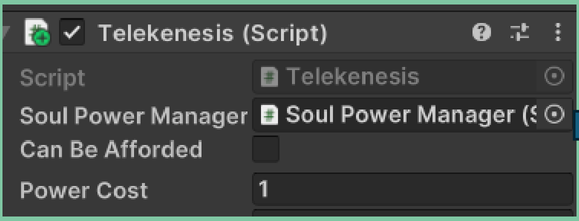
Sprint Two

Sprint One Recap:

During Sprint one, our main goal was to create a playable prototype of our game that contained the core functionality together. Much of my time was spent creating basic assets and scripting the players abilities which led to telekinesis and necromancy working. At the end of the sprint, we had a functional game with working basic mechanics. During our feedback sessions, we were told to narrow our scope and focus on the core gameplay loops we had.

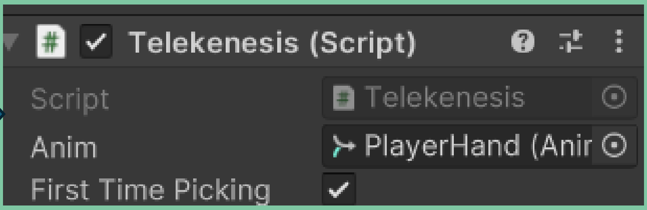
Feedback:

Initially, our response to the feedback was to have a meeting to discuss the changes we could make to the project. On my part, I began looking into ways the player could have their UI displayed mechanically and whether certain mechanics were needed at all. To this extent, we all decided it was best that the mana resource system should be removed in order to extend the playtime and fun our players experience.



Telekenesis (Script)

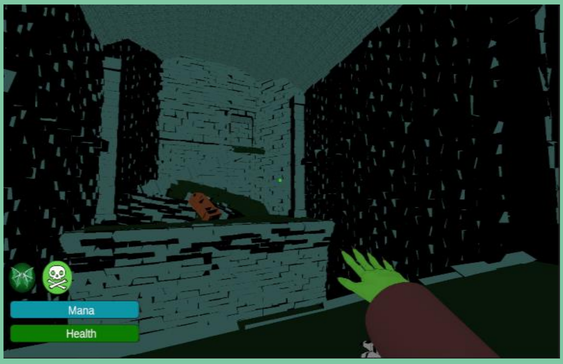
Script	Telekenesis
Soul Power Manager	Soul Power Manager (S
Can Be Afforded	☐
Power Cost	1

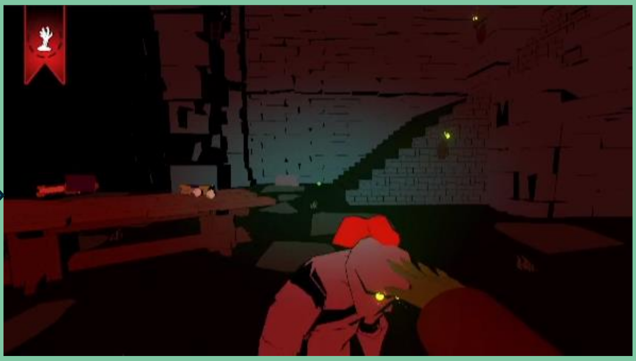


Telekenesis (Script)

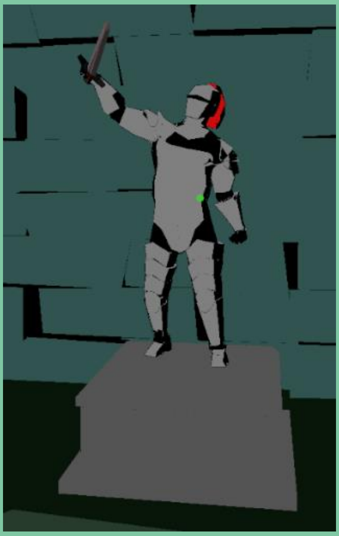
Script	Telekenesis
Anim	PlayerHand (Anir
First Time Picking	✓

