Retreat Protocol

구현기술 의사코드 정리 종합본

계명대학교 게임소프트웨어전공

5702341 김기석

5645866 구기현

5702600 이창민

목차

이창민 (5702600)

Retreat Protocol

요약본

본 문서에서는 게임의 전체적인 특징을 요약하여 기술한다.

개요

Retreat Protocol은 우주 공간에서 목표 지점을 향해 이동하는 3D 싱글플레이어 TPS 게임이다. 플레이어는 적대적 NPC와 전투를 하면서, 게임 시작 시 주어지는 무작위 목표 지점을 향해 이동해야 한다.

조작

마우스와 키보드를 사용하여 조작하는 PC 게임이다.

플레이어는 다음과 같은 방법으로 함선을 조종할 수 있다. 함선은 직진 이동만 가능하며, 함선을 기울여 방향을 조절할 수 있지만, 직접적인 4방향 이동은 지원하지 않는다.

- **마우스:** 마우스를 움직여서 플레이어의 주변을 관찰
- **M1:** 바라보는 방향으로 선택한 무기를 발사
- **M2:** 누르고 있는 동안 바라보는 방향을 줌 인
- **W, S:** 함선을 수직으로 기울여 이동 방향을 조정
- **A, D:** 함선을 수평으로 기울여 이동 방향을 조정
- **SPACE:** 누르고 있는 동안 함선의 이동속도를 천천히 가속
- **SHIFT:** 누르고 있는 동안 함선의 이동속도를 천천히 감속
- **1, 2, 3, 4:** 다음에 발사할 무기를 선택
- **Q, E:** 생존에 도움이 되는 스킬을 사용

전투 요소

플레이어는 무기와 스킬을 활용하여 전투를 벌일 수 있다. 무기와 스킬은 한 번 사용한 후엔 재사용하기 위해 잠시 동안 대기시간을 가진다.

- **무기:** 플레이어가 공격 시 사용할 무기. 무기에 따라 발사되는 포탄의 특징이 다르다.
- **스킬:** 적을 직접 공격하기 보단 플레이어 함선의 능력을 향상시키는 스킬.

능력치

이창민 (5702600)

플레이어 함선은 다음과 같은 특징을 가진다.

- **HP:** 적대적 NPC의 공격을 받으면 공격의 일정하게 줄어드는 수치. 이 수치가 1 미만으로 떨어질 경우 플레이어는 패배한다. 적을 격추하여 경험치를 모으고, 경험치를 모아 레벨 업 하면 HP가 회복된다.
- **연료:** 플레이어가 이동하는 동안 서서히 줄어드는 수치. 이 수치가 1 미만으로 떨어질 경우 플레이어는 패배한다. 적을 격추하면 연료를 회복할 수 있다.
- **이동속도:** 플레이어가 이동하는 속도. 이동속도가 높을수록 연료 소모량이 증가한다. 플레이어의 이동속도는 최대 이동속도를 넘을 수 없다.

성장

플레이어가 적을 처치하면 일정량의 경험치를 얻고, 충분한 양의 경험치를 모아서 플레이어의 함선을 레벨 업 할 수 있다.

함선이 레벨 업 할 때마다 무기 포인트를 1점 얻는다. 무기 포인트를 소모하여 원하는 무기를 해금하거나 강화할 수 있고, 하나의 무기는 4단계까지 강화할 수 있다.

4레벨, 8레벨에는 스킬 포인트 1점을 함께 얻는다. 스킬 포인트로 스킬을 해금할 수 있고, 스킬은 추가로 강화할 수 없다.

게임 종료 조건

- **승리:** 플레이어가 생존하여 목표 지점에 무사히 도달한 경우.
- **패배:** 플레이어가 목표 지점에 도달하기 전에 적대적 NPC의 공격으로 인해 HP가 1 미만으로 떨어지거나, 연료가 1 미만으로 떨어진 경우.

Git 링크

https://github.com/Liberty11346/UDP.git

사용한 에셋

스카이박스: https://assetstore.unity.com/packages/2d/textures-materials/diverse-space-skybox-11044
폭발 파티클: https://assetstore.unity.com/packages/vfx/particles/particle-pack-127325
함선 엔진 불꽃, 플레이어 및 적 포탄 파티클: https://assetstore.unity.com/packages/vfx/particles/fire-explosions/low-poly-fire-244190

이창민 (5702600)

