Ellie개발문서

유니티 팀 프로젝트

Monster & Behaviour Tree

목차

1. 기획내용

1-1. 기획팀 내용

1-2. 개발팀 추가 요청 내용

2. 몬스터 설계

2-1. Diagram

2-2. Monster

3. 몬스터 공격

3-1. Diagram

3-2. Monster Attacks

4. Behaviour Tree

4-1. Behaviour Tree & Node

4-2. Monster Behaviour Tree

4-3. Sequence Behaviour Tree

4-4. Action Node

4-5. Composite Node

4-6. Decorater Node

5. 기타

5-1. Monster Data Class

5-2. Behaviour Tree Enum

[게임 영상 보러가기 [유튜브]](https://youtu.be/zzI1e1QEVNM?si=YWv9Q7n6C1ybh-9t)

**1. 기획내용**

1-1. 기획팀 기획 내용

- 몬스터 기획서



- 근접공격 다이어그램

A diagram of a company

Description automatically generated

- 원거리 몬스터 다이어그램

A screenshot of a computer

Description automatically generated

1-2. 개발팀 기획 요청 내용

1-2.1 몬스터 공통 Stat



1-2.2 몬스터 공격

- 공격 여부 표시



- 근접공격



- 달려들기



1-2.3 몬스터 행동 우선순위



**2. 몬스터 설계**

2-1. Diagram

A diagram of a computer

Description automatically generated

2-2. Monster

A screenshot of a computer program

Description automatically generated

2-2.1 MonsterBase

- 공통 사항

모든 몬스터의 부모 클래스로 몬스터가 가지고 있어야 하는 공격, Behaviour Tree, 몬스터 Stat, 플레이어를 인지하는 AI와 쫓아가는 AI, Patrol Points를 기본적으로 가지고 있다.

- 공격

모든 공격의 부모 클래스 MonsterAttackBase를 사용하여 몬스터의 공격을 AddAttack()함수를 통해 추가 할 수 있고, ExecueteAttack을 통해 공격을 실행할 수 있다.

AddAttack으로 공격을 추가하면 몬스터 자식으로 해당 공격의 script를 가지고 있는 empty object가 몬스터 자식으로 추가된다.

**using** **System.Collections**;

**using** **System.Collections.Generic**;

**using** **Assets.Scripts.Combat**;

**using** **Assets.Scripts.Managers**;

**using** **Assets.Scripts.Monsters.Utility**;

**using** **Assets.Scripts.UI.Monster**;

**using** **Assets.Scripts.Utils**;

**using** **Channels.Combat**;

**using** **Channels.Components**;

**using** **Channels.Type**;

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterBase** : MonoBehaviour, ICombatant

{

**private** Dictionary<MonsterSkill, MonsterAttackBase> attacks;

[SerializeField] **protected** MonsterBehaviourTree tree;

**public** MonsterStatData monsterStat;

**protected** DetectAI detectPlayer;

**protected** DetectAI detectChase;

[SerializeField] **public** MonsterCenter MonsterCenter { **get**; **private** **set**; }

[SerializeField] **public** List<Transform> patrolPoints;

**private** NavMeshAgent agent;

**private** MonsterAudioController audioController;

**private** Animator animator;

**private** TicketMachine ticketMachine;

**private** float currentHP;

**private** bool isHeadshot;

**private** UIMonsterBillboard billboard;

**private** Transform billboardObject;

**private** bool isBillboardOn;

**private** Transform cameraObj;

**private** Coroutine battleCoroutine;

**private** **readonly** MonsterDataContainer dataContainer = **new**();

**private** **void** Awake()

{

audioController = GetComponent<MonsterAudioController>();

animator = GetComponent<Animator>();

ticketMachine = gameObject.GetOrAddComponent<TicketMachine>();

ticketMachine.AddTickets(ChannelType.Combat, ChannelType.Monster);

SetDetectAI();

tree = GetComponent<MonsterBehaviourTree>();

}

**protected** **virtual** **void** Start()

{

tree.AddMonsterData<GameObject>(MonsterData.v3SpawnPosition, MonsterCenter.Player);

}

**private** **void** Update()

{

MonsterOnPlayerForward();

}

*// >> : Set Datas*

**protected** **void** SetMonsterData(MonsterStatData data)

{

*// UI*

currentHP = monsterStat.HP;

dataContainer.MaxHp = (int)monsterStat.HP;

dataContainer.PrevHp = (int)monsterStat.HP;

dataContainer.CurrentHp.Value = (int)currentHP;

dataContainer.Name = monsterStat.name;

billboardObject = Functions.FindChildByName(gameObject, "Billboard").transform;

cameraObj = Camera.main.transform;

billboard = UIManager.Instance.MakeStatic<UIMonsterBillboard>(billboardObject, UIManager.UIMonsterBillboard);

HideBillboard();

billboard.InitBillboard(billboardObject);

*// Agent*

agent.speed = monsterStat.speed;

billboard.InitData(dataContainer);

}

**private** **void** SetDetectAI()

{

GameObject detectPlayerObj = **new** GameObject("DetectPlayer");

detectPlayerObj.transform.SetParent(transform);

detectPlayerObj.transform.localPosition = Vector3.zero;

detectPlayerObj.transform.localRotation = Quaternion.Euler(Vector3.zero);

detectPlayerObj.transform.localScale = Vector3.one;

detectPlayer = detectPlayerObj.AddComponent<DetectAI>();

GameObject detectChaseObj = **new** GameObject("DetectChase");

detectChaseObj.transform.SetParent(transform);

detectChaseObj.transform.localPosition = Vector3.zero;

detectChaseObj.transform.localRotation = Quaternion.Euler(Vector3.zero);

detectChaseObj.transform.localScale = Vector3.one;

detectPlayer = detectChaseObj.AddComponent<DetectAI>();

detectChaseObj.AddComponent<SphereCollider>();

}

**protected** **void** AddAttack(MonsterSkill skill, MonsterAttackBase attack)

{

GameObject attackObj = **new** GameObject(skill.ToString());

attackObj.transform.SetParent(transform);

attackObj.transform.localPosition = Vector3.zero;

attackObj.transform.localRotation = Quaternion.Euler(Vector3.zero);

attackObj.transform.localScale = Vector3.one;

**switch** (skill)

{

**case** MonsterSkill.Melee:

attack = attackObj.AddComponent<MonsterMeleeAttack>();

**break**;

**case** MonsterSkill.Projectile:

attack = attackObj.AddComponent<MonsterProjectileAttack>();

**break**;

**case** MonsterSkill.Weapon:

attack = attackObj.AddComponent<MonsterWeaponeAttack>();

**break**;

**case** MonsterSkill.Flee:

attack = attackObj.AddComponent<MonsterFleeSkill>();

**break**;

}

**if** (attack == **null**)

{

Debug.LogFormat("{0} Failed To Add Skill {1}", transform.name, skill.ToString());

**return**;

}

attacks.Add(skill, attack);

}

*// >> : Battles*

**public** bool ExecuteAttack(MonsterSkill skill)

{

MonsterAttackBase attack;

**if** (attacks.TryGetValue(skill, **out** attack))

{

attack.ExecuteAttack();

**return** **true**;

}

Debug.LogFormat("Trying To Access Attack Does Not Have : {0}, {1}", transform.name, skill.ToString());

**return** **false**;

}

**public** **void** Attack(IBaseEventPayload payload)

{

CombatPayload combatPayload = payload **as** CombatPayload;

ticketMachine.SendMessage(ChannelType.Combat, combatPayload);

}

**public** **void** ReceiveDamage(IBaseEventPayload payload)

{

CombatPayload combatPayload = payload **as** CombatPayload;

UpdateHP(combatPayload.Damage);

}

*// >> : MonsterDamaged or Dead*

**private** **void** SetMonsterDead()

{

audioController.PlayAudio(MonsterAudioType.Dead);

**if** (animator == **null**)

{

animator = GetComponent<Animator>();

}

animator.Play("Dead");

GetComponent<Collider>().enabled = **false**;

MonsterPayload monsterPayload = **new**();

monsterPayload.RespawnTime = monsterStat.respawnTime;

monsterPayload.Monster = transform;

monsterPayload.ItemDrop = monsterStat.itemTableNum;

ticketMachine.SendMessage(ChannelType.Monster, monsterPayload);

}

**private** **void** UpdateHP(float damage)

{

**if** (tree.GetBTData<bool>(BTData.bReturning))

**return**;

**if** (isHeadshot) damage \*= monsterStat.weeknessRatio;

**if** (battleCoroutine != **null**) StopCoroutine(battleCoroutine);

battleCoroutine = StartCoroutine(EndBattleState());

currentHP -= damage;

tree.SetBTData<float>(BTData.iCurrentHP, currentHP);

**if** (currentHP < 1)

{

SetMonsterDead();

HideBillboard();

isBillboardOn = **false**;

}

**else**

{

**if** (isHeadshot)

audioController.PlayAudio(MonsterAudioType.HeadShot);

**else** audioController.PlayAudio(MonsterAudioType.Hit);

}

isHeadshot = **false**;

}

*// >> : Billboard*

**private** IEnumerator EndBattleState()

{

**yield** **return** **new** WaitForSeconds(MonsterCenter.battleStateTime);

HideBillboard();

isBillboardOn = **false**;

}

**private** **void** MonsterOnPlayerForward()

{

**if** (!isBillboardOn) **return**;

Vector3 direction = transform.position - cameraObj.position;

float dot = Vector3.Dot(direction.normalized, cameraObj.forward.normalized);

**if** (dot > 0)

ShowBillboard();

**else**

HideBillboard();

}

**private** **void** HideBillboard()

{

billboardObject.transform.localScale = Vector3.zero;

}

**private** **void** ShowBillboard()

{

billboardObject.transform.localEulerAngles = Vector3.one;

isBillboardOn = **true**;

}

}

}

2-2.2 Monsters

몬스터 정보를 파싱을 위한 Parsing Number와 해당 몬스터가 가지고 있는 공격 정보를 가지고 있다.

