**Retreat Protocol**

**구현 기술 정리 및 의사코드 정리**

**플레이 캐릭터 중거리형 캐릭터 구현**

**MiddleWeaponFirst(기본 포탄)**

 void BasicWeapon()

    {

        Vector3 firePos = new Vector3(playerCameraTransform.position.x, playerCameraTransform.position.y, playerCameraTransform.position.z);

        GameObject fireProjectile = Instantiate(projectile, firePos, playerCameraTransform.rotation);

        fireProjectile.GetComponent<PlayerBulletBasic>().Clone(this ,currentLevel);

    }

의도 : 카메라가 바라보는 방향으로 직진하는 포탄을 생성.

    public virtual void Update()

    {

        // 플레이어의 공격 투사체는 기본적으로 직진만 한다.

        transform.Translate( Vector3.forward \* moveSpeed \* Time.deltaTime );

        // 플레이어와 일정 거리 이상 떨어지면 스스로를 삭제

        if( Vector3.Distance(transform.position, player.transform.position) > 300 ) Destroy(gameObject);

    }

 void OnTriggerEnter(Collider other)

    {

        if( other.tag == "Enemy" )

        {

            Enemy enemy = other.GetComponent<Enemy>();

            enemy.GetDamage(attackDamage); // 맞은 적에게 피해를 입힌다

            ActivateWhenHit(other); // 오버라이딩 된 추가 효과가 있다면 발동

            Destroy(gameObject); // 스스로를 삭제

        }

    }

MiddleBulletFirst를 구현하기 위한 핵심 코드 부모 클레스인 PlayerBulletBasic의 위 두 함수를 받아와 기본적으로 직진하는 투사체를 소환하고, 적과 부딪혔을 경우 적에게 데미지를 주고 스스로를 삭제하는 포탄인 MiddleBulletFirst를 소환하는 포신.

**MiddleBulletSeconde(다크 메터)**

 projectile = Resources.Load<GameObject>("Middle/MiddleBullet2");

MiddleBullet2를 소환하는 포신

* MiddleBullet2를 구현하기 위한 핵심 코드
* private GameObject FindTarget()
* {
* GameObject[] enemies = GameObject.FindGameObjectsWithTag("Enemy");
* GameObject closeEnemy = null;
* float closeDistance = Mathf.Infinity;
* foreach (GameObject enemy in enemies)
* {
* float distance = Vector3.Distance(transform.position, enemy.transform.position);
* if (distance < closeDistance)
* {
* closeDistance = distance;
* closeEnemy = enemy;
* }
* }
* if (closeEnemy != null)
* {
* Debug.Log("Closest enemy found: " + closeEnemy.name);
* }
* else
* {
* Debug.LogWarning("No enemy found within range.");
* }
* return closeEnemy;
* }

적을 끌어당기기 위해 포탄 오브젝트 근처의 적을 찾음

적을 찾고 그 정보를 저장하기 위한 배열을 생성하고, 가장 가까운 적의 게임 오브젝트를 생성하고 그 오브젝트를 null로 초기화 후 가장 가까운 적의 위치를 실수 값으로 저장하기 위한 변수를 생성해주고 무한대로 초기화 여기서 무한대로 초기화 하는 이유는 어떤 값과 비교하더라도 그 값이 작은 값이 되기 때문에 무한대로 초기화 한다. 이후 foreach를 이용하여 거리 계산을 하고 가장 가까운 적의 위치를 반환한다.

  private IEnumerator PullEnemy(GameObject enemy, float duration)

    {

        float elapsedTime = 0f;

        while (elapsedTime < duration && enemy != null)

        {

            elapsedTime += Time.deltaTime;

            enemy.transform.position = Vector3.MoveTowards(enemy.transform.position, transform.position, ForceSpeed \* Time.deltaTime);

            yield return null;

        }

        Debug.Log("Enemy pulled successfully.");

    }

적을 당겨주기 위한 코루틴.