목표 지점 오브젝트 모델: https://assetstore.unity.com/packages/3d/vehicles/space/space-station-free-3d-asset-hdrp-urp-built-in-188734
플레이어 및 적 함선 모델: https://assetstore.unity.com/packages/3d/vehicles/space/low-poly-spaceships-set-209758 조준점 스프라이트: https://assetstore.unity.com/packages/2d/gui/icons/crosshairs-216732
스킬 아이콘 스프라이트: https://game-icons.net/tags/science-fiction.html
나눔고딕 폰트(한글 폰트): https://hangeul.naver.com/fonts/search?f=nanum
스타크래프트 폰트(영어 폰트): https://www.mewpot.com/sound-effects/1139
게임 중 배경 음악: https://youtu.be/q_R_g6I7YwM?si=azpP__lKQ8nnkjfw
타이틀 화면 및 엔딩 배경 음악: https://noonsol.net/download (웅장한 음악 모음집 → Far Away(2012))
플레이어 포탄 발사 효과음: https://www.mewpot.com/sound-effects/836
플레이어 레벨 업 효과음: https://www.mewpot.com/sound-effects/461
플레이어 무기 선택 효과음: https://www.mewpot.com/sound-effects/1139

김기석 (5702341)

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 ,currentLe
vel);
    }
```

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
        }
    }
```

김기석 (5702341)

MiddleBulletFirst를 구현하기 위한 핵심 코드 부모 클래스인 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

김기석 (5702341)

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 실행시키

김기석 (5702341)

고 다크메터 오브젝트를 삭제시킨다.

MiddleWeawponThird(빨간 버튼)

```
public override void Fire()
    {
        // 이미 생성된 MiddleBullet2 를 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 (0.5 초 대기)
        yield return new WaitForSeconds(0.5f);

        // trackedBullet 이 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 시

김기석 (5702341)

키고 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

김기석 (5702341)

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 최대 체력으로 만드는 코드이다.

중거리 캐릭터의 스킬2(긴급 회피)

```
 public IEnumerable AccelerationSpeed()
```

김기석 (5702341)

```
    {
        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 목표 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 생성해준다.

구기현 (5645866)

Retreat Protocol

구현 기술 및 의사코드 정리

플레이어 조작

Move() 함수 의사코드

Move 함수:
  직진 방향으로 현재 속도 * 시간만큼 이동
 	
  스페이스 바를 누르고 있으면:
  현재 속도를 최대 속도로 천천히 증가
 	
  왼쪽 쉬프트를 누르고 있으면:
  현재 속도를 최소 속도로 천천히 감소
 	
  "Vertical" 축 입력값 * 시간 * 회전 속도만큼 X축 회전 값 감소
 	
  "Horizontal" 축 입력값 * 시간 * 회전 속도만큼 Y축 회전 값 증가
 	
  X축, Y축 회전 값을 이용해 함선의 회전을 적용

Move() 함수 설명
함선은 기본적으로 일정한 속도로 직진합니다. Space Bar를 누르는 동안 함선은 서서히 가속하고 Left SHIFT를 누르는 동안 함선은 서서히 감속합니다. W, S를 통해 상하 이동을 할 수 있고, A, D를 통해 좌우 이동을 할 수 있습니다.
Attack() 함수 의사코드

Attack 함수:
  마우스 왼쪽 버튼을 누르고 있으면:
  선택된 무기 인덱스를 발사할 무기로 설정
 	
  선택된 무기를 사용할 수 있으면:
  무기 발사
  발사한 무기의 쿨타임 시작
  포탄 발사 소리 재생
 	
  마우스 오른쪽 버튼을 누르고 있으면:
  플레이어 카메라 활성화
  메인 카메라 비활성화

플레이어 카메라의 시야각을 줌 인 시야각으로 설정 현재 카메라를 플레이어 카메라로 설정 마우스 오른쪽 버튼을 누르고 있지 않으면: 플레이어 카메라 비활성화 메인 카메라 활성화 플레이어 카메라의 시야각을 기본 시야각으로 설정 현재 카메라를 메인 카메라로 설정
Attack() 함수 설명
마우스의 왼쪽 버튼을 누르고 있으면, 숫자키를 통해 선택한 무기를 발사하고, 무기 쿨타임을 측정하기 위한 코루틴을 실행하고, 무기 발사 소리를 출력합니다. 마우스의 오른쪽 버튼을 누르고 있지 않으면, 플레이어는 메인(기본) 카메라를 통해 플레이어의 함선, 적의 함선과 발사체, 배경, 골 오브젝트 등 게임 화면을 자유롭게 살펴볼 수 있습니다. 마우스의 오른쪽 버튼을 누르고 있으면, 메인 카메라에서 플레이어 카메라로 전환합니다. 플레이어 카메라는 메인 카메라보다 더 확대되어, 플레이어가 적을 보다 쉽게 조준할 수 있도록 돕습니다.
SelectType() 함수 의사코드
SelectType 함수: 숫자 키 1을 누르면: 선택된 무기 인덱스를 0으로 설정 무기 선택 UI 효과 적용 숫자 키 2를 누르면: 선택된 무기 인덱스를 1으로 설정 무기 선택 UI 효과 적용 숫자 키 3을 누르면: 선택된 무기 인덱스를 2으로 설정 무기 선택 UI 효과 적용 숫자 키 4를 누르면: 선택된 무기 인덱스를 3으로 설정 무기 선택 UI 효과 적용
Skill() 함수 의사코드
Skill 함수: Q 키를 누르고 있으면: 0번 스킬이 사용 가능한 상태라면: 0번 스킬을 활성화 사용한 스킬의 쿨타임 시작 스킬 사용 소리 재생 E 키를 누르고 있으면: 1번 스킬이 사용 가능한 상태라면:

```
            1번 스킬을 활성화
            사용한 스킬의 쿨타임 시작
            스킬 사용 소리 재생
```

