# Adding in 3D Models and Animations

We've got a fairly complete small game so far but it needs some models to make it look nice, this next set of tutorials will help improve this. They are all about importing models into Unity, configuring them and setting up animations.

The models I've chosen for this tutorial are pulled from Mixamo.com, mixamo used to be an indie site but was taken over by Adobe. It's undergoing a restructuring at time of writing but as a consequence provides a huge library of free models and animations to match. If you can't find what you want there, check the Asset store, if not there, some of the many other 3d models sites.

# Straight import

There's two ways we could do this, start afresh or change some of the parameters of the capsule we're already using. For simple models it's easier to change and replace the mesh but actually, since the model we're using for our character is fairly complex we're going to start afresh.

Drag Adam into the scene, you can see him to the right, the mesh is made up of multiple parts all contained and setup for you. We need to add the components used in the current player object, they are;

- A Character Controller with it's center raised by 1
- A shooting script
- A health script
- An interaction script
- A LookX script
- A Player Movement script

Check your Player you've already made if you're not sure, you'll also need to drag and drop and reposition the Main Camera from the other player to a relevant position. Then make sure you delete the existing player. Make a prefab of this player, make sure the prefab is tagged as "Player".

If you hit play you'll see the model in but not animated, to animate it we need to setup a few more things.

# **Using Mixamo**

MooICT

Mixamo is a state based animation system that allows you to setup complex animation settings with minimal scripting. Animations play when you transition from one state to another, for instance going over a speed.

🔻 🚰 🗹 Animator

Update Mode

Culling Mode

Apply Root Motion

Not initialized

Controller

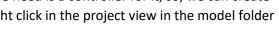
Avatar

If you click on Adam at the moment you'll see he has an Animator component attached, this is what fuels the animations and allows you to control them.

What we need is a controller for it, so, we can create one, right click in the project view in the model folder



Adam\_Clothes Adam\_Eyes Adam\_Skin mixamorig:Hips



None (RuntimeAnimatorCo ◎

👙 AdamAvatar

Cull Update Transforms

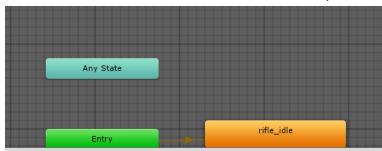
Normal

0

#

**‡** 

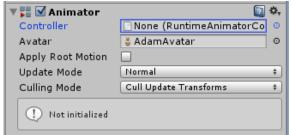
and create-> Animator Controller, call it what you like, Adam is a good bet. If you double click it, it



the orange means it is the default animation. It's what will play by default.

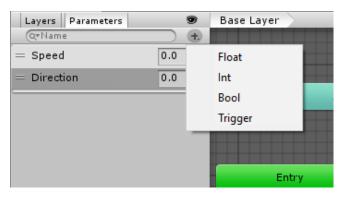
Drag the controller onto the slot in the animator component and press play. It should play idle animation now.

# will open up showing two boxes, entry and any state. These are the default entry points for the state machine, in the picture to the left you can see that there is another box connected to entry "rifle\_idle". This is because I have dragged that onto the animator view and it has become a new state we can transition to,



# Moving between animations

Right now we're playing the default animation but nothing else, we need to tell Mecanim to move between the idle and run animation. We want to do that when we get above a certain speed, so we'll need to add some code for this and some settings in Mecanim.



In the top left of the animator window you should see a button labelled parameters. These are the bits of information we use to decide what animation to play. Click the little plus and you'll get a dropdown as shown, click Float and you'll get a new parameter. Do this twice, once for Speed and once for Direction.

Drag run\_forwards onto the animator too, we're going to set it up to transition between

idle and running based on speed of the character.

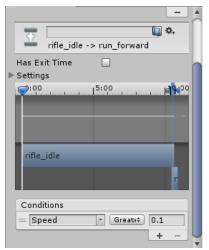
Select rifle\_idle, right click it and then make transition, clicking on run\_forwards when the arrow appears, now do the same in reverse, from run\_forwards to rifle\_idle.

