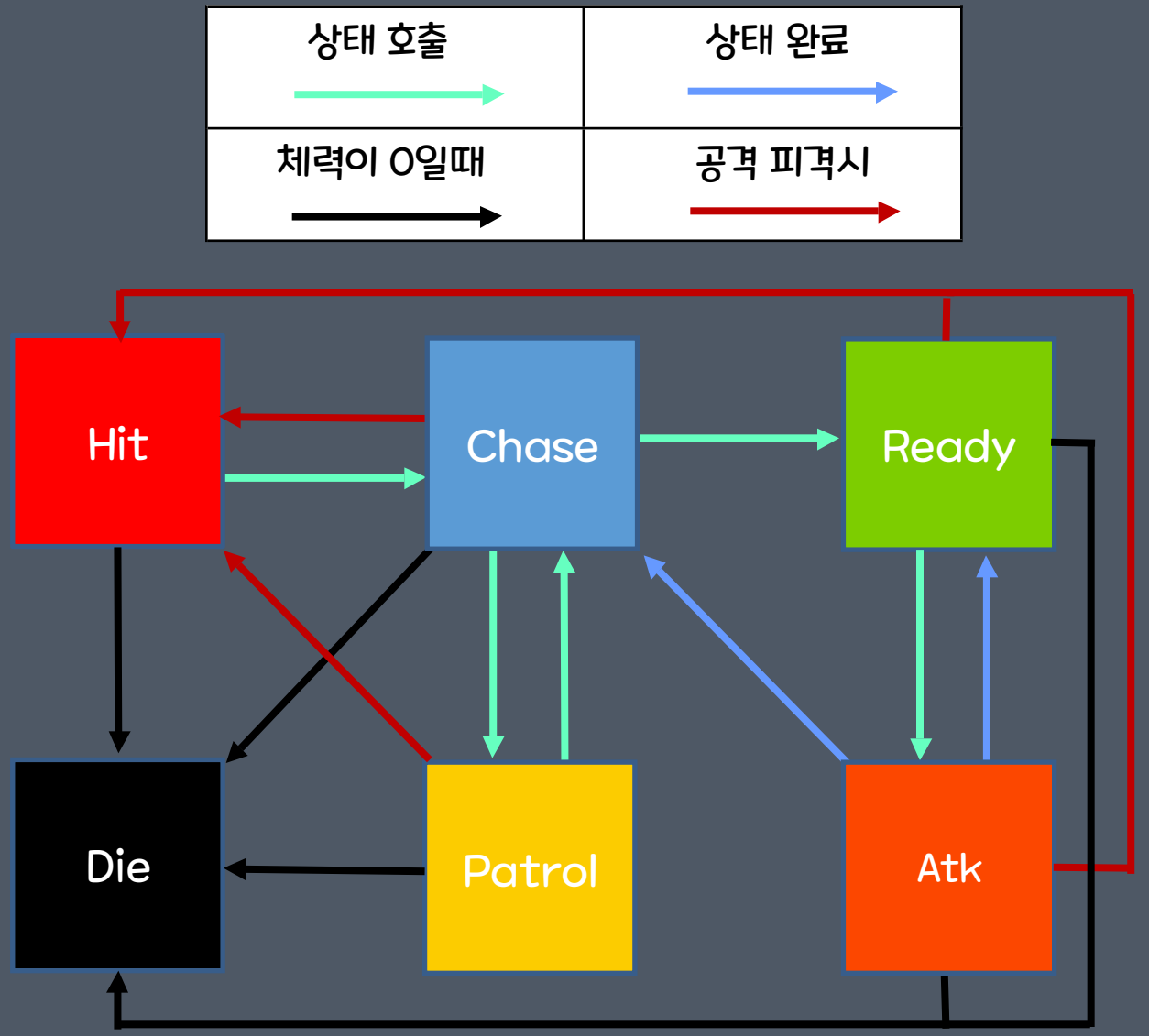
PortFolio

Dragon
Soul

Contents

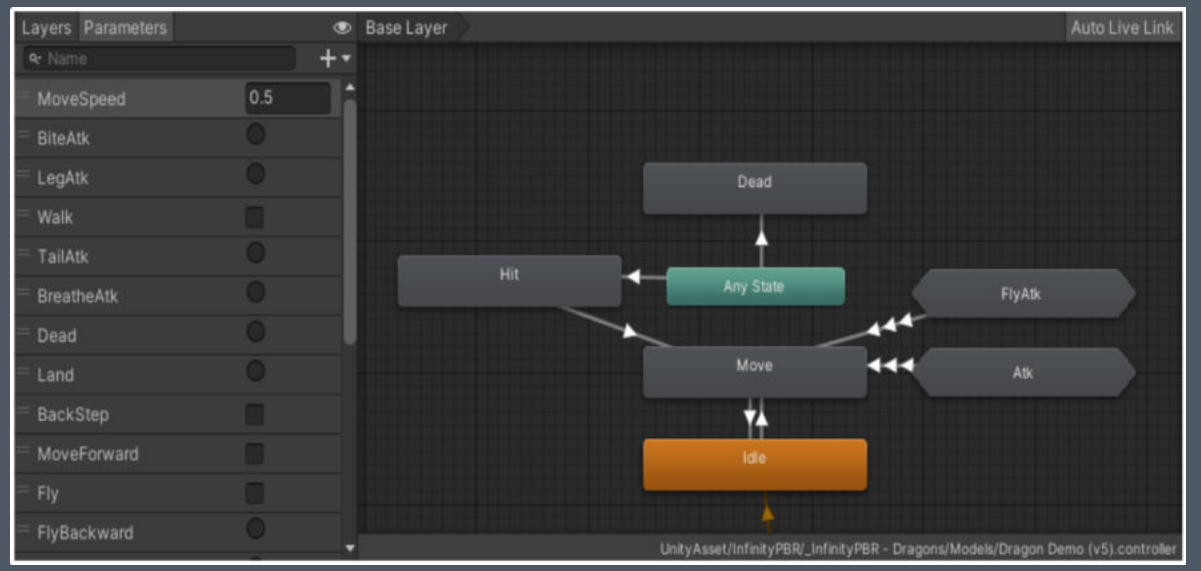
FSM(Dragon)

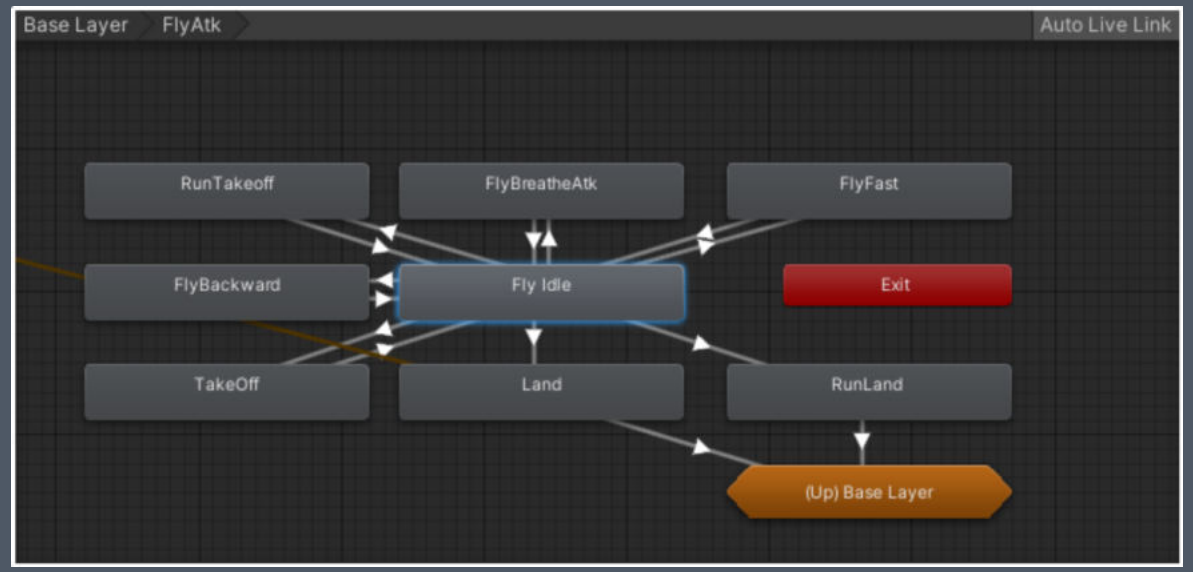
- 드래곤의 FSM 구조
- Patrol과 Chase는 백그라운드 스크립트인 Enemy_Ai 에서
  플레이어와의 거리를 비교해 결정됩니다.
- 드래곤이 공격을 마친 후의 상태는
  드래곤이 공중에 떠있는지, 아닌지에 따라 달라집니다.
  공중에 있을 경우 Ready상태로 전환되고
  지상에 있을 경우 Chase상태로 전환됩니다.
- Hit 및 Die 상태는 어떤 상태에서든
  조건을 충족하면 전환 됩니다.