Parsing 이후 MonsterBase에 있는 attack Dictionary에 해당 몬스터가 가지고 있는 공격을 추가 해준다.

- 평범한 스켈레톤 (NormalSkeleton)

**using** **System.Collections**;

**using** **Assets.Scripts.Managers**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **NormalSkeleton** : MonsterBase

{

**private** **enum** ParsingData

{

MonsterStat = 1900,

MeleeAttack = 2000,

RunToPlayer = 2005,

}

**private** MeleeAttackData meleeAttackData;

**private** RunToPlayerData runData;

**protected** **override** **void** Start()

{

**base**.Start();

StartCoroutine(InitParsingData());

}

**private** IEnumerator InitParsingData()

{

**yield** **return** DataManager.Instance.CheckIsParseDone();

monsterStat = DataManager.Instance.GetIndexData<MonsterStatData, MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);

meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);

runData = DataManager.Instance.GetIndexData<RunToPlayerData, MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);

SetSkills();

SetMonsterData(monsterStat);

tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);

tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);

tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);

}

**private** **void** SetSkills()

{

MonsterMeleeAttack meleeAttack = **new**();

meleeAttack.SetInitialData(meleeAttackData);

AddAttack(MonsterSkill.Melee, meleeAttack);

}

}

}

- 모험가 스켈레톤 (AdventureSkeleton)

**using** **System.Collections**;

**using** **Assets.Scripts.Managers**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **AdventureSkeleton** : MonsterBase

{

**private** **enum** ParsingData

{

MonsterStat = 1901,

MeleeAttack = 2001,

RunToPlayer = 2006,

WeaponAttack = 2100,

}

**private** MeleeAttackData meleeAttackData;

**private** RunToPlayerData runData;

**private** WeaponAttackData weaponAttackData;

**protected** **override** **void** Start()

{

**base**.Start();

StartCoroutine(InitParsingData());

}

**private** IEnumerator InitParsingData()

{

**yield** **return** DataManager.Instance.CheckIsParseDone();

monsterStat = DataManager.Instance.GetIndexData<MonsterStatData, MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);

meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);

runData = DataManager.Instance.GetIndexData<RunToPlayerData, MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);

weaponAttackData = DataManager.Instance.GetIndexData<WeaponAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.WeaponAttack);

SetSkills();

SetMonsterData(monsterStat);

tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);

tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);

tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);

tree.AddMonsterData<WeaponAttackData>(MonsterData.Weapon, weaponAttackData);

}

**private** **void** SetSkills()

{

MonsterMeleeAttack meleeAttack = **new**();

meleeAttack.SetInitialData(meleeAttackData);

AddAttack(MonsterSkill.Melee, meleeAttack);

MonsterWeaponeAttack weaponeAttack = **new**();

weaponeAttack.SetInitialData(weaponAttackData);

AddAttack(MonsterSkill.Weapon, weaponeAttack);

}

}

}

- 마법사 스켈레톤 (WizardSkeleton)

**using** **System.Collections**;

**using** **Assets.Scripts.Managers**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **WizardSkeleton** : MonsterBase

{

**private** **enum** ParsingData

{

MonsterStat = 1902,

Flee = 2008,

ProjectileAttack = 2009,

}

**private** FleeSkillData fleeData;

**private** ProjectileAttackData projectileData;

**protected** **override** **void** Start()

{

**base**.Start();

StartCoroutine(InitParsingData());

}

**private** IEnumerator InitParsingData()

{

**yield** **return** DataManager.Instance.CheckIsParseDone();

monsterStat = DataManager.Instance.GetIndexData<MonsterStatData, MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);

fleeData = DataManager.Instance.GetIndexData<FleeSkillData, MonsterAttackDataparsingInfo>((int)ParsingData.Flee);

projectileData = DataManager.Instance.GetIndexData<ProjectileAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.ProjectileAttack);

SetSkills();

SetMonsterData(monsterStat);

tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);

tree.AddMonsterData<FleeSkillData>(MonsterData.Melee, fleeData);

tree.AddMonsterData<ProjectileAttackData>(MonsterData.RunToPlayer, projectileData);

}

**private** **void** SetSkills()

{

MonsterFleeSkill fleeAttack = **new**();

fleeAttack.SetInitialData(monsterStat, fleeData);

AddAttack(MonsterSkill.Flee, fleeAttack);

MonsterProjectileAttack projectileAttack = **new**();

projectileAttack.SetInitialData(projectileData);

AddAttack(MonsterSkill.Projectile, projectileAttack);

}

}

}

- 공대원 스켈레톤

**using** **System.Collections**;

**using** **Assets.Scripts.Managers**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **GuildguardSkeleton** : MonsterBase

{

**private** **enum** ParsingData

{

MonsterStat = 1903,

MeleeAttack = 2002,

RunToPlayer = 2007,

WeaponAttack = 2101,

}

**private** MeleeAttackData meleeAttackData;

**private** RunToPlayerData runData;

**private** WeaponAttackData weaponAttackData;

**protected** **override** **void** Start()

{

**base**.Start();

StartCoroutine(InitParsingData());

}

**private** IEnumerator InitParsingData()

{

**yield** **return** DataManager.Instance.CheckIsParseDone();

monsterStat = DataManager.Instance.GetIndexData<MonsterStatData, MonsterStatDataParsingInfo>((int)ParsingData.MonsterStat);

meleeAttackData = DataManager.Instance.GetIndexData<MeleeAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.MeleeAttack);

runData = DataManager.Instance.GetIndexData<RunToPlayerData, MonsterAttackDataparsingInfo>((int)ParsingData.RunToPlayer);

weaponAttackData = DataManager.Instance.GetIndexData<WeaponAttackData, MonsterAttackDataparsingInfo>((int)ParsingData.WeaponAttack);

SetSkills();

SetMonsterData(monsterStat);

tree.AddMonsterData<MonsterStatData>(MonsterData.MonsterStat, monsterStat);

tree.AddMonsterData<MeleeAttackData>(MonsterData.Melee, meleeAttackData);

tree.AddMonsterData<RunToPlayerData>(MonsterData.RunToPlayer, runData);

tree.AddMonsterData<WeaponAttackData>(MonsterData.Weapon, weaponAttackData);

}

**private** **void** SetSkills()

{

MonsterMeleeAttack meleeAttack = **new**();

meleeAttack.SetInitialData(meleeAttackData);

AddAttack(MonsterSkill.Melee, meleeAttack);

MonsterWeaponeAttack weaponeAttack = **new**();

weaponeAttack.SetInitialData(weaponAttackData);

AddAttack(MonsterSkill.Weapon, weaponeAttack);

}

}

}

**3. 몬스터 공격**

3-1. Diagram

A screenshot of a computer

Description automatically generated

3-2. Monster Attacks

3-2.1 MonsterAttackBase

모든 공격의 부모 클래스로 공격을 실행하는 ExecuteAttack을 가상함수로 가지고 있어 자식 클래스에서 override하여 사용한다

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterAttackBase** : MonoBehaviour

{

**private** MonsterBase monster;

**protected** MonsterAudioController audioController;

**protected** MonsterParticleController particleController;

**protected** **virtual** **void** Awake()

{

**if** (audioController == **null**)

audioController = GetComponent<MonsterAudioController>();

**if** (particleController == **null**)

particleController = GetComponent<MonsterParticleController>();

}

**public** **virtual** **void** ExecuteAttack() {}

**public** **void** Attack(IBaseEventPayload payload)

{

monster.Attack(payload);

}

}

}

3-2.2 Monster Skills

- MonsterMeleeAttack (BoxcolliderAttack)

설정된 크기와 위치에 BoxCollider을 ExecuteAttack을 통해 해당 Collider을 Enable, Disable하는 방식으로 공격