Click on the arrow that appears, going from idle to forward, the bottom one in the below screenshot, you can use this to trigger moving from one state and animation to another.



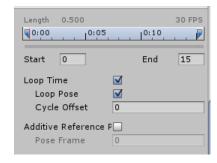
To do this you need to do two things, in the inspector you should see a window like the right one appear, (you may need to scroll down) uncheck "Has Exit Time" and press the plus to add a condition. In this case speed, which we want to be greater than 0.1 to transition.

Do the same for the reverse one back to rifle\_idle from forward but instead of greater than, set it as less than 0.1.



Finally, we need to set the animations to loop and to ensure they are not applying any motion to our characters. To do this, double click on the animation to access its properties, scroll down and tick loop time and loop motion.

We then need to do the same for run left, drag it in, add two transitions to and from idle and click on the transitions to bring it up and disable "has Exit Time". This time add a parameter of Direction and from idle



you want "Less than -0.1" and "Greater than -0.1" for back . This is because an Input in Unity if you remember goes -1 for all the way back and +1 for all the way forwards, making all left -1.

Challenge 1 – do the same for run backwards and run right you'll need to make sure that loop time and loop motion is enabled.

So we've set up our animations to transition at certain points, now we need to update that information to actually trigger our parameters.

Open up your playerMovement script, we need to add the bits highlighted in yellow which are new. We get the animator attached (the thing that drives the animation) then in update, whenever we get input, we send that to the animator and it does the rest. Easy right?

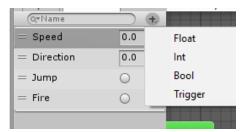
```
[SerializeField] float moveSpeed = 5.0f;
public float h;
public float v;
Animator anim;
// Use this for initialization
void Start () {
    charController = GetComponent<CharacterController>();
   anim = GetComponent<Animator>();
}
// Update is called once per frame
void Update () {
    h = Input.GetAxis ("Horizontal");
    v = Input.GetAxis ("Vertical");
   anim.SetFloat("Speed", v);
    anim.SetFloat("Direction", h);
    Vector3 direction = new Vector3(h, 0, v);
```

#### **Last Animations**

There are a few player animations that we still haven't sorted, namely jumping and firing, we'll deal with jumping first as it is the easiest. The jumping animation is one that is fixed in place, so we don't

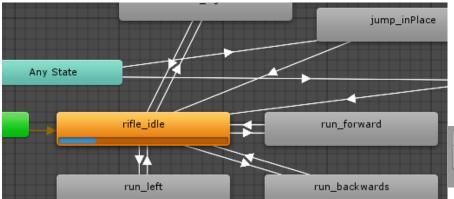
need to worry about looping it or fixing it's pose. Drag jump in place animation to the animator base layer, we want this to just play once, to do this we use a new parameter type called Trigger. A trigger is something that when set to true, runs then sets itself back to false once the animation is finished.

Add a new parameter as before, but this time of Trigger, whilst we're at it go ahead and add a Fire one, a trigger the same. We're going to add some transitions, this time though slightly differently.



For jumping we want to be able to transition from virtually any state, so, to do this we need to transition from the any state box in mecanim. See the screenshot below for an example.

Create a transition from any state to jump\_inPlace, set the parameter to "Jump" and then a



transition back from jump\_inPlace to rifle\_idle because once you've jumped you want to return back to whatever state is needed.



Finally, we need to add a trigger for this into the code, this is very similar to before, in the PlayerMovement script, scroll down to the jump bit as shown and add the line highlighted.

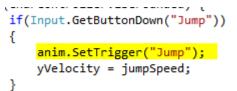
Give it a try, when you press jump, he should jump and move if he doesn't you've probably not dragged the controller into the Animator's controller variable slot. See the screenshot below. Drag the controller we created earlier into this slot.

