vectors!
Vector3 uses operators to combine with other Vector3's and float values

Addition of two vectors

```
// adding two vectors
transform.position = vectorA + vectorB;
```

Subtracting vectors

```
// subtracting two vectors
transform.position = vectorA - vectorB;
```

Multiply by other vectors

```
// multiplying two vectors
Vector3 multiplyVector = new Vector3
{
    x = vectorA.x * vectorB.x,
    y = vectorA.y * vectorB.y,
    z = vectorA.z * vectorB.z,
};
transform.position = multiplyVector;
```

Multiply by a float

```
//multiplying by a float
transform.position = vectorA * value;
```

Dividing by a float
NOTE: do not divide a Vector3 zero! You may experience crashes or other issues

```
// dividing by a float
// NOTE: unity do not like dividing by zero, may give warnings
transform.position = vectorA / value;
```

Vector Direction

There are a few different methods of looking at or facing things and getting a distance between objects in Unity.

Transform.LookAt will rotate towards another Transform

```
// look at the target - calculates the direction for you!
transform.LookAt(target);
```

NOTE: LookAt will rotate in ALL directions, this may not be the functionality you want

Vector3.Distance will return the distance in unity units (metres) between two vectors

```
// distance using Vector3 distance
distance = Vector3.Distance(transform.position, target.position);
```

To get more control over the direction we can do some simple calculations:
Get a heading between the two positions

```
// heading between two points
// NOTE: put target position first to get a facing direction
heading = target.position - transform.position;
```

A directional Vector3 has a property we can use for distance - magnitude
In Trigonometry, the magnitude is the Hypotenuse of a triangle (the angled line)

```
// distance using magnitude
distance = heading.magnitude;
```

Dividing the heading with the magnitude will give us a direction
This is a vector facing TOWARDS the target position, relative to our position

```
// we can get a direction from the heading
direction = heading / distance;
```

We can set any direction we don't wish to rotate towards before applying it to our transform
Setting the Y direction will stop our transform from rotating on the X and Z axes

```
// we can control which direction our transform looks at
direction.y = 0;
```

Here, the transforms forward direction is set to influence the rotation
A transform has a direction for each axis

- Forward = Z axis (blue arrow)
- Up = Y axis (green arrow)
- Right = X axis (red arrow)

```
// we can set the transform to rotate towards a direction
transform.forward = direction;
```

```
// X
transform.right = direction; // right
```

```
// Y
transform.up = direction; // up
```

Tips

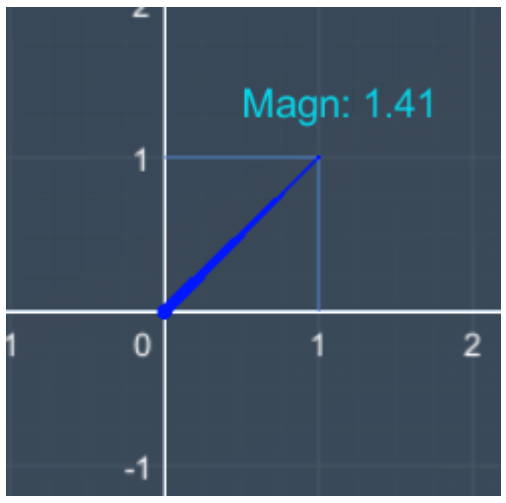
NOTE: Don't set more than one direction at the same time as they influence each other.!
Pick the direction you are most interested in:

- Aircraft may be interested in the Up direction to measure altitude etc
- Ground based characters are interested in the forward direction to rotate towards the direction of travel

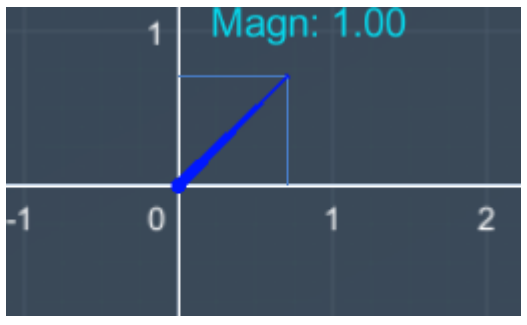
Vector normals

A “normalised” vector reduces (usually) the distance of a vector but not the direction
A normalised vector always has a magnitude of one