각 Monster에서 설정된 MeleeAttack 정보를 토대로 BoxCollider의 위치, 크기, 지속시간 등 필요한 정보를 모두 설정한다

**using** **System.Collections**;

**using** **Assets.Scripts.Combat**;

**using** **Assets.Scripts.StatusEffects**;

**using** **Channels.Combat**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterMeleeAttack** : MonsterAttackBase

{

**private** MeleeAttackData data;

**private** BoxCollider collider;

**private** ParticleSystem particle;

**protected** **override** **void** Awake()

{

**base**.Awake();

*// Collider Setting*

**if** (collider == **null**)

collider = gameObject.AddComponent<BoxCollider>();

collider.isTrigger = **true**;

collider.size = data.colliderSize;

gameObject.transform.localPosition = data.colliderOffset;

collider.enabled = **false**;

}

**public** **void** SetInitialData(MeleeAttackData meleeData)

{

data = meleeData;

}

**public** **override** **void** ExecuteAttack()

{

**if**(data==**null**)

Debug.LogFormat("{0} Has Not Initialized MeleeAttack : {0}, MeleeAttack" + transform.name);

collider.enabled = **true**;

StartCoroutine(DisableCollider());

}

**private** IEnumerator DisableCollider()

{

**yield** **return** **new** WaitForSeconds(data.colliderDuration);

collider.enabled = **false**;

}

**private** **void** OnTriggerEnter(Collider other)

{

**if**(other.CompareTag("Player"))

{

**if**(other.gameObject.GetComponent<ICombatant>()!=**null**)

{

audioController.PlayAudio(MonsterAudioType.MeleeAttackHit);

**if** (particle == **null**)

{

particle = particleController.GetParticle(MonsterParticleType.MeleeHit);

}

particle.transform.position = other.transform.position;

particle.Play();

SetAndAttack(other.transform);

}

}

}

**private** **void** SetAndAttack(Transform otherTransform)

{

CombatPayload payload = **new**();

payload.Type = data.combatType;

payload.Attacker = transform;

payload.Defender = otherTransform;

payload.AttackDirection = Vector3.zero;

payload.AttackStartPosition = transform.position;

payload.AttackPosition = otherTransform.position;

payload.StatusEffectName = StatusEffectName.WeakRigidity;

payload.statusEffectduration = 0.3f;

payload.Damage = data.damage;

Attack(payload);

}

}

}

- MonsterWeaponAttack (내적 계산 공격)

몬스터가 무기를 휘둘러 공격하는 기술로, 무기와 상관 없이 주어진 반지름과 각도를 기준으로 부채꼴 형태의 범위에 내적을 사용하여 플레이어가 부채꼴 안에 있는지 아닌지를 확인하여 공격 여부를 판단한다

**using** **System.Collections**;

**using** **Assets.Scripts.StatusEffects**;

**using** **Channels.Combat**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterWeaponeAttack** : MonsterAttackBase

{

**private** WeaponAttackData data;

**private** GameObject target;

**protected** **override** **void** Awake()

{

**base**.Awake();

target = GameObject.Find("Player");

}

**public** **void** SetInitialData(WeaponAttackData WeaponData)

{

data = WeaponData;

}

**public** **override** **void** ExecuteAttack()

{

StartCoroutine(AttackFanshape());

}

**public** IEnumerator AttackFanshape()

{

float accumTime = 0.0f;

**while** (accumTime <= data.duration)

{

**if** (CaculateDotProduct())

{

**if** (target.CompareTag("Player"))

{

audioController.PlayAudio(MonsterAudioType.WeaponAttackHit);

particleController.PlayParticle(MonsterParticleType.WeaponHit, target.transform);

SetAndAttack(data, target.transform);

**break**;

}

}

accumTime += Time.deltaTime;

**yield** **return** **null**;

}

}

**private** bool CaculateDotProduct()

{

Vector3 interV = target.transform.position - transform.position;

float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);

float theta = Mathf.Acos(dot);

float degree = Mathf.Rad2Deg \* theta;

**if** (degree <= data.angle / 2.0f)

{

interV.y = 0;

**if** (interV.sqrMagnitude <= data.radius \* data.radius)

{

**return** **true**;

}

}

**return** **false**;

}

**private** **void** SetAndAttack(WeaponAttackData data, Transform otherTransform)

{

CombatPayload payload = **new**();

payload.Type = data.combatType;

payload.Attacker = transform;

payload.Defender = otherTransform;

payload.AttackDirection = Vector3.zero;

payload.AttackStartPosition = transform.position;

payload.AttackPosition = otherTransform.position;

payload.StatusEffectName = StatusEffectName.WeakRigidity;

payload.statusEffectduration = 0.5f;

payload.Damage = data.damage;

Attack(payload);

}

}

}

- MonsterProjectileAttack(투사체 공격)

몬스터의 투사체 공격으로 몬스터에게 투사체 Prefab을 발사하여 공격하는 방법

**using** **Assets.Scripts.Managers**;

**using** **Assets.Scripts.StatusEffects**;

**using** **Channels.Combat**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterProjectileAttack** : MonsterAttackBase

{

**private** **const** string path = "Prefabs/Monster/Projectiles/";

**private** ProjectileAttackData projectileData;

**private** GameObject projectileObj;

**private** ProjectileBase projectile;

**protected** **override** **void** Awake()

{

**base**.Awake();

string finalPath = path + projectileData.name;

projectileObj = Instantiate(ResourceManager.Instance.LoadExternResource<GameObject>(finalPath), transform);

**if** (projectileObj == **null**)

Debug.LogFormat("{0} Trying To Instanciate {1} But No Prefab On {2} : {0}, {1}", transform.name, projectileData.name, path);

**else**

projectile = projectileObj.GetComponent<ProjectileBase>();

}

**public** **void** SetInitialData(ProjectileAttackData projectileData)

{

**this**.projectileData = projectileData;

}

**public** **override** **void** ExecuteAttack()

{

**if** (projectileData == **null**)

Debug.LogFormat("{0} Has Not Initialized {1} : {0}, {1}" + transform.name, projectileData.name);

**else** projectile.Fire();

}

**public** **void** Explode(Transform transform)

{

particleController.PlayParticle(MonsterParticleType.ProjectileHit, transform);

audioController.PlayAudio(MonsterAudioType.ProjectileHit, transform);

}

**public** **void** SetAndAttack(Transform otherTransform)

{

CombatPayload payload = **new**();

payload.Type = projectileData.combatType;

payload.Attacker = transform;

payload.Defender = otherTransform;

payload.AttackDirection = Vector3.zero;

payload.AttackStartPosition = transform.position;

payload.AttackPosition = otherTransform.position;

payload.StatusEffectName = StatusEffectName.Burn;

payload.statusEffectduration = 3.0f;

payload.Damage = (int)projectileData.damage;

Attack(payload);

}

}

}

- MonsterFleeSkill (도망가기)

플레이어와 멀리 떨어진 곳으로 도망가는 스킬로 최우선 적으로 몬스터 뒤쪽으로 Ray를 통해 벽 유무, NavMesh인 곳인지를 확인하고, 실패 시 랜덤 방향으로 총 5번의 시도를 한다. 성공 시 빠른 속도로 해당 위치로 이동하게 된다.

**using** **System.Collections**;

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterFleeSkill** : MonsterAttackBase

{

**private** **const** int constMaxAttempts = 5;

**private** FleeSkillData fleeData;

**private** MonsterStatData statData;

**private** NavMeshAgent agent;

**private** Vector3 directionVector;

**private** Vector3 runAwayVector;

**private** Vector3 fleeVector;

**private** int attemptFlee;

**private** GameObject player;

**protected** **override** **void** Awake()

{

**base**.Awake();

player = GameObject.Find("Player");

agent = GetComponent<NavMeshAgent>();

}

**public** **void** SetInitialData(MonsterStatData statData, FleeSkillData fleeData)

{

**this**.statData = statData;

**this**.fleeData = fleeData;

}

**public** **override** **void** ExecuteAttack()

{

**if**(statData==**null**||fleeData==**null**)

{

Debug.LogFormat("{0} Has Not Initialized WeaponAttack : {0}, WeaponAttack" + transform.name);

**return**;

}

attemptFlee = 0;

directionVector = player.transform.position - transform.position;

directionVector.y = 0.0f;

runAwayVector = directionVector.normalized \* -fleeData.fleeDistance;

fleeVector = transform.position + runAwayVector;

*// Check Walls*

RaycastHit hit;

**do**

{

bool isHittedWall = **true**;

**if** (Physics.Raycast

(transform.position, runAwayVector.normalized, **out** hit, fleeData.fleeDistance))

{

**if** (hit.collider.tag == "Wall")

{

runAwayVector = Random.onUnitSphere;

