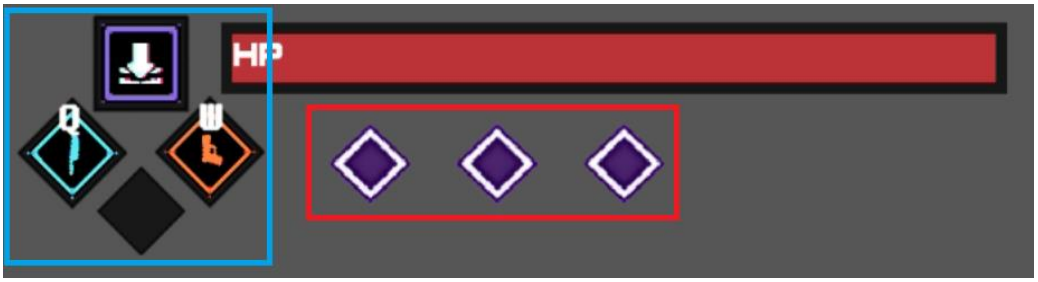
[무기 능력 및 교체 시스템 관련 코드]

1. **SkillManager** : 무기 교체 기능 및 연계 스킬의 작동 타이밍을 조절하는 기능을 하는 클래스이다. 필살기와 스킬을 쓰기 위한 마나를 관리하는 기능도 가진다.

[코드 요약]

```
private void Start()
{
    owner = GetComponent<Entity>();

    if (hudControl) // 플레이어가 보는 UI에서 currentWeapon 값에 해당하는 무기가 무엇인지 보여줌.
        hudControl.ChangeCurrentWeapon(currentWeapon);

    ChangeWeaponSkill(false, currentWeapon); // currentWeapon 값에 해당하는 무기로 현재 무기 바꿈,
    CheckMPUI(); // Gauge 값을 체크하여 Gauge가 얼마나 모였는지 표시함. (Gauge는 보라색 마름모로 표시함. Gauge가 3이면 보라색 마름모가 3개이다.)
}

Unity 메시지 | 참조 0개
private void Update()
{
    if (owner)
        owner.aManager.ani.SetInteger("Weapon", currentWeapon); // 애니메이터의 파라미터 값을 바꿔 현재 무기에 맞는 애니메이션이 재생되도록 함.

    if (hudControl)
    {
        if (Input.GetKeyDown(KeyCode.Q))
        {
            ChangeWeaponSkill(true); // 왼쪽무기와 현재무기교체 (현재 무기는 왼쪽으로 이동)
        }

        if (Input.GetKeyDown(KeyCode.E))
        {
            ChangeWeaponSkill(false); // 오른쪽무기와 현재무기교체 (현재 무기는 오른쪽으로 이동)
        }
        if (owner.GetHp() > 0 && Input.GetKeyDown(KeyCode.W)) // 체력 회복
        {
            int curGauge = owner.aManager.ani.GetInteger("Gauge"); // 애니메이터에서 Gauge 값을 가져옴. Gauge는 스킬, 필살기 사용에 필요한 값이다.
            if (curGauge > 0) // Gauge 값이 0보다 크면 게이지 값 1을 소모하여 체력 회복이 가능하다.
            {
                MedalControlSystem.hpBeforeChange = (int)owner.GetHp();
                Instantiate(healEffect, owner.transform.position, Quaternion.identity).transform.parent = owner.transform;
                owner.SetHp(owner.GetHp() + owner.maxHP * 0.3f); // 체력 회복 퍼센테이지 0.3
                isHealthEvent = true;
                ReduceGauge();
            }
        }
    }
    if (owner.GetMp() == owner.maxMp) // 현재 MP값이 최대치일 때 처리
    {
        int curGauge = owner.aManager.ani.GetInteger("Gauge");
        if (curGauge < 3) // 값이 3이 아닐 경우 게이지 값이 증가한다.
        {
            owner.aManager.ani.SetInteger("Gauge", curGauge + 1);
            CheckMPUI();
        }
        owner.SetMp(0); // MP값이 0으로 초기화됨.
    }
}
```

- Start 함수에선 Entity(캐릭터에 대한 정보(hp, mp 등)가 저장됨) 컴포넌트를 변수 owner 에 저장하고, 캐릭터의 현재 무기와 현재 게이지를 UI 에 표시한다.