WeaponLevelUp() 함수 의사코드

```
WeaponLevelUp 함수:
  Ctrl 키를 누르고 있으면:
      숫자 키 1을 누르면:
          0번 무기의 레벨이 3 미만이면
              0번 무기의 레벨을 1 증가
              무기 포인트 1 감소
      숫자 키 2을 누르면:
          1번 무기의 레벨이 3 미만이면
              1번 무기의 레벨을 1 증가
              무기 포인트 1 감소
      숫자 키 3을 누르면:
          2번 무기의 레벨이 3 미만이면
              2번 무기의 레벨을 1 증가
              무기 포인트 1 감소
      숫자 키 4을 누르면:
          3번 무기의 레벨이 3 미만이면
              3번 무기의 레벨을 1 증가
              무기 포인트 1 감소
```

SkillLevelUp() 함수 의사코드

```
SkillLevelUp 함수:
  Ctrl 키를 누르고 있으면:
      Q 키를 누르면:
          0번 스킬의 레벨이 3 미만이면
              0번 스킬의 레벨을 1 증가
              스킬 포인트 1 감소
      E 키를 누르면:
          1번 스킬의 레벨이 3 미만이면
              1번 스킬의 레벨을 1 증가
              스킬 포인트 1 감소
```

메인 카메라

MainCameraCtrl 스크립트 의사코드

```
Start 함수:
```

구기현 (5645866)

플레이어 오브젝트 찾기 Update 함수: 플레이어 오브젝트가 존재하면: Move 함수 실행 Rotate 함수 실행 Move 함수: 마우스 입력을 기반으로 회전값 계산 카메라 위치 = 플레이어 위치 – (회전 방향 * 거리) 카메라 위치 업데이트 카메라가 항상 플레이어를 바라보게 설정 Rotate 함수: MouseX += "Mouse X" 입력값 * 감도 * 시간 MouseY += "Mouse Y" 입력값 * 감도 * 시간 MouseY를 -90과 90 사이로 제한
MainCameraCtrl 스크립트 설명
플레이어가 마우스를 움직이면 메인 카메라는 그에 맞춰 회전합니다. 메인 카메라는 일정 거리 뒤에서 플레이어를 따라갑니다.

근거리 함선의 무기 및 스킬

MeleeWeaponFirst, Second, Third, Fourth 스크립트 의사코드
PlayerWeaponBasic을 상속받음 Start 함수: 무기 이름 설정 무기 설명 설정 무기의 레벨 별 수치 설정 플레이어 카메라 정보 로드 포탄 객체 로드 Fire 함수: FireProjectile 함수 호출 FireProjectile 함수: 발사할 포탄 수 설정

구기현 (5645866)

발사할 포탄 수만큼 반복: 랜덤 위치 값 생성 랜덤 회전 값 생성 발사 위치에 생성한 랜덤 위치 값, 회전 값을 적용해 발사 포탄에 대한 레벨을 설정
MeleeWeaponFirst, Second, Third, Fourth 스크립트 설명
각각의 스크립트는 근거리 함선의 첫번째, 두번째, 세번째, 네번째 무기입니다. 세번째 무기만 한 발의 포탄을 발사하고, 이외의 무기는 모두 산탄 형태로 포탄을 발사합니다.
MeleeBulletSecond 스크립트 의사코드
PlayerBulletBasic을 상속받음 ActivateWhenHit 함수 (충돌 객체): 적 객체 가져오기 적의 방어력을 50만큼 감소 3초 후 적의 방어력 회복
MeleeBulletSecond 스크립트 설명
근거리 함선의 두번째 무기로, 적에게 적중하면 3초 동안 플레이어가 해당 적에게 가하는 피해가 50%만큼 증가합니다.
MeleeBulletThird 스크립트 의사코드
PlayerBulletBasic을 상속받음 ActivateWhenHit 함수 (충돌 객체): 플레이어 오브젝트 찾기 플레이어 스크립트 가져오기 적 오브젝트 찾기 적 객체 가져오기 적의 위치를 목표 위치로 설정 플레이어의 원래 속도를 저장 플레이어의 속도를 60으로 변경 Move 코루틴 실행 Move 코루틴 (플레이어 오브젝트, 목표 위치, 이동 시간, 원래 속도): 경과 시간 초기화 플레이어 오브젝트의 현재 위치 저장 3초동안 반복: 플레이어의 위치를 목표 위치로 부드럽게 이동 경과 시간 업데이트

