Contents

Links

Video:

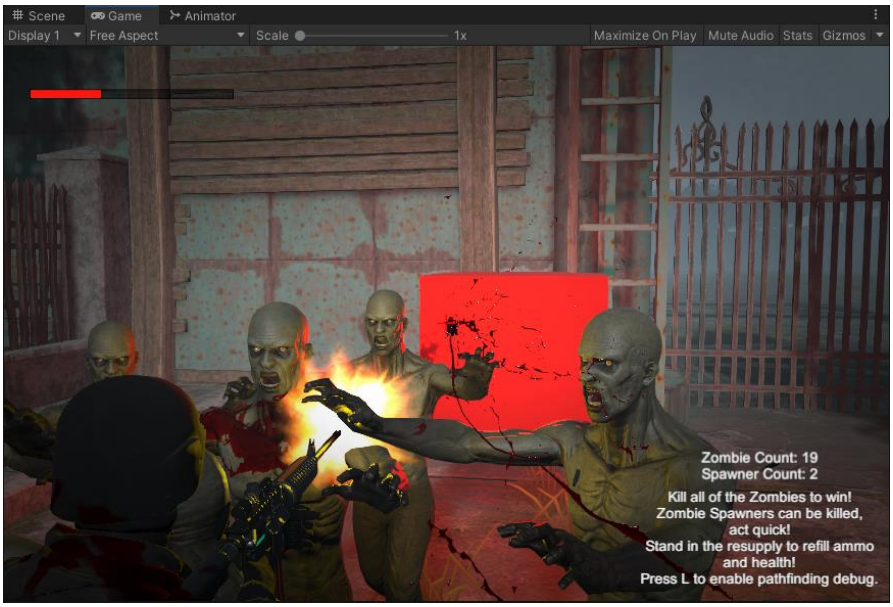
https://www.youtube.com/watch?v=inPjR9CPi7k

Zombie Shooter Project

Introduction

I was tasked with creating and extending a zombie shooter tutorial to demonstrate the implementation of simple AI techniques in Unity. I chose to utilize a simple third person shooter "wave"-style game to demonstrate different AI states and player interactions with AI.