}

**else** isHittedWall = **false**;

}

**else** isHittedWall = **false**;

*// Check NavMesh*

**if** (!isHittedWall)

{

NavMeshHit navMeshHit;

**if** (NavMesh.SamplePosition(fleeVector, **out** navMeshHit, 1.0f, NavMesh.AllAreas))

{

*// Found Runaway Point*

agent.destination = navMeshHit.position;

agent.speed = fleeData.fleeSpeed;

StartCoroutine(FinishUsingFlee());

**break**;

}

}

attemptFlee++;

} **while** (attemptFlee < constMaxAttempts);

}

**private** IEnumerator FinishUsingFlee()

{

**yield** **return** **new** WaitForSeconds(fleeData.duration - fleeData.animationHold);

agent.speed = statData.speed;

}

}

}

**4. Behaviour Tree**

4-1. Behaviour Tree & Node

A screenshot of a computer

Description automatically generated

- Tree

MonsterBehaviourTree의 부모 클래스로 root 노드 설정 추상함수와 Update마다 root 노드를 Evaluate한다

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **abstract** **class** **Tree** : MonoBehaviour

{

**protected** Node root = **null**;

**private** **void** Update()

{

**if** (root != **null**) root.Evaluate();

}

**protected** **abstract** **void** SetupTree(Node root);

}

}

- MonsterBehaviourTree

행동트리에서는 Dictionary를 통해 3가지 정보를 관리한다.

1. 몬스터가 가지고 있는 모든 Component 관리

2. 몬스터의 기본 Stat과 공격 정보 관리

3. 행동 트리에서 필요한 boolean 변수, 플레이어와의 거리 등 행동트리 내에서 사용되고 지속적으로 update되어야 하는 변수 관리

모든 MonsterNode는 MonsterBehaviourTree를 변수로 가지고 있으며, 각 몬스터가 가지고 있는 MonsterNode는 동일한 MonsterBehaviourTree를 공유한다.

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterBehaviourTree** : Tree

{

**protected** Dictionary<MonsterComponentData, object> monsterComponentData = **new**();

**protected** Dictionary<BTData, object> BtData = **new**();

**protected** Dictionary<MonsterData, object> monsterData;

**protected** MonsterBehaviourTree()

{

InitialComponentData();

InitialBTData();

}

**protected** **override** **void** SetupTree(Node root)

{

**this**.root = root;

}

*// >> : ComponentData*

**private** **void** InitialComponentData()

{

NavMeshAgent agent = GetComponent<NavMeshAgent>();

monsterComponentData.Add(MonsterComponentData.AGENT, agent);

MonsterBase monster = GetComponent<MonsterBase>();

monsterComponentData.Add(MonsterComponentData.MONSTER, monster);

MonsterParticleController particle = GetComponent<MonsterParticleController>();

monsterComponentData.Add(MonsterComponentData.PARTICLE, particle);

AudioSource audioSource = GetComponent<AudioSource>();

monsterComponentData.Add(MonsterComponentData.AUDIO, audioSource);

MonsterAudioController audioController = GetComponent<MonsterAudioController>();

monsterComponentData.Add(MonsterComponentData.AUDIO\_CON, audioController);

Transform transform = GetComponent<Transform>();

monsterComponentData.Add(MonsterComponentData.TRANSFORM, transform);

Animator animator = GetComponent<Animator>();

monsterComponentData.Add(MonsterComponentData.ANIMATOR, animator);

Transform patrolPoints = transform.Find("PatrolPoints");

monsterComponentData.Add(MonsterComponentData.PATROL\_POINTS, patrolPoints);

DetectAI detectPlayer = transform.Find("DetectPlayerAI").GetComponent<DetectAI>();

monsterComponentData.Add(MonsterComponentData.PlayerDetectAI, detectPlayer);

DetectAI detectChase = transform.Find("DetectChaseAI").GetComponent<DetectAI>();

monsterComponentData.Add(MonsterComponentData.ChaseDetectAI, detectChase);

Vector3[] patorlPoints = transform.Find("PatrolPoints").GetComponent<PatrolPoints>().GetPatrolPointst();

monsterComponentData.Add(MonsterComponentData.PATROL\_POINTS, patrolPoints);

}

**public** T GetComponentData<T>(MonsterComponentData data)

{

object obj = **null**;

**if** (monsterComponentData.TryGetValue(data, **out** obj))

{

**if** (obj **is** T) **return** (T)obj;

}

**else**

{

Debug.LogFormat("Trying To Get Type Does Not Match : {0}, {1} " + data.ToString(), transform.name);

**return** default(T);

}

Debug.LogFormat(gameObject.name + "Trying To Access Object Does Not Have : {0}, {1}" + data.ToString(), transform.name);

**return** default(T);

}

*// >> : BTData*

**private** **void** InitialBTData()

{

SetBTData<bool>(BTData.bOnSpawnPosition, **true**);

SetBTData<bool>(BTData.bOvertraveld, **false**);

SetBTData<bool>(BTData.bReturning, **false**);

}

**public** T GetBTData<T>(BTData data)

{

object obj;

**if** (BtData.TryGetValue(data, **out** obj))

{

**if** (obj **is** T) **return** (T)obj;

**else**

{

Debug.LogFormat("Trying To Get Component Does Not Match : {0}, {1} " + data.ToString(), transform.name);

**return** default(T);

}

}

**else**

{

T defaultValue = **default**(T);

BtData.Add(data, defaultValue);

**return** defaultValue;

}

}

**public** **void** SetBTData<T>(BTData data, T **value**)

{

**if** (BtData.ContainsKey(data))

{

**if** (BtData[data] **is** T)

{

BtData[data] = **value**;

**return**;

}

**else**

{

Debug.LogFormat("Trying To Get Data Does Not Match : {0}, {1} " + data.ToString(), transform.name);

**return**;

}

}

Debug.LogFormat("Trying to access data that does not exist: {0}, {1}", data.ToString(), transform.name);

}

*// >> : MonsterData*

**public** T GetMonsterData<T>(MonsterData data)

{

object obj;

**if** (monsterData.TryGetValue(data, **out** obj))

{

**if** (obj **is** T) **return** (T)obj;

**else**

{

Debug.LogFormat("Trying To Get Monster Data Does Not Match : {0}, {1} " + data.ToString(), transform.name);

**return** default(T);

}

}

Debug.LogFormat(gameObject.name + "Trying To Access Monster Data Does Not Have : {0}, {1}" + data.ToString(), transform.name);

**return** default(T);

}

**public** **void** AddMonsterData<T>(MonsterData key, T **value**)

{

**if**(monsterData.ContainsKey(key))

{

Debug.LogFormat("{0} Is Trying To Add Key {1} Already Exists : {0}, {1}", transform.name, key.ToString());

**return**;

}

monsterData.Add(key, **value**);

}

}

}

- Node

MonsterNode의 부모 클래스로 ChildNode관리

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **enum** NodeState

{

RUNNING,

SUCCESS,

FAILURE,

}

**public** **class** **Node**

{

**protected** NodeState state;

**protected** Transform transform;

**public** Node parent;

**protected** List<Node> children;

**protected** Tree baseTree;

**protected** bool isStarted;

**protected** bool isExited;

**public** Node()

{

parent = **null**;

}

**public** Node(List<Node> children)

{

**foreach** (Node child **in** children)

Attach(child);

}

**public** Node(Node child)

{

Attach(child);

}

**protected** **virtual** **void** OnStart() { }

**protected** **virtual** **void** OnExit() { }

**public** **virtual** NodeState Evaluate() => NodeState.FAILURE;

**protected** **void** SetChildren(List<Node> children)

{

**foreach** (Node child **in** children)

Attach(child);

}

**private** **void** Attach(Node node)

{

node.parent = **this**;

children.Add(node);

}

}

}

- MonsterNode

MonsterBehaviourTree를 가지고 있는 Monster에서 사용되는 Node

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterNode** : Node

{

**protected** MonsterBehaviourTree tree;

**public** MonsterNode() : **base**()

{

tree = baseTree **as** MonsterBehaviourTree;

**if** (tree == **null**)

Debug.LogFormat("Tree DownCasting To MonsterBehaviourTree Failed : {0}", transform.name);

transform = tree.GetComponentData<Transform>(MonsterComponentData.TRANSFORM);

}

**public** MonsterNode(List<Node> children) : **base**(children) { }

**protected** **void** DebugNull(Transform trns, MonsterComponentData component)

{

Debug.LogFormat("{0} Is Trying To Access {1}, Which It Does Not Have : {0} / {1}", trns.name, component.ToString());

}

}

}

4-2. Monster Behaviour Tree

각 몬스터의 BehaviourTree로 만들어진 Sequence들을 필요에 따라 추가한다. 몬스터의 기본 우선순위 행동은 공격->움직임->플레이어 감지 순으로 행동한다.