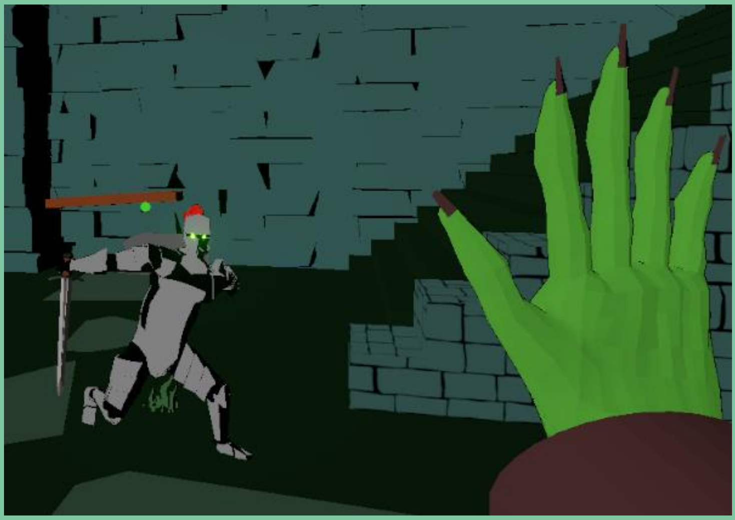
In practice, this meant removing the cost for telekinesis and replacing UI elements with a simple vignette to simplify it.





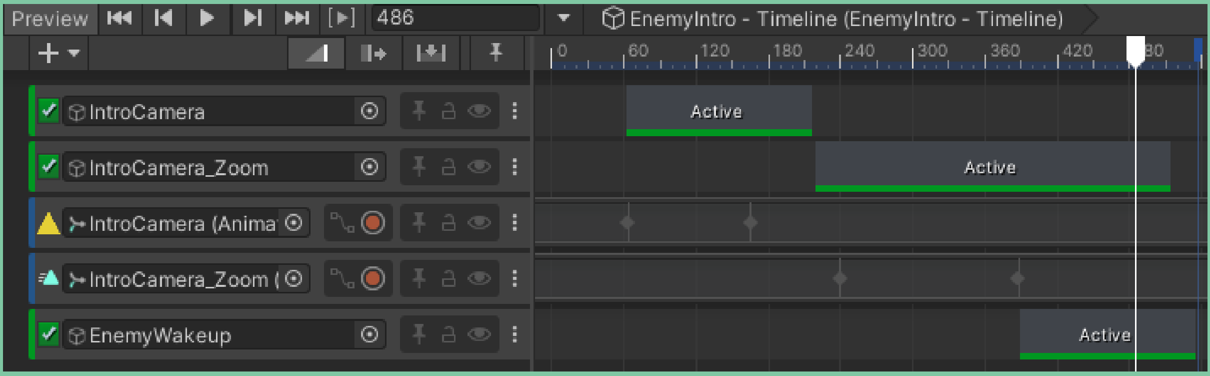
Besides the player, the enemies also underwent an overhaul which led to their animations all being switched out and their AI becoming more of a threat in smaller numbers. Their eyes were also altered to give them a more magical aesthetic and also act as a visual indicator as to whether they are “active” or not

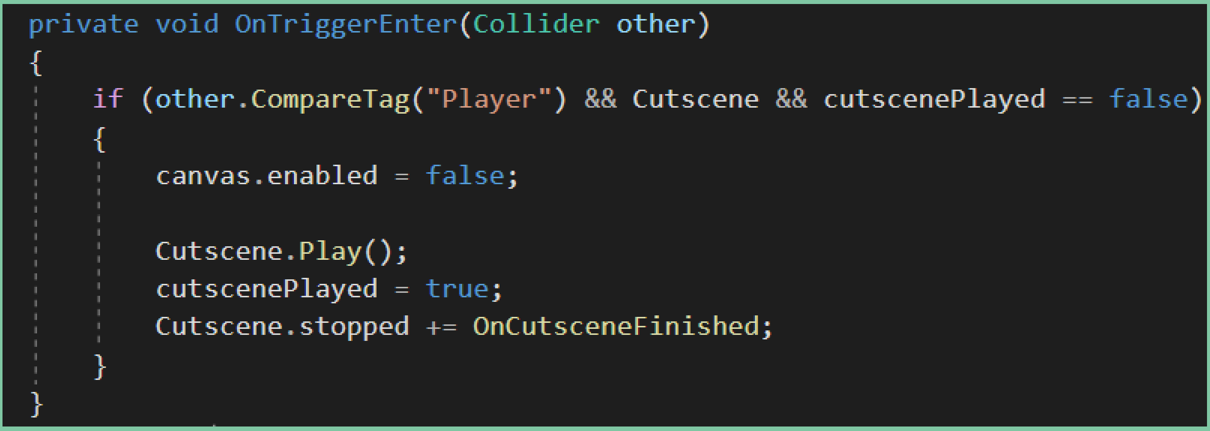




Inclusion of Cutscenes:

As a method for diegetic storytelling, cutscenes were introduced as a way to present enemies and objectives to the players. This meant I took time to create a "timeline manager" which would manage what variables had to be true before a cutscene would play.





