

CI606 Virtual Reality Systems

Gabriel Kelly

Table of contents

Table of contents.....	2
Section 1: Introduction	3
1.1 Demo Idea	3
1.2 User Requirements	3
1.3 Motivation	4
1.4 Demo Video	4
Section 2: Implementation	5
2.1 Setting Up the VR system	5
2.2 State machine.....	6
2.2.1 Idle State Implementation.....	8
2.2.2 Walking State Implementation	9
2.2.3 Chasing and Flying State Implementation	12
2.3 Weapons	14
2.3.1 Projectile Weapon.....	14
2.3.2 Ray cast Weapon.....	15
Section 3: Testing	17
Section 4: Finished Project.....	20
4.1 Conclusion	20
References	21
Websites	21
Assets used.....	22

Section 1: Introduction

1.1 Demo Idea

My main idea was to create a Demo game set at Brighton Beach where naturally spawning seagulls/birds would patrol the beach and when the player got close to them, they would start to chase the player and steal the player's chips, which they run away with. The player can disperse the birds by picking up a projectile weapon and shooting them with it.

This idea was generated because I knew in this project, I would like to explore a few features such as:

- Modelling, rigging, and animating to be used in Unity
- Creating a State Machine to control AI
- Exploring how AI animation can traverse the game world
- Overall set-up of an XR system in Unity
- XR Grab Interactions with interactable events
- The use of ray casting with a VR system

1.2 User Requirements

Functional Requirements

- The user should be able to play the demo by using a Virtual Reality system (Oculus Quest)
- The user should be able to interact with the world by using the Quest controllers
- The user should be able to look around the world by using the headset screen
- The user should be able to teleport around the allocated areas of the game world
- The user should be able to interact with the AI enemies

Non-functional requirements

- The AI should be able to spawn in the world every few seconds
- The AI should be able to move around any type of environment in the game world
- The AI should chase the player controller when they get near each other
- The AI should flee when they collide with the player
- The AI should be destroyed when colliding with projectiles

1.3 Motivation

I wanted to use this project as an opportunity to explore the process of modelling rigging and animating a character. I already have some experience in 3D modelling but never tried to use them in a Game Engine. I specifically wanted to see how my animated character would react to the environment such as flat areas, hills, or steps. This gives me proof of concept that I can compare back to in my Final Year Project where I use Procedural Animation to animate characters using an algorithm rather than traditional keyframe animation.

VR has always been something I've been interested in, especially how VR controllers can interact with objects around the game world. When I started, I didn't want to create guns for the player to interact with as I thought this was too violent for the theme of my game but after doing more research into XR interactions it gave more opportunities to use a gun/projectile weapon to show proof of a wide variety of skills.

1.4 Demo Video

<https://youtu.be/pEEatWcaLD8>

Section 2: Implementation

2. Describe the implementation along with any relevant aspects regarding the Virtual Reality system (e.g. how do you achieve your cameras, transformations, engagement with content)

2.1 Setting Up the VR system

My biggest concern when starting the project was the complexity of the VR system and how it might pair up with my system, but this was simpler than I thought. I can access my unity application by using a link cable to access my computer on the headset and whenever the unity game is in play mode it automatically uploads the footage of the game onto the headset.

At this moment unity can show up on the VR system however the camera and controllers have not been set up to use yet. I can do this by using the XR plugin management built-in VR and downloading the OpenXR and Oculus plugins which allows me to collectively create a VR system using OpenXR and more importantly for an oculus setup which is the headset I am using. For unity, I will be using a single-pass instance for rendering the application which allows me to render the entire scene once as it is quite small which also reduces the CPU usage significantly. I chose single-pass rendering over multi-pass rendering as I knew my project wouldn't be visually detailed so there was no need for richer imagery that multi-pass enabled and the usage on the CPU would not be worth it due to the small size of my Project.

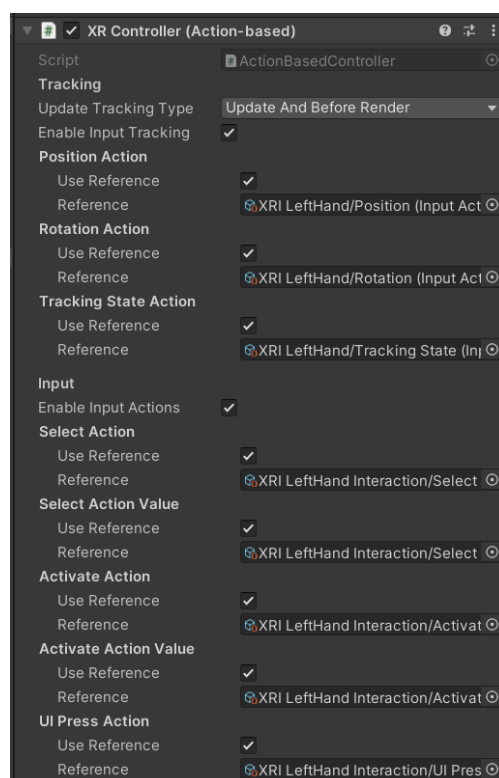


Figure 2.1 – Controller Input Setup Screenshot

With the headset rendering setup, there was still no way of using the controllers, this creates some problems for compatibility especially if I was selling this game commercially. When setting up the controllers in Unity, shown in figure 2.1, I need to set up every input corresponding with a certain button on the controller. The main problem here is I have only set up the controls with the quest 2 controllers, if someone wanted to play this game with a different VR system they would not be able to, as the controls have not been synced with that particular controller and this creates a big problem in the commercial aspect of the project. To fix this I would need to set up the controls for every system out there, which due to the wide variety of VR controllers would take a very long time. As VR systems are still a developing technology, more and more different controllers will be made, which may not have plugins fitted, which prevents them from being set up, meaning the developer must wait for Unity to update their XR plugins.

