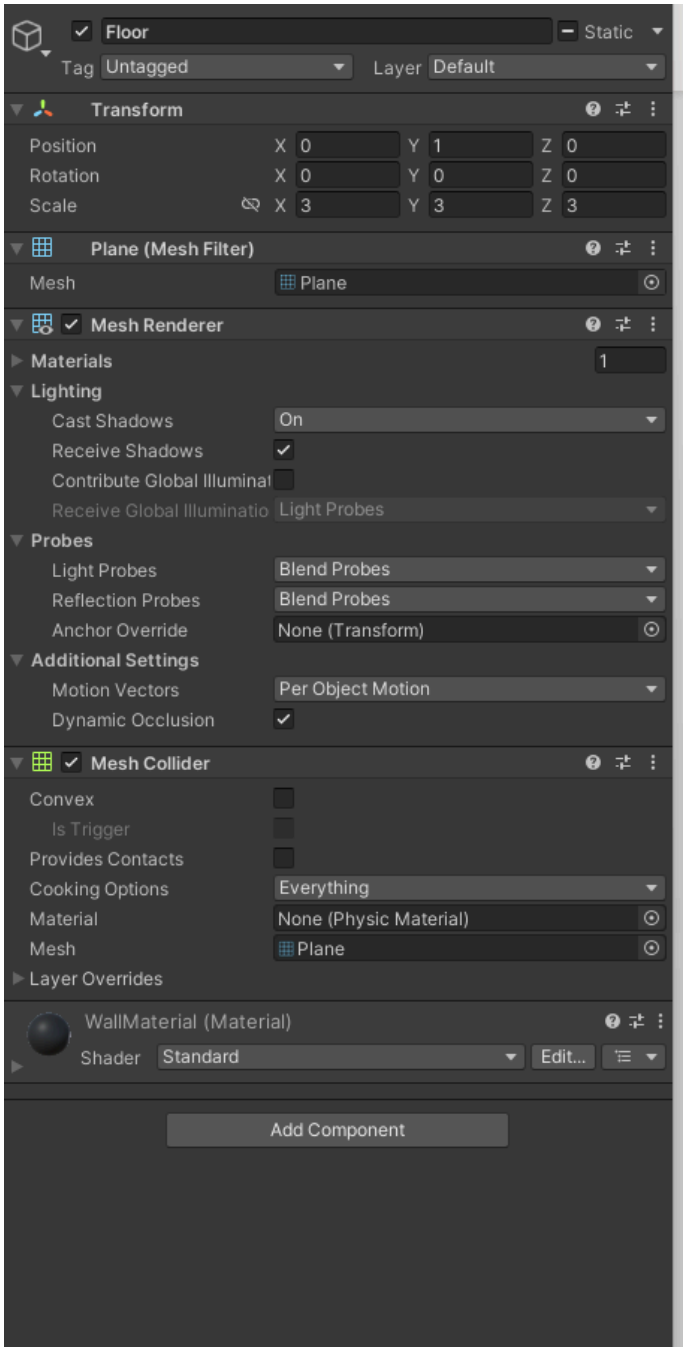
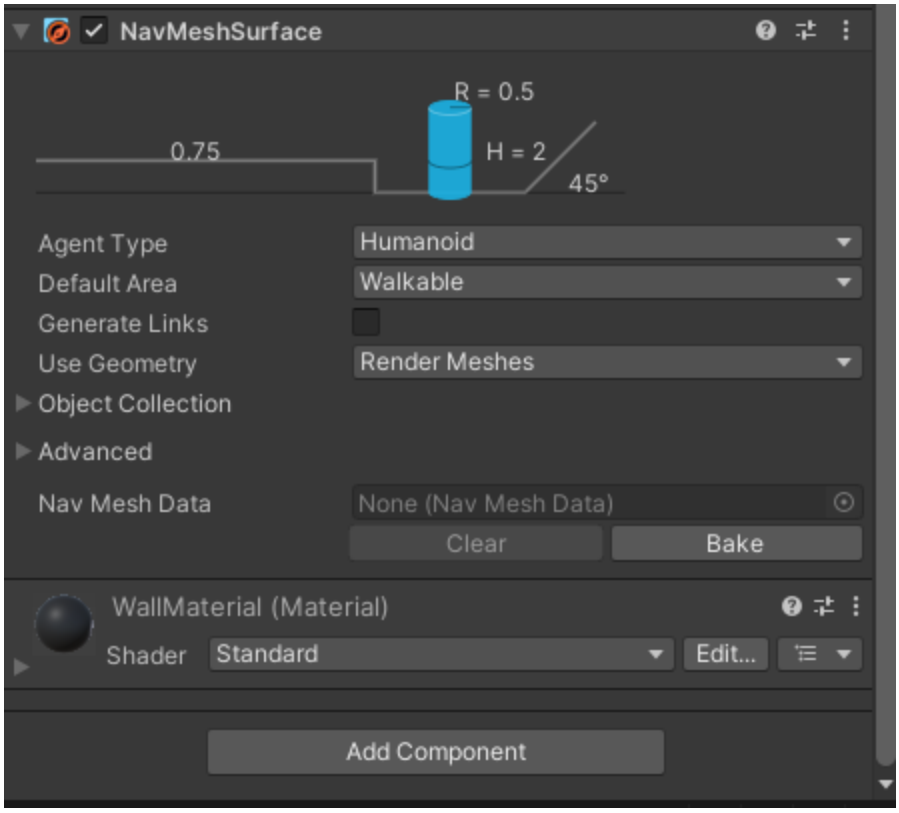
Lab 7 Navigation :