4-2.1 평범한 스켈레톤

- 설계

A diagram of a computer program

Description automatically generated

- 코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **NormalSkeletonBT** : MonsterBehaviourTree

{

**private** **void** Start()

{

*// Monster Attack Sequences*

Select attackSelect = **new**(**new** List<Node>

{

**new** SeqRunToPlayer(),

**new** SeqMeleeAttack()

});

*// Monster Cant Attack, But Can Move Sequences*

Select movementSelect = **new**(**new** List<Node>

{

**new** SeqOvertravel(),

**new** SeqNearPlayer(),

**new** SeqChasePlayer(),

**new** SeqPatrol(),

});

*// Monster Cant detect player Sequences*

Select noDetectSelect = **new**(**new** List<Node>

{

**new** SeqReturnSpawnPosition(),

**new** SeqNoDetection(),

});

*// Combine All Sequences*

Select allSequence = **new**(**new** List<Node>

{

attackSelect,

movementSelect,

noDetectSelect,

});

*// Combine Update Data*

Sequence normalSkeletonNodes = **new**(**new** List<Node>

{

**new** ActionUpdateData(),

allSequence,

});

SetupTree(**new** Repeater(normalSkeletonNodes));

}

}

}

4-2.2 모험가 스켈레톤

- 설계

A diagram of a software system

Description automatically generated

- 코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **AdventureSkeletonBT** : MonsterBehaviourTree

{

**private** **void** Start()

{

Select attackSelect = **new**(**new** List<Node>

{

**new** SeqRunToPlayer(),

**new** SeqWeaponAttack(),

**new** SeqMeleeAttack()

});

Select movementSelect = **new**(**new** List<Node>

{

**new** SeqOvertravel(),

**new** SeqNearPlayer(),

**new** SeqChasePlayer(),

**new** SeqPatrol(),

});

Select noDetectSelect = **new**(**new** List<Node>

{

**new** SeqReturnSpawnPosition(),

**new** SeqNoDetection(),

});

Select allSequence = **new**(**new** List<Node>

{

attackSelect,

movementSelect,

noDetectSelect,

});

Sequence adventureSkeletonNodes = **new**(**new** List<Node>

{

**new** ActionUpdateData(),

allSequence,

});

SetupTree(**new** Repeater(adventureSkeletonNodes));

}

}

}

4-2.3 마법사 스켈레톤

- 설계

A diagram of a computer program

Description automatically generated

- 코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **WizardSkeletonBT** : MonsterBehaviourTree

{

**private** **void** Start()

{

Select attackSelect = **new**(**new** List<Node>

{

**new** SeqFleeSkill(),

**new** SeqProjectileAttack(),

});

Select movementSelect = **new**(**new** List<Node>

{

**new** SeqOvertravel(),

**new** SeqChasePlayer(),

**new** SeqPatrol(),

});

Select noDetectSelect = **new**(**new** List<Node>

{

**new** SeqReturnSpawnPosition(),

**new** SeqNoDetection(),

});

Select allSequence = **new**(**new** List<Node>

{

attackSelect,

movementSelect,

noDetectSelect,

});

Sequence wizardSkeletonNodes = **new**(**new** List<Node>

{

**new** ActionUpdateData(),

allSequence,

});

SetupTree(**new** Repeater(wizardSkeletonNodes));

}

}

}

4-2.4 공대원 스켈레톤

- 설계

A diagram of a software system

Description automatically generated

- 코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **GuildGuardSkeletonBT** : MonsterBehaviourTree

{

**private** **void** Start()

{

Select attackSelect = **new**(**new** List<Node>

{

**new** SeqRunToPlayer(),

**new** SeqWeaponAttack(),

**new** SeqMeleeAttack()

});

Select movementSelect = **new**(**new** List<Node>

{

**new** SeqOvertravel(),

**new** SeqNearPlayer(),

**new** SeqChasePlayer(),

**new** SeqPatrol(),

});

Select noDetectSelect = **new**(**new** List<Node>

{

**new** SeqReturnSpawnPosition(),

**new** SeqNoDetection(),

});

Sequence allSequence = **new**(**new** List<Node>

{

attackSelect,

movementSelect,

noDetectSelect,

});

Sequence guildguardSkeletonNodes = **new**(**new** List<Node>

{

**new** ActionUpdateData(),

allSequence,

});

SetupTree(**new** Repeater(guildguardSkeletonNodes));

}

}

}

4-3. Sequence Behaviour Tree

4-3.1 플레이어 미감지

- ReturnSpawnPosition

플레이어가 감지된 상태에서 다시 감지하지 못할 때 자신의 Spawn 위치로 돌아가는 Sequence

설계

A diagram of a sequence

Description automatically generated

코드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqReturnSpawnPosition** : Sequence

{

**public** SeqReturnSpawnPosition()

{

List<Node> children = **new**()

{

**new** Inverter(**new** ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOnSpawnPosition))),

**new** ActionPlayAudio(MonsterAudioType.Move1, **true**, **true**),

**new** ActionPlayAnimation(AnimationType.WALK),

**new** ActionSetAgent(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition)),

**new** ActionReturnSpawnPosition(),

**new** ActionSetBoolean(BTData.bOnSpawnPosition, **true**),

};

SetChildren(children);

}

}

}

- NoDetection

플레이어를 감지하지 못한 상태에서 대기와 대기 중 플레이어 감지 했을 때의 행동 Sequence

설계

A diagram of a computer flowchart

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqNoDetection** : Sequence

{

**public** SeqNoDetection()

{

List<Node> children = **new**()

{

*// undetected player*

**new** Inverter(**new** ActionDetect(DetectType.PLAYER)),

**new** ActionPlayAudio(MonsterAudioType.Idle,**false**, **true**),

**new** ActionPlayAnimation(AnimationType.SIT, **true**),

**new** ActionDetect(DetectType.PLAYER),

*// if detected player*

**new** ActionSetBoolean(BTData.bOnSpawnPosition, **false**),

**new** ActionPlayAnimation(AnimationType.STANDUP, **true**),

};

SetChildren(children);

}

}

}

4-3.2 움직임

- Overtravel

몬스터가 spawn위치 기준으로 허용된 범위 밖으로 나갈 때 자신의 Spawn위치로 돌아오는 Sequence

설계

A diagram of a sequencer

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqOvertravel** : Sequence

{

**public** SeqOvertravel()

{

List<Node> children = **new**()

{

**new** ActionAssertBoolean(tree.GetBTData<bool>(BTData.bOvertraveld)),

**new** ActionPlayAudio(MonsterAudioType.Move1),

**new** ActionPlayAnimation(AnimationType.WALK),

**new** SeqReturnSpawnPosition(),

**new** ActionPlayAnimation(AnimationType.SIT),

**new** ActionSetBoolean(BTData.bOnSpawnPosition, **true**),

**new** ActionSetBoolean(BTData.bOvertraveld, **false**),

};

SetChildren(children);

}

}

}

- NearPlayer

설정된 범위 안에 플레이어가 있을 때 플레이어를 바라보게 돌아주는 Sequence

설계

A diagram of a sequencer

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqNearPlayer** : Sequence

{

**public** SeqNearPlayer()

{

List<Node> children = **new**()

{

**new** ActionCheckPlayerIsNear(),

**new** ActionPlayAudio(MonsterAudioType.IdleAttack1),

**new** ActionPlayAnimation(AnimationType.WALK),

**new** ActionLookPlayer(),

**new** ActionPlayAnimation(AnimationType.IDLE\_ATTACK),

};

SetChildren(children);

}

}

}

- ChasePlayer

플레이어를 쫓아가는 Sequence

설계

A diagram of a sequencer

Description automatically generated

코드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqChasePlayer** : Sequence

{

**public** SeqChasePlayer()

{

List<Node> children = **new**()

{

**new** ActionDetect(DetectType.CHASE),

**new** ActionPlayAudio(MonsterAudioType.Move1),

**new** ActionPlayAnimation(AnimationType.WALK),

**new** ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),

};

SetChildren(children);

}

}

}

Patrol

플레이어가 감지 되었으나 쫓아가는 감지 범위 밖에 있을 때 미리 설정된 Point들을 정찰하는 Sequence

설계

A diagram of a network

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqPatrol** : Sequence

{

**public** SeqPatrol()

{

List<Node> children = **new**()

{

**new** ActionDetect(DetectType.PLAYER),

**new** ActionPlayAudio(MonsterAudioType.Move1),

**new** ActionPlayAnimation(AnimationType.WALK),

**new** ActionPatrolToPoint(),

**new** ActionPlayAnimation(AnimationType.IDLE),

**new** ActionHold(1.0f),

};

SetChildren(children);

}

}

}

4-3.3 공격

몬스터의 공격 가능 여부 (플레이어와의 거리, 공격의 Interval)을 확인 후 공격 실행한다. 공격 실행은 해당 공격의 ExecuteAttack함수를 호출하여 행동트리가 아닌 몬스터 공격 script에서 실행된다. 행동트리는 해당 공격의 Audio와 Animation을 필요할 때 실행시켜 준다.