구기현 (5645866)

플레이어 속도를 원래 속도로 변경
MeleeBulletThird 스크립트 설명
근거리 함선의 세번째 무기로, 적에게 적중하면 플레이어를 빠르게 적의 위치로 이동시킵니다.
MeleeBulletFourth 스크립트 의사코드
PlayerBulletBasic을 상속받음 ActivateWhenHit 함수 (충돌 객체): 적 객체 가져오기 Explosion 코루틴 실행 Explosion 코루틴 (적 객체): 2초 대기 적이 존재하면: 추가 피해 적용 explosionDamage 함수: 레벨 1: 20% 추가 피해 반환 레벨 2: 30% 추가 피해 반환 레벨 3: 40% 추가 피해 반환 레벨 4: 50% 추가 피해 반환 기본적으로 20% 추가 피해 반환
MeleeBulletFourth 스크립트 설명
근거리 함선의 네번째 무기로, 적에게 탄환이 적중하면 2초 후 해당 탄환이 한 번 더 폭발하여 레벨에 따라 추가 피해를 입힙니다.
MeleeSkillFirst 스크립트 의사코드
PlayerSkillBasic을 상속받음 Start 함수: 스킬 이름 설정 스킬 설명 설정 스킬 수치 설정 Activate 함수: 방어막 오브젝트가 없으면: 플레이어 위치에 방어막 오브젝트 생성 방어막 활성화
Barrier 스크립트 의사코드
Start 함수: 플레이어 오브젝트 찾기

구기현 (5645866)

부모를 플레이어 오브젝트로 설정

Update 함수:
방어막 오브젝트가 활성화 되어있으면:
경과 시간 업데이트
경과 시간이 지나면:
Deactive 함수 호출

Deactive 함수:
방어막 오브젝트 비활성화
경과 시간 초기화

OnTriggerEnter 함수:
충돌한 객체가 적의 총알이면:
적의 총알 파괴

MeleeSkillFirst, Barrier 스크립트 설명

스킬을 사용하면, 플레이어의 위치에 3초 동안 적의 공격을 차단하는 투명한 방어막이 생성됩니다. 3초가 지나면 방어막이 비활성화 됩니다.

MeleeSkillSecond 스크립트 의사코드

PlayerSkillBasic을 상속받음

Start 함수:
스킬 이름 설정
스킬 설명 설정
스킬 수치 설정
플레이어 오브젝트 찾기
플레이어 렌더러 가져오기
플레이어의 원래 색상 저장
플레이어 스크립트 가져오기

Activate 함수:
적에게 입힌 피해량 초기화
스킬 활성화
플레이어의 색상을 초록색으로 변경
경과 시간 업데이트
Skill 코루틴 실행

Skill 코루틴:
5초 동안:

적에게 입힌 피해량 누적 경과 시간 업데이트 플레이어 색상을 원래 색상으로 설정 스킬 비활성화 적에게 입힌 피해량이 0보다 크면: 플레이어의 체력 회복
MeleeSkillSecond 스크립트 설명
플레이어는 5초 동안 적에게 입힌 피해만큼 함선의 체력을 회복할 수 있습니다. 스킬이 활성화되면 함선의 색이 초록색으로 바뀌어, 플레이어는 스킬의 활성화 여부를 쉽게 확인할 수 있습니다.

타이틀 화면, 함선 선택 화면, 엔딩 화면 연출

타이틀 화면(TitleManager.cs)

타이틀 화면에는 게임의 제목, 플레이어 함선, 그리고 Play, Tutorial, Exit 버튼이 표시됩니다.

- Play 버튼을 누르면 함선 선택 화면으로 이동하여, 함선에 대한 정보를 확인하고 선택할 수 있습니다.
- Tutorial 버튼을 누르면 게임 조작법과 목표를 배울 수 있는 튜토리얼로 이동합니다.
- Exit 버튼을 누르면 게임을 종료합니다.

함선 선택 화면(MenuManager.cs)

함선 선택 화면에는 Melee, Middle, Range 버튼과 Back 버튼이 있습니다.

