敵が自動で追いかけるスクリプト

public enum Enemy1

{

idle,

walk,

chase,

attack,

attack2,

}

private CharacterController characterController;

private Animator animator;

//　トロールが登場した最初の位置

private Vector3 defaultPos;

//　トロールの状態

private Enemy1 enemy = Enemy1.idle;

//　目的地

private Vector3 destination;

//　移動範囲

[SerializeField]

private float movementRange = 20f;

//　移動速度

private Vector3 velocity = Vector3.zero;

//　歩くスピード

[SerializeField]

private float walkSpeed = 0.5f;

//　追いかけるスピード

[SerializeField]

private float chaseSpeed = 1f;

//　向きを回転する速さ

[SerializeField]

private float rotateSpeed = 2f;

//　idle状態の経過時間

private float elapsedTimeOfIdleState = 0f;

//　idle状態で留まる時間

[SerializeField]

private float timeToStayInIdle = 3f;

//　攻撃対象のTransform

private Transform attackTargetTransform;

//　攻撃時の対象の位置

private Vector3 attackTargetPos;

// Use this for initialization

void Start()

{

characterController = GetComponent<CharacterController>();

animator = GetComponent<Animator>();

defaultPos = transform.position;

SetRandomDestination();

}

// Update is called once per frame

void Update()

{

//　トロールの状態によって処理を変える

if (enemy == Enemy1.idle)

{

Idle();

}

else if (enemy == Enemy1.walk)

{

Walk();

}

else if (enemy == Enemy1.chase)

{

Chase();

}

else if (enemy == Enemy1.attack)

{

Attack();

}

else if (enemy == Enemy1.attack2)

{

Attack2();

}

//　共通するCharacterControllerの移動処理

velocity.y += Physics.gravity.y \* Time.deltaTime;

characterController.Move(velocity \* Time.deltaTime);

}

//　目的地を設定する

void SetRandomDestination()

{

//　最初の位置から有効範囲内のランダム位置を取得

var randomPos = defaultPos + Random.insideUnitSphere \* movementRange;

var ray = new Ray(randomPos + Vector3.up \* 10f, Vector3.down);

RaycastHit hit;

//　目的地が地面になるように再設定

if (Physics.Raycast(ray, out hit, 100f, LayerMask.GetMask("Field")))

{

destination = hit.point;

}

}

//　状態変更メソッド

public void SetState(Enemy1 tmpState, Transform playerTransform = null)

{

enemy = tmpState;

if (enemy == Enemy1.idle)

{

velocity = new Vector3(0f, velocity.y, 0f);

animator.SetFloat("Walk", 0f);

animator.SetBool("Chase", false);

SetRandomDestination();

Debug.Log("アイドル");

}

else if (enemy == Enemy1.walk)

{

Debug.Log("パトロール");

}

else if (enemy == Enemy1.attack)

{

attackTargetTransform = playerTransform;

attackTargetPos = attackTargetTransform.position;

velocity = new Vector3(0f, velocity.y, 0f);

animator.SetTrigger("Attack");

animator.SetBool("Chase", false);

Debug.Log("通常攻撃");

}

else if (enemy == Enemy1.attack2)

{

attackTargetTransform = playerTransform;

attackTargetPos = attackTargetTransform.position;

velocity = new Vector3(0f, velocity.y, 0f);

animator.SetTrigger("Attack2");

animator.SetBool("Chase", false);

Debug.Log("衝撃波攻撃");

}

else if (enemy == Enemy1.chase)

{

animator.SetBool("Chase", true);

attackTargetTransform = playerTransform;

Debug.Log("チェイス");

}

}

//　状態取得メソッド

public Enemy1 GetState()

{

return enemy;

}

//　Idle状態の時の処理

private void Idle()

{

elapsedTimeOfIdleState += Time.deltaTime;

//　一定時間が経過したらpatrol状態にする

if (elapsedTimeOfIdleState >= timeToStayInIdle)

{

elapsedTimeOfIdleState = 0f;

SetState(Enemy1.walk);

}

}

//　Patrol状態の時の処理

private void Walk()

{

//　通常移動処理

if (characterController.isGrounded)

{

velocity = Vector3.zero;

//　目的地の方向を計算し、向きを変えて前方に進める

var direction = (destination - transform.position).normalized;

animator.SetFloat("Walk", direction.magnitude);

var targetRot = Quaternion.Lerp(transform.rotation, Quaternion.LookRotation(destination - transform.position), Time.deltaTime \* rotateSpeed);

transform.rotation = Quaternion.Euler(transform.eulerAngles.x, targetRot.eulerAngles.y, transform.eulerAngles.z);

velocity = transform.forward \* walkSpeed;

}

//　目的地に着いたらidle状態にする

if (Vector3.Distance(transform.position, destination) < 0.5f)

{

SetState(Enemy1.idle);

}

}

//　Chase状態の時の処理

private void Chase()

{

//　目的地を毎回設定し直す

destination = attackTargetTransform.position;

//　追いかける処理

if (characterController.isGrounded)

{

velocity = Vector3.zero;

//　追いかける時はキャラクターの向きに回転して進ませる

var direction = (destination - transform.position).normalized;

var targetRot = Quaternion.Lerp(transform.rotation, Quaternion.LookRotation(destination - transform.position), Time.deltaTime \* rotateSpeed);

transform.rotation = Quaternion.Euler(transform.eulerAngles.x, targetRot.eulerAngles.y, transform.eulerAngles.z);

velocity = transform.forward \* chaseSpeed;

}

}

//　Attack状態の時の処理

private void Attack()

{

//　攻撃状態になった時のキャラクターの向きを計算し、徐々にそちらの向きに回転させる

var targetRot = Quaternion.Lerp(transform.rotation, Quaternion.LookRotation(attackTargetPos - transform.position), Time.deltaTime \* 2f);

transform.rotation = Quaternion.Euler(transform.eulerAngles.x, targetRot.eulerAngles.y, transform.eulerAngles.z);

//　Attackアニメーションが終了したらIdle状態にする

if (animator.GetCurrentAnimatorStateInfo(0).IsName("Attack")

&& animator.GetCurrentAnimatorStateInfo(0).normalizedTime >= 1f)

{

SetState(Enemy1.idle);

}

}

//　ShockwaveAttackの時の処理

private void Attack2()

{

//　攻撃状態になった時のキャラクターの向きを計算し、徐々にそちらの向きに回転させる

var targetRot = Quaternion.Lerp(transform.rotation, Quaternion.LookRotation(attackTargetPos - transform.position), Time.deltaTime \* 2f);

transform.rotation = Quaternion.Euler(transform.eulerAngles.x, targetRot.eulerAngles.y, transform.eulerAngles.z);

//　ShockwaveAttackアニメーションが終了したらIdle状態にする

if (animator.GetCurrentAnimatorStateInfo(0).IsName("Attack2")

&& animator.GetCurrentAnimatorStateInfo(0).normalizedTime >= 1f)

{

SetState(Enemy1.idle);

}

}

APのゲージを減らすスクリプト

//最大のAP数

int maxAP = 100;

//現在のAP数

float currentAP;

GameObject AP;

// Start is called before the first frame update

void Start()

{

AP = GameObject.Find("AP");

}

// Update is called once per frame

void Update()

{

AP.GetComponent<AP>().APFull(currentAP, maxAP);

}

private void FixedUpdate()

{

if(0<=currentAP)

{

currentAP = maxAP - Time.time \* 100;

Debug.Log("a");

}

}