The Game

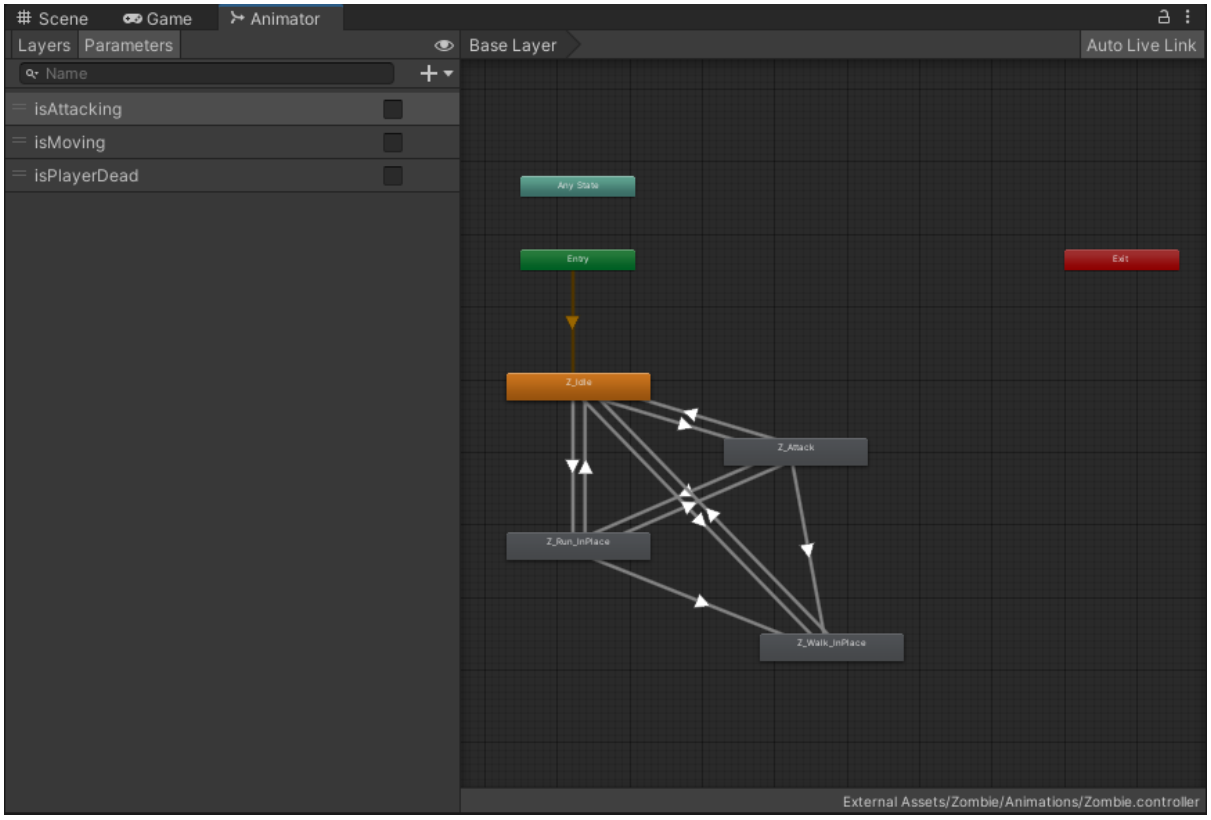


The player starts out in a house with a resupply cabinet and armed with a pistol and assault rifle. The street quickly fills with zombies coming out of mysterious red cubes. If the player manages to shoot the cube before being overrun by the horde, the light fades and it falls to the ground. The player should realise their objective is to kill both cubes as fast as possible before eliminating the rest of the zombies.

The zombies are slower than the player at running speed, but if running backwards and shooting they are quick to catch up. There is no collision with the zombies, however they 'scrape' the player, dealing very little amounts of damage which can quickly add up if running through a horde. If the zombies get close and begin attacking, a single zombie can quickly deal lethal amounts of damage. The game ends when all zombies are eliminated.

Method

The zombie has four animation states; idle, attack, run and walk. These are activates by three different Booleans within the script; “isAttacking”, “isMoving” and “isPlayerDead”.



When the player and zombie is alive, the script starts a co-routine which paths the nav-mesh agent to the player. The animation state is changed depending on the velocity of the agent. When the agent reaches a certain range from the player, they transfer to the attacking state, and rotate to face the player. The co-routine is called every second as it achieves a convincing effect without having to check the player’s position in every frame.

```
private void Update()
{
    if (dead == false && playerHealthScript.HealthValue != 0)
    {

        Debug.Log("Chasing Player!");

        StartCoroutine(SetTarget());

        if (agent.velocity.magnitude < 0.5f)
        {
            anim.SetBool("isMoving", false);
        }
        else
        {
            anim.SetBool("isMoving", true);
        }

        if (Vector3.Distance(target.transform.position, transform.position) < 20f)
        {
            anim.SetBool("isAttacking", true);
            Vector3 lookPos = target.position - transform.position;
            lookPos.y = 0;
            gameObject.transform.rotation = Quaternion.LookRotation(lookPos);
            //standStill = true;
        }
        else
        {
            anim.SetBool("isAttacking", false);
            //standStill = false;
        }

        /*
        if (standStill == false)
        {
            StartCoroutine(SetTarget());
        }
        else
        {
            StopCoroutine(SetTarget());
        }
        */
    }
```

When the player is dead, they stop pathing and transfer to the ‘wander’ state, involving a slow speed and walk animation. The wander function finds a random co-ordinate on the nav-mesh and paths the agent towards that destination.

```
else if (dead == false && playerHealthScript.HealthValue == 0)
{
    wandering = true;
    anim.SetBool("isPlayerDead", true);
    agent.speed = 1;
    if (agent.velocity.magnitude < 0.5f)
    {
        anim.SetBool("isMoving", false);
    }
    else
    {
        anim.SetBool("isMoving", true);
    }

    if (wandering == true)
    {
        Debug.Log("Wandering");
        if (agent.remainingDistance < 1f)
        {
            agent.SetDestination(Wander());
        }
    }
    else
    {
        StopCoroutine(SetTarget());
        Debug.Log("Player Dead!");
    }
```

```
Vector3 Wander()
{
    Debug.Log("Wander Initiated");
    Vector3 wanderPos = Random.insideUnitSphere * 40;
    wanderPos += transform.position;
    NavMeshHit navHit;
    Vector3 newPos = Vector3.zero;
    NavMesh.SamplePosition(wanderPos, out navHit, 40, NavMesh.AllAreas);
    newPos = navHit.position;
    return newPos;
}
```

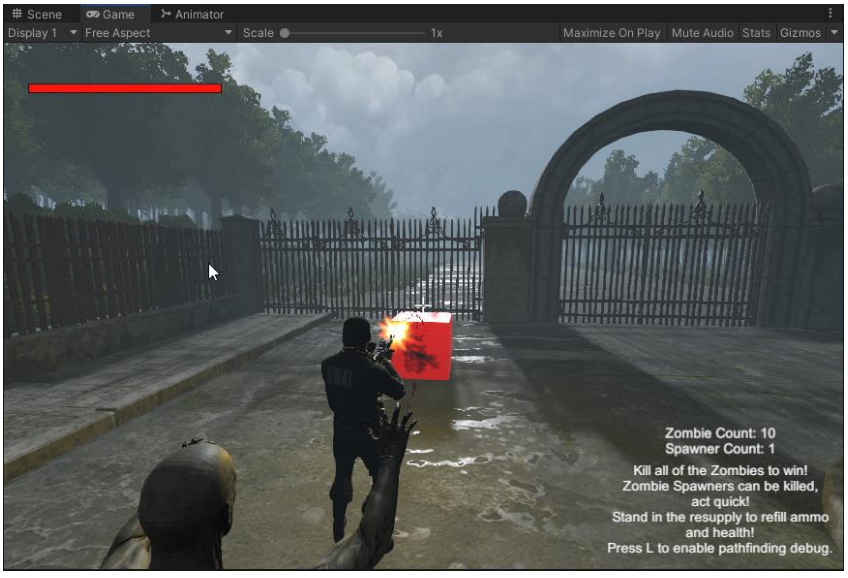
The zombie also has a health value, which responds to a hit event from the character controller. When this value reaches zero, the rigid-body stops being kinematic, the agent stops and the animator is disabled, triggering the ragdoll to take over.

```
public void Death()
{
    foreach(Rigidbody rb in rigidbodies)
    {
        rb.isKinematic = false;
    }
    anim.enabled = false;
    agent.isStopped = true;
    dead = true;
    if (deathCounted == false)
    {
        deathCounted = true;
        logicObject.ZombieCount--;
    }
}

1 reference
void OnImpact(float amount, Vector3 position, Vector3 forceDirection, GameObject attacker, object attackerObject, Collider hitCollider)
{
    Debug.Log("Shot");
    health -= 30;
}
```