- Event Trigger의 Pointer Enter 기능을 이용해 Melee, Middle, Range 버튼에 마우스를 올리면 각 함선의 설명, 무기, 스킬 등을 확인할 수 있습니다. 플레이어가 원하는 함선을 선택하면 메뉴 UI가 비활성화되고, 플레이어 UI가 활성화됩니다. 또한, 선택한 함선의 타입이 게임 매니저로 전달되어 플레이어가 선택한 함선으로 게임을 진행할 수 있게 됩니다.
- Back 버튼을 누르면 타이틀 화면으로 돌아갑니다.

구기현 (5645866)

엔딩 화면(Ending.cs, EndingManager.cs)

플레이어가 연료와 체력이 모두 바닥나지 않고 목적지인 골 오브젝트에 도달하면 엔딩 화면으로 전환됩니다.

- 플레이어 함선이 골 오브젝트를 향해 직선으로 날아가면서 사라집니다.
- 이후 축하 메시지와 함께 타이틀 화면으로 돌아갈 수 있는 Menu 버튼과 게임을 종료할 수 있는 Exit 버튼이 표시됩니다.

게임 밸런스 설정

게임 개발 과정에서, 게임의 플레이 타임과 난이도 조정을 위한 몇 가지 수정이 필요했습니다.

첫번째로, 플레이어는 연료와 체력이 바닥나지 않은 상태로 골 오브젝트(목적지)까지 도달해야 하며, 연료를 얻기 위해서는 적을 반드시 파괴해야 합니다. 그러나 적의 이동 속도와 플레이어의 총알 속도가 거의 비슷하여, 플레이어가 적을 공격하기가 매우 어려웠습니다. 따라서, 모든 함선의 총알 속도를 적절하게 조정하여 플레이어가 보다 쉽게 적을 공격할 수 있도록 수정했습니다.

두번째로, 게임 초반부터 레벨 업에 필요한 경험치가 지나치게 높게 설정되어 있어 레벨 업이 어려웠습니다. 따라서, 경험치를 적절히 조정하여 레벨을 원활하게 올릴 수 있도록 수정했습니다.

세번째로, 적이 플레이어 근처에서, 자주 생성되었고, 그로 인해 적의 총알이 너무 가까운 거리에서 많이 날아와 피하기 어려웠습니다. 따라서, 적의 생성 위치와 생성 주기를 적절히 조정하여 플레이어가 보다 쉽게 적의 공격을 피할 수 있도록 수정했습니다.

마지막으로, 경과 시간에 따라 적의 최대 체력이 증가하도록 설정하여 시간이 흐를수록 점점 더 강력한 적들이 등장해 게임의 난이도를 점진적으로 높일 수 있도록 수정했습니다.

이창민 (5702600)

Retreat Protocol

기술 문서: 적 AI

본 문서에서는 게임에 구현된 적대적 NPC의 AI의 구현 과정에 대해 기술한다.

이동

NPC는 플레이어와 일정 거리를 유지한 채 플레이어를 따라다닌다. NPC도 플레이어와 마찬가지로 직진만 가능하며, 함선을 기울여 직진 방향을 변경함으로써 이동 경로를 수정한다.

NPC는 생성 시 플레이어와 유지할 최대 거리와 최소 거리를 구한다. 최소 거리는 30 ~ 50 사이의 무작위 값이며, 최대 거리는 최소 거리에 30 ~ 100 사이의 무작위 값을 더한 값이다.

이후 NPC는 매 프레임마다 NPC와 플레이어 사이의 거리를 계산하면서, 다음과 같은 조건에 맞춰 목표 방향을 설정한다:

- NPC와 플레이어 사이의 거리가 최소 거리보다 짧다면, NPC는 플레이어를 등지는 방향을 목표 방향으로 설정한다.
- NPC와 플레이어 사이의 거리가 최대 거리보다 멀다면, NPC는 플레이어를 바라보는 방향을 목표 방향으로 설정한다.
- 그 외의 경우 현재 이동 방향을 유지한다.

목표 방향은 플레이어와 NPC의 위치 벡터의 뺄셈으로 구한다. 이 때 목표 방향은 Vector3 형으로 계산되는데, 실제로 이 값을 사용하려면 Quaternion형이 필요하기 때문에 Quaternion.LoockAt()을 호출하여 Quaternion형으로 변환한다.

이렇게 목표 방향을 설정하였다면, Quaternion.Slerp 함수로 계산한 보간 값을 이용하여 목표 방향에 도달할 때까지 서서히 rotation 값을 조절하여 NPC의 이동 경로를 수정한다.