You'll notice we're missing one important thing, a gun, it's included in the package, the CQAssaultRifle from the asset store. I've dragged it onto the mixamo model, attaching it to the hand of

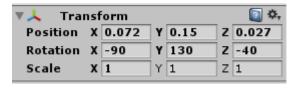
▼ mixamorig:Hips
 ▶ mixamorig:LeftUpLeg
 ▶ mixamorig:Spine
 ▼ mixamorig:Spine1
 ▼ mixamorig:Spine2
 ▶ mixamorig:LeftShoulder
 ▶ mixamorig:Neck
 ▼ mixamorig:RightShoulder
 ▼ mixamorig:RightArm
 ▼ mixamorig:RightForeArm
 ▼ mixamorig:RightHand
 ▶ CQAssaultRifle

the model which can be seen expanded to the left. The position and rotation relevant to the hand are shown below, it's essentially horizontal positioned so the hand looks like it grips the rifle. The animations will change the position so it

makes every animation look like it's holding on.





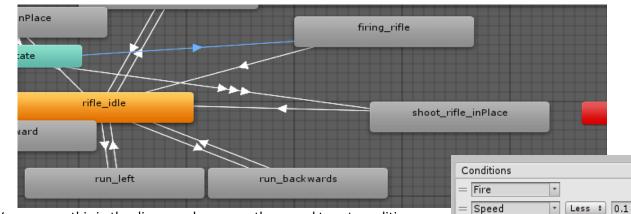


Now for firing, for firing there are two animations, one for moving and one for standing, first of all drag the two shooting animations in, we'll deal with moving first but we need to think about when we want each to play.

Standing and shooting plays when fire is pressed AND direction is between -0.1 and 0.1 AND speed is between -0.1 and 0.1. In other words, when fire is pressed and we're not moving.

Moving and shooting plays when fire is pressed AND EITHER direction is greater than 0.1 OR direction is less than -0.1 OR speed is greater than 0.1 OR speed is less than -0.1.

Let's start with a stationary shooting, that's firing\_rifle in the animations, create a transition from any state to firing\_rifle and then back from firing\_rifle to idle.



You can see this in the diagram above, you then need to set conditions, these are shown to the right and should be easily understandable. These are called AND conditions, we leave the condition for the return as it was, letting it exit after the animation has finished.

For the next one we want to add four different cases,

- Speed greater than 0.1 and fire pressed
- Speed less than 0.1 and fire pressed
- Direction greater than 0.1 and fire pressed
- Direction less than 0.1 and fire pressed

These are called OR conditions, we set these up by making four separate transitions from any state to shoot\_rifle\_inPlace. The arrow will turn to a

triple arrow and you'll see a list of transitions if you click on it. If you click on each of the transitions (as shown) you can scroll down and set the conditions at the bottom of the window.

Open up your shoot script and make some changes to add and update the code to use and trigger these conditions. Again we have a variable to store the animator, get a reference to it in Start() and then trigger it BEFORE the raycast as shown.

```
[SerializeField] int damageDealt=20;
[SerializeField]
LayerMask layermask;
Animator anim;

// Use this for initialization
void Start () {
    anim = GetComponent<Animator>();
    Cursor.lockState=CursorLockMode.Locked;
    Cursor.visible = false;
}
```

```
if (Input.GetButtonDown ("Fire1")) {
    //Make a raycast with the line starting
    Cursor.lockState = CursorLockMode.Locke
    Cursor.visible = false;
    Ray mouseRay = GetComponentInChildren<C
    RaycastHit hitInfo;
    anim.SetTrigger("Fire");
    //Send the raycast and if the raycast h
    if (Physics.Raycast (mouseRay, out hitI)</pre>
```

= Speed

Direction

= Direction

Transitions

4 Transitions

Greate# -0.1

Greate# -0.1

Solo Mute

- Less + 0.1

AnyState -> shoot\_rifle\_inPlac

AnyState -> shoot\_rifle\_inPla \_\_\_

AnyState -> shoot\_rifle\_inPla 🗌

AnyState -> shoot\_rifle\_inPla 🗌

AnyState -> shoot\_rifle\_inPla 🗌

MooICT

Spawning new Enemies

That should be it now for shooting, give it a try and explore your level before we move on to the enemies.