// 빨간색 박스에 들어있는 부분은 게이지를 나타내고, 하늘색 박스에 들어있는 부분은 보유한 무기를 나타낸다.

- Update 함수의 경우엔 아래와 같은 작업을 진행한다.

  1. 무기에 따라 다른 애니메이션을 재생하도록 파라미터 값을 조절한다.

  - ◆ 해당 파라미터를 바꾸는 이유는 총을 제외한 모든 무기가 애니메이션 자체에서 애니메이션 이벤트 등을 이용해 모든 공격 및 스킬, 필살기 등의 처리가 이뤄지기 때문이다.
  - ◆ 총은 추가적으로 해당 파라미터 뿐만 아니라 다른 클래스가 같이 해당 무기를 제어하게 된다.

  2. 무기 교체(Q, E 버튼으로 가능)에 대한 처리와 게이지를 사용한 체력 회복도 진행하며 mp 가 최대값이 될 때 게이지가 1 상승하도록 한다.

```
public void ChangeWeaponSkill(bool isLeftWeapon, int newWeapon = -1) // 첫번째 매개변수는 왼쪽 무기와 교체하는지 여부, 두번째 매개변수는 바꿀 무기의 번호이다.
{
    var previousWeapon = currentWeapon; // 교체 전 무기를 변수로 저장
    if (newWeapon == -1)  // 무기의 번호를 매개변수로 전달 안 할 경우
    {
        if (isLeftWeapon) // 왼쪽으로 교체하는 버튼을 눌렀을 경우
            currentWeapon = currentLeftWeapon; // 현재 무기를 기본 값으로 설정된 왼쪽무기로 바꿈.
        else
            currentWeapon = currentRightWeapon; // 현재 무기를 기본 값으로 설정된 오른쪽무기로 바꿈.
    }
    else
        currentWeapon = newWeapon; // 무기 번호에 맞는 무기로 바꿈.

    if(haveWeaponNum < currentWeapon) // 바꾼 무기의 번호(무기 번호는 얻는 순서와 같음)가 현재 보유한 무기의 수보다 작을 경우
    {
        currentWeapon = previousWeapon; // 이전 무기로 다시 바꿈.
        return;
    }

    hudControl.ChangeCurrentWeapon(currentWeapon); // 현재 무기를 UI에 반영

    bool left = false;
    for (int i = 0; i < 3; i++) // 다음 왼쪽 무기와 오른쪽 무기를 번호 순서에 따라 정함.
    {
        if (i != (currentWeapon - 1))
        {
            if (!left)
            {
                left = true;
                currentLeftWeapon = i + 1;
            }
            else
                currentRightWeapon = i + 1;
        }
    }
}
```

- ChangeWeaponSkill 함수에선 현재 무기를 왼쪽 또는 오른쪽 무기와 교체하는 작업을 진행한다.
- 단, 무기 번호를 두번째 매개변수 값으로 전달하면 해당 무기로 바로 전환 가능하다.
  - ◆ 예외 처리 : 보유하지 않는 무기로는 바꿀 수 없게 처리했다.
- 무기를 교체한 이후엔 왼쪽 무기와 오른쪽 무기가 자동으로 무기 번호 순서대로 바뀌게 되어있다.

2. **AnimationManager** : 특정 키에 반응하여 애니메이션을 재생하는 클래스이다. 애니메이션이 공격 애니메이션일 경우 공격이 나가기에 일반적으로 무기의 사용을 담당한다.