이동 (의사 코드)

```
void Update()
{
    바라보는 방향(Vector3.foward)으로 이동한다.
    목표_방향_계산_함수 호출
    목표 방향을 바라보도록 회전(Quaternion.Slerp로 계산한 보간 값 활용)
}

void 목표_방향_계산_함수()
{
```

```
    Vector3.Distance 함수를 활용하여 플레이어와 자기 자신 사이의 거리를 계산
    if( 최소 거리보다 짧은 경우 ) 플레이어를 등지는 방향으로 목표 방향을 설정
    else if( 최대 거리보다 긴 경우 ) 플레이어를 바라보는 방향으로 목표 방향을 설정
    else 아무것도 하지 않는다. (현재 이동 방향을 유지)
}
```

공격

NPC는 공격이 준비될 때마다 플레이어의 예상 위치를 계산하고, 이 위치를 목표로 직선 방향으로 날아가는 공격 투사체를 발사한다. 플레이어의 속도와 방향이 일정할 것을 전제하기 때문에, 플레이어는 이동 경로를 수정하여 NPC의 공격을 회피할 수 있다.

현재 플레이어의 위치와 이동 방향, 이동 속도, 현재 NPC의 위치와 NPC 포탄의 이동 속도를 기반으로, 다음과 같은 절차를 따라 플레이어의 미래의 예상 위치를 계산하여 목표 위치로 삼는다:

1. 플레이어가 이동 중인 방향과 이동 속도 정보를 담은 벡터 v1을 구한다.
2. NPC의 위치에서 플레이어의 위치로 향하는 벡터 v2를 구한다.
3. NPC가 발사하는 포탄의 이동속도와 벡터 v2의 크기를 기반으로, NPC의 위치에서 벡터 v2 방향으로 발사한 포탄이 플레이어의 위치에 도달하는 데 걸리는 시간 t1을 계산한다.
4. 시간 t1 이후, 플레이어가 벡터 v1을 따라 이동한 결과 위치 p1을 계산한다.
5. NPC의 위치에서 위치 p1으로 향하는 벡터 v3를 구한다.
6. NPC가 발사하는 포탄의 이동속도와 벡터 v3의 크기를 기반으로, NPC의 위치에서 벡터 v3 방향으로 발사한 포탄이 위치 p1에 도달하는 데 걸리는 시간 t2를 계산한다.
7. 시간 t2 이후, 플레이어가 벡터 v1을 따라 이동한 결과 위치 p2를 계산한다.
8. 이렇게 계산한 p2를 조준하여 포탄을 발사한다.

NPC의 명중률이 너무 높아 게임 난이도 조절에 어려움이 있을 경우 5~7번 단계를 생략하고, 4번 단계에서 계산한 p1을 조준한다. 명중률이 너무 낮은 경우, 5~7번 단계를 반복하여 더욱 정밀한 위치를 계산한다.

공격 (의사 코드)

```
void 공격_실행
{
    공격 속도만큼 시간이 지날 때가지 대기
    If( 시간이 충분히 지나면 )
    {
        공격_위치_계산 함수를 호출하여 공격 목표 위치를 계산
        목표 위치를 향해 날아가는 투사체 생성
```

이창민 (5702600)

```
        공격 속도 초기화
    }
}

Vector3 공격_위치_계산
{
    현재 발사할 포탄의 이동속도 참조
    플레이어의 현재 위치와 이동 방향, 속도 정보를 담은 벡터 v1 생성
    현재 위치에서 플레이어로 향하는 방향과 포탄의 이동속도 정보를 담은 벡터 v2 생성
    벡터 v2 방향으로 날아가는 포탄이 벡터 v1 위치에 도달할 시간 t 계산
    시간 t 이후 플레이어의 예상 위치 정보를 담은 위치 p1 생성
    위치 p1을 반환
    // 명중률이 낮을 경우, 플레이어의 현재 위치에 p1을 대입하여 이 과정을 반복.
}
```

이창민 (5702600)

Retreat Protocol

기술 문서: 플레이어 캐릭터(원거리형)

본 문서에서는 플레이어가 게임 중 선택할 수 있는 세 가지 스타일 중 하나인 원거리형 스타일의 구현 과정에 대해 기술한다.

무기

원거리형 스타일은 다음과 같은 네 가지 무기를 사용할 수 있다:

1	**철갑소이탄**	짧은 간격으로 두 번 발사하는 2연장 함포.
2	**근접유도탄**	가까운 적을 추적하여 큰 피해를 입히는 2연장 함포.
3	**EMP자폭탄**	주변의 적을 탐지하면 폭발하여 피해를 입히면서 밀어내는 포탄.
4	**플라즈마 포**	2초 동안 에너지를 충전하여 강력한 피해를 입히는 레이저 무기.