# Adding an enemy- Vampires

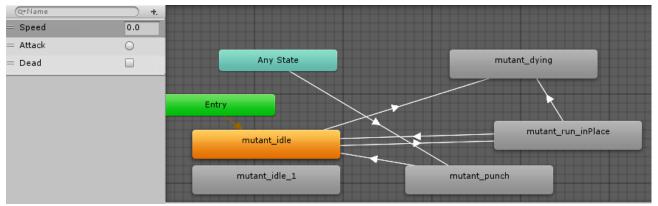
We'll move over to adding our scripts to the enemy now, we're going to add a vampire model, again from mixamo with it's animations too. Go ahead and drag the vampire into your scene, we then go about adding to the vampire all the things we'll need for it to be an "enemy". This is where Unity's component based design shines.

#### Add;

- A NavMeshAgent
- A health script
- A Navmesh movement script (either the basic one or waypoints will do) make sure to update it's target or waypoint variables by dragging the relevant construct in
- A Character Controller (we're not moving with it, we're instead using it for our physics)
- Duplicate (select and press Ctrl+D) the enemyAttack object attached to one of our prior made enemies and drag it to a relevant place on the vampire.

So, the NavMeshAgent and companion script will control movement, we shouldn't need to alter that, the health script will control health and the enemyAttack object and it's attached trigger and script should do the attacking.

Let's drag our animations in and set them up, the screenshot below is an indicator of how it should be setup. Take a look at the parameters and transitions, I'll list what the transitions should be below.



#### Transitions;

- Idle to dying- Dead=true hasExit time ticked
- Mutant Run to Dying Dead=true hasExit time ticked
- Idle to Mutant Run Speed Greater than 0.1 hasExit time unticked
- Mutant Run to Idle Speed Less than 0.1 hasExit time ticked
- Any State to mutant punch Attack (trigger) hasExit time unticked
- Punch to Idle none has Exit time ticked

For running and idle we want the animation to play over and over so we double click it and go to loop time and loop pose to ensure this is the case.

As a quality of life thing we make a change to EnemyNavMovement, see the bit highlighted. This ensures that when close the model always looks at the player.

What we need now is a script to power the movement animations; we can do most of

```
// Update is called once per frame
void Update () {
    agent = GetComponent<NavMeshAgent> ();
    agent.SetDestination (target.position);
    if(agent.remainingDistance<(agent.stoppingDistance+0.5f))
    {
        transform.LookAt(target.transform);
    }
}</pre>
```

it in current scripts but doing it inside the movement scripts means if you make changes you'll have to update them. Instead we take a more modular approach.

Create a new script called EnemyNavAnimations and attach it to the vampire, this will power our animations.

We setup in a fairly standard way, getting the animator and the navmeshagent attached;

We're going to be using both of these below.

```
// Update is called once per frame
void Update () {
    print(agent.velocity.magnitude);
    if (agent)
    {
        anim.SetFloat("Speed", agent.velocity.magnitude);
    }
}
```

```
NavMeshAgent agent;
Animator anim;

// Use this for initialization
void Start () {
   agent = GetComponent<NavMeshAgent > ();
   anim = GetComponent<Animator>();
```

In Update we first print the velocity of the navmeshagent- I.e. how fast he's going. As agent. Velocity returns a vector (i.e. how fast it's going in the 3 directions) we use magnitude to get the overall speed.

This is just a debug thing and can (and should) be deleted if you wish. We use an if statement to check to make sure there is still an agent attached then we pass the animator the speed of the navmeshagent.

You should be able to try it now and see the animation playing.

Let's move on to the next animation state, punching, whenever the enemy gets close enough we

want to attack with a punch. This is actually quite easy, we just trigger the animation when we attack. Open EnemyAttack script and add the now ubiquitous reference to animator that's obtained in start, like so;

```
Animator anim;
void Start()
{
    anim = GetComponentInParent<Animator>();
}
```

```
if (other.tag == "Player"&&Time.time>=nextTimeAttackIsAllowed) {
    Health playerHealth = other.GetComponent<Health>();
    anim.SetTrigger("Attack");
    playerHealth.Damage(damageDealt);
    nextTimeAttackIsAllowed = Time.time + attackDelay;
```

We then, in the OnTriggerStay section, just before we actually damage the player set the trigger for attack.