2.2 State machine

Before setting up my state machine I created a 3D model and set up an Armature shown in figure 2.2 that allows me to create 4 animation states: Idle, walking, attacking and flying state. The concept behind this was to change the animation of the bird depending on the action it is doing, for example it will be in its idle state when standing still and transfer to the walking state when moving slowly around the game world.

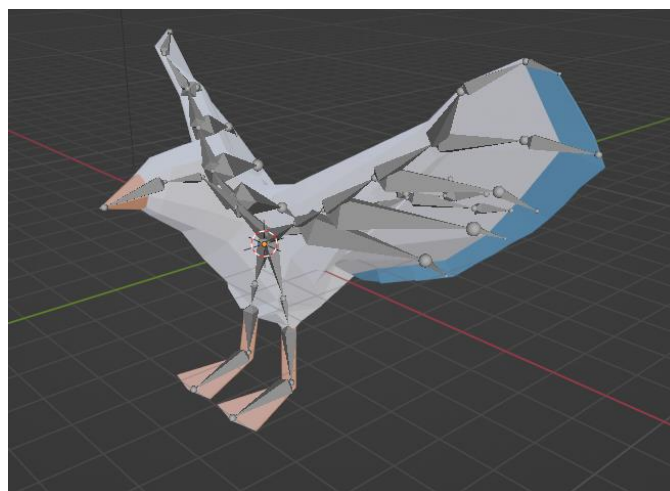


Figure 2.2 – Model and rigged screenshot

When modelling my character I needed to decide the level of detail of the object, if I made it too detailed it would look nice and realistic, however will use thousands of triangles that need to be rendered every time this object is spawned into the world, which will significantly slow down the runtime and performance of the project. So instead, I opted to make a more stylised model in a low poly style, which uses significantly fewer triangles but still effectively showing what the object is.

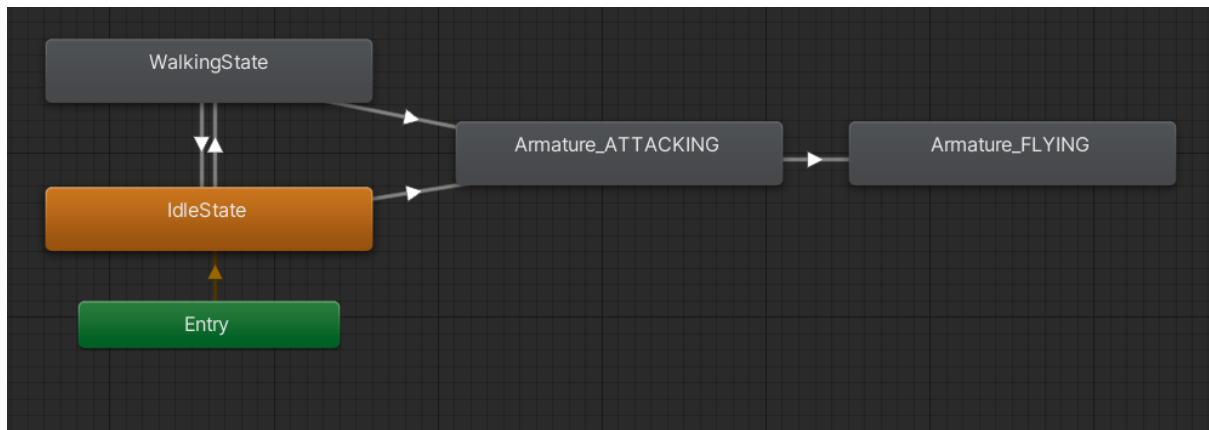


Figure 2.3 – Animator transition of states

After exporting the fbx files from blender into Unity I had my rigged model and animations ready to use. In the animator page of Unity, I have set up the transitions between the animation states shown in figure 2.3. Each transition is activated under certain conditions shown in figure 2.4 where the idle state transitions into the walking state under the Boolean condition that is patrolling equals true.