- RunToPlayer

설계

A diagram of a sequencer

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqRunToPlayer** : Sequence

{

**public** SeqRunToPlayer()

{

RunToPlayerData runToPlayer = tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);

List<Node> children = **new**()

{

**new** ActionSkillUseable(runToPlayer.interval, runToPlayer.attackableDistance),

**new** ActionPlayAudio(MonsterAudioType.MoveSkill),

**new** ActionPlayAnimation(AnimationType.RUN),

**new** ActionRunToPlayer()

};

SetChildren(children);

}

}

}

- MeleeAttack

설계

A diagram of a diagram

Description automatically generated

코드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqMeleeAttack** : Sequence

{

**public** SeqMeleeAttack()

{

MeleeAttackData meleeAttack = tree.GetMonsterData<MeleeAttackData>(MonsterData.Melee);

List<Node> children = **new**()

{

**new** ActionSkillUseable(meleeAttack.interval, meleeAttack.attackableDistance),

**new** ActionStop(),

**new** ActionPlayAudio(MonsterAudioType.MeleeAttack,**true**, **false**),

**new** ActionPlayAnimation(AnimationType.MELEE),

**new** ActionHold(meleeAttack.animationHold),

**new** ActionPlayParticle(MonsterParticleType.MeleeAttack),

**new** ActionExecuteAttack(MonsterSkill.Melee),

**new** ActionHold(meleeAttack.duration-meleeAttack.animationHold),

**new** ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),

};

SetChildren(children);

}

}

}

- WeaponAttack

설계

A diagram of a diagram

Description automatically generated

코드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqWeaponAttack** : Sequence

{

**public** SeqWeaponAttack()

{

WeaponAttackData weaponData = tree.GetMonsterData<WeaponAttackData>(MonsterData.Weapon);

List<Node> children = **new**()

{

**new** ActionSkillUseable(weaponData.interval, weaponData.attackableDistance),

**new** ActionStop(),

**new** ActionPlayAudio(MonsterAudioType.WeaponAttackCast),

**new** ActionPlayAnimation(AnimationType.WEAPON),

**new** ActionHold(weaponData.animationHold),

**new** ActionPlayParticle(MonsterParticleType.WeaponSwing),

**new** ActionPlayAudio(MonsterAudioType.WeaponAttackPerform),

**new** ActionExecuteAttack(MonsterSkill.Weapon),

**new** ActionHold(weaponData.duration-weaponData.animationHold),

**new** ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),

};

SetChildren(children);

}

}

}

- FleeSkill

설계

A diagram of a sequencer

Description automatically generated

코드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqFleeSkill** : Sequence

{

**public** SeqFleeSkill()

{

FleeSkillData fleeData = tree.GetMonsterData<FleeSkillData>(MonsterData.Flee);

List<Node> children = **new**()

{

**new** ActionSkillUseable(fleeData.interval, fleeData.attackableDistance),

**new** ActionPlayAudio(MonsterAudioType.Move2),

**new** ActionPlayAnimation(AnimationType.FLEE),

**new** ActionHold(fleeData.animationHold),

**new** ActionExecuteAttack(MonsterSkill.Flee),

**new** ActionHold(fleeData.duration-fleeData.animationHold),

};

SetChildren(children);

}

}

}

- ProjectileAttack

설계

A diagram of a projector

Description automatically generated

코드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **SeqProjectileAttack** : Sequence

{

**public** SeqProjectileAttack()

{

ProjectileAttackData data = tree.GetMonsterData<ProjectileAttackData>(MonsterData.Projectile);

List<Node> children = **new**()

{

**new** ActionSkillUseable(data.interval, data.attackableDistance),

**new** ActionStop(),

**new** ActionPlayAudio(MonsterAudioType.ProjectileAttack, **true**, **false**),

**new** ActionPlayAnimation(AnimationType.PROJECTILE\_CAST,**true**),

**new** ActionPlayAnimation(AnimationType.PROJECTILE\_FIRE, **false**),

**new** ActionHold(data.animationHold),

**new** ActionPlayAudio(MonsterAudioType.ProjectileFire),

**new** ActionExecuteAttack(MonsterSkill.Projectile),

**new** ActionSetAgent(tree.GetBTData<Vector3>(BTData.v3PlayerPosition)),

};

SetChildren(children);

}

}

}

4-4. ActionNode

4-4.1 공통

- ActionAssertBoolean

Boolean이 true면 성공, fail이면 실패를 반환하며 반대로 사용하는 방법은 Invertal과 함께 사용한다

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionAssertBoolean** : MonsterNode

{

**private** bool boolean;

**public** ActionAssertBoolean(bool boolean)

{

**this**.boolean = boolean;

}

**public** **override** NodeState Evaluate()

{

**if** (boolean) **return** NodeState.SUCCESS;

**else** **return** NodeState.FAILURE;

}

}

}

- ActionDetect

몬스터가 플레이어를 인지, 추적 감지 여부를 판단하여 성공, 실패를 반환한다.

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionDetect** : MonsterNode

{

**private** DetectType type;

**public** ActionDetect(DetectType type)

{

**this**.type = type;

}

**public** **override** NodeState Evaluate()

{

**switch** (type)

{

**case** DetectType.PLAYER:

**if** (tree.GetComponentData<DetectAI>(MonsterComponentData.PlayerDetectAI).IsDetected)

**return** NodeState.SUCCESS;

**break**;

**case** DetectType.CHASE:

**if** (tree.GetComponentData<DetectAI>(MonsterComponentData.ChaseDetectAI).IsDetected)

**return** NodeState.SUCCESS;

**break**;

}

**return** NodeState.FAILURE;

}

}

}

- ActionHold

몬스터가 다음 ActionNode를 실행하기 전 대기해야 할 때 사용된다.

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionHold** : MonsterNode

{

**private** float holdTime;

**private** float accumTime=0;

**public** ActionHold(float holdTime= 1.0f) : **base**()

{

**this**.holdTime = holdTime;

}

**protected** **override** **void** OnStart()

{

accumTime = 0;

}

**public** **override** NodeState Evaluate()

{

**if** (accumTime >= holdTime) **return** NodeState.SUCCESS;

accumTime += Time.deltaTime;

**return** NodeState.RUNNING;

}

}

}

- ActionPlayAnimation

몬스터의 Animation을 실행해주는 ActionNode로 Animation을 실행하고 바로 다음 ActionNode로 넘어갈지, 아니면 해당 Animation이 끝날 때 까지 기다릴지 설정해줄 수 있다.

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionPlayAnimation** : MonsterNode

{

Animator animator;

**private** AnimationType animation;

**private** bool waitToEnd;

**private** float animationLength;

**private** float accumTime;

**public** ActionPlayAnimation(AnimationType animation, bool waitToEnd = **false**)

{

**if**(animator==**null**)

animator = tree.GetComponentData<Animator>(MonsterComponentData.ANIMATOR);

**if** (animator == **null**)

DebugNull(transform, MonsterComponentData.ANIMATOR);

**this**.animation = animation;

**this**.waitToEnd = waitToEnd;

}

**protected** **override** **void** OnStart()

{

animator.SetTrigger(animation.ToString());

**if** (waitToEnd)

{

accumTime = 0;

AnimatorClipInfo[] clipInfo = animator.GetCurrentAnimatorClipInfo(0);

animationLength = clipInfo[0].clip.length;

}

}

**public** **override** NodeState Evaluate()

{

**if**(waitToEnd)

{

**if** (accumTime <= animationLength)

{

accumTime += Time.deltaTime;

**return** NodeState.RUNNING;

}

**else** **return** NodeState.SUCCESS;

}

**return** NodeState.SUCCESS;

}

}

}

- ActionPlayAudio

몬스터의 Audio를 실행해주는 ActionNode로 ScriptableObject에 준비된 Audio를 찾아 실행해준다. 다른 Audio가 실행 중일 때 중단하고 실행할 것인지, 반복재생해줄 것인지 설정해줄 수 있다

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionPlayAudio** : MonsterNode

{

**private** AudioSource audioSource;

**private** MonsterAudioController audioController;

**private** MonsterAudioType audioType;

**private** bool isLoop;

**private** bool isInteruptable;

**public** ActionPlayAudio(MonsterAudioType audioType, bool isInteruptable = **true**, bool isLoop = **true**) : **base**()

{

**if** (audioSource == **null**)

audioSource = tree.GetComponentData<AudioSource>(MonsterComponentData.AUDIO);

**if** (audioSource == **null**)

DebugNull(transform, MonsterComponentData.AUDIO);

**if** (audioController == **null**)

audioController = tree.GetComponentData<MonsterAudioController>(MonsterComponentData.AUDIO\_CON);

**if** (audioController == **null**)

DebugNull(transform, MonsterComponentData.AUDIO\_CON);