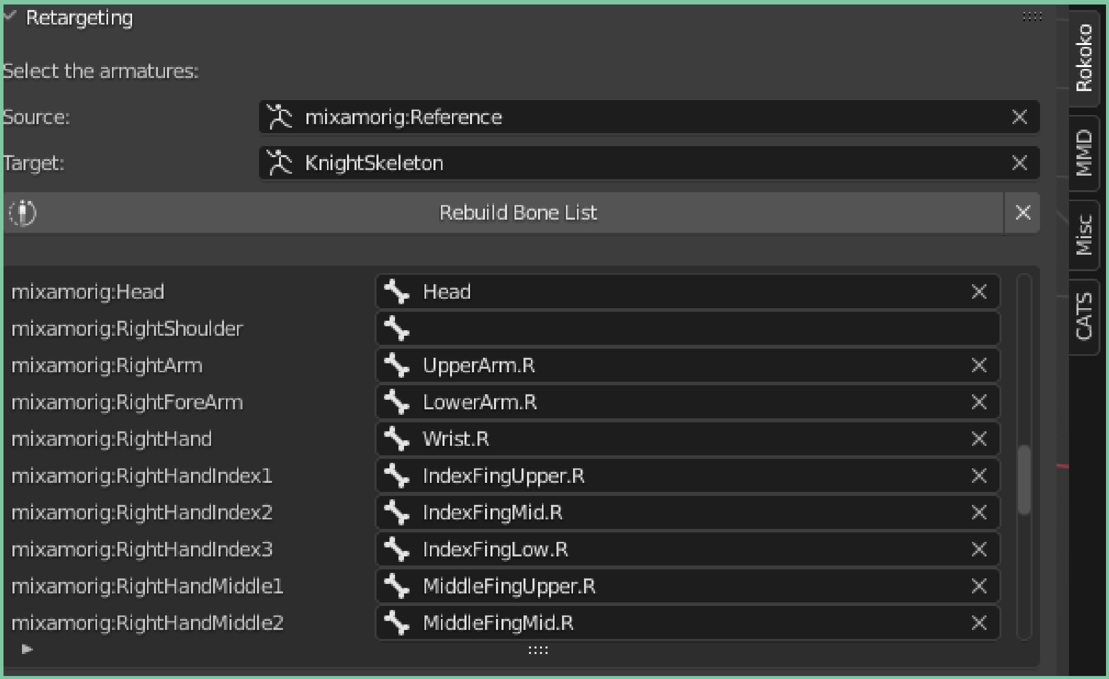
```
private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player") && Cutscene && cutscenePlayed == false)
    {
        canvas.enabled = false;

        Cutscene.Play();
        cutscenePlayed = true;
        Cutscene.stopped += OnCutsceneFinished;
    }
}
```

Through cutscenes, the player is better directed to where their attention should be held, and it also serves as a way to present lore. The most difficult part of this for me was getting timings correct and making sure everything ran correctly.

Inclusion of Motion Capture Animations:

As a way to enhance our gameplay and be much more efficient with the way we animate. This involved me putting the suit on, performing some basic movements, and then eventually retargeting the bones of the suit to the bones of our characters.

In order to accomplish this, I used a free add-on from Rokoko which would help match the bones up and get the animation working correctly.



Retargeting

Select the armatures:

Source: mixamorig:Reference

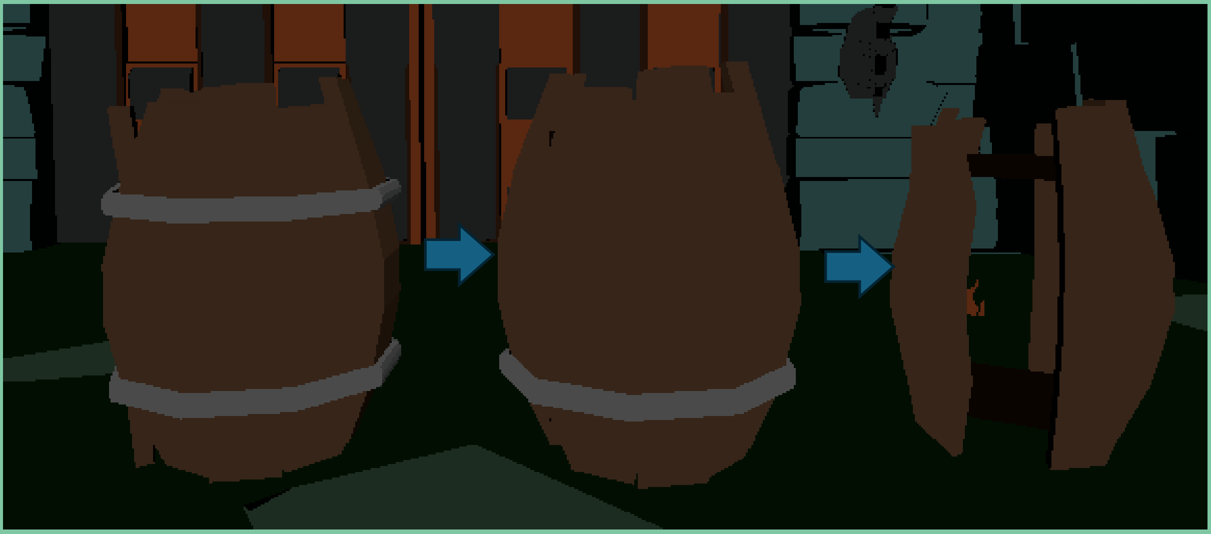
Target: KnightSkeleton

Rebuild Bone List

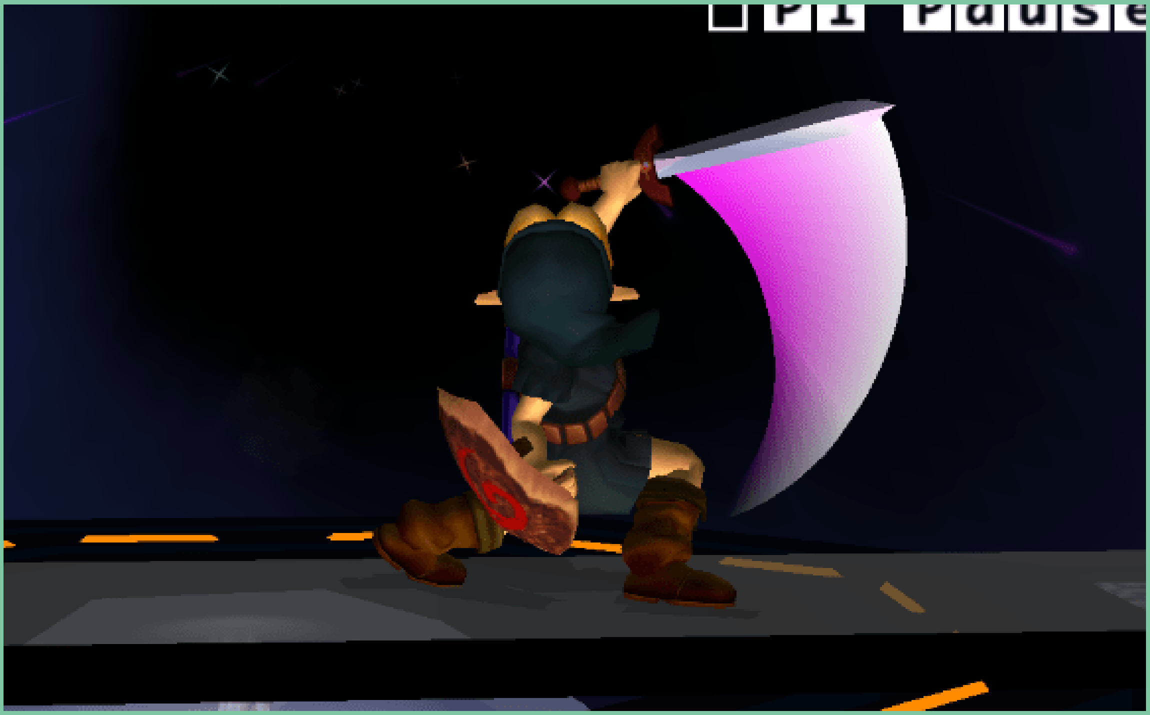
mixamorig:Head	Head
mixamorig:RightShoulder	
mixamorig:RightArm	UpperArm.R
mixamorig:RightForeArm	LowerArm.R
mixamorig:RightHand	Wrist.R
mixamorig:RightHandIndex1	IndexFingUpper.R
mixamorig:RightHandIndex2	IndexFingMid.R
mixamorig:RightHandIndex3	IndexFingLow.R
mixamorig:RightHandMiddle1	MiddleFingUpper.R
mixamorig:RightHandMiddle2	MiddleFingMid.R