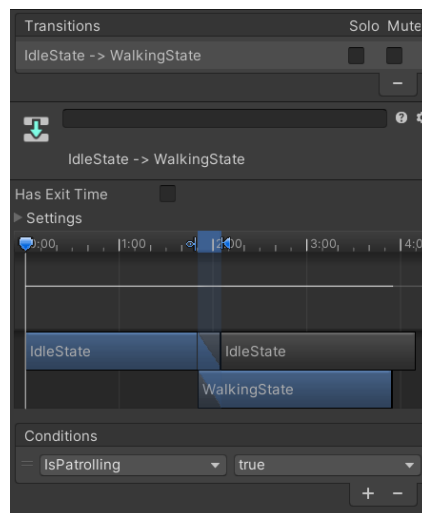
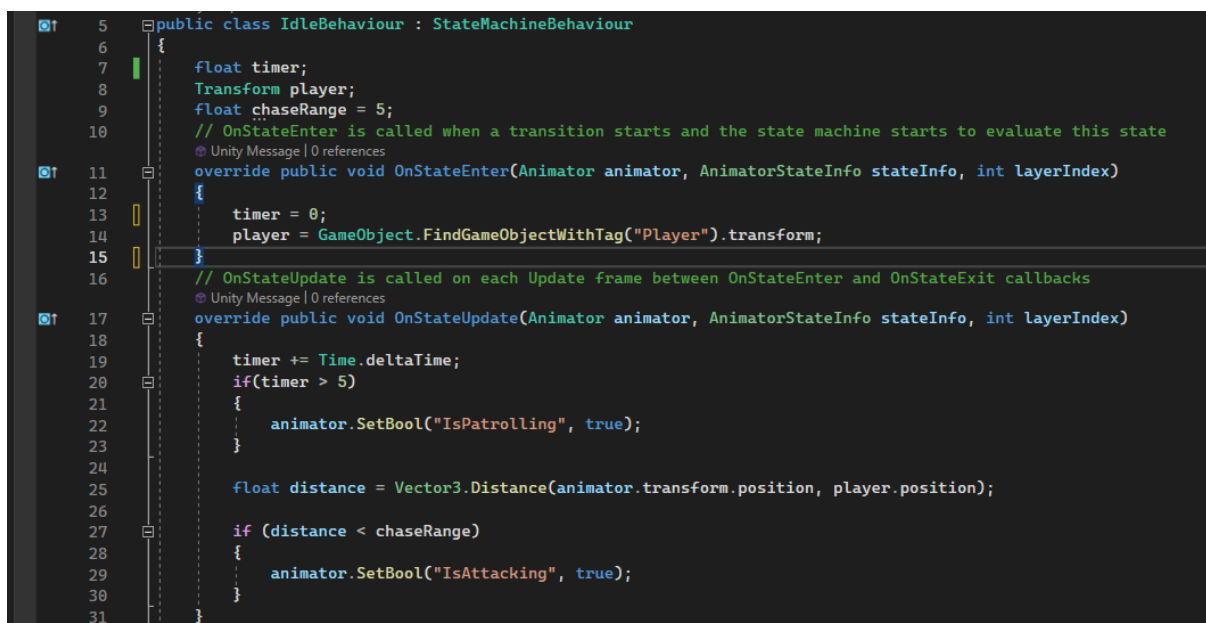


Figure 2.4 – idle to walking transition screenshot

2.2.1 Idle State Implementation

I have created my state machine using the inbuilt State machine behaviour scripts unity provides. These scripts are attached to their corresponding states in the animator shown in figure 2.3. In figure 2.5 this is activated whenever a new bird is instantiated into the world. It uses a simple timer activated with `Time.deltaTime` to transition to a new state after 5 seconds into the patrol state, which sets the Boolean “isPatrolling” true that activates the walking state script shown in figure 2.6. When the new Boolean is set to true, it deactivates the idle state as the animation is not running anymore but starts the walking animation state that activates the new script and behaviour.

I have also allowed it, so this state can transition to multiple states, it finds the vector 3 positions of both the animator, which is what’s being animated and the bird, which is how it knows what object it needs to allocate the code towards. It also finds the position of the player and compares them to find the distance between the player and the bird, so when the bird is close to the player it sets the Boolean `isAttacking` true, which changes the behaviour state and animation.

A screenshot of a C# script named `IdleBehaviour` in a code editor. The script inherits from `StateMachineBehaviour`. It contains two main methods: `OnStateEnter` and `OnStateUpdate`. `OnStateEnter` initializes a `timer` to 0 and finds the `player` transform. `OnStateUpdate` increments the `timer` by `Time.deltaTime` each frame. When the timer reaches 5 seconds, it sets the `IsPatrolling` boolean to true. Additionally, it calculates the distance between the animator's position and the player's position. If the distance is less than a `chaseRange` of 5, it sets the `IsAttacking` boolean to true.

```
5 public class IdleBehaviour : StateMachineBehaviour
6 {
7     float timer;
8     Transform player;
9     float chaseRange = 5;
10    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
11    // Unity Message | 0 references
12    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
13    {
14        timer = 0;
15        player = GameObject.FindGameObjectWithTag("Player").transform;
16    }
17    // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
18    // Unity Message | 0 references
19    override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
20    {
21        timer += Time.deltaTime;
22        if(timer > 5)
23        {
24            animator.SetBool("IsPatrolling", true);
25        }
26
27        float distance = Vector3.Distance(animator.transform.position, player.position);
28
29        if (distance < chaseRange)
30        {
31            animator.SetBool("IsAttacking", true);
32        }
33    }
34 }
```

Figure 2.5 – idle state code screenshot

2.2.2 Walking State Implementation

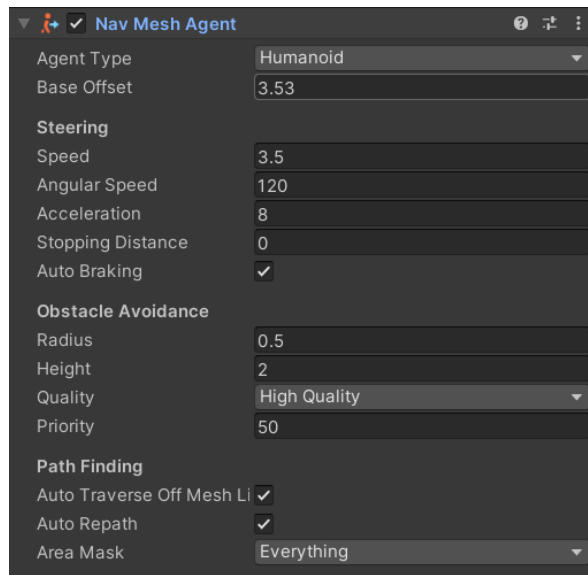
When the Boolean `ispatrolling` is set to true, this activates the scripts shown in figure 2.6 that shows how the bird moves around the world. Around the game world are simple spheres placed everywhere, which have been allocated to the tag "waypointGroup". In figure 2.6 the code finds where all the locations of all these waypoints are in the world and chooses one for the bird to move towards, when they get to the waypoint it sets a new destination to a different waypoint which creates the illusion that the enemy is moving around freely.

```
7 public class WalkingBehaviour : StateMachineBehaviour
8 {
9     List<Transform> waypoints = new List<Transform>();
10    NavMeshAgent agent;
11    Transform player;
12    float chaseRange = 7;
13
14    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
15    // Unity Message | 0 references
16    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
17    {
18        Transform waypointsObject = GameObject.FindGameObjectWithTag("WaypointGroup").transform;
19        foreach (Transform t in waypointsObject)
20        {
21            waypoints.Add(t);
22        }
23        agent = animator.GetComponent<NavMeshAgent>();
24        agent.SetDestination(waypoints[0].position);
25        player = GameObject.FindGameObjectWithTag("Player").transform;
```