**this**.audioType = audioType;

**this**.isLoop = isLoop;

**this**.isInteruptable = isInteruptable;

}

**public** **override** NodeState Evaluate()

{

**if** (isLoop)

audioSource.loop = **true**;

**else** audioSource.loop = **false**;

**if** (!isInteruptable)

{

**if** (audioSource.isPlaying)

**return** NodeState.SUCCESS;

}

AudioClip clip = audioController.GetAudio(audioType);

**if**(clip==**null**)

{

Debug.Log(transform.name + "Try To Play Audio Does Not Have: " + audioType.ToString());

**return** NodeState.FAILURE;

}

audioSource.clip = clip;

audioSource.Play();

**return** NodeState.SUCCESS;

}

}

}

- ActionPlayParticle

Scriptable Object로 준비된 Particle을 실행시켜주는 ActionNode

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionPlayParticle** : MonsterNode

{

**private** MonsterParticleController particleController;

**private** MonsterParticleType particleType;

**public** ActionPlayParticle(MonsterParticleType particleType)

{

particleController = tree.GetComponentData<MonsterParticleController>(MonsterComponentData.PARTICLE);

**this**.particleType = particleType;

}

**public** **override** NodeState Evaluate()

{

particleController.PlayParticle(particleType);

**return** NodeState.SUCCESS;

}

}

}

- ActionSetAgent

몬스터의 NavMeshAgent의 목표 지점을 설정해주는 Node

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionSetAgent** : MonsterNode

{

**private** NavMeshAgent agent;

**private** Vector3 destination;

**public** ActionSetAgent(Vector3 destination)

{

agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);

**if** (agent == **null**)

DebugNull(transform, MonsterComponentData.AGENT);

**this**.destination = destination;

}

**public** **override** NodeState Evaluate()

{

agent.destination = destination;

**return** NodeState.SUCCESS;

}

}

}

- ActionSetBoolean

MonsterBehaviourTree에 있는 Tree에서 사용되는 정보들 중 Boolean을 설정할 수 있는 Node

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionSetBoolean** : MonsterNode

{

**private** BTData data;

**private** bool boolean;

**public** ActionSetBoolean(BTData data, bool boolean)

{

**this**.data = data;

**this**.boolean = boolean;

}

**public** **override** NodeState Evaluate()

{

tree.SetBTData<bool>(data, boolean);

**return** NodeState.SUCCESS;

}

}

}

- ActionStop

몬스터가 공격 등 움직임 말고 다른 행동을 실행할 때 몬스터가 자리에 멈추게 하는 Node

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionStop** : MonsterNode

{

**private** NavMeshAgent agent;

**public** ActionStop()

{

agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);

}

**public** **override** NodeState Evaluate()

{

agent.destination = transform.position;

**return** NodeState.SUCCESS;

}

}

}

- ActionUpdateData

지속적으로 Update되어야 하는 정보들을 관리하는 Node

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionUpdateData** : MonsterNode

{

**public** ActionUpdateData() { }

**public** **override** NodeState Evaluate()

{

Vector3 playerPosition = tree.GetMonsterData<GameObject>(MonsterData.v3SpawnPosition).transform.position;

tree.SetBTData<Vector3>(BTData.v3PlayerPosition, playerPosition);

float playerDistance = Vector3.SqrMagnitude(playerPosition - transform.position);

tree.SetBTData<float>(BTData.fPlayerDistanceSqr, playerDistance);

float spawnDistance = Vector3.SqrMagnitude(tree.GetMonsterData<Vector3>(MonsterData.v3SpawnPosition));

tree.SetBTData<float>(BTData.fSpawnDistanceSqr, spawnDistance);

**if** (spawnDistance > tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).overtravelDist)

tree.SetBTData<bool>(BTData.bOvertraveld, **true**);

**return** NodeState.SUCCESS;

}

}

}

4-4.2 플레이어 미감지

- ActionReturnSpawnPosition

몬스터가 Spawn된 위치로 복귀를 완료 하였는지 확인하는 Node

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionReturnSpawnPosition** : MonsterNode

{

**public** **override** NodeState Evaluate()

{

float distance = tree.GetBTData<float>(BTData.fSpawnDistanceSqr);

**if** (distance > 0.5f)

**return** NodeState.SUCCESS;

**else** **return** NodeState.RUNNING;

}

}

}

4-4.3 움직임

- ActionCheckPlayerIsNear

플레이어가 몬스터의 설정된 범위 안에 들어왔는지 확인하는 Node

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionCheckPlayerIsNear** : MonsterNode

{

**public** **override** NodeState Evaluate()

{

float playerDistance = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);

float playerStopDistance = tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).stopDistance;

**if** (playerDistance < playerStopDistance)

**return** NodeState.SUCCESS;

**return** NodeState.FAILURE;

}

}

}

- ActionLookPlayer

몬스터가 플레이어를 향해 바라보는 Node

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionLookPlayer** : MonsterNode

{

**private** float accumTime;

**private** **const** float angle = 90.0f;

**private** **const** float radius = 5.0f;

**public** **override** NodeState Evaluate()

{

Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);

Vector3 monsterPos = transform.position;

Vector3 interV = playerPos - monsterPos;

float dot = Vector3.Dot(interV.normalized, transform.forward.normalized);

float theta = Mathf.Acos(dot);

float degree = Mathf.Rad2Deg \* theta;

**if** (degree <= angle / 2.0f)

{

interV.y = 0;

**if** (interV.sqrMagnitude <= radius \* radius)

**return** NodeState.SUCCESS;

}

**else**

{

Quaternion targetRotation;

Vector3 directionVector = interV;

directionVector.y = 0;

directionVector.Normalize();

targetRotation = Quaternion.LookRotation(directionVector, Vector3.up);

transform.rotation= Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime \* 90.0f);

accumTime += Time.deltaTime;

**if** (accumTime >= 0.5f) **return** NodeState.FAILURE;

}

**return** NodeState.RUNNING;

}

}

}

- ActionPatrolToPoints

미리 설정된 Point들에 몬스터가 차례로 방문하며 정찰하는 Node

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionPatrolToPoint** : MonsterNode

{

NavMeshAgent agent;

Vector3[] patrolPoints;

**private** int patrolNum;

**private** int patrolQuant;

**public** ActionPatrolToPoint()

{

**if** (patrolPoints == **null**)

patrolPoints = tree.GetComponentData<Vector3[]>(MonsterComponentData.PATROL\_POINTS);

**if** (patrolPoints == **null**)

DebugNull(transform, MonsterComponentData.PATROL\_POINTS);

**if** (agent == **null**)

agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);

**if** (agent == **null**)

DebugNull(transform, MonsterComponentData.AGENT);

patrolQuant = patrolPoints.Length;

patrolNum = 0;

}

**public** **override** NodeState Evaluate()

{

agent.destination = patrolPoints[patrolNum];

float distance = Vector3.SqrMagnitude(transform.position - patrolPoints[patrolNum]);

**if**(distance<0.1f)

{

patrolNum++;

**if** (patrolNum == patrolQuant)

patrolNum = 0;

**return** NodeState.SUCCESS;

}

**return** NodeState.RUNNING;

}

}

}

4-4.4 공격

- ActionExecuteAttack

몬스터가 가지고 있는 공격을 실행시켜주는 Node

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionExecuteAttack** : MonsterNode

{

MonsterSkill monsterSkill;

MonsterBase monster;

**public** ActionExecuteAttack(MonsterSkill monsterSkill)

{

**if**(monster==**null**)

monster = tree.GetComponentData<MonsterBase>(MonsterComponentData.MONSTER);

**if** (monster == **null**)

DebugNull(transform, MonsterComponentData.MONSTER);

**this**.monsterSkill = monsterSkill;

}

**public** **override** NodeState Evaluate()

{

**if** (monster.ExecuteAttack(monsterSkill))

**return** NodeState.SUCCESS;

Debug.LogFormat("{0} Is Trying To Attack With {1} That Does Not have : {0}, {1}", transform.name, monsterSkill.ToString());

**return** NodeState.FAILURE;

}

}

}

- ActionRunToPlayer

몬스터가 플레이어를 향해 달려가는, agent의 속도를 변경해주는 Node

**using** **UnityEngine**;

**using** **UnityEngine.AI**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionRunToPlayer** : MonsterNode

{

**private** RunToPlayerData skill;

**private** NavMeshAgent agent;

**private** float originalSpeed;

**private** float accumTime;

**public** ActionRunToPlayer()

{

skill = tree.GetMonsterData<RunToPlayerData>(MonsterData.RunToPlayer);

**if** (agent == **null**)

agent = tree.GetComponentData<NavMeshAgent>(MonsterComponentData.AGENT);