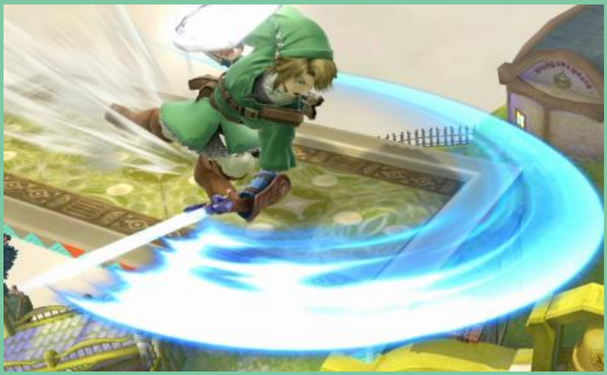
Sprint 2 Wrap up

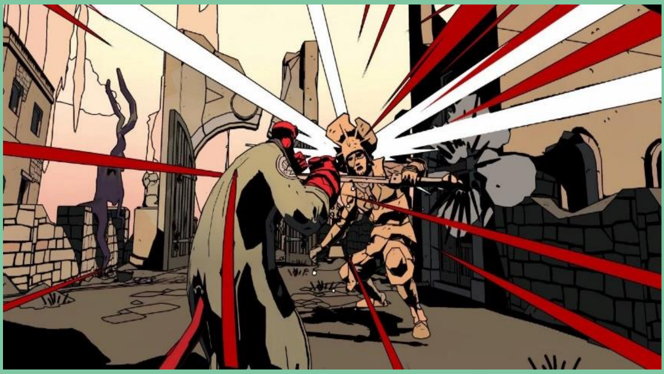
Towards the end of the second sprint, my focus was on the feedback the player got when they were throwing objects. To help with this, I created a script that had objects change mesh to a more “Damaged” state and also have objects “leak” parts of themselves which aided with showing health of objects.



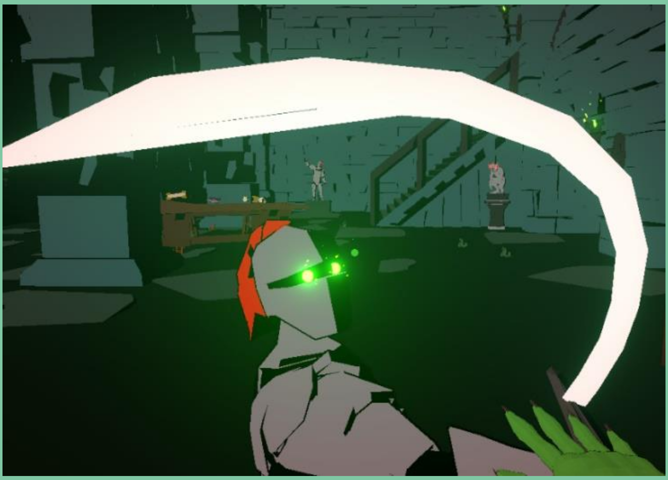
Alongside this, I also took a look into how I could make enemy attacks fit more into the theme. To do this, I researched how games such use trail lines as a method of feedback for the player. Specifically, I used Super Smash Bro’s and Hellboy Web of Wyrd as reference points