2.6 – walking state code screenshot

```
26 // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
27 // Unity Message | 0 references
28 override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
29 {
30     if(agent.remainingDistance <= agent.stoppingDistance)
31     {
32         agent.SetDestination(waypoints[Random.Range(0, waypoints.Count)].position);
33     }
34     float distance = Vector3.Distance(animator.transform.position, player.position);
35
36     if(distance < chaseRange)
37     {
38         animator.SetBool("IsAttacking", true);
39     }
40 }
41 // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
42 // Unity Message | 0 references
43 override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
44 {
45     agent.SetDestination(agent.transform.position);
46 }
```

2.7 – walking state code screenshot



2.8 – Nav mesh Agent screenshot

When this script was activated, I was surprised to see that nothing happens as the bird doesn't move towards the waypoints. This is because I had not setup a nav mesh area across the world shown in figure 2.9 I have selected all the areas I want the enemy to walk and baked the walkable area. So when a new nav mesh agent is put on this area it understands where it can move around. I was surprised to see how advanced my AI got from this simple implementation. It now understands:

- when there are obstacles in the way and can walk around them
- depending on the height of steps it can walk up different heights and jump off cliffs
- and, moves over, round and can negotiate sharp hills.

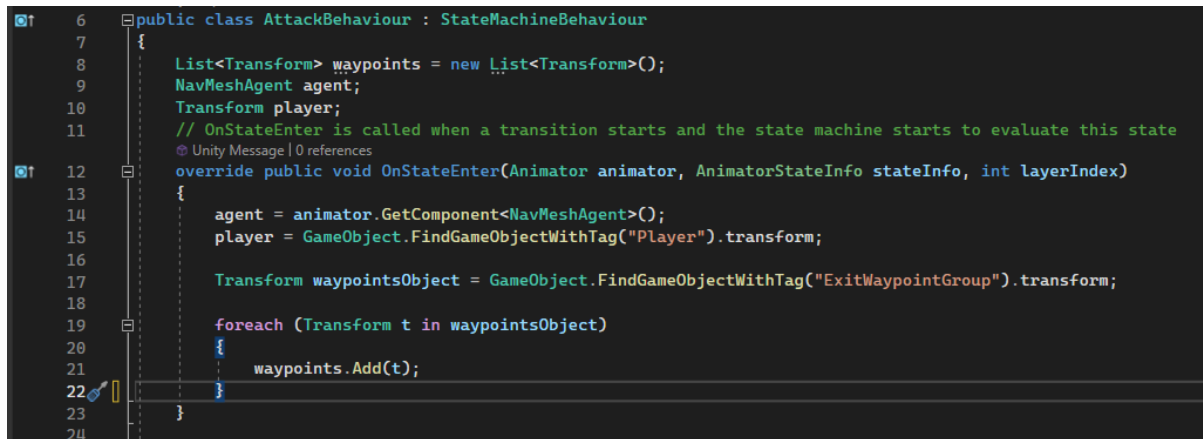
However, this does come with some visual problems, such as the nav mesh agent doesn't know a character is taking a step, so doesn't add a little bounce for realism, so the character does look like they are ice skating. The character also moves at the same speed no matter how steep a platform is.



2.9 – Nav mesh walkable area screenshot

2.2.3 Chasing and Flying State Implementation

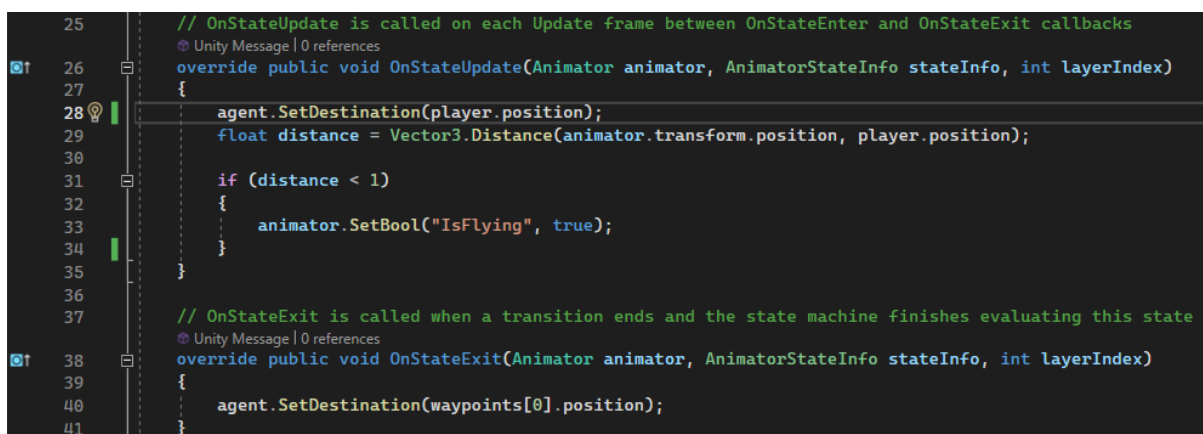
Attacking behaviour is quite simple where it gets the location of the player and sets the navmeshes destination to the player's position, which creates the illusion that it is chasing the player and whenever the enemy touches the player it activates the flying script, which works the same as the walking script where it moves towards one waypoint. However, when it gets there it doesn't move to a new waypoint and the enemy gets destroyed.



```
6 public class AttackBehaviour : StateMachineBehaviour
7 {
8     List<Transform> waypoints = new List<Transform>();
9     NavMeshAgent agent;
10    Transform player;
11    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
12    // Unity Message | 0 references
13    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
14    {
15        agent = animator.GetComponent<NavMeshAgent>();
16        player = GameObject.FindGameObjectWithTag("Player").transform;
17
18        Transform waypointsObject = GameObject.FindGameObjectWithTag("ExitWaypointGroup").transform;
19
20        foreach (Transform t in waypointsObject)
21        {
22            waypoints.Add(t);
23        }
24    }
```