**if** (agent == **null**)

DebugNull(transform, MonsterComponentData.AGENT);

originalSpeed = tree.GetMonsterData<MonsterStatData>(MonsterData.MonsterStat).speed;

accumTime = 0.0f;

}

**protected** **override** **void** OnStart()

{

agent.speed = skill.speed;

isStarted = **true**;

}

**public** **override** NodeState Evaluate()

{

**if** (!isStarted)

OnStart();

Vector3 playerPos = tree.GetBTData<Vector3>(BTData.v3PlayerPosition);

agent.destination = playerPos;

float distance = Vector3.SqrMagnitude(playerPos - transform.position);

**if**(accumTime<skill.duration)

{

**if** (distance < skill.stopDistance)

{

OnExit();

**return** NodeState.SUCCESS;

}

accumTime += Time.deltaTime;

**return** NodeState.RUNNING;

}

OnExit();

**return** NodeState.SUCCESS;

}

**protected** **override** **void** OnExit()

{

agent.speed = originalSpeed;

accumTime = 0.0f;

isStarted = **false**;

}

}

}

- ActionSkillUseable

몬스터의 스킬 사용 가능 여부를 플레이어와의 거리, 쿨타임을 기준으로 확인하는 Node

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **ActionSkillUseable** : MonsterNode

{

float attackableDistance = 0.0f;

float interval = 0.0f;

**private** float lastTime;

**public** ActionSkillUseable(float interval, float attackableDistance)

{

**this**.interval = interval;

**this**.attackableDistance = attackableDistance;

lastTime = 0.0f;

}

**public** **override** NodeState Evaluate()

{

float playerDistanceSqr = tree.GetBTData<float>(BTData.fPlayerDistanceSqr);

**if** (playerDistanceSqr > attackableDistance)

**return** NodeState.FAILURE;

**if** (Time.time - lastTime < interval)

**return** NodeState.FAILURE;

lastTime = Time.time;

**return** NodeState.SUCCESS;

}

}

}

4-5. Composite Node

- Select

하위 노드들 중 하나가 성공할 때 까지 순차적으로 평가하여 하나가 성공하면 즉시 성공으로 간주되는 노드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **Select** : Node

{

**private** int current;

**public** Select() : **base**() { }

**public** Select(List<Node> childrens) : **base**(childrens) { }

**protected** **override** **void** OnStart()

{

current = 0;

isStarted = **true**;

}

**public** **override** NodeState Evaluate()

{

**if** (!isStarted)

OnStart();

**for**(int i=current;i<children.Count;i++)

{

current = i;

var child = children[current];

**switch** (child.Evaluate())

{

**case** NodeState.FAILURE:

**continue**;

**case** NodeState.SUCCESS:

**return** NodeState.SUCCESS;

**case** NodeState.RUNNING:

**return** NodeState.RUNNING;

**default**:

**continue**;

}

}

**return** NodeState.FAILURE;

}

}

}

- Sequence

하위 노드들을 순차적으로 시행하며 모든 하위 노드가 성공하면 전체가 성공으로 간주되는 노드

**using** **System.Collections.Generic**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **Sequence** : MonsterNode

{

**public** Sequence() : **base**() { }

**public** Sequence(List<Node> childrens) : **base**(childrens) { }

**public** **override** NodeState Evaluate()

{

bool anyChildIsRunning = **false**;

**foreach**(Node child **in** children)

{

**switch**(child.Evaluate())

{

**case** NodeState.FAILURE:

**return** NodeState.FAILURE;

**case** NodeState.SUCCESS:

**return** NodeState.SUCCESS;

**case** NodeState.RUNNING:

anyChildIsRunning = **true**;

**continue**;

**default**:

**return** NodeState.SUCCESS;

}

}

**return** anyChildIsRunning ? NodeState.RUNNING : NodeState.SUCCESS;

}

}

}

4-6. Decorator

- Inverter

하위 노드의 성공과 실패를 반대로 반환하는 노드

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **Inverter** : Node

{

**public** Inverter(Node node) : **base**(node){}

**public** **override** NodeState Evaluate()

{

**switch** (children[0].Evaluate())

{

**case** NodeState.RUNNING:

**return** NodeState.RUNNING;

**case** NodeState.SUCCESS:

**return** NodeState.FAILURE;

**case** NodeState.FAILURE:

**return** NodeState.SUCCESS;

}

**return** NodeState.FAILURE;

}

}

}

- Repeater

하위 노드를 계속해서 반복하는 노드

**using** **System.Collections.Generic**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **Repeater** : Node

{

bool repeatOnSuccess;

bool repeatOnFailure;

**public** Repeater(Node child, bool repeatOnSuccess = **true**, bool repeatOnFailure = **true**) : **base**(child)

{

**this**.repeatOnFailure = repeatOnFailure;

**this**.repeatOnSuccess = repeatOnSuccess;

}

**public** Repeater(List<Node> childrens, bool repeatOnSuccess = **true**, bool repeatOnFailure = **true**) : **base**(childrens)

{

**this**.repeatOnFailure = repeatOnFailure;

**this**.repeatOnSuccess = repeatOnSuccess;

}

**public** **override** NodeState Evaluate()

{

**if** (children.Count != 1)

{

Debug.Log(transform.name + "Repeater Node Has No or More Than 1 Child");

**return** NodeState.FAILURE;

}

**switch** (children[0].Evaluate())

{

**case** NodeState.RUNNING:

**return** NodeState.RUNNING;

**case** NodeState.SUCCESS:

**if** (repeatOnSuccess) **return** NodeState.RUNNING;

**return** NodeState.SUCCESS;

**case** NodeState.FAILURE:

**if** (repeatOnFailure) **return** NodeState.FAILURE;

**return** NodeState.FAILURE;

}

**return** NodeState.RUNNING;

}

}

}

**5. 기타**

5-1. MonsterData Class

**using** **System.Collections.Generic**;

**using** **Channels.Combat**;

**using** **UnityEngine**;

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **class** **MonsterStatData**

{

**public** MonsterType monsterType;

**public** string name;

**public** float HP;

**public** float speed;

**public** float rotationSpeed;

**public** float detectPlayerDist;

**public** float detectChaseDist;

**public** float overtravelDist;

**public** float overtravelReturnSpeed;

**public** float stopDistance;

**public** float weeknessRatio;

**public** float respawnTime;

**public** List<int> itemTableNum;

**public** Vector3 spawnPosition;

}

**public** **class** **RunToPlayerData**

{

**public** float interval;

**public** float attackableDistance;

**public** float speed;

**public** float duration;

**public** float stopDistance;

}

**public** **class** **MeleeAttackData**

{

**public** CombatType combatType;

**public** int damage;

**public** float interval;

**public** float attackableDistance;

**public** float duration;

**public** float animationHold;

**public** float colliderDuration;

**public** Vector3 colliderOffset;

**public** Vector3 colliderSize;

}

**public** **class** **WeaponAttackData**

{

**public** CombatType combatType;

**public** int damage;

**public** float interval;

**public** float attackableDistance;

**public** float duration;

**public** float animationHold;

**public** float angle;

**public** float radius;

}

**public** **class** **FleeSkillData**

{

**public** float fleeSpeed;

**public** float interval;

**public** float attackableDistance;

**public** float duration;

**public** float fleeDistance;

**public** float animationHold;

}

**public** **class** **ProjectileAttackData**

{

**public** CombatType combatType;

**public** string name;

**public** float damage;

**public** float interval;

**public** float attackableDistance;

**public** float duration;

**public** float animationHold;

**public** float projectileSpeed;

}

}

5-2. BehaviourTree Enum

**namespace** **Scripts.BehaviourTrees.Monster**

{

**public** **enum** MonsterComponentData

{

MONSTER,

AGENT,

AUDIO,

AUDIO\_CON,

TRANSFORM,

ANIMATOR,

PATROL\_POINTS,

PARTICLE,

MONSTER\_CENTER,

PlayerDetectAI,

ChaseDetectAI,

}

**public** **enum** BTData

{

bOnSpawnPosition,

bOvertraveld,

bReturning,

fPlayerDistanceSqr,

fSpawnDistanceSqr,

v3PlayerPosition,

iCurrentHP,

}

**public** **enum** MonsterData

{

v3SpawnPosition,

*// Datas*

MonsterStat,

RunToPlayer,

Melee,

Weapon,

Flee,

Projectile,

}

**public** **enum** MonsterSkill

{

RunToPlayer,

Melee,

Projectile,

Weapon,

Flee,

}

**public** **enum** DetectType

{

PLAYER,

CHASE,

}

**public** **enum** AnimationType

{

IDLE,

STANDUP,

WALK,

SIT,

IDLE\_ATTACK,

RUN,

MELEE,

WEAPON,

FLEE,

PROJECTILE\_CAST,

PROJECTILE\_FIRE,

}

**public** **enum** MonsterType

{

NormalSkeleton,

AdventureSekeleton,

WizardSkeleton,

GuildguardSkeleton

}

}