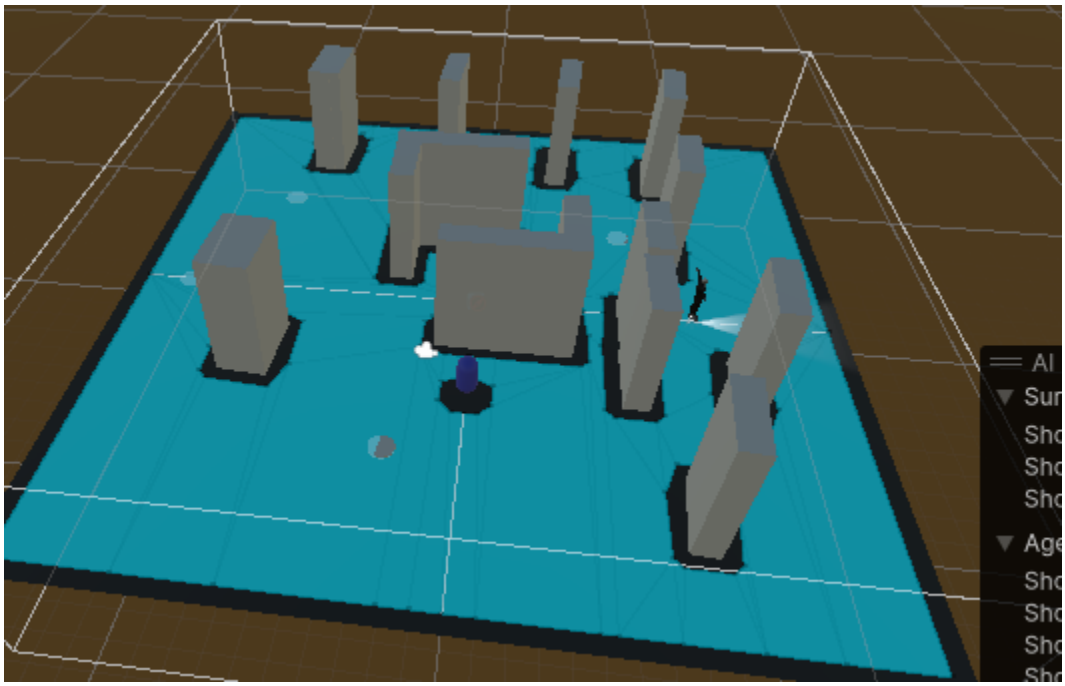
Accept the github assignment : https://classroom.github.com/a/r_Z9oWZ-

Step 1: Setup your scene

1. **In the inspector add a NavMesh Surface component to the Floor**



2. 