Figure 2.10 - attacking state code screenshot

When the enemy is chasing the player, there's not a lot of indication to tell the player that this is happening, the only thing that happens is the animation changes and the enemy moves towards the player. Some ways this could enhance the player's engagement, is by adding a sound indication when chase is activated to alert the player that they are being chased. The use of the VR headset will work in my benefit because when the sound is played they are able to hear its position, so for example, if the enemy is to the right of the player the sound will be more dominate in the right side of the headset so when the player turns their head to the right they can see the enemy chasing them increasing engagement.



```
25 // OnStateUpdate is called on each Update frame between OnStateEnter and OnStateExit callbacks
26 // Unity Message | 0 references
27 override public void OnStateUpdate(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
28 {
29     agent.SetDestination(player.position);
30     float distance = Vector3.Distance(animator.transform.position, player.position);
31
32     if (distance < 1)
33     {
34         animator.SetBool("IsFlying", true);
35     }
36 }
37 // OnStateExit is called when a transition ends and the state machine finishes evaluating this state
38 // Unity Message | 0 references
39 override public void OnStateExit(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
40 {
41     agent.SetDestination(waypoints[0].position);
42 }
```

Figure 2.11 - attacking state code screenshot

The sound indications will work well for the game. However, it is not very accessible as some people playing may lack hearing or play with the sound off. This could be improved by using a visual tell to show when an enemy is chasing, such as an arrow above the enemy's head, which moves with the camera. When the player moves its head away from the enemy that is out of view the arrow can still be seen in the headset pointing towards the enemy maintaining player engagement.

For the flying state, I planned for it to move to a waypoint high in the sky to give the illusion the bird is flying away with the chips. However, because I used the nav mesh agent, it prevented me from doing this as there was not a walkable area in the sky making the nav mesh agent fix to the ground. One way I could fix this is when flying state is initialised it disables the nav mesh so it is able to move away from the navigation area into the sky. This will successfully make the enemy move in the sky. However this comes with its own new problems such as the enemy will not be facing the direction it is travelling in as the nav mesh agent rotates along the x-axis to face the moving direction. Also, if there are any obstacles in the sky the enemy wouldn't know it's there so would go straight through them.

```
7 public class FlyingBehaviour : StateMachineBehaviour
8 {
9     List<Transform> waypoints = new List<Transform>();
10    NavMeshAgent agent;
11
12    // OnStateEnter is called when a transition starts and the state machine starts to evaluate this state
13    // Unity Message | 0 references
14    override public void OnStateEnter(Animator animator, AnimatorStateInfo stateInfo, int layerIndex)
15    {
16        Transform waypointsObject = GameObject.FindGameObjectWithTag("ExitWaypointGroup").transform;
17
18        foreach (Transform t in waypointsObject)
19        {
20            waypoints.Add(t);
21        }
22
23        agent = animator.GetComponent<NavMeshAgent>();
24        agent.SetDestination(waypoints[0].position);
25    }
```

Figure 2.12- flying state code screenshot

2.3 Weapons

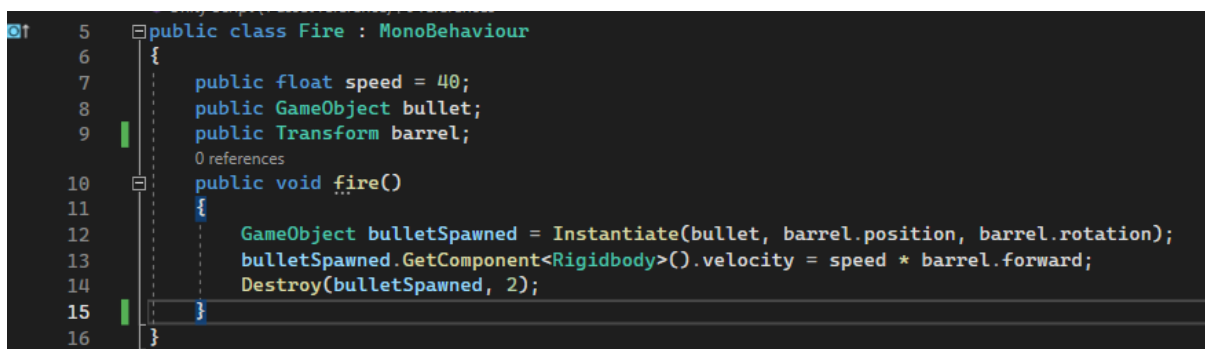
When interacting with objects in the world I used a ray interactor for the controller, which is a ray casted line from the position of the controller hand that I am able to point and, in this example, pick up items, which have an XR grab interactions script. However, I felt picking up items with the ray interactor was not very immersive, as you can interact with items as far away as possible. So instead, for picking up items in the right-hand controller I have used an XR direct interactor, which uses a sphere collider, where an object is able to be interacted when inside the sphere collider giving the illusion that you are picking something up.