[코드 요약]

```
void PlayerAnimation() // 조종하는 플레이어 캐릭터의 애니메이션 관리 -> 입력에 반응
{
    if (Input.GetKeyDown(Punch))
    {
        ani.SetTrigger("Punch");
        ani.SetBool("waitLinkAttack", true);
        StartCoroutine(WaitLink(1f));
    }
    if (ani.GetInteger("Weapon") != 2 &&
            (groundCheck.GetOnGround || !additionalJump) && Input.GetKeyDown(Jump) && !Input.GetKey(DownArrow))
    {
        if (!onGround)
        {
            additionalJump = true;
            if (Ec && Ec.jumpDustEffectPrefab)
            {
                GameObject eff = GameObject.Instantiate(Ec.jumpDustEffectPrefab);
                eff.transform.position = gameObject.transform.position - new Vector3(0,0.5f,0);
            }
        }

        if (ani.GetInteger("Weapon") != 2)
        {
            State = AnimationState.Jump;
            ani.SetTrigger("Jump");
        }
```

```
참조 1개
IEnumerator WaitLink(float delay)
{
    yield return new WaitForSeconds(delay);
    ani.SetBool("waitLinkAttack", false);
}
```

- 클래스 내에서 플레이어의 입력을 받아 무기를 이용한 공격이나 스킬을 사용하는 부분은 PlayerAnimation 함수 부분이다. 해당 함수에선 연계 스킬에 대한 부분도 구현되어 있는데 무기 사용 중에 다른 무기로 교체할 경우 연계 스킬이 발동하도록 하였다.
    - ◆ 무기 사용 중엔 waitLinkAttack 라는 파라미터의 값이 애니메이터로 전달되는데 해당 값이 true 여야 연계 스킬이 발동된다.
    - ◆ 코루틴 WaitLink 를 실행하여 특정 시간 동안만 waitLinkAttack 값이 true 이도록 만들어 연계 스킬이 적절한 타이밍에만 가능하도록 했다.

3. **ShootingControl** : 총의 사용을 위해 만들어진 클래스로 마우스 커서의 모양을 바꾸고 총 무기의 사격 및 폭탄 투척 스킬 등을 구현하였다. 또한, 단순히 총에만 사용되진 않고 무언가 생성하는 기술의 경우엔 모두 해당 클래스를 사용한다.

[코드 요약]

```
void Update()
{
    if(owner.movement.PlayerType)
    if(!owner.isDie)
    if (WhenTargeting) // 플레이어가 총을 무기로 사용하고 있으며 사격 가능한 상태
    {
        Cursor.SetCursor(targetingImage, new Vector2(targetingImage.width/3f, targetingImage.height / 3f), CursorMode.Auto); // 커서를 사격 조준경으로 바꿈.
        AtargetingPos = Camera.main.ScreenToWorldPoint(Input.mousePosition + new Vector3(20, -20, 0)); // 마우스 위치를 변수에 저장함.

        // gun은 총 오브젝트를 의미하며 아래 변수들은 마우스의 방향에 따라 총의 회전값이 변하도록 하였다.
        var dir = Input.mousePosition - Camera.main.WorldToScreenPoint(transform.position);
        var angle = MathF.Atan2(dir.x, dir.y) * MathF.Rad2Deg;

        gun.transform.rotation = Quaternion.AngleAxis(-angle, Vector3.forward);

        // 총을 사용 가능한 상태에선 gunAnimator라는 애니메이터를 이용해 총의 발사 애니메이션을 보여준다.
        // 아래는 gunAnimator를 가진 게임 오브젝트의 회전 값을 바꾸어 총의 위치가 제대로 위치하도록 하는 코드.
        gunAnimator.transform.localEulerAngles = new Vector3(gunAnimator.transform.localEulerAngles.x, owner.transform.localEulerAngles.y, gunAnimator.transform.localEulerAngles.z);

        if (bulletCount < bulletMaxCount) // 총의 발사 횟수 제한이 있고, 해당 횟수를 넘어서면 일정 시간 동안 재장전이 이뤄진다.
        {
            if (Input.GetKeyUp(KeyCode.Mouse0))
            {
                // 아래는 총을 발사하는 과정을 나타낸 코드이다.

                bulletCount++;
                if (GunGauge.fillAmount > 0.8f)
                {
                    GunGauge.fillAmount -= 0.15f;
                }
                else
                {
                    GunGauge.fillAmount -= 0.17f;
                }
                var obj = Instantiate(bullet, fireTr.position, Quaternion.identity);
                obj.GetComponent<HitCollider>().owner = owner;
                Destroyer d = obj.GetComponent<Destroyer>();

                obj.transform.LookAt(AtargetingPos);
                d.moveSpeed = bulletSpeed;
                d.haveTarget = true;

                if (gunAnimator != null)
                {
                    gunAnimator.SetTrigger("Shoot");
                }
                owner.aManager.ResetAttackTriggerEvent();
            }
            if (bulletCount == bulletMaxCount)
                StartCoroutine(Reload()); // 재장전을 담당하는 코루틴 호출
        }
    }
```