The zombie spawners communicate with a ‘Game Logic’ game object which determines how many zombies can be alive at once. The difficulty of the game could be configured with the variables in the spawner script. Every time the timer reaches zero, a wave of zombies is spawned, and the next wave is incremented. If the next wave of zombies would go over the maximum zombie limit, the number is reduced so that the limit is upheld.

```
public int startNumber;
public GameObject objectToSpawn;

public int spawnIncrement = 0;
public int incrementAmount = 2;

public int health = 100;
public bool alive = true;
bool aliveCounted = false;
Light aliveLight;

public float timer = 5;
public float respawnTime = 10;

GameLogic logicObject;
```

```
void Update()
{
    if (health <= 0)
    {
        alive = false;
        if (aliveCounted == false)
        {
            logicObject.spawnerCount--;
            aliveCounted = true;
        }
    }

    if (alive == true)
    {
        if (timer > 0)
        {
            timer -= Time.deltaTime;
        }
        else
        {
            timer = respawnTime;

            if (logicObject.ZombieCount < logicObject.maxZombieCount)
            {
                int newSpawnNumber = 0;

                if (logicObject.ZombieCount + startNumber + spawnIncrement < logicObject.maxZombieCount)
                {
                    newSpawnNumber = startNumber + spawnIncrement;
                }
                else
                {
                    newSpawnNumber = logicObject.maxZombieCount - logicObject.ZombieCount;
                }

                for (int n = 0; n < newSpawnNumber; n++)
                {
                    GameObject newZombie = Instantiate(objectToSpawn, transform.position, Quaternion.identity);
                    logicObject.ZombieCount++;
                }
            }

            spawnIncrement += incrementAmount;
        }
    }
    else
    {
        aliveLight.enabled = false;
        gameObject.AddComponent<Rigidbody>();
        gameObject.GetComponent<Rigidbody>().mass = 20;
    }
}
```



```
void Update()
{
    zombieText.text = "Zombie Count: " + ZombieCount + "\nSpawner Count: " + spawnerCount;

    if (playerHealthScript.HealthValue == 0)
    {
        gameOverText.enabled = true;

        if(Input.GetKeyDown(KeyCode.R))
        {
            SceneManager.LoadScene("Main Scene");
        }
    }

    if (startTimer > 0)
    {
        startTimer = startTimer - Time.deltaTime;
    }
    else
    {
        if(ZombieCount == 0)
        {
            victoryText.enabled = true;

            if (Input.GetKeyDown(KeyCode.R))
            {
                SceneManager.LoadScene("Main Scene");
            }
        }
    }

}
```

Zombies cause damage to the player controller through the use of a collider on their hands. When running through non-attacking zombies, they deal 1 damage, whereas getting hit by an attacking zombie does 30.

```
private void OnTriggerEnter(Collider other)
{
    if(other.tag == "PlayerHitbox")
    {
        if (other.GetComponentInParent<CharacterHealth>() != null)
        {
            if(brain.dead == false)
            {
                if (anim.GetCurrentAnimatorStateInfo(0).IsName("Z_Attack"))
                {
                    Debug.Log("Attacking");
                    other.GetComponentInParent<CharacterHealth>().Damage(30);
                }
                else
                {
                    other.GetComponentInParent<CharacterHealth>().Damage(1);
                }
            }
        }
    }

}
```

External Assets

Environment Art:

https://assetstore.unity.com/packages/3d/environments/flooded-grounds-48529

Character Controller:

https://assetstore.unity.com/packages/tools/game-toolkits/ultimate-character-controller-99962

Zombie Model:

https://assetstore.unity.com/packages/3d/characters/humanoids/zombie-30232

Player Model:

https://www.mixamo.com/#/?page=1&query=police&type=Character

Blood Asset:

https://assetstore.unity.com/packages/vfx/particles/volumetric-blood-fluids-173863