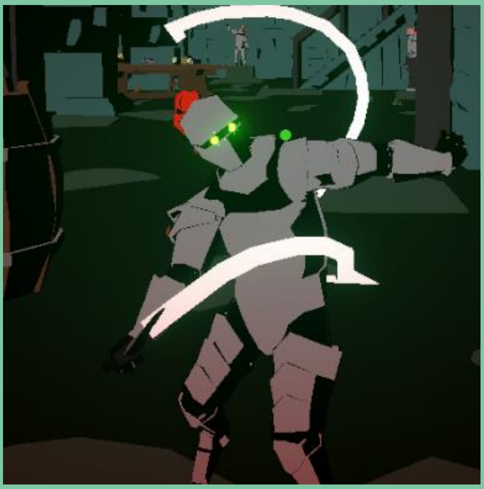




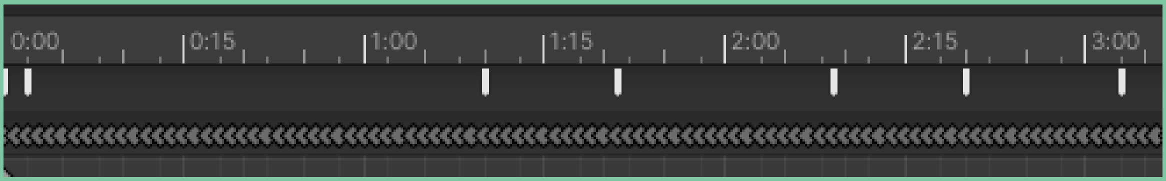


After this research, I adapted the style onto my enemy character attacks. This both give the player feedback for when the enemy is attacking and also gave them an idea of how far they need to back up when the enemy attacks.



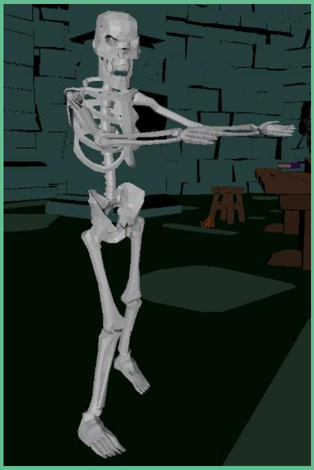


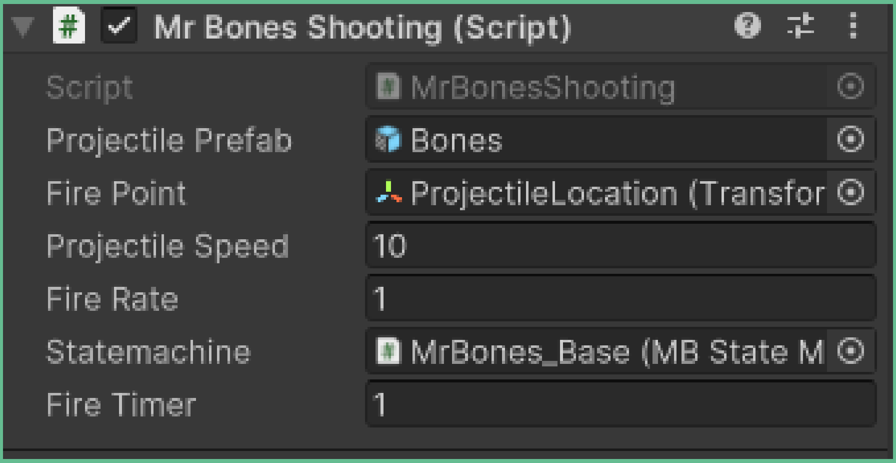
much of it was done by toggling on and off the actual trail line during the animation. This took a lot of time as each movement had to be correctly synced to when the trail should appear and disappear.



When reaching the end of sprint 2, it was heavily debated whether we should include a second enemy type or not to keep gameplay engaging. This was brought up by play testers as they found that elements of the game were too easy when facing a single ranged enemy type.

To test this out, I created the brain and framework for a ranged enemy which used an incomplete skeleton model.