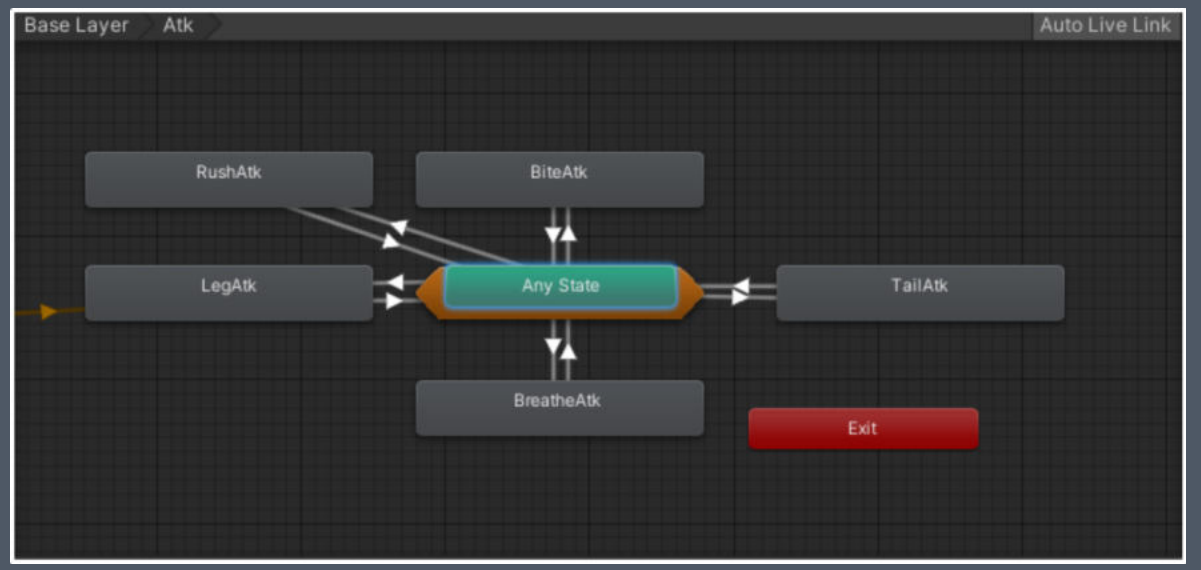


Animator

- 드래곤의 Animator
- 이동, 피격, 사망, 공격 상태가 있으며
  드래곤의 비행여부에 따라 할 수 있는 공격이 달라집니다.
  공중에서 공격이 끝나면 Fly Idle 상태로 전환되고
  지상에서 공격이 끝나면 Move 상태로 전환됩니다.

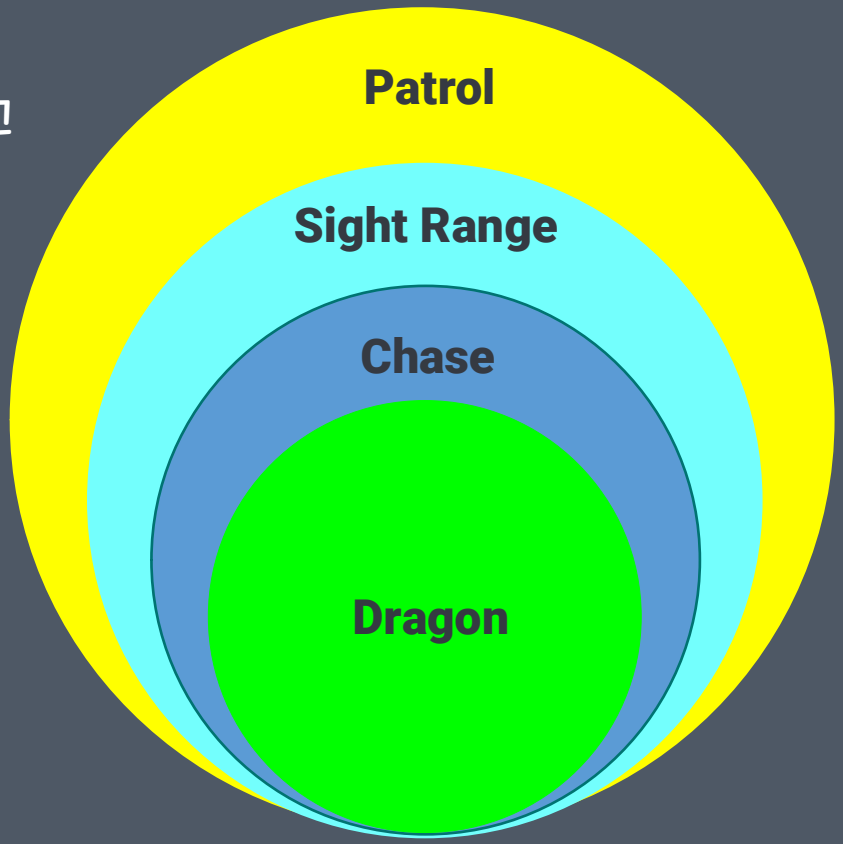






AI

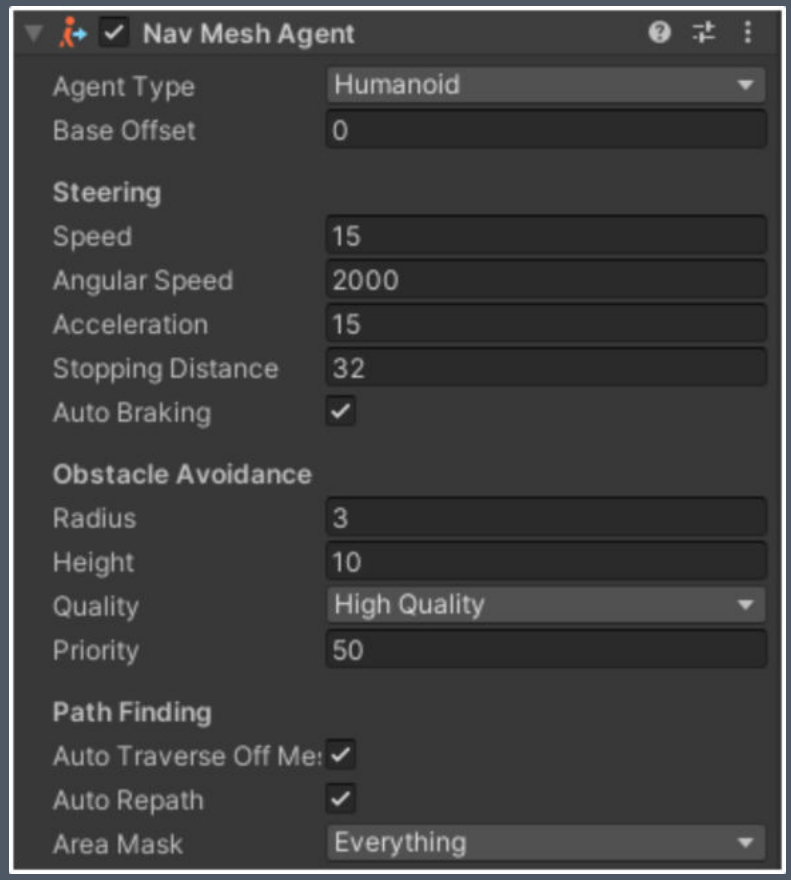
- 드래곤의 AI와 이동은 Nav Mesh Agent 컴포넌트를 기반으로 구현했으며,
  플레이어의 거리차에 따라 상태가 전환되도록 했습니다.
  Patrol 상태일 때 플레이어가 시야범위에 들어오면 Chase상태로 전환 되고
  Chase 상태일 때 시야범위 밖으로 벗어나면 Patrol 상태로 전환됩니다.




```
private void Update()
{
    if (dragon.IsState(Dragon.EnemyState.PATROL))
    {
        float distance = (dragon.target.position - dragon.transform.position).sqrMagnitude;
        if (distance <= dragon.SightRange * dragon.SightRange)
        {
            dragon.ChangeState(Dragon.EnemyState.CHASE);
        }
    }
    else if (dragon.IsState(Dragon.EnemyState.CHASE))
    {
        float distance = (dragon.target.position - dragon.transform.position).sqrMagnitude;
        if (distance >= dragon.SightRange * dragon.SightRange)
        {
            dragon.ChangeState(Dragon.EnemyState.PATROL);
        }
    }
}
```