매개 변수로 적 오브젝트와 다크 메터가 존재하는 시간을 실수값으로 받아준다. 그리고 당겨주고 최소 있는 시간을 실수 값으로 받아주고 while 반복문을 사용하여 적 오브젝트가 있고, 당겨주고 있는 최소 시간이 다크메터 존재 시간보다 작으면 최소시간을 증가시키고 적의 위치를 포탄 오브젝트 위치로 당겨주기 위해 MoveTowards함수를 사용하여 위치를 이동 시켜주고, 적이 순간이동 하는 것 처럼 이동하는 것을 막기 위하여 이동 하는 속도에 DeltaTime을 곱해주어 적의 이동을 부드럽게 만들어 주었다.

 private void DecreaseEnemySpeed(GameObject enemy)

    {

        if (enemy != null)

        {

            // 적의 이동속도를 20% 감소시킴

            Enemy enemyScript = enemy.GetComponent<Enemy>();

            if (enemyScript != null)

            {

                enemyScript.moveSpeed \*= 0.8f;  // 20% 감소

                Debug.Log("Enemy's speed reduced by 20%");

            }

        }

    }

또한 다크 메터는 다크 메터 주위에 있는 적들에게 슬로우를 거는데 위 함수는 슬로우를 걸기 위한 함수이다. 위 함수는 매개변수로 적의 오브젝트의 정보를 받아주고, 적 오브젝트 정보가 null 값이 뜨지 않게 하기 위해 enemy스크립트를 컴포넌트로 받아준다. 그리고 받아준 컴포넌트로 enemy 스크립트의 moveSpeed에 접근하여 적의 스피드를 감소시킨다.

Enemy = FindTarget();  // Enemy가 없다면 찾기

    // 적이 폭발 범위 내에 있을 때 폭발 대신 이동속도 감소

    if (!hasExploded && Enemy != null && Vector3.Distance(transform.position, Enemy.transform.position) < explodeRadius)

    {

        DecreaseEnemySpeed(Enemy);  // 적의 이동속도 감소

        hasExploded = true;  // 폭발이 발생했음을 표시

    }

    // 왼쪽 마우스 클릭 (0) 감지

    if (Input.GetMouseButtonDown(0) && Enemy != null)  // 왼쪽 마우스 클릭 시

    {

        Debug.Log("Left mouse button clicked!");

        StartCoroutine(PullEnemy(Enemy, destroyDelay));

    }

    // 플레이어와 일정 거리 이상 떨어지면 적을 끌어당기고 4초 뒤 사라짐

    if (Vector3.Distance(transform.position, player.transform.position) > 300)

    {

        Debug.Log("DarkMatter is too far from the player, destroying.");

        Enemy = FindTarget();

        if (Enemy != null)

        {

            StartCoroutine(PullEnemy(Enemy, destroyDelay));

        }

        Destroy(gameObject, destroyDelay);

    }

MiddleBullet2의 Update()함수 안에서 적을 끌어당기고, 적의 이동속도를 감소시키는 함수를 실행시키는데 적이 다크 메터가 폭발하는 범위 안에 있으면 적의 이동속드를 감소시키는 함수를 실행시키고 또한 왼쪽 마우스를 클릭하면 적을 끌어당긴다.

다크메터가 플레이어의 위치에서 어느정도 멀어지면 적을 끌어당기는 코루틴을 실행시키고 다크메터 오브젝트를 삭제시킨다.

**MiddleWeawponThird(빨간 버튼)**

 public override void Fire()

    {

        // 이미 생성된 MiddleBullet2를 추적하여 폭발 시킴

        if (trackedBullet != null)

        {

            StartCoroutine(TriggerExplosion());

        }

        else

        {

            Debug.LogWarning("No MiddleBullet2 object to track.");

        }

    }

    private IEnumerator TriggerExplosion()

    {

        // 일정 시간 대기 후 폭발 트리거 (0.5초 대기)

        yield return new WaitForSeconds(0.5f);

        // trackedBullet이 존재하면 폭발 발생

        if (trackedBullet != null)

        {

            trackedBullet.MadeExplode(trackedBullet.transform.position);  // 해당 위치에서 폭발 발생

            Debug.Log("폭발 발생");

        }

        else

        {

            Debug.LogWarning("MiddleBullet2 has been destroyed before explosion.");

        }

    }

    // 외부에서 MiddleBullet2 객체를 설정할 메서드

    public void SetTrackedBullet(MiddleBullet2 bullet)

    {

        trackedBullet = bullet;

    }

빨간 버튼을 개발하기 위한 핵심 코드

빨간 버튼을 사용했을 때 다크 메터를 폭파시키기 위한 코루틴인 TriggerExplosion을 생성 이 코루틴은 WaitForSeconds를 사용하여 0.5초 대기 시키고 MiddleBullet2를 받아온 TrackedBullet이 존재하면 MiddleBullet2에 있는 MadeExpolode 함수를 실행시킨다.

public void MadeExplode(Vector3 position)

    {

        // 폭발 효과 생성

        Instantiate(Explosion, position, Quaternion.identity);

        // "폭발" 로그 출력

        Debug.Log("Explosion triggered!");

        // 폭발 후 0.4초 뒤에 삭제

        Destroy(gameObject, 0.4f);  // 폭발 후 오브젝트 삭제

        GameObject[] enemyList = GameObject.FindGameObjectsWithTag("Enemy");

        foreach (GameObject enemy in enemyList)

        {

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

            // 폭발 범위 내에 있는 적들에게 피해를 주고 밀쳐냄

            if (distance < explodeRadius + currentLevel)

            {

                Debug.Log("Enemy within explosion range. Applying damage and force.");

                attackDamage = 100;

                enemy.GetComponent<Enemy>().GetDamage(attackDamage);

                Debug.Log("데미지 : " + attackDamage);

                Rigidbody enemyRb = enemy.GetComponent<Rigidbody>();

                if (enemyRb != null)

                {

                    // 폭발력 적용: 폭발의 방향에 따라 밀쳐냄

                    enemyRb.AddForce((enemy.transform.position - transform.position).normalized \* explosionForce, ForceMode.Impulse);

                }

            }

        }

    }

빨간 버튼에서 호출한 MadeExplosion함수이다. Instantiate를 사용하여 폭발 효과 파티클을 생성하고 다크메터와 마찬가지로 가장 가까운 적의 위치를 저장하기 위해 배열을 하나 생성하고 적의 위치를 배열에 저장한다. 그리고 Vector3.Distance를 사용하여 적과 포탄 사이의 위치를 계산한다. 그리고 이 거리가 폭파 범위보다 작으면 데미지를주고 폭발력을 주기 위한 AddForce 함수를 실행시킨다.

MiddleWeaponFourth(연속 공격)

private IEnumerator FireBurst()

    {

        for(int j = 0; j < burstCount; j++){

           yield return new WaitForSeconds(1f);

        for (int i = 0; i < burstCount; i++) // 설정한 개수만큼 총을 발사

        {

            BasicWeapon(); // 총알 발사

            yield return new WaitForSeconds(fireRate); // 각 발사 간의 간격을 설정

        }

    }

    }

위 코드는 연속 공격을 개발하기 위한 핵심 코드로 단순히 MiddleBullet1 4개를 4번 발사하는 코루틴이다.

중거리 캐릭터의 스킬1(긴급 수리)

public override void Activate()

    {

        if(player != null)

        {

            player.currentHealth += (int)(player.maxHealth \* 0.2f);

            player.currentHealth = Mathf.Clamp(player.currentHealth, 0, player.maxHealth);

        }

    }

위 코드는 긴급 수리를 개발하기 위한 핵심코드로 단순히 플레이어의 현재 채력을 최대체력으로 만드는 코드이다.

중거리 캐릭터의 스킬2(긴급 회피)

 public IEnumerable AccelerationSpeed()

    {

        player.currentSpeed = player.maxSpeed;

        yield return new WaitForSeconds(5f);

        player.currentSpeed = 10f;

    }

위 코드는 긴급 회피를 개발하기 위한 핵심코드로 플레이어의 moveSpeed를 받아와 5초동안 최대 속도로 만드는 코드이다.

**적 생성 매커니즘**

 void SpawnMonsters()

    {

        // 플레이어 위치를 중심으로 무작위 방향 벡터와 거리를 생성한 후

        Vector3 randomVector = new Vector3(Random.Range(-1f, 1f), Random.Range(-1f, 1f), Random.Range(-1f, 1f));

        float spawnMinDistance = 30;

        float spawnRandomDistance = Random.Range(10, 100);

        // 생성한 방향으로, 생성한 거리만큼 떨어진 곳에 적을 생성한다.

        float spawnDistance = spawnMinDistance + spawnRandomDistance;

        Vector3 spawnPosition = randomVector \* spawnDistance;

        // 현재 플레이어와 목표 지점 사이의 거리를 구한다.

        float currentDistance = Vector3.Distance(player.transform.position, goal.transform.position);

        // 적을 생성하고 플레이어와 묙표 지점 사이의 거리에 따라 스탯을 재설정

        GameObject spawnedEnemy = Instantiate(enemyPrefab, spawnPosition, Quaternion.identity);

        Enemy enemyScript = spawnedEnemy.GetComponent<Enemy>();

        enemyScript.SetStatus(currentDistance, 2100);

        // 적의 수 증가

        currentMonsterCount++;

    }

플레이어로부터 -1 부터 1까지의 랜덤한 x, y, z 축을 백터 값으로 생성하고, 특정 범위의 랜덤한실수 값을 생성하여 위 벡터 값에 곱해 적 생성 위치를 정해주고 적을 생성해 준다.