각 무기의 최대 사거리에 따로 제한이 있진 않지만, 포탄이 플레이어로부터 300거리 이상 떨어진 경우 게임에서 제거된다.

무기1 – 철갑소이탄

조준점을 기준으로 x축에서 ±1만큼 떨어진 위치에 서로 평행하게 날아가는 두 개의 포탄을 0.2초 간격으로 두 번 발사하여 총 4발의 포탄을 발사한다. 철갑소이탄에 명중한 적은 4초동안 방어력이 20% 감소한다.

철갑 소이탄에 적중한 적의 방어력을 떨어트린 후, 4초 후에 다시 방어력을 복구해야 하는데, 이 시점엔 이미 철갑소이탄이 적중하여, 방어력을 복구할 포탄 오브젝트가 사라진 상태이다. 이를 해결하기 위해 적의 코드에서 방어력을 초기화하는 함수를 구현하고, 해당 함수를 호출하는 방법으로 구현한다.

```
class 철갑소이탄_무기
{
    void 투사체_발사
    {
        투사체_생성 함수 호출
        0.2초간 대기
        투사체_생성 함수 호출
    }

    void 투사체_생성
    {
```

이창민 (5702600)

```
        for( int i = -1 ; i < 2 ; i += 2 )
        {
            카메라.position.x + i 위치에 카메라와 동일한 rotation을 가지는 투사체 생성
        }
    }
}

class 철갑소이탄_포탄
{
    void Update()
    {
        바라보는 방향으로 이동한다
        플레이어와 자기 자신 사이의 거리를 계산
        if( 현재 거리 > 300 ) 자기 자신을 삭제
    }

    void OnTriggerEnter(Collider 적)
    {
        적.HP -= 포탄 공격력
        적.방어력 = -20
        적.방어력_복구(4) 함수 호출
        자기 자신을 삭제
    }
}

class 적
{
    void 방어력_복구(int 대기시간(초))
    {
        대기시간(초) 동안 대기
        방어력 초기화
    }
}
```

무기2 – 근접유도탄

조준점을 기준으로 x축에서 ±1만큼 떨어진 위치에 서로 평행하게 날아가는 두 개의 포탄을 발사하여 총 2발의 포탄을 발사한다. 근접유도탄은 날아가는 도중 주변의 적을 발견하면 해당 적의 위치로 방향을 변경한다.

이창민 (5702600)

```
class 근접유도탄_무기
{
    void 투사체_발사
    {
        for( int i = -1 ; i < 2 ; i += 2 )
        {
            카메라.position.x + i 위치에 카메라와 동일한 rotation을 가지는 투사체 생성
        }
    }
}

class 근접유도탄_포탄
{
    void Update()
    {
        바라보는 방향으로 이동
        if( 적 발견 = false ) 적_탐색 함수 호출
    }

    void 적_탐색
    {
        Enemy 태그를 가진 모든 오브젝트를 검색하여 적_리스트 생성
        foreach( 적 in 적_리스트 )
        {
            자기 자신과 오브젝트 사이의 거리를 측정(Vector3.Distance)
            if( 거리 < 탐색 거리 ) LookAt(오브젝트)
        }
    }

    void OnTriggerEnter(Collider 적)
    {
        적.HP -= 포탄 공격력
        스스로를 삭제
    }
}
```

무기3 - EMP자폭탄

이창민 (5702600)

조준점 방향으로 정확하게 날아가는 하나의 포탄을 발사한다. EMP자폭탄은 날아가는 도중 주변의 적을 발견하면 폭발하며, 폭발에 휘말린 적들을 피해를 입고, 폭발의 중심부로부터 외각 방향으로 밀쳐진다.

```
class EMP자폭탄_무기
{
    void 투사체_발사
    {
        EMP자폭탄 오브젝트를 생성하여 바라보는 방향으로 rotation 설정
    }
}

class EMP자폭탄_포탄
{
    void Update()
    {
        바라보는 방향으로 이동한다
        적_탐색 함수 호출
        플레이어와 자기 자신 사이의 거리를 계산
        if( 현재 거리 > 300 ) 스스로를 삭제
    }

    void 적_탐색
    {
        Enemy 태그를 가진 모든 오브젝트를 검색하여 적_리스트 생성
        foreach( 적 in 적_리스트 )
        {
            자기 자신과 오브젝트 사이의 거리를 측정(Vector3.Distance)
            if( 거리 < 탐색 거리 ) EMP폭발(적_리스트) 함수 호출
        }
    }

    }

    void EMP폭발(list 적_리스트)
    {
        폭발 연출용 오브젝트를 생성
        foreach( 적 in 적_리스트 )
        {
```

```
            자기 자신과 오브젝트 사이의 거리를 측정(Vector3.Distance)
            if( 거리 < 폭발 거리 )
            {
                적.HP -= 포탄 공격력
                폭발의 외각 방향으로 밀쳐낸다(AddForece.Impulse)
            }
        }
        스스로를 삭제
    }
}
```

무기4 – 플라즈마 포

발사 시 조준 방향을 기억하고, 2초 후 기억한 방향으로 발사되어 접촉하는 모든 적에게 피해를 입히는 레이저 포를 발사한다.