Nav Mesh

- 드래곤의 이동은 Nav Mesh Agent 컴포넌트를 이용해 구현 했습니다.
- 순찰 상태 일때는 웨이포인트를 목적지로 설정해 주위를 배회하도록 했고,
- 추격 상태 일때는 플레이어를 목적지로 설정해 추격을 하도록 했습니다.



Patrol

- 특정 위치값을 배열로 받아 네비메쉬의 목적지로 설정해서
  드래곤이 계속 배회하도록 했고 배회중
  플레이어가 시야 범위 안에 들어 오면 Chase 상태로 전환됩니다.

```
public void OnUpdate(Dragon dragon)
{
    if (dragon.ArrivePoint==false &&
        dragon.RemainDistance <= dragon.AttackRange)
    {
        GotoNextPoint(dragon);
    }
}
참조 2개
public void GotoNextPoint(Dragon dragon)
{
    if (dragon.WayPoints.Length == 0)
        return;

    dragon.navMeshAgent.destination = dragon.WayPoints[patrolIndex].position;

    patrolIndex = (patrolIndex + 1) % dragon.WayPoints.Length;
}
```

Chase

- 드래곤의 시야 범위에 들어온 플레이어를
  네비메쉬의 목적지로 설정해 계속 추격하고,
  공격 사거리까지 접근 했다면 Ready 상태로 전환됩니다

```
public void OnUpdate(Dragon dragon)
{
    float Distance = (dragon.target.position - dragon.transform.position).magnitude;

    dragon.dragonAnimator.SetFloat("MoveSpeed", dragon.navMeshAgent.velocity.magnitude);

    if (Distance <= dragon.AttackRange &&
                    dragon.Attacking == false)
    {
        dragon.ChangeState(Dragon.EnemyState.READY);
    }
    else
    {
        dragon.navMeshAgent.SetDestination(dragon.target.transform.position);
    }
}
```

Ready

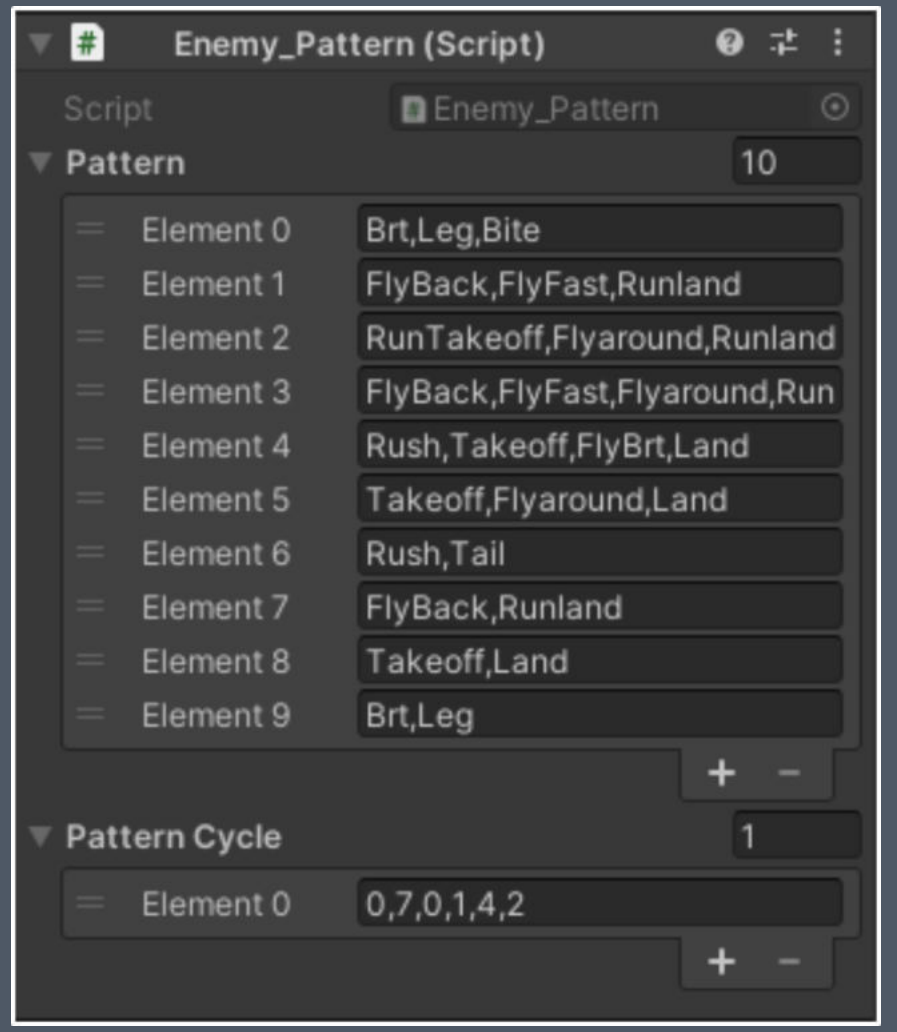
- 공격을 준비하는 상태이며 드래곤의 이동을 멈추고,
  일정 시간 동안 플레이어를 바라보도록 회전 시킨 뒤
  공격 상태로 전환하는 NextState 함수를 실행 시킵니다.

```
public IEnumerator AtkReadyCoroutine(Dragon dragon)
{
    dragon.Attacking = true;

    while (dragon.ActReadyTime >= 0f)
    {
        dragon.ActReadyTime -= Time.deltaTime;
        Rotation(dragon);
        yield return null;
    }
    dragon.Enemy_pattern.NextState(dragon);
    dragon.ActReadyTime = dragon.actReadyTime;
}

public void Rotation(Dragon dragon)
{
    var targetPos = dragon.target.position;
    targetPos.y = dragon.transform.position.y;
    var targetDir = Quaternion.LookRotation(targetPos - dragon.transform.position);
    dragon.transform.rotation = Quaternion.Slerp(dragon.transform.rotation, targetDir,
                                        dragon.RotationSpeed * Time.deltaTime);
}
```

Pattern

- 드래곤의 공격은 미리 설정된 패턴의 순서를 따르며,
  공격을 조합하기 편하도록 리스트 형태로 구현하였습니다.
- 각 공격들의 순서를 정해 하나의 패턴을 만든 뒤,
  패턴의 번호를 사이클에 넣게 되면 입력한 순서대로 실행됩니다.

```
public void NextState(Dragon dragon)
{
    if (Pattern.Length == 0)
    {
        return;
    }
    else if (currentPattern > patternCount - 1)
    {
        currentPattern = 0;
    }

    List<string> currentpattern = ParseCommands(Pattern[Convert.ToInt32(patternStorage[currentPattern])]);
    StartCoroutine(NextStateCoroutine(dragon, currentpattern));
    patternIndex += 1;
}
참조 1개
IEnumerator NextStateCoroutine(Dragon dragon, List<string> currentpattern)
{
    SetState(dragon, currentpattern[patternIndex]);

    if (patternIndex >= currentpattern.Count - 1)
    {
        yield return new WaitForSeconds(0.1f);
        currentPattern += 1;
        patternIndex = 0;
    }
}
```