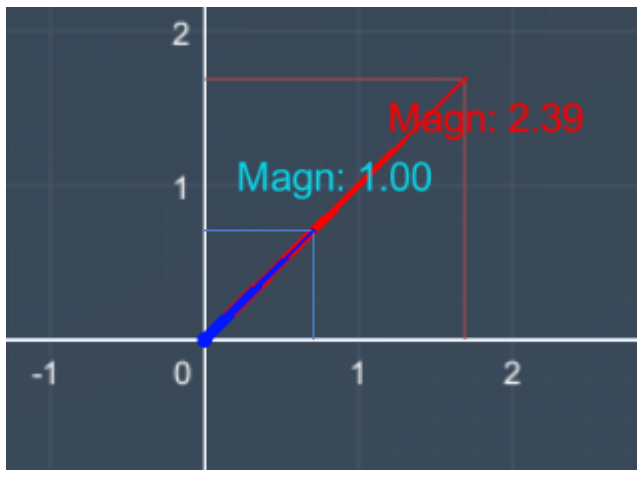
An “unnormalised” vector can have a magnitude of greater than one



A normalised vector has a magnitude of one, but still maintains the direction of the vector

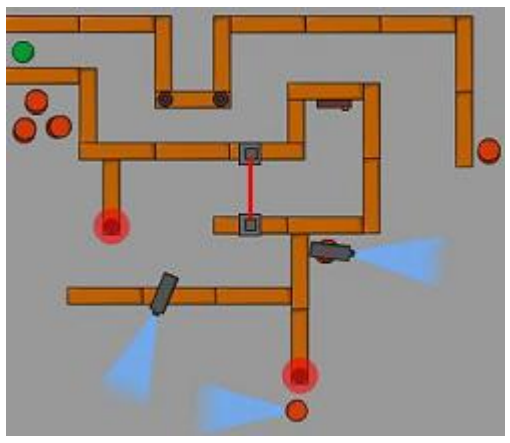


We can multiply a normalised vector by a float to control the distance directly
This is useful for detecting things within a specific range

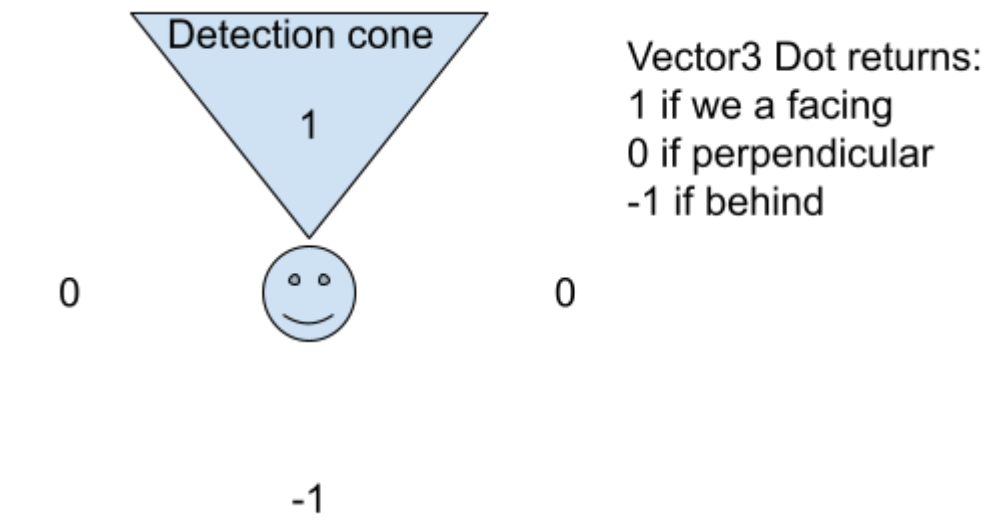


Vector Dot

We can detect if we are facing an object within a specific angle
For example, a game has security cameras that can spot the player within a sight radius



Vector Dot can detect if a position is in front or behind, but cannot detect if it is to the left or right of us



Vector3 Dot returns:
1 if we a facing
0 if perpendicular
-1 if behind

We can convert the value to a 360 degree representation to place a detection cone

First we get a heading, or direction of travel by subtracting the target position from our own

```
// we get the direction between the transform and target
Vector3 direction = target.position - transform.position;
```

Vector3.Dot returns a float value in radians (mathematical degrees)
Dot takes two directions to calculate the facing angle in radians

```
// use Vector3.Dot to tell if we are facing the target or not
// 1 = facing target, 0 = perpendicular to target, -1 = target is behind us
// the angle is in radians, not degrees!
float radians = Vector3.Dot(direction.normalized, transform.forward);
```

Here we convert the radians angle into degrees
We also have to apply a MathF.Acos (arc cosine) to “normalised” our radians
NOTE: this does not have be be done when converting angles using Quaternion, we can just use Mathf.Rad2Deg
To ensure the cone of vision extends correctly we multiply the value by 2

```
// we can convert the angle from radians into degrees
// multiply by two so the angle covers left and right side
degrees = (Mathf.Acos(radians) * Mathf.Rad2Deg) * 2;
```

Ball Vectors

This is an example of smooth movement with acceleration/deceleration without using a Rigidbody physics component.
Often, developers have a smooth movement system without the need for physics forces; for example, they may want to restrict movement to certain areas using a navmesh instead of collision boundaries. The physics system may not always be the best form of player movement but can still be part of other game systems.