Finally, we have an animation state for dying, that's right instead of just disappearing our enemies now die with style. Sadly, this one isn't as simple as just added a trigger reference, open your health script and we'll make some modifications.

Once again, add a variable to hold animator and use getComponent in start to update it. On the next page you'll see the Damage function with a few bits added in.

```
public void Damage(int damageValue)
    currentHealth -= damageValue;
    if (currentHealth <= 0) {</pre>
        if (gameObject.tag != "Player")
            if(anim)
            {
                anim.SetBool("Dead",true);
            }
            Destroy(GetComponent<EnemyNavMovement>());
            Destroy(GetComponent<NavMeshAgent>());
            Destroy(GetComponent<CharacterController>());
            Destroy(GetComponentInChildren<EnemyAttack>());
        }
        else
        {
            Destroy(gameObject);
   }
```

The circled bit is the new bit, we check to see if the object the script is attached to is tagged as player.

If not, we check to see if there is an animator attached.

If there is, we set the Boolean for whether it's dead to true. This means that the animation plays and, because we have no transition back, no other's play.

Afterwards, we destroy the components attached to the enemy, the nav script and agent to stop it moving around. The controller to stop us colliding with thin air and the enemyattack script to stop us walking through it and getting hit.

If health is attached to the player, we destroy it same as before.

Your enemy should be done, don't forget to make a prefab of the new Vampire enemy.

# Firing effects

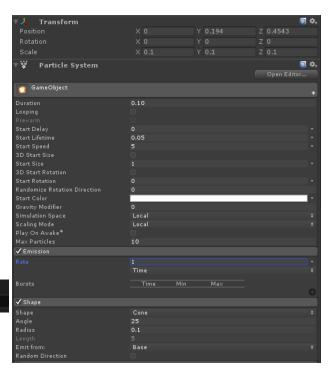
At the moment we have no idea if we've fired or hit except for the animation, we want two effects, a gun shot and a blood splatter.

WE'll do the gun shot first, it's easier but both rely on the same premise. Find the gun object attached and right click on it to add a new empty game object. Position it at the barrel of the gun, scale it down to 0.1,0.1,0.1 and to this GameObject add a particleSystem and set it up similar to the screenshot shown.

Finally, add a material and modify your shootingscript with the second line shown below.

```
anim.SetTrigger("Fire");
GetComponentInChildren<ParticleSystem>().Play();
```

I've attached a fire material that will work. You should now see a flash of fire from the end of your gun.



## Blood splatter

Now we have muzzle flash we want to add in some blood splatter to show where you get hit, we're not going to do anything too flashy, just use some red particles to add some blood splatter. Create

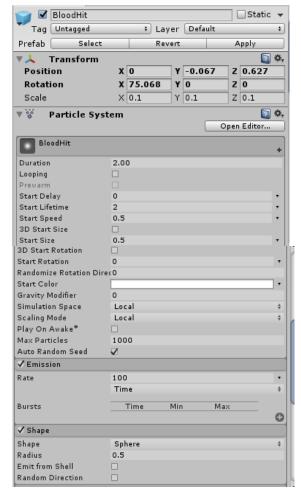
an empty gameObject, name it BloodHit and add a particle system to it like you do above, the settings are shown below drag the bloodSplat material supplied on;

We scale it down to 0.1,0.1,0.1- we don't want a huge blood hit, then tweak the settings until they are as shown, this isn't perfect but I liked the way it looked. Create a prefab of this by dragging it into project view.

Now we need to use this prefab, we're going to change the shooting script to use this prefab and place it every time we hit something.