AtkState

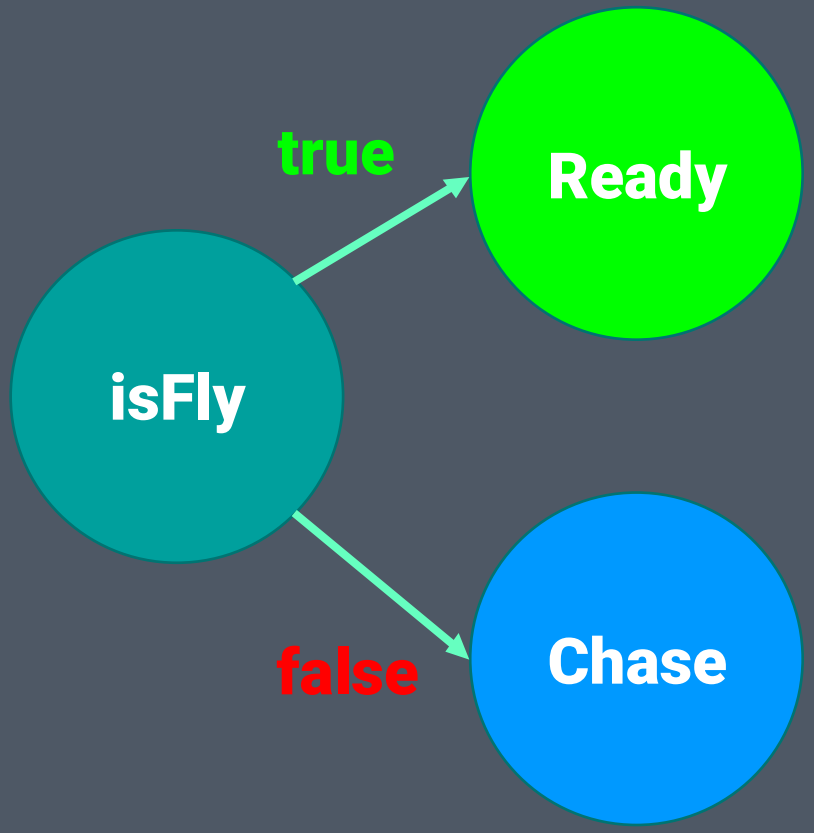
- SetState 함수의 인자값으로 원하는 공격 스테이트를 넣게 되면, 해당 스테이트로 변경되고 고유의 코루틴이 실행되며, 애니메이션의 진행도를 조건으로 설정해 원하는 타이밍에 이펙트, 공격 콜리전등을 활성화 시킬수 있습니다.
- 추가적으로 AnimationEndCheck를 계속 실행시켜 애니메이션이 끝났는지 지속적으로 확인합니다.

```
참조 2개
public void OnUpdate(Dragon dragon)
{
    dragon.enemy_ai.AnimationEndCheck();
}

참조 1개
IEnumerator BiteAtkCoroutine(Dragon dragon)
{
    dragon.a_id = "BiteAtk";
    dragon.dragonAnimator.SetTrigger(dragon.a_id);

    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.4f);

    dragon.BiteAtkColision.SetActive(true);

    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.5f);

    dragon.BiteAtkColision.SetActive(false);
}
```

AI

- Animation EndCheck 함수는
  드래곤의 애니메이션이 끝났는지 확인하는 역할을 합니다.
- IsFly가 true 일때는 Ready상태로 보내줘서
  공중에서의 다음 공격을 바로 준비 할수있게해주고
- IsFly가 false일때는 지상에 있다는 뜻이므로
  Chase로 변경시켜줍니다.

```
public void AnimationEndCheck()
{
    if (dragon.AnimationName && dragon.AnimationProgress >= 0.9f)
    {
        if (dragon.isFly == true)
        {
            dragon.ChangeState(Dragon.EnemyState.READY);
        }
        else
        {
            dragon.ChangeState(Dragon.EnemyState.CHASE);
        }

        dragon.Attacking = false;
    }
}
```



Take Off

- 드래곤이 날기위해서는 특정 애니메이션이 실행되어야 합니다.
  애니메이션 진행도에 따라 IsFly 변수를 true로 바꿔
  공격이 끝나도 Chase상태로 가지않게 해주며,
  드래곤을 공중에 띄워 주기위해 네비메쉬를 비활성화 해주고
  y값이 FlyHeight 값과 같아질 때까지 서서히 증가 시킵니다

```
참조 1개
IEnumerator RunTakeoffCoroutine(Dragon dragon)
{
    dragon.a_id = "RunTakeoff";
    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.1f);
    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.5f);
    GameObject SmokeEffect = ObjectPoolingManager.Instance.GetObject_Noparent("Smoke", dragon.SmokePos);
    dragon.isFly = true;
    ShakeCamera.instance.OnShakeCamera(0.3f, 0.3f);
    dragon.navMeshAgent.enabled = false;
    dragon.StartCoroutine(FlyCoroutine(dragon));
    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.9f);
    ObjectPoolingManager.Instance.ReturnObject("Smoke", SmokeEffect);

}
//
```