- Update 함수에서 마우스 커서를 바꾸고, 총의 조준 방향 조절 및 총알 발사, 총의 애니메이션 재생 등에 대한 처리 또한 같이 진행한다.

    ◆ 총은 무한으로 발사할 수 없도록 발사 횟수의 제한을 두어 횟수 초과 시 재장전을 하는 대기 시간을 주었다.

    ◆ 총의 조준 방향은 마우스 위치에 따라 달라지며, 총 오브젝트도 마우스에 따라 회전하도록 했다.

// 커서의 모양이 변한 모습

```
public void SimpleShoot(float speed)
{
    Vector2 len = this.gameObject.GetComponent<Movement>().mousePos - this.gameObject.transform.position;
    float z = Mathf.Atan2(len.y, len.x) * Mathf.Rad2Deg;

    var obj = Instantiate(bullet, GunfireTr.position, Quaternion.Euler(0f, 0f, z));
    obj.GetComponent<HitColider>().owner = owner;
    Destroyer d = obj.GetComponent<Destroyer>();
    d.moveSpeed = speed;
}
참조 0개
public void ThrowBomb()
{
    var obj = Instantiate(bombPrefab, fireTr.position, Quaternion.identity);
    Destroyer d = obj.GetComponent<Destroyer>();
    obj.transform.LookAt(AtargetingPos);
    d.moveSpeed = bulletSpeed;
    d.haveTarget = true;
}
```

- Update 함수 이외에는 호출할 수 있는 여러가지 총과 관련된 함수들을 만들었다.
    - ◆ SimpleShoot 과 ThrowBomb 모두 마우스가 총의 발사 방향을 결정한다.
    - ◆ ThrowBomb 의 경우 총알이 아닌 폭탄이 생성되도록 했다.

```
public void ThrowStone()
{
    float tempPosX;
    if (owner.transform.rotation.y == 0) tempPosX = 1.5f;
    else tempPosX = -1.5f;
    // 바위를 생성해 앞쪽으로 날린다.
    var stone = Instantiate(StonePrefab, owner.transform.position + new Vector3(tempPosX, 0.3f),
        Quaternion.identity);
    stone.GetComponent<Rigidbody2D>().AddForce(owner.transform.right * 10000);

    // 바위는 플레이어가 맞지 않도록 만든 오브젝트라 별도 처리가 필요 없다.
}

참조 0개
public void ShockWave()
{
    float tempPosX;
    if (owner.transform.rotation.y == 0) tempPosX = 0.5f;
    else tempPosX = -0.5f;

    // 참격을 생성해 앞쪽으로 날린다.
    var shockWave = Instantiate(ShockWavePrefab, owner.transform.position + new Vector3(tempPosX, -0.5f),
        Quaternion.identity);

    // 참격의 경우 플레이어가 맞지 않게 설정함.
    shockWave.GetComponent<HitColider>().owner = owner;
    shockWave.GetComponent<Rigidbody2D>().AddForce(owner.transform.right * 10000);
}
```

- 총과 관련되지 않은 다른 함수도 해당 클래스 내에 존재하며 대부분 공격을 위한 오브젝트를 생성해 발사하는 코드가 작성되어 있다.