To create a more immersive experience for the user I could have used the audio and haptic feedback implementation fitted with the XR grab interaction script, where on certain interactions I could play audio or vibrate the controller. In my case when the player first picks up the weapon it could play a cocking of a gun audio clip to signify it is ready and play a shot sound when the trigger was pulled. I did add haptic feedback for when the gun was fired, however this became very annoying to hold the controller as I was firing quite a lot so decided to turn it off instead. I believe a good place to use haptic feedback in a VR setting is when the player might be holding a much bigger object, creating the illusion that the object is heavy, or if the player is near its next goal the haptic feedback intensifies to show where the next goal is helping the player along the game.

When designing my game, I wanted to stay away from using guns/ projectile weapons as I thought they were way too violent for the theme of my game. VR systems are becoming more and more commercially available, widening the audience and especially to include under-18s, and games featuring guns as weapons are less likely to appeal. However, after doing more research I found creating a gun-like interaction allowed me to show a more diverse range of new skills.

2.3.1 Projectile Weapon

I found it very easy to interact with objects that are being held, by using the XR direct interaction script, shown in figure 2.14. I'm able to locate the object I want to activate when holding it by using the script shown in figure 2.13. So whenever I press the trigger on the right control it activates the script shown in figure 2.13 to fire a bullet out of the gun that I am holding. This works perfectly for what I am doing but there is another way to do it using code, by adding event listeners for each interaction, which adds a lot more versatility, for example there could be a 10% chance that the gun doesn't shoot when the trigger is activated creating the illusion that the gun is jammed.

A screenshot of a code editor showing a C# script named 'Fire' that inherits from 'MonoBehaviour'. The script contains a float variable 'speed' set to 40, and two public variables: 'bullet' of type 'GameObject' and 'barrel' of type 'Transform'. A 'fire()' method is defined, which instantiates a new 'bullet' object at the 'barrel's position and rotation, sets its velocity to 'speed * barrel.forward', and destroys it after 2 seconds.

```
5 public class Fire : MonoBehaviour
6 {
7     public float speed = 40;
8     public GameObject bullet;
9     public Transform barrel;
10
11     public void fire()
12     {
13         GameObject bulletSpawned = Instantiate(bullet, barrel.position, barrel.rotation);
14         bulletSpawned.GetComponent<Rigidbody>().velocity = speed * barrel.forward;
15         Destroy(bulletSpawned, 2);
16     }
17 }
```

Figure 2.13 – Fire projectile code Screenshot

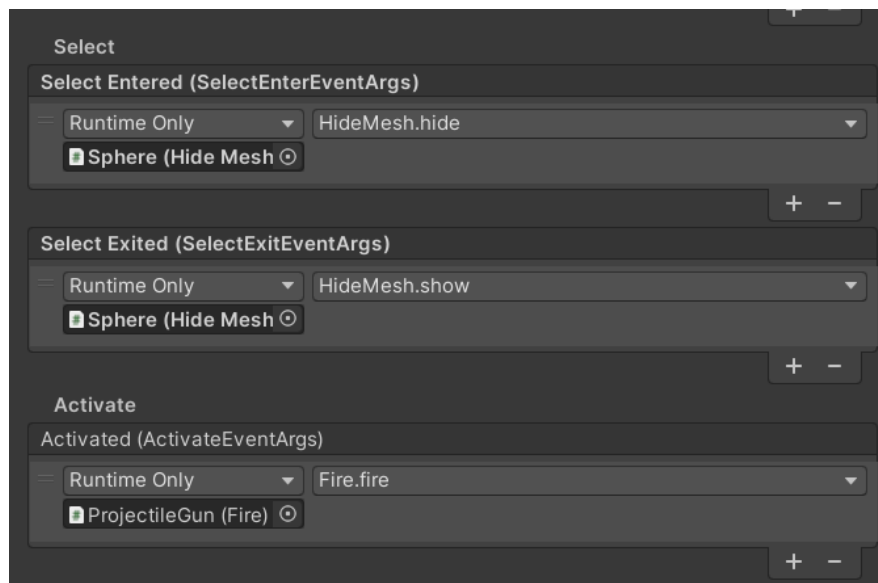


Figure 2.14 – XR direct interactions script screenshot

2.3.2 Ray cast Weapon

When starting this project, I knew ray casts are very important for creating a successful VR system, for example my left controller for the game uses a XR ray interactor, which allows it to interact with anything the ray touches and in my case when the ray is pointed at the ground and the trigger is pressed the player will be teleported to the ray's collision point. I have used rays again to create a similar weapon, which works like the projectile weapon but instead of shooting out a projectile it shoots out a ray. When the script in figure 2.15 is activated, the ray is shot from the start of the barrel locations in a forward direction across an infinite range and if it hits the target layer, which is the one the enemies are on, it activates the ray, killing the enemy and adding to the score.

```

5 public class RayCastFire : MonoBehaviour
6 {
7     public Transform barrel;
8     public LayerMask targetLayer;
9
10    LineRenderer laserLine;
11
12    Unity Message | 0 references
13    private void Awake()
14    {
15        laserLine = GetComponent<LineRenderer>();
16    }
17    0 references
18    public void fire()
19    {
20        RaycastHit hit;
21        laserLine.SetPosition(0, barrel.position);
22        if(Physics.Raycast(barrel.position, barrel.transform.forward, out hit, Mathf.Infinity, targetLayer))
23        {
24            laserLine.SetPosition(1, hit.point);
25            Destroy(hit.transform.gameObject);
26            ScoreManager.Instance.Addkills();
27        }
28    }

```

Figure 2.15 – ray cast shoot code example

The main problem I had with this is when fire is being activated the ray was not showing up. However, I knew it was shooting an invisible ray because it was destroying the enemy. This is when I realised, I needed to add a line renderer shown in figure 2.16, which I think is very interesting as this allows me to customise the ray that shoots out. For example, increase its width and shape, or make the ray bend around and change the colour like I did where I gave it a gradient from orange to red, which gave a more interesting look.