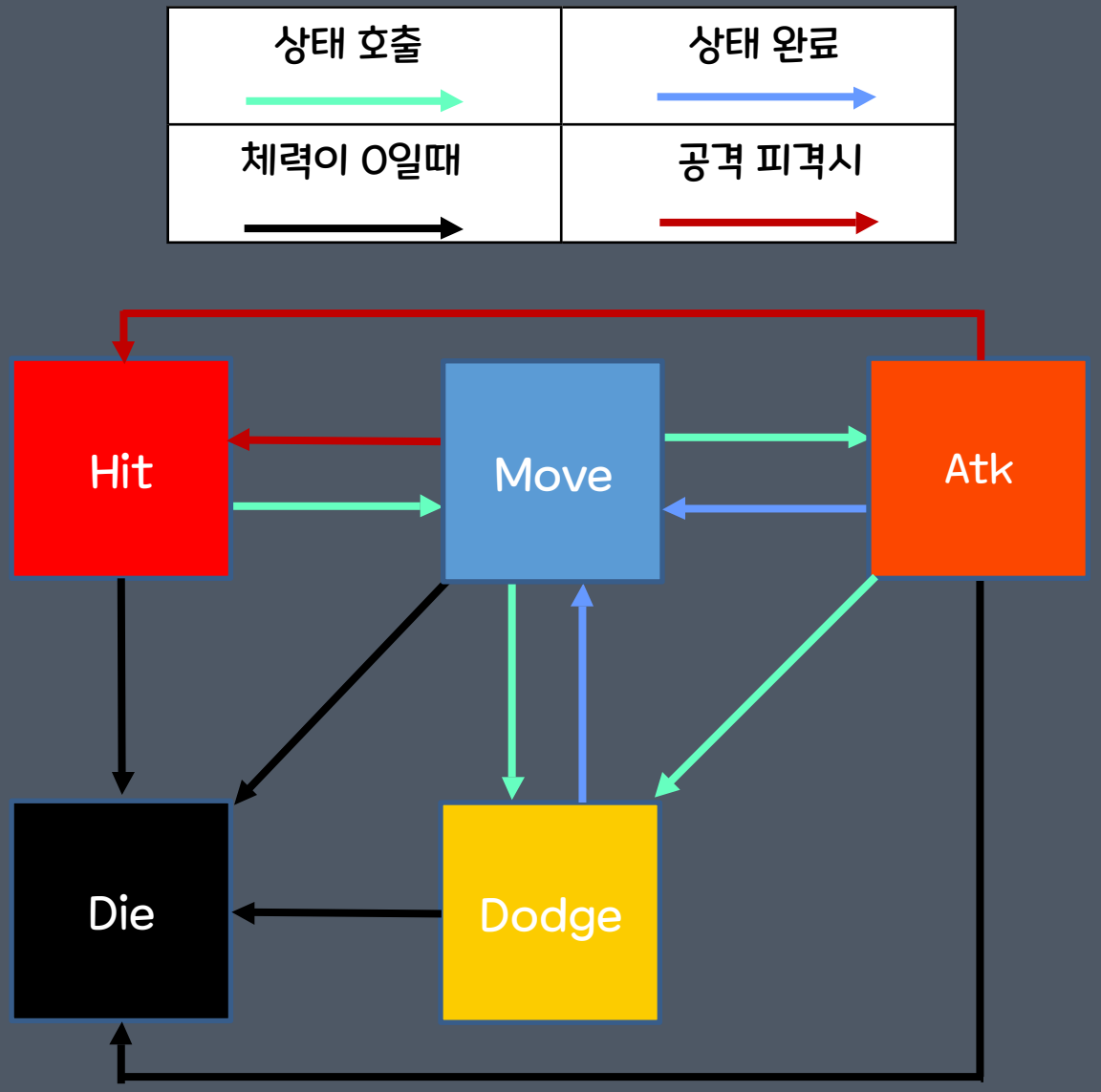
Land

- 드래곤이 날아오를때와 마찬가지로
  땅에 내려올때도 특정애니메이션을 실행시켜줍니다.
- 애니메이션 진행도에 맞춰 isFly를 false로 바꿔
  공격이 끝난뒤 지상에서의 행동이 가능하게 해주고,
  드래곤의 y값을 0까지 낮추고, 네비메쉬를 활성화 시킵니다.

```
IEnumerator RunLandCoroutine(Dragon dragon)
{
    dragon.navMeshAgent.enabled = false;
    dragon.a_id = "RunLand";
    dragon.dragonAnimator.SetTrigger("RunLand");

    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.4f);
    dragon.isFly = false;
    dragon.navMeshAgent.enabled = true;
    ShakeCamera.instance.OnShakeCamera(0.4f, 0.4f);
    GameObject LandEffect = ObjectPoolingManager.Instance.GetObject("Ground", dragon.LandPos);
    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.45f);
    yield return new WaitUntil(() => dragon.AnimationName && dragon.AnimationProgress >= 0.55f);
    dragon.StartCoroutine(LightningCoroutine(dragon));
    yield return new WaitForSeconds(1f);
    ObjectPoolingManager.Instance.ReturnObject("Ground", LandEffect);
}
```

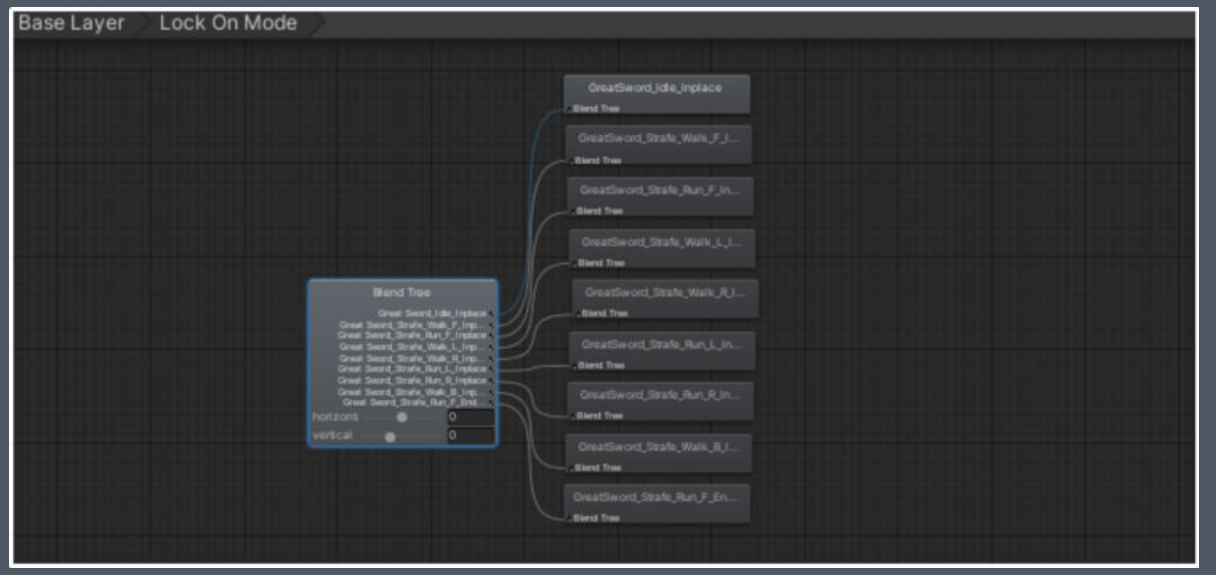
FSM(Player)

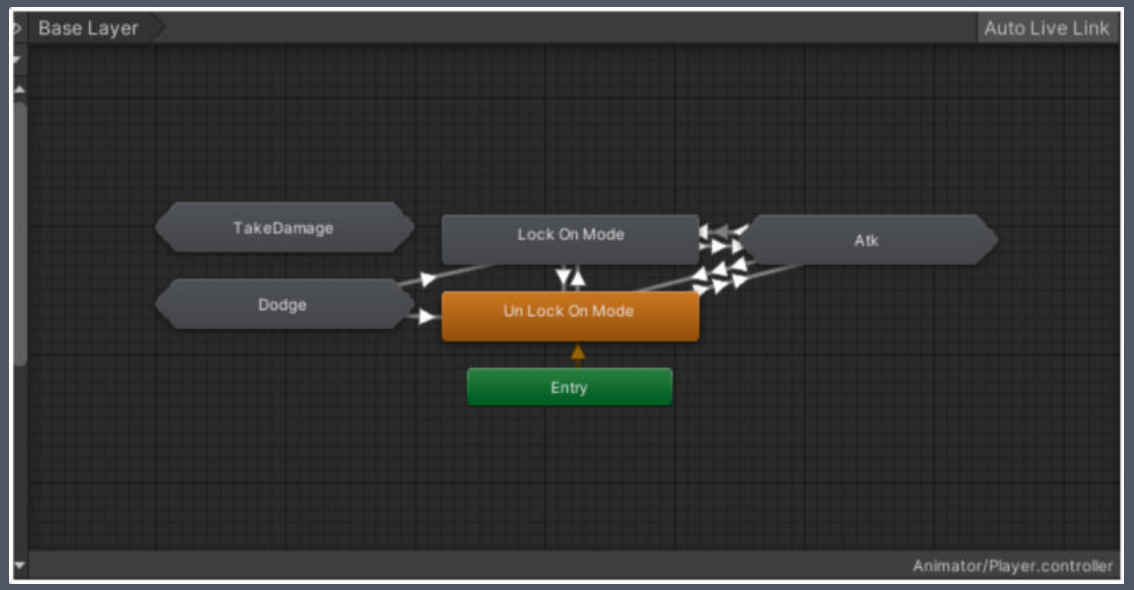
- 플레이어의 FSM 구조
- 플레이어의 입력값에 따라 상태가 변화하며,
- Move 상태에서 Atk, Dodge 상태로 전환 가능하고
- Atk 상태에서 특정 키를 누르면 Dodge 상태로 전환합니다
- Hit 와 Die 상태는 어떤 상태에서도 전환 가능합니다.