3. **Then click Bake to bake the navmesh**
4. 

Step 2: Create the AI State Machine Classes

3.1 Base State Class

Create an interface called IState:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public interface IState
{
    StateType Type { get; }
    void Enter();
    void Execute();
    void Exit();
}
```

3.2 Individual State Classes

For each behavior (Idle, Patrol, Chase, and Attack), create a derived class that inherits from IState:

IdleState:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class IdleState : IState
{
    private AIController aiController;
    private float idleDuration = 2f;
    private float idleTimer;

    public StateType Type => StateType.Idle;

    public IdleState(AIController aiController)
```

```
    {
        this.aiController = aiController;
    }

    public void Enter()
    {
        idleTimer = 0f;
        //aiController.Animator.SetBool("isMoving", false);
    }

    public void Execute()
    {
        idleTimer += Time.deltaTime;
        if (idleTimer >= idleDuration)
        {
            aiController.StateMachine.TransitionToState(StateType.Patrol);
        }
    }

    public void Exit()
    {
        // Cleanup if necessary
    }
}
```

PatrolState:

```
public class PatrolState : IState
{
    private AIController aiController;
    private int currentWaypointIndex = 0;

    public StateType Type => StateType.Patrol;

    public PatrolState(AIController aiController)
```

```
    {
        this.aiController = aiController;
    }

    public void Enter()
    {
        //aiController.Animator.SetBool("isMoving", true);
        MoveToNextWaypoint();
    }

    public void Execute()
    {
        if (aiController.CanSeePlayer())
        {
            aiController.StateMachine.TransitionToState(StateType.Chase);
            return;
        }

        if (!aiController.Agent.pathPending &&
aiController.Agent.remainingDistance <= aiController.Agent.stoppingDistance)
        {
            MoveToNextWaypoint();
        }
    }

    public void Exit()
    {
        // Cleanup if necessary
    }

    private void MoveToNextWaypoint()
    {
        if (aiController.Waypoints.Length == 0)
            return;

        aiController.Agent.destination =
aiController.Waypoints[currentWaypointIndex].position;
        currentWaypointIndex = (currentWaypointIndex + 1) %
aiController.Waypoints.Length;
```

```
    }
}
```

Chase State

```
public class ChaseState : IState
{
    private AIController aiController;

    public StateType Type => StateType.Chase;

    public ChaseState(AIController aiController)
    {
        this.aiController = aiController;
    }

    public void Enter()
    {
       aiController.Animator.SetBool("isChasing", true);
        // No animations, so no need to set any animator parameters
    }

    public void Execute()
    {
        if (!aiController.CanSeePlayer())
        {
            aiController.StateMachine.TransitionToState(StateType.Patrol);
            return;
        }

        if (aiController.IsPlayerInAttackRange())
        {
            aiController.StateMachine.TransitionToState(StateType.Attack);
            return;
        }

        aiController.Agent.destination = aiController.Player.position;
```

```
    }

    public void Exit()
    {
        // No cleanup necessary
    }
}
```

