**AI Script Design**

**AIHandler:**

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class AIHandler : MonoBehaviour {

[SerializeField] int range;

[SerializeField] float rotateSpeed;

[SerializeField] LayerMask coverLayer;

[SerializeField] Collider coverPoint;

[SerializeField] NavMeshAgent agent;

[SerializeField] Transform target;

private float speedX;

private float speedY;

private string state = "idle";

public bool isAtCover;

private Rigidbody rb;

private Vector3 destination;

private Collider currentCoverCollider;

private Transform agentTransform;

private Rigidbody[] bones;

private Animator anim;

private void Start()

{

bones = GetComponentsInChildren<Rigidbody>();

rb = GetComponent<Rigidbody>();

anim = GetComponent<Animator>();

agent = GetComponent<NavMeshAgent>();

target = FindObjectOfType<Character>().transform;

agentTransform = gameObject.transform;

for (int i = 0; i < bones.Length; i++)

{

if (!bones[i].isKinematic)

{

bones[i].isKinematic = true;

}

}

}

private void Update()

{

HandleCover();

HandleAgentRotation();

HandleMovementAnimations();

HandleMovementAnimations();

}

private void MoveToPosition(Vector3 targetTransform)

{

agent.isStopped = false;

OnSetDestination(targetTransform);

destination = targetTransform;

}

private void CalculateDirectionalSpeed()

{

Vector3 velocity = rb.velocity;

speedX = velocity.x;

speedY = velocity.y;

}

private void HandleCover()

{

Collider col = FindClosestCover();

if (col == null)

return;

Vector3 dirToTarget = target.position - col.transform.position;

dirToTarget.Normalize();

Vector3 targetPosition = col.transform.position + (dirToTarget \* -1);

MoveToPosition(targetPosition);

}

private Collider FindClosestCover()

{

Collider[] colliders = Physics.OverlapSphere(agentTransform.position, range, coverLayer);

float mDist = float.MaxValue;

Collider closest = null;

for (int i = 0; i < colliders.Length; i++)

{

if (currentCoverCollider == colliders[i])

continue;

float tDist = Vector3.Distance(colliders[i].transform.position, agentTransform.position);

if (tDist < mDist)

{

mDist = tDist;

closest = colliders[i];

}

}

return closest;

}

private void HandleMovementAnimations()

{

CalculateDirectionalSpeed();

float maxSpeed = agent.speed;

float velocityX = maxSpeed / speedX;

float velocityY = maxSpeed / speedY;

Debug.Log("X = " + speedX);

Debug.Log("Y = " + speedY);

Debug.Log("Max Speed = " + maxSpeed);

anim.SetFloat("SpeedX", velocityX);

anim.SetFloat("SpeedY", velocityY);

}

private void HandleAgentRotation()

{

if (target != null)

{

if (agent.updateRotation)

agent.updateRotation = false;

Vector3 dir = target.position - agentTransform.position;

dir.y = 0;

if (dir == Vector3.zero)

dir = agentTransform.forward;

Quaternion targetRot = Quaternion.LookRotation(dir);

Quaternion rotation = Quaternion.Slerp(agentTransform.rotation, targetRot, rotateSpeed);

agentTransform.rotation = rotation;

}

else

{

if (!agent.updateRotation)

agent.updateRotation = true;

}

}

private void OnSetDestination(Vector3 destination)

{

Vector3 navMeshPos = destination;

NavMeshHit hit;

if (NavMesh.SamplePosition(navMeshPos, out hit, 2, NavMesh.AllAreas))

{

navMeshPos = hit.position;

agent.SetDestination(destination);

}

}

private void OnCoverReached()

{

float distance = Vector3.Distance(transform.position, destination);

if (transform.position == destination)

{

isAtCover = true;

}

else

{

isAtCover = false;

}

}

}