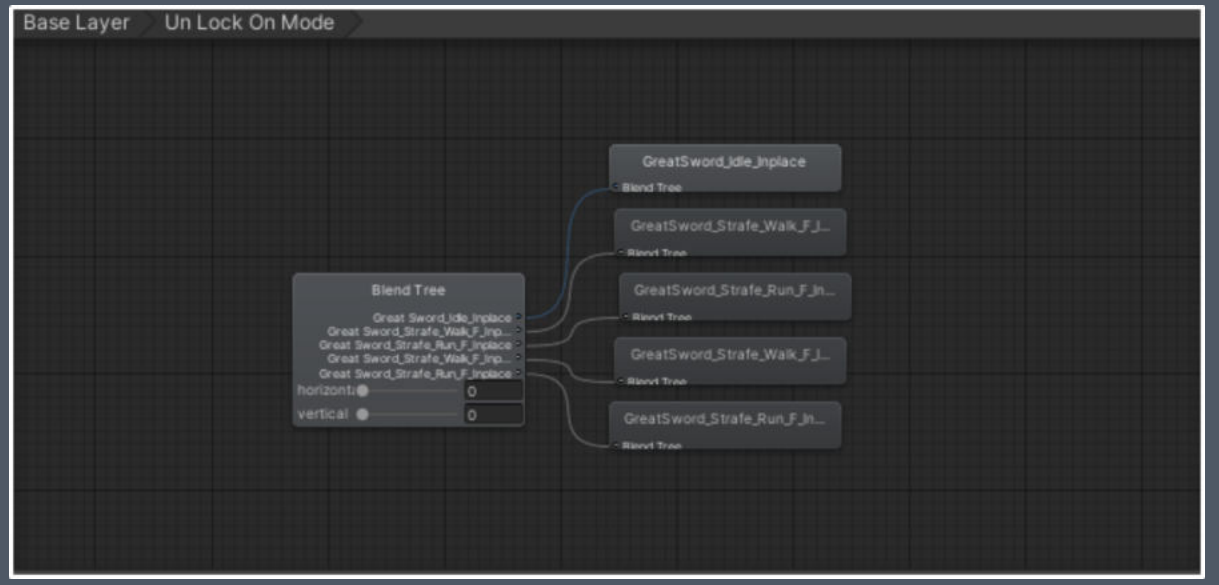


Animator

- 플레이어의 애니메이터
- 이동, 공격, 회피, 피격, 사망 상태가 있으며
  락온모드와 언락온모드에 따라
  이동 애니메이션과 회피 애니메이션이 달라집니다.







Move

- 플레이어는 Move상태로 시작하며
  InputMovement 함수를 통해 방향키로 이동이 가능합니다.
- 특정 입력값에 따라 공격, 회피, 락온이 가능하며
  입력시 해당 상태로 전환되게 됩니다.

```
public void InputMovement()
{
    float horizontalInput = Input.GetAxis("Horizontal");
    float verticalInput = Input.GetAxis("Vertical");

    moveDirection = new Vector3(horizontalInput, 0, verticalInput);
    float inputMagnitude = Mathf.Clamp01(moveDirection.magnitude);
    player.PlayerAnimator.SetFloat("horizontal", horizontalInput, 0.1f, Time.deltaTime);
    player.PlayerAnimator.SetFloat("vertical", verticalInput, 0.1f, Time.deltaTime);
    float speed = inputMagnitude * maximumSpeed;

    moveDirection = Quaternion.AngleAxis(cameraTransform.rotation.eulerAngles.y,
                                         Vector3.up) * moveDirection;
    moveDirection.Normalize();
    velocity = moveDirection * speed;
}
```

AtkState

- 이동상태에서 마우스 왼쪽버튼을 누르면 공격으로 전환되며 애니메이션의 이름, 진행상태를 체크해 지정한 타이밍에 검기를 활성화 해주거나 공격콜리전을 활성화 해줍니다.
- 공격중 다른 상태로 전환 할 수 있도록 ComboAtkCheck를 매번 실행시켜 줍니다.

```
참조 2개
public void OnUpdate(Player player)
{
    player.inputmanager.ComboAtkCheck(Player.eState.NORMALATK2);
}

참조 1개
IEnumerator NormalAtk1Coroutine(Player player)
{
    player.a_id = "NormalAtk1";
    player.PlayerAnimator.SetTrigger("NormalAtk1");
    yield return new WaitUntil(() => player.AnimationName && player.AnimationProgress >= 0.22f);
    player.AtkColision.SetActive(true);
    GameObject Slash = ObjectPoolingManager.Instance.GetObject("Slash", player.WeaponPos[0]);

    yield return new WaitUntil(() => player.AnimationName && player.AnimationProgress >= 0.25f);
    player.AtkColision.SetActive(false);
    yield return new WaitUntil(() => player.AnimationName && player.AnimationProgress >= 0.5f);
    ObjectPoolingManager.Instance.ReturnObject("Slash", Slash);
}
```

ComboAtk

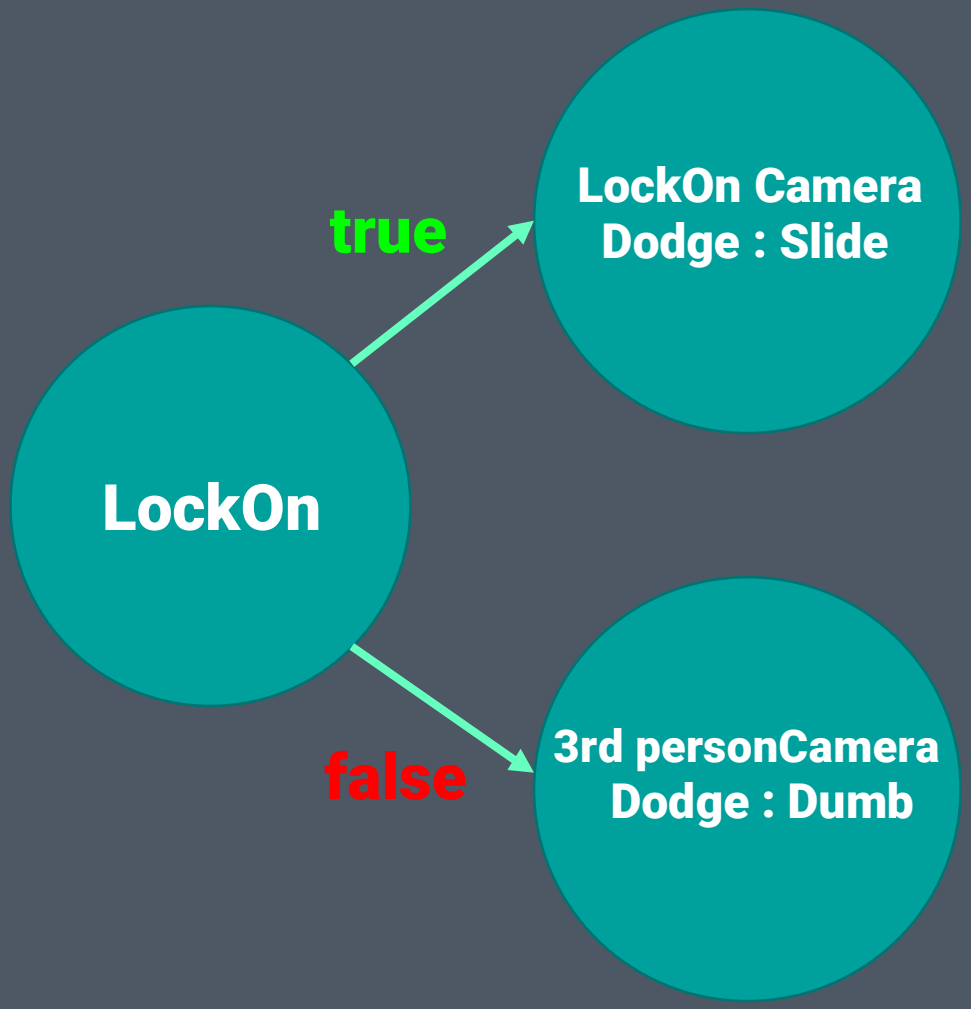
- ComboAtkCheck함수는 캐릭터의 콤보공격을 구현하기 위해 사용 되었으며 공격도중 플레이어의 입력값에 따라 다음 행동을 실행시키는 역할을 합니다.
- 특정 타이밍에 공격버튼을 누르면 콤보공격이 나가고, Ctrl 버튼을 누르면 공격을 캔슬하고 회피하며 아무 버튼을 누르지 않는다면 이동상태로 돌아가게 됩니다.

```
참조 3개
public void ComboAtkCheck(Player.eState state)
{
    if (player.AnimationName && player.AnimationProgress >= 0.6f && Input.GetMouseButtonDown(0))
    {
        player.ChangeState(state);
    }

    if (player.AnimationName && player.AnimationProgress >= 0.9f)
    {
        player.ChangeState(Player.eState.MOVE);
    }

    if (Input.GetKeyDown(KeyCode.LeftControl))
    {
        player.ChangeState(Player.eState.DODGE);
    }
}
```