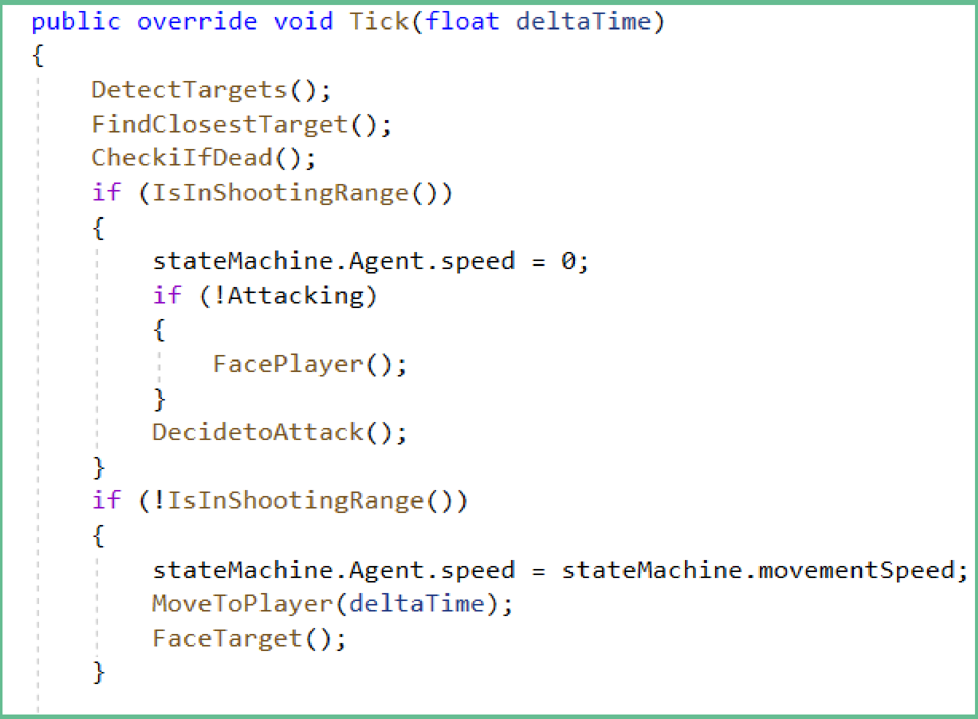




Mr Bones Shooting (Script)

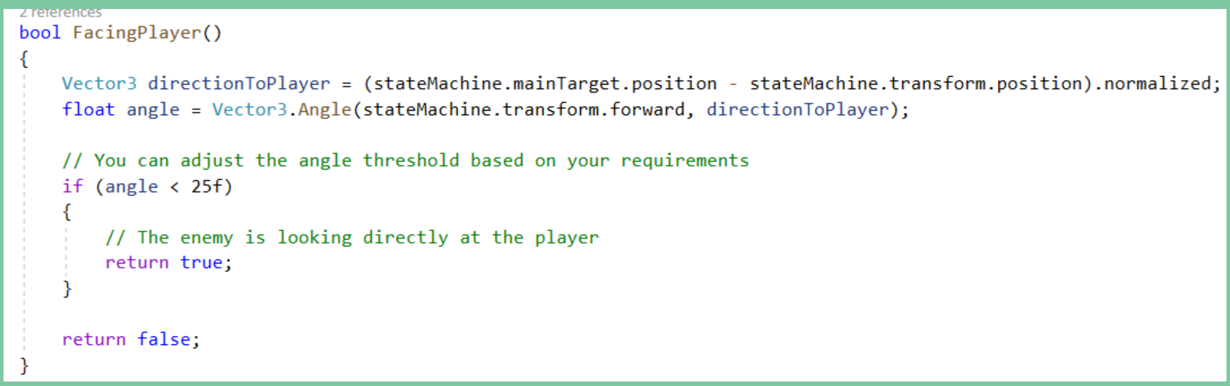
Script	MrBonesShooting
Projectile Prefab	Bones
Fire Point	ProjectileLocation (Transfor
Projectile Speed	10
Fire Rate	1
Statemachine	MrBones_Base (MB State M
Fire Timer	1

Dubbed as “Mr Bones”, He shared a similar framework as the knight but with the inclusion of a few new states. One of these states was used to refer to this “Shooting” script which handled all the values including what projectile he would throw.



```
public override void Tick(float deltaTime)
{
    DetectTargets();
    FindClosestTarget();
    CheckiIfDead();
    if (IsInShootingRange())
    {
        stateMachine.Agent.speed = 0;
        if (!Attacking)
        {
            FacePlayer();
        }
        DecidetoAttack();
    }
    if (!IsInShootingRange())
    {
        stateMachine.Agent.speed = stateMachine.movementSpeed;
        MoveToPlayer(deltaTime);
        FaceTarget();
    }
```

Much of the scripting involved various checks to make sure the player was in view and making sure the enemy did not continue walking towards the player when they start attacking. This also meant I had to define a range for the skeleton so to keep balance within the game.



```
2 references
bool FacingPlayer()
{
    Vector3 directionToPlayer = (stateMachine.mainTarget.position - stateMachine.transform.position).normalized;
    float angle = Vector3.Angle(stateMachine.transform.forward, directionToPlayer);

    // You can adjust the angle threshold based on your requirements
    if (angle < 25f)
    {
        // The enemy is looking directly at the player
        return true;
    }

    return false;
}
```