AttackState:

```
using UnityEngine;
using UnityEngine.SceneManagement;

public class AttackState : IState
{
    private AIController aiController;

    public StateType Type => StateType.Attack;

    public AttackState(AIController aiController)
    {
        this.aiController = aiController;
    }

    public void Enter()
```

```
    {

        aiController.Animator.SetBool("isAttacking", true);
        aiController.Agent.isStopped = true; // Stop the AI agent movement
    }

    public void Execute()
    {
        // Check if the player is within attack range
        if (Vector3.Distance(aiController.transform.position,
aiController.Player.position) > aiController.AttackRange)
        {
            // If the player moves away, transition back to ChaseState
            aiController.StateMachine.TransitionToState(StateType.Chase);
            return;
        }
    }

    public void Exit()
    {
        aiController.Animator.SetBool("isAttacking", true);
        aiController.Agent.isStopped = false; // Resume the AI agent movement
    }
}
```

Step 4: Create a class called StateMachine:

Create the State Machine Class

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class StateMachine
```

```
{
    private Dictionary<StateType, IState> states = new Dictionary<StateType,
IState>();
    private IState currentState;

    public StateType GetCurrentStateType()
    {
        return currentState.Type;
    }

    public void AddState(IState state)
    {
        if (!states.ContainsKey(state.Type))
        {
            states.Add(state.Type, state);
        }
    }

    public void TransitionToState(StateType type)
    {
        currentState?.Exit();
        currentState = states[type];
        currentState.Enter();
    }

    public void Update()
    {
        currentState?.Execute();
    }
}
```

Step 5: Create the AI Controller Class

```
using UnityEngine;
```

```
using UnityEngine.AI;



public class AIController : MonoBehaviour

{

    public StateMachine StateMachine { get; private set; }

    public NavMeshAgent Agent { get; private set; }

    public Animator Animator { get; private set; } // Not needed since we're not
using animations

    public Transform[] Waypoints;

    public Transform Player;

    public float SightRange = 10f;

    public float AttackRange = 2f; // New attack range variable

    public LayerMask PlayerLayer;

    public StateType currentState;

    void Start()

    {

        Agent = GetComponent<NavMeshAgent>();

        Animator = GetComponent<Animator>(); // Commented out since we're not
using animations



        StateMachine = new StateMachine();

        StateMachine.AddState(new IdleState(this));