LockOn

- 플레이어는 T버튼을 통해 락온모드와 일반모드로 전환할수 있으며,
  카메라 , 이동, 회피 애니메이션은 락온상태의 유무에 따라 달라 집니다.

```
if (Input.GetKeyDown(KeyCode.T))
{
    if (player.Lockon)
    {
        player.Lockon = false;
        player.PlayerAnimator.SetBool("LockOnMode", false);
    }
    else
    {
        player.Lockon = true;
        player.PlayerAnimator.SetBool("LockOnMode", true);
    }
}
```



Camera

- 카메라는 기본적으로 3인칭 카메라지만
  Lockon값이 ture가 되면 락온 카메라로 전환 되며,
  플레이어의 시야는 드래곤에 고정 되고
  이동 모션과 회피 모션이 변경 됩니다.

```
public void LockCamera()
{
    Vector3 current = pivotPoint;
    Vector3 target = player.transform.position + Vector3.up;
    pivotPoint = Vector3.MoveTowards(current, target, Vector3.Distance(current, target) * cameraSlack);

    transform.position = pivotPoint;
    transform.LookAt((enemy.position + player.position) / 2);

    transform.position -= transform.forward * cameraDistance;
}
참조 1개
public void UnLockCamera()
{
    x += Input.GetAxis("Mouse X") * xMoveSpeed * Time.deltaTime;
    y -= Input.GetAxis("Mouse Y") * yMoveSpeed * Time.deltaTime;
    y = ClampAngle(y, yMinLimit, yMaxLimit);
    transform.rotation = Quaternion.Euler(y, x, 0);

    distance -= Input.GetAxis("Mouse ScrollWheel") * wheelSpeed * Time.deltaTime;
    distance = Mathf.Clamp(distance, minDistance, maxDistance);
}
```

Dodge

- 플레이어는 Move 상태에서
  ctrl버튼을 누르면 회피상태로 전환되며 무적상태가 됩니다.
  락온상태 일때는 미끄러지는듯한 슬라이드 애니메이션이 실행되고
  아닐때에는 구르기 애니메이션이 실행됩니다.

```
public void OnEnter(Player player)
{
    player.p_takedamage.GodMode = true;
    if (player.Lockon)
    {
        player.a_id = "Slide";
        player.PlayerAnimator.SetTrigger("Slide");
    }

    else
    {
        player.a_id = "Dumb";
        player.PlayerAnimator.SetTrigger("Dumb");
    }
}
```

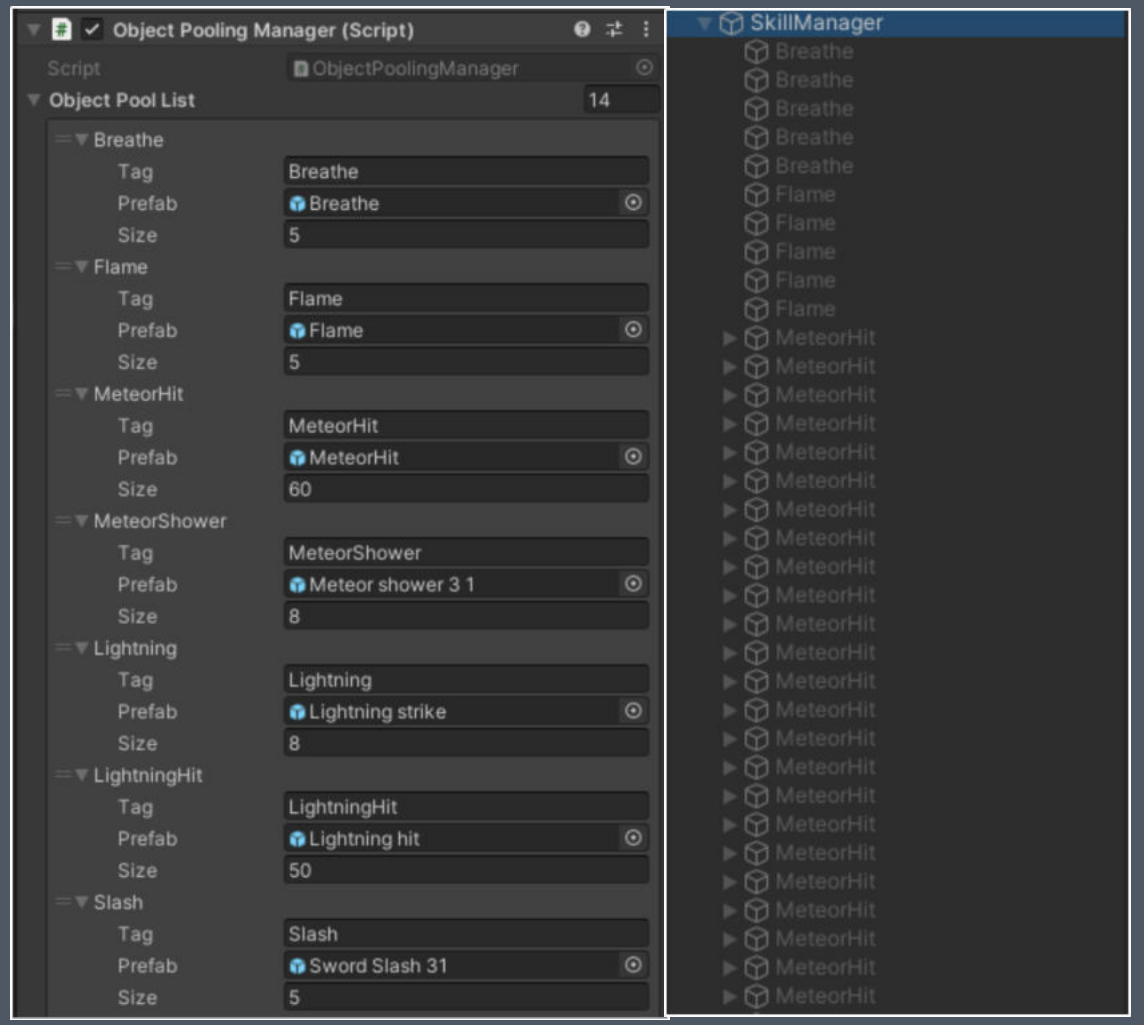
Hit

- 플레이어는 드래곤의 공격에 피격 당할 경우
  Hit 상태로 들어오고 잠시 무적상태가 됩니다.
  강한공격에 맞은경우에는 넉백이 되고
  약한공격에 맞은경우에는 경직상태가 됩니다.

```
public void OnEnter(Player player)
{
    player.p_takedamage.GodMode = true;
    if (player.SmashHit)
    {
        player.a_id = "Falldown";
        player.PlayerAnimator.SetTrigger("TakeDamage_Knockback");
    }
    else
    {
        player.a_id = "Hit";
        player.PlayerAnimator.SetTrigger("TakeDamage");
    }
}
```

Object Pooling

- 게임 플레이중 발생할 수 있는 과부하를 방지하기 위해
- 플레이어와 드래곤의 모든 이펙트는
  오브젝트풀링으로 관리합니다.
- 이펙트를 등록할 때는 인스펙터창에 리스트 추가 후
  태그와 갯수를 정해준 뒤 이펙트를 등록하고,
  게임이 시작 되면 SkillManager라는 오브젝트에
  사전에 등록한 모든 이펙트가 생성됩니다.