The ball has max speed and max acceleration settings as sliders for convenience

```
// the maximum speed we can travel at
[SerializeField, Range(0f, 100f)]
float maxSpeed = 10f;

// the maxumim acceleration we get achieve
[SerializeField, Range(0f, 100f)]
float maxAcceleration = 10f;
```

We store our current calculated velocity as a vector3
NOTE: we are only moving along the X and Z axis, but this can be adapted for other situations (underwater, space etc)

```
// our current velocity
private Vector3 velocity = Vector3.zero;
```

Inside the Update method we get the player input from joystick or keys
We use a vector2 to store the values
NOTE: this is the C# recommended way to create new objects with initial values

```
// create a Vector2 for input from a joystick or keys
Vector2 playerInput = new Vector2
{
    x = Input.GetAxis("Horizontal"),
    y = Input.GetAxis("Vertical")
};
```

We need to ensure the player input values are always less than one
ClampMagnitude will “clamp” the X and Y values of a vector2 and return those updated values

```
// here we make sure the magnitude of player input to 1
playerInput = Vector2.ClampMagnitude(playerInput, 1f);
```

The “desired” velocity is the movement speed we want to achieve in this update frame
We multiply the player input by our max speed to get the desired velocity
NOTE: we assign the player input Y to the Z axis of our desired velocity, as we are moving along the X and Z axes

```
// create a vector3 for the desired velocity
Vector3 desiredVelocity = new Vector3(playerInput.x, 0f, playerInput.y);
```

```
// multiply by our max speed field
desiredVelocity *= maxSpeed;
```

Max speed change is the change in speed per update frame
This “locks” our velocity calculations into the speed unity currently runs at (Time.deltaTime)
We can use max speed change to “clamp” our velocity

```
// store our change in speed over time
// this will clamp our velocity to match our max speed
float maxSpeedChange = maxAcceleration * Time.deltaTime;
```

The velocity X and Z values are set here - velocity will be “clamped” by our max speed change created above
We use MathF.MoveTowards for each axis to interpolate from our current velocity to the desired velocity
MoveTowards will “clamp” the new velocity using a “delta”, we are using max speed change as our delta

```
// apply the velocity from our current to desired velocity for the X and Z axes
// Mathf.MoveTowards limits the amount we can move per update using a delta (maxSpeedChange)
velocity.x = Mathf.MoveTowards(velocity.x, desiredVelocity.x, maxSpeedChange);
velocity.z = Mathf.MoveTowards(velocity.z, desiredVelocity.z, maxSpeedChange);
```

This is where we ensure no velocity is applied to the y position
If you wish to use the Y axis in your calculations, remove this line

```
// make sure our Y velocity is zero for top down movement
velocity.y = 0;
```

The displacement is the velocity for this update frame
We multiply the velocity by Time.deltaTime to get an exact velocity movement this frame

```
// store the calulated velocity over time
Vector3 displacement = velocity * Time.deltaTime;
```

Finally we apply the displacement to our transform position!

```
// apply our calculated velocty to the transform
transform.position += displacement;
```

Links

Vectors

Vector3
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3.html

Vector3.Distance
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3.Distance.html

Vector3.magnitude
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3-magnitude.html

Vector3.Dot
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3.Dot.html

Vector3.Cross
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector3.Cross.html

Vector2.ClampMagnitude
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Vector2.ClampMagnitude.html

MathF

MathF.Acos
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Mathf.Acos.html

Mathf.Rad2Deg
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Mathf.Rad2Deg.html

Mathf.MoveTowards
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Mathf.MoveTowards.html

Transform

Transform.LookAt
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform.LookAt.html

Transform.forward
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform-forward.html

Transform.right
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform-right.html

Transform.up
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Transform-up.html

Debug

Debug.DrawRay
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Debug.DrawRay.html

Debug.DrawLine
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Debug.DrawLine.html

Input

Input.GetKeyDown
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Input.GetKeyDown.html

Input.GetAxis
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Input.GetAxis.html

KeyCode
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/KeyCode.html

Other

UnityEvent
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Events.UnityEvent.html

Range Attribute
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/RangeAttribute.html

SelectionBase Attribute
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/SelectionBaseAttribute.html

OnTriggerStay
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/MonoBehaviour.OnTriggerStay.html

AnimationCurve
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/AnimationCurve.html

Time.deltaTime
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/Time-deltaTime.html

TrailRenderer
https://docs.unity3d.com/2020.2/Documentation/ScriptReference/TrailRenderer.html