We need to change this so that whenever we damage the enemy we instantiate this bloodHit we've created. We first get the point we hit from hitInfo – the information output from the raycast about where it hit.- this let's us get the point we hit. Then we sets the rotation using FromToRotation, transforming from Vector3.forward to the line coming out of where we hit, just like blood would.

Then we Instantiate it, feeding all this info in- not forgetting to set a new variable at the top to hold the bloodHIt and dragging the bloodHit prefab onto the bloodHit variable slot- in the player prefab.



```
[SerializeField] GameObject bloodHit;

if(enemyHealth !=null)
{
    enemyHealth.Damage(damageDealt);
    Vector3 bloodHitPos = hitInfo.point;
    Quaternion bloodHitRot = Quaternion.FromToRotation(Vector3.forward,hitInfo.normal);
    Instantiate(bloodHit, bloodHitPos,bloodHitRot);
}
```

#### All the bloodsplatter objects

You'll see the blood splatter objects stack up- we're making lots but not doing anything with them, it's an easy fix. Create a new script on the BloodHit prefab called "bloodDestroy" this is just going to destroy our bloodHit after a set time.

```
void Start () {
    Destroy(gameObject, 1);
}
```

That's all this is, the 1 could be 2 if you'd like, to show the full burst, or you could change it higher still, it's just the numbed of seconds. So the object is created, then destroyed after a set time.

#### Player Blood splatter

Now we have some blood splatter for the enemies when hit, how about when players are hit, open up EnemyAttack and modify it to this.

```
[SerializeField] GameObject bloodHit;

void OnTriggerStay(Collider other)
{
    if (other.tag == "Player"&&Time.time>=nextTimeAttackIsAllowed) {
        Health playerHealth = other.GetComponent<Health>();
        anim.SetTrigger("Attack");
        playerHealth.Damage(damageDealt):

        Vector3 hitDirection = (transform.root.position - other.transform.position).normalized;
        Vector3 hitEffectPos = other.transform.position + (hitDirection * 0.01f) + (Vector3.up * 1.5f);
        Quaternion hitEffectRotation = Quaternion.FromToRotation(Vector3.forward, hitDirection);
        Instantiate(bloodHit, hitEffectPos, hitEffectRotation);

        nextTimeAttackIsAllowed = Time.time + attackDelay;
    }
}
```

This works different to the raycast and isn't as easy to follow, the first bit should be familiar by now. The second bit to add starts with a Vector3 to hold the direction that we hit in, that just takes the position of the enemy and the position of the thing that was hit and subtracts them to work out the rough direction.

The second part, hitEffectPos takes the position of what we hit, adds a little bit in the direction away from the thing we hit and up.

The third is the rotation, working out the rotation towards the direction we hit and the fourth actually creates the bloodHit with these parameters.

Again, drag the bloodHit prefab onto the PREFAB's bloodHit variable on EnemyAttack script/object. You should see some blood splatter when the enemy hits you- it will though depend on

# Decorating the rest of the scene

There are loads of options out there for decorating your scene straight from the asset store for free, one of my favourites is Low Poly Street Pack by Dynamic Art, there's also Stylized Simple Cartoon City by Area730 for a low poly look and tons more between of small or large asset packs to build any environment. For the best look do it yourself, bring it in and compose it for yourself, but I was feeling lazy and just used one of the free Environments, in my case the Free low poly city block. I've not included it for download- you need to download it yourself, or another, but it'll look cool, if you did some of the earlier challenges about importing a terrain and other decorations, you might just want to bring your own decorations in.

Don't forget, you'll need to rebake the navmesh, setting anything that doesn't move as navigation static. A good starting point is to set everything that doesn't move static and then go through and add the walkable bits.

More challenges below, less this time, part of the challenge in this one is just rearranging the assets and models to look nice.

# Challenges

### Challenge 1

Add a second enemy type (or for the second enemy type) change it's models and animations to something different. (Hint: if you use the same animation names you can keep the code the same).

#### Challenge 2

Have different blood effects for the different monsters; tweak the colour volume and amount of blood

#### Challenge 3

There's a second idle animation for the vampire, add it in and have them randomly play one or the other.