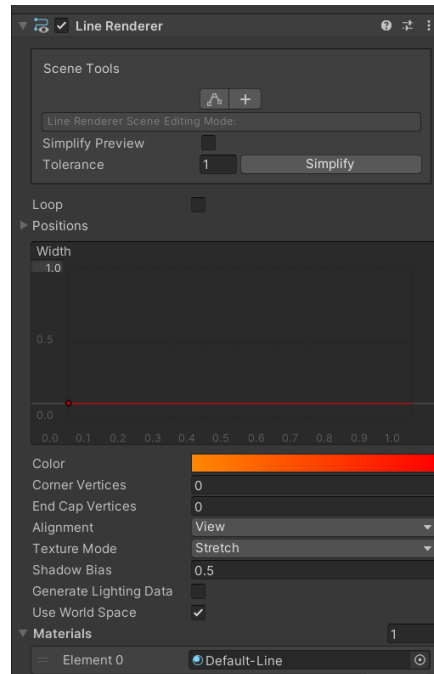


Figure 2.16 – Line renderer screenshot

Section 3: Testing

Present any testing you have done for your system, this can either be user testing or a testing technique such as white/black box testing.

What is being tested	What should happen	What did happen	Was it successful	How was it fixed
Functional Requirements				
The user should be able to play the demo by using a Virtual Reality system (Oculus Quest)	The user should be able to look around the game world and move controllers around when in a VR system	The player was able to look around the Gameworld however the camera was in a fixed position and the controls didn't appear	No, the problem was the XR player controller wasn't set up yet so the game was being shown from the stationary default camera	In the hierarchy, I needed to add an XR origin which added cameras and controller input with all the correct scripts which successfully enables VR in unity
The user should be able to interact with the world by using the Quest controllers	The user should be able to pick up objects around the world	The ray interactor successfully pickups up objects however the direct interactor doesn't allow for any interaction	No, when the direct interactor trigger is enabled near an object nothing happens when the player should be able to pick up the object	The problem was with the direct interactor I needed to add a collider to it so that when the collider touched an interactable object, and the trigger is pushed it should pick up the object
The user should be able to look around the world by using the headset screen	When the game is being played in the headset the player should be able to view the world using 3 degrees of freedom	The user can look around the entire world with 3 degrees of freedom	Yes, it does work however was using a continuous turn provider script which allowed me to look around using the trigger as well which made playing the game very nauseous.	Easy fix by taking off the continuous turn provider script and just allowing the user to view the world by turning their head instead.
The user should be able to teleport around the allocated areas of the game world	Using the ray interaction, the user should be able to teleport to the location where the ray is pointed increasing to 6 degrees of freedom	When the ray interaction was pointed at the floor and the trigger pushed it successfully teleported the player close to the location of the ray	Yes, however, I was not able to teleport to every location in the game world	The problem was that not all the flooring in the world had colliders so the ray just went straight through it when colliders were added I was able to teleport to every place in the game world

The user should be able to interact with the AI enemies	The player should be able to shoot the enemy which destroys the object	While holding the gun and the trigger is pulled it shoots out a projectile and when that projectile interacts with the AI, they are destroyed	Yes, the player was able to successfully interact with the object	To improve this further when the enemy is destroyed it could turn into a ragdoll and particle effects could appear where it gets hit to make the game more visually exciting
Non-functional requirements				
The AI should be able to spawn in the world every few seconds	Every 10 seconds at the spawn points the game should instantiate an AI object	At the spawn areas every 10 seconds it spawned a brand-new enemy AI	Yes, it does work but it is very predictable when a new enemy is spawned	To increase the difficulty of the game to decrease the spawn time as the game timer increases, this will make the spawn time less predictable and increase the number of enemies in the game at once
The AI should be able to move around any type of environment in the game world	When an enemy is in the game world It should be able to move around all areas of the world	When the enemy is spawned it starts walking around points around the entire world	Yes, however, the speed it moves is the same no matter the environment	To make the game more immersive depending on the angle the AI is walking on such as up a steep hill the AI should go to a slower place to show it is struggling to get up the hill
The AI should chase the player controller when they get near each other	When the AI gets close to the player controller the AI should change to its chase state and start moving towards the player	When the AI gets a few meters close to the player it changes to its chase state and changes its destination to be the player	Yes	
The AI should fly away when they collide with the player	When the AI collides with the player it should change to its flying state and run back to its spawn location	When the AI's distance equals the same as the player controller's distance it changes to its file state and	Yes	

		runs away from the player running back to its spawn location.		
The AI should be destroyed when colliding with projectiles	When the AI collides with the projectiles from the gun it should be destroyed	When the projectile collides with an ai object it destroys the AI and the projectile bullet	Yes, however just makes the AI disappear	To improve this when the AI is shot it could turn into its ragdoll mode to create a more exciting death

Section 4: Finished Project

Present and discuss the resulting system.