일반적인 포탄과 달리 투사체가 아니고, 접촉하는 모든 적에게 피해를 입혀야 하기 때문에 BoxCollider 컴포넌트를 가진 광선 오브젝트를 생성하는 방식으로 구현한다.

기본적으로 BoxCollider는 박스의 한 가운데를 중점으로 삼기 때문에 광선이 플레이어로부터 양옆으로 발사되는 모양을 보이는데, 이는 Center 값을 조절함으로써 콜라이더의 중점 위치를 변경하여 해결할 수 있다.

```
class 플라즈마포_무기
{
    void 광선_발사
    {
        현재 카메라의 각도 저장
        2초간 대기
        저장해둔 각도와 동일한 rotation을 가지는 광선 오브젝트 생성
    }
}

class 플라즈마포_광선
{
    void Start()
    {
        4초간 대기 후 스스로를 삭제 (Invoke)
    }
```

이창민 (5702600)

```
    void OnTriggerEnter(Collider 적)
    {
        적.HP -= 광선 공격력
        스스로를 삭제
    }
}
```

스킬

원거리형 스타일은 다음과 같은 두 가지 스킬을 사용할 수 있다:

Q	**전자기장**	함선 주변에 있는 적 포탄의 이동을 멈춘다.
E	**비상 발전**	다음 번에 사용하는 무기의 대기시간을 제거하여 곧바로 충전시킨다.

스킬1 – 전자기장

사용 시 씬 내의 적 포탄을 모두 검색한 후, 플레이어 함선 주변의 포탄만 골라 이동속도를 0으로 만들어 이동을 멈추게 한다. 사용 후 12초가 지나면 스킬을 재사용할 수 있다. 전자기장에 의해 이동이 멈춘 포탄은 시간이 지나도 이동속도를 회복하지 않는다.

```
void 사용
{
    EnemyBullet 태그를 가진 모든 오브젝트를 검색하여 적포탄_리스트 생성
    foreach( 적_포탄 in 적포탄_리스트 )
    {
        플레이어와 적_포탄 사이의 거리를 측정 (Vector3.Distance)
        if( 측정 거리 < 기준 거리 ) 적_포탄.moveSpeed = 0
    }
}
```

스킬2 – 비상 발전

사용 시 플레이어는 비상 발전 상태에 돌입하고, 비상 발전 상태에서 발사하는 첫번째 무기의 대기시간을 즉시 0으로 만들어 제거한다. 이후 비상 발전 상태를 해제한다. 사용 후 (비상 발전 상태에 돌입한 순간으로부터) 12초가 지나면 재사용할 수 있다.

비상 발전을 사용한 후 12초 동안 아무런 무기도 발사하지 않고 있다가, 한 번 더 비상 발전을 사용하더라도 대기시간이 두 번 제거되진 않는다.

플레이어가 무기를 선택하여 발사한 후, 무기 코드에서 스스로 자신의 대기시간(쿨타임)을 초기

이창민 (5702600)

화 하는 게 아니라, 플레이어 코드에서 방금 발사한 무기의 대기시간을 초기화하는 함수를 호출해주는 구조로 구현되어 있다. 여기서, 비상발전을 사용한 후에는 쿨타임을 초기화하지 않도록 조건문을 추가하는 방법으로 구현한다.

```
class 비상발전
{
    void 사용
    {
        플레이어.비상발전_버프 = true
    }
}

class 플레이어
{
    void 공격
    {
        선택된_무기.포탄_발사 함수 호출
        if( 비상발전_버프 == false ) 선택된_무기.쿨타임_시작 함수 호출
        else 비상발전_버프 = false
    }
}
```

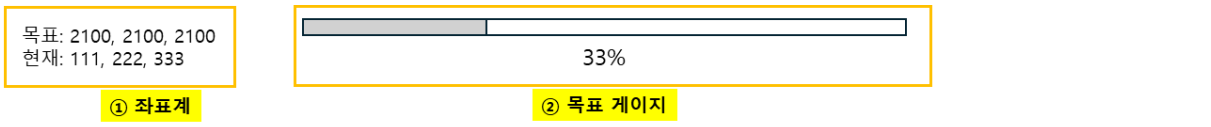
이창민 (5702600)