        StateMachine.AddState(new PatrolState(this));
```

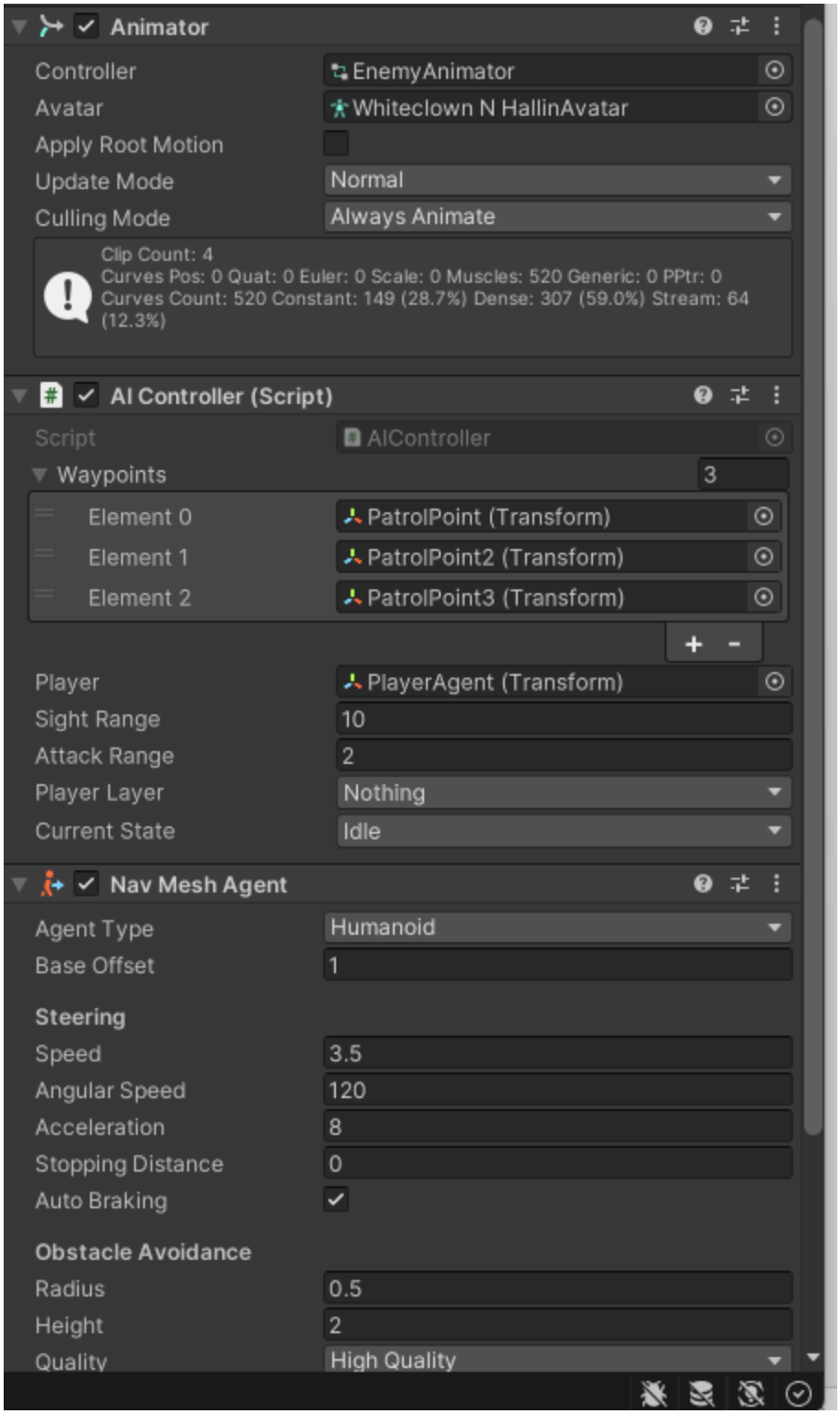
```
        StateMachine.AddState(new ChaseState(this));
        StateMachine.AddState(new AttackState(this)); // Add the new AttackState


        StateMachine.TransitionToState(StateType.Idle);
    }


    void Update()
    {
        StateMachine.Update();
        Animator.SetFloat("CharacterSpeed", Agent.velocity.magnitude);
        currentState = StateMachine.GetCurrentStateType();
    }


    public bool CanSeePlayer()
    {
        float distanceToPlayer = Vector3.Distance(transform.position,
Player.position);
        if (distanceToPlayer <= SightRange)
        {
            // Optionally, add line of sight checks here using Raycast
            return true;
        }
        return false;
```

```
    }



    // New method to check if the AI is within attack range

    public bool IsPlayerInAttackRange()

    {

        float distanceToPlayer = Vector3.Distance(transform.position,
Player.position);

        return distanceToPlayer <= AttackRange;

    }

}
```

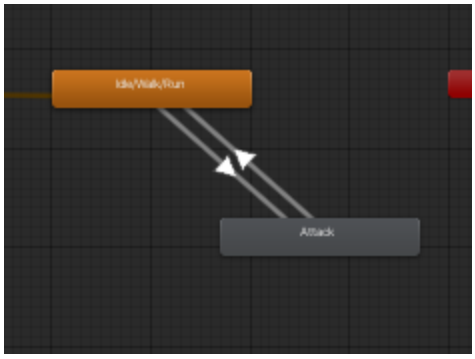
Step 8: Assign Components in Unity

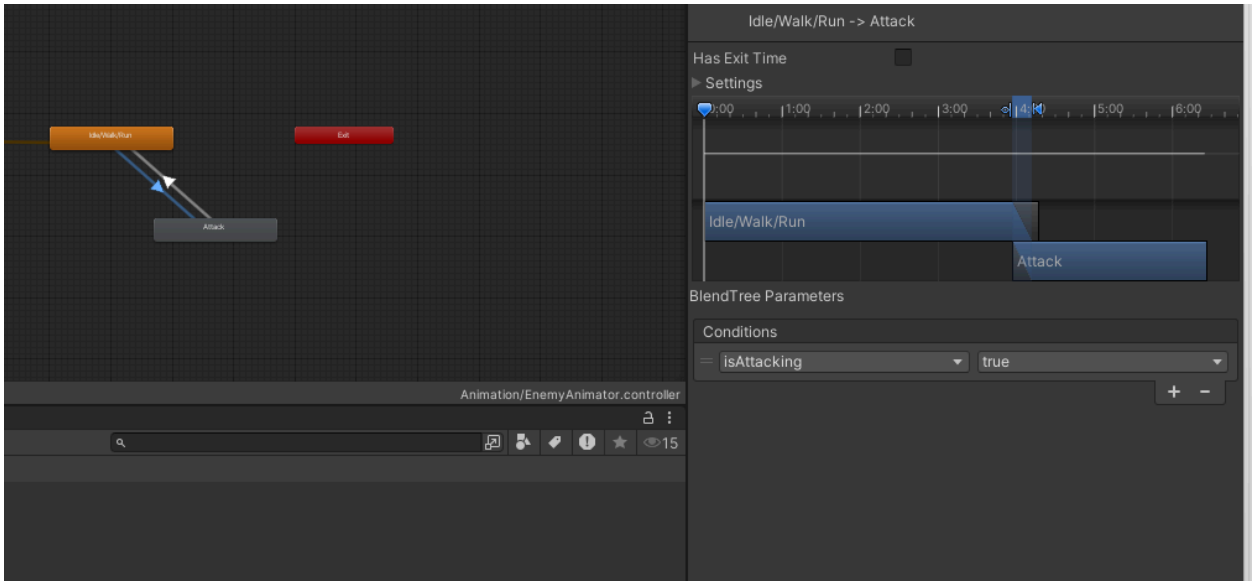
1. Attach the AIController script to the Zombie Character.
2. Assign the Animator and NavMeshAgent in the inspector.
3. Set the Player transform to the player's GameObject and add waypoints in PatrolPoints.

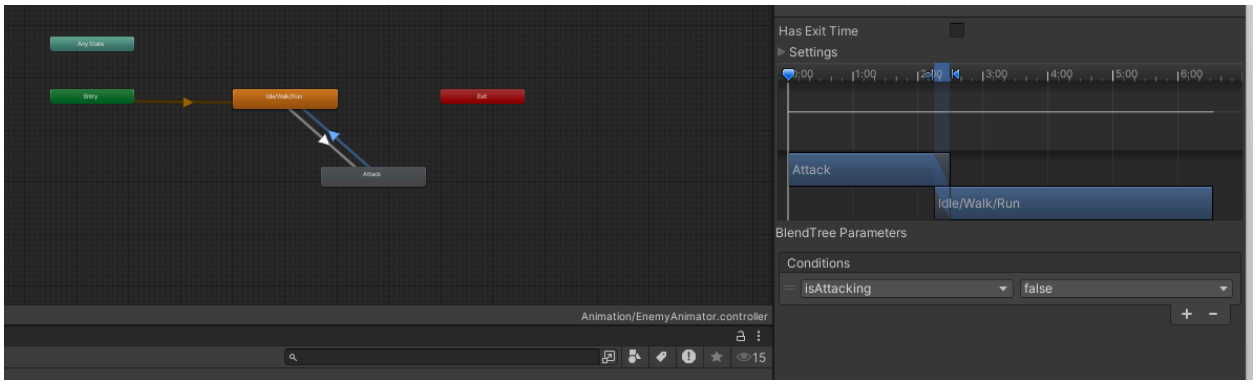
4. Make sure the NavMeshAgent base offset is set to 1.



Setup the animator controller







Make sure the hasExitTime is set to false.