4.1 Conclusion

I believe the demo game I have created is a very good starting point for any framework VR application built in Unity, which shows a wide variety of skills used and understanding of theories, such as single and multi-pass rendering, 6 degrees of freedom, openXR etc.

The core framework for the base game is there, which can be developed into a commercial market with the user requirements in mind. Some improvements to the game could include:

- swap the violent weapons with comedic weapons such as ketchup
- more levels implemented with increase difficulties
- the inclusion of more ai with different patterns and states
- a way for the ai to fight back which decreases the players health, creating the ability to lose the game
- multiplayer where you can work together to fight of the ai or battle each other
- music which changes with the momentum of the game
- NPC which roams around the game world who are able to aid you which brings life to the game

I recognise that there is a discordance between the use of violent weapons in the game with the uplifting theme of seagulls and chips. This is an aspect which could be improved even further by including less contentious weaponry such as ketchup bottles firing, pickled onion grenades, ice cream able to freeze enemies, breadcrumbs used to distract AI or rock candy as a melee weapon.

References

Websites

Unity XR - Unity Technologies (2019). *Unity - Manual: VR overview*. [online] Unity3d.com. Available at: <https://docs.unity3d.com/Manual/VROverview.html>.

How to setup quest in unity - [www.youtube.com](https://www.youtube.com/watch?v=lh3-wh85dds&ab_channel=JonnosVr). (n.d.). *How to make a VR game in Unity - 2022 tutorial for Oculus Quest 2 (Part 1, The Basics)*. [online] Available at: https://www.youtube.com/watch?v=lh3-wh85dds&ab_channel=JonnosVr [Accessed 20 Nov. 2022].

Oculus development in unity - [developer.oculus.com](https://developer.oculus.com/documentation/unity/). (n.d.). *Oculus App Development in Unity | Oculus Developers*. [online] Available at: <https://developer.oculus.com/documentation/unity/>.

XR ray interactor - [docs.unity3d.com](https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-ray-interactor.html#:~:text=Interactor%20used%20for%20interacting%20with). (n.d.). *XR Ray Interactor | XR Interaction Toolkit | 2.0.4*. [online] Available at: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-ray-interactor.html#:~:text=Interactor%20used%20for%20interacting%20with> [Accessed 20 Nov. 2022].

XR direct interactor - [docs.unity3d.com](https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-direct-interactor.html). (n.d.). *XR Direct Interactor | XR Interaction Toolkit | 2.0.4*. [online] Available at: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-direct-interactor.html> [Accessed 20 Nov. 2022].

XR grab interactor - [docs.unity3d.com](https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-grab-interactable.html). (n.d.). *XR Grab Interactable | XR Interaction Toolkit | 2.0.4*. [online] Available at: <https://docs.unity3d.com/Packages/com.unity.xr.interaction.toolkit@2.0/manual/xr-grab-interactable.html> [Accessed 20 Nov. 2022].

XR origin - [docs.unity3d.com](https://docs.unity3d.com/Packages/com.unity.xr.core-utils@2.0/manual/xr-origin.html#:~:text=The%20XR%20Origin%20represents%20the). (n.d.). *XR Origin | XR Core Utilities | 2.0.1*. [online] Available at: <https://docs.unity3d.com/Packages/com.unity.xr.core-utils@2.0/manual/xr-origin.html#:~:text=The%20XR%20Origin%20represents%20the> [Accessed 20 Nov. 2022].

Raycasting - Unity Technologies (2019). *Unity - Scripting API: Physics.Raycast*. [online] Unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Physics.Raycast.html>.

Animator - Technologies, U. (n.d.). *Unity - Scripting API: Animator*. [online] [docs.unity3d.com](https://docs.unity3d.com/ScriptReference/Animator.html). Available at: <https://docs.unity3d.com/ScriptReference/Animator.html>.

Exporting from blender to unity - [www.youtube.com](https://www.youtube.com/watch?v=ysl0qYq5p9w&list=WL&index=8&ab_channel=RoyalSkies). (n.d.). *Blender 2.8 Exporting FBXs to Unity 3D (In 60 Seconds!)*. [online] Available at: https://www.youtube.com/watch?v=ysl0qYq5p9w&list=WL&index=8&ab_channel=RoyalSkies [Accessed 28 Nov. 2022].

Line renderer - Technologies, U. (n.d.). *Unity - Manual: Line Renderer component*. [online] docs.unity3d.com. Available at: <https://docs.unity3d.com/Manual/class-LineRenderer.html> [Accessed 1 Dec. 2022].

Nav mesh agent - Technologies, U. (n.d.). *Unity - Scripting API: NavMeshAgent*. [online] docs.unity3d.com. Available at: <https://docs.unity3d.com/ScriptReference/Al.NavMeshAgent.html>.

Assets used

10 weapon pack - assetstore.unity.com. (n.d.). *10 Weapons Pack / 3D Weapons / Unity Asset Store*. [online] Available at: <https://assetstore.unity.com/packages/3d/props/weapons/10-weapons-pack-184260> [Accessed 12 Jan. 2023].

low poly water - assetstore.unity.com. (n.d.). *LowPoly Water / Particles/Effects / Unity Asset Store*. [online] Available at: <https://assetstore.unity.com/packages/tools/particles-effects/lowpoly-water-107563>.

Desert - assetstore.unity.com. (n.d.). *Desert - Low Poly Toon Battle Arena / Tower Defense Pack / 3D Environments / Unity Asset Store*. [online] Available at: <https://assetstore.unity.com/packages/3d/environments/desert-low-poly-toon-battle-arena-tower-defense-pack-124507> [Accessed 12 Jan. 2023].