Object Pooling

- 게임실행시 리스트에 등록된 오브젝트들을 태그로 분류하여 입력한 갯수만큼 생성시켜 줍니다.
- 플레이어와 드래곤 양쪽에서 사용하기 때문에 쉽게 접근이 가능하도록 싱글톤을 사용했고 관리를 용이하게 하기위해 딕셔너리로 구성했습니다

```
private void Awake()
{
    if (Instance == null)
        Instance = this;

    else if (Instance != this)
        Destroy(this.gameObject);
}

public List<ObjectPool> ObjectPoolList;
public Dictionary<string, Queue<GameObject>> ObjectPoolDictionary;

Unity 메시지 | 참조 0개
private void Start()
{
    ObjectPoolDictionary = new Dictionary<string, Queue<GameObject>>();

    foreach(ObjectPool pool in ObjectPoolList)
    {
        Queue<GameObject> objectPool = new Queue<GameObject>();

        for (int i=0;i< pool.Size; i++)
        {
            obj=  Instantiate(pool.Prefab) as GameObject;
            obj.name = pool.Prefab.name;
            obj.transform.SetParent(gameObject.transform);
            obj.SetActive(false);
            objectPool.Enqueue(obj);
        }
        ObjectPoolDictionary.Add(pool.tag, objectPool);
    }
```

Object Pooling

- GetObject는 인자값으로 태그와 부모를 받아
  해당 오브젝트를 활성화 시킨 뒤
  부모의 위치로 꺼내오는 역할을 하고
- ReturnObject는 태그를 인자값으로 받아
  사용이 완료된 오브젝트를 비활성화 하여
  다시 원래 위치(Skill Manager)로 넣어줍니다.

```
참조 22개
public GameObject GetObject(string tag,GameObject Parent)
{
    if (!ObjectPoolDictionary.ContainsKey(tag))
        return null;
    GameObject SpawnObject;
    SpawnObject = ObjectPoolDictionary[tag].Dequeue();
    SpawnObject.transform.SetParent(Parent.transform);
    SpawnObject.transform.position = Parent.transform.position;
    SpawnObject.transform.rotation = Parent.transform.rotation;
    SpawnObject.SetActive(true);
    return SpawnObject;
}
```

```
public GameObject ReturnObject(string tag, GameObject Object)
{
    ObjectPoolDictionary[tag].Enqueue(Object);
    Object.transform.SetParent(gameObject.transform);
    Object.SetActive(false);
    return Object;
}
```

Collider Event

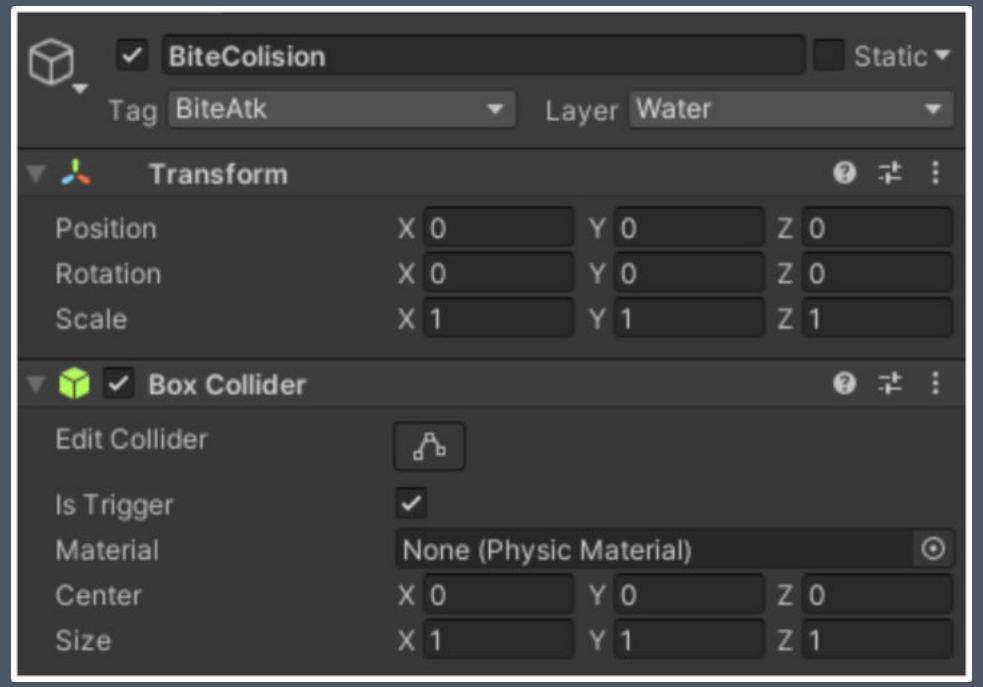
- 공격을 구현하기 위해 플레이어의 검이나 드래곤의 머리등에 콜라이더를 달아주었으며, 공격시 짧은 시간 활성화 됩니다.
- 서로의 콜라이더가 다른 대상에게 접촉하면 때린 상대의 공격력만큼 맞은 캐릭터의 체력을 소모 시켜줍니다.

```
public static float EnemyDamage = 20f;

Unity 메시지 | 참조 0개
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Player"))
        other.GetComponent<Player_TakeDamage>().PlayerTakeDamage(EnemyDamage);
}
```

```
public void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Enemy"))
    {
        ShakeCamera.instance.OnShakeCamera(0.2f, 0.2f);
        other.GetComponent<Enemy_TekeDamage>().TakeDamage(Player.PlayerDamage);
    }
}
```

Collider Event

- 공격콜라이더에 캐릭터가 닿게되면
  태그를 비교해 해당 공격 콜라이더에 해당하는
  데미지를 받게되고, Hit 상태로 전환되며
- 체력이 0이하가 됐을 경우에는 Dead 상태로 전환 됩니다.




```
public void PlayerTakeDamage(float damage)
{
    if (GodMode)
        return;
    PlayerHp -= damage;

}
Unity 메시지 | 참조 0개
private void OnTriggerEnter(Collider other)
{
    if (PlayerHp <= 0f)
    {
        player.ChangeState(Player.eState.DEAD);
    }
    else if (other.gameObject.CompareTag("BreatheAtk") && !GodMode)
    {
        StartCoroutine(BurnHitCoroutine());
        PlayerTakeDamage(10f);
        player.Hit = true;
        player.ChangeState(Player.eState.HIT);
    }
```

Shake Camera

- 게임의 타격감, 연출을 살리기 위해 쉐이크 카메라 기능을 구현했습니다.
- 원하는 타이밍에 OnShakeCamera 함수를 호출해서 사용하며, 시간과 세기를 인자값으로 넘겨 원하는 대로 조정 할 수 있습니다.

```
참조 18개
public void OnShakeCamera(float shakeTime = 0.2f, float shakeIntensity = 0.2f)
{
    this.shakeTime = shakeTime;
    this.shakeIntensity = shakeIntensity;

    if (isOnShake == true)
        return;
    StopCoroutine("ShakeByRotation");
    StartCoroutine("ShakeByRotation");
}

참조 0개
private IEnumerator ShakeByRotation()
{
    isOnShake = true;
    Vector3 startRotation = transform.eulerAngles;
    float power = 10f;

    while (shakeTime > 0.0f)
    {
        float x = 0;
        float y = 0;
        float z = Random.Range(-1f, 1f);
        transform.rotation = Quaternion.Euler(startRotation + new Vector3(x, y, z) * shakeIntensity * power);
        shakeTime -= Time.deltaTime;
        yield return null;
    }
    transform.rotation = Quaternion.Euler(startRotation);
    isOnShake = false;
```

Video&Document

- [포트폴리오 영상] : https://url.kr/d65ylh
- [깃허브 소스코드] : https://github.com/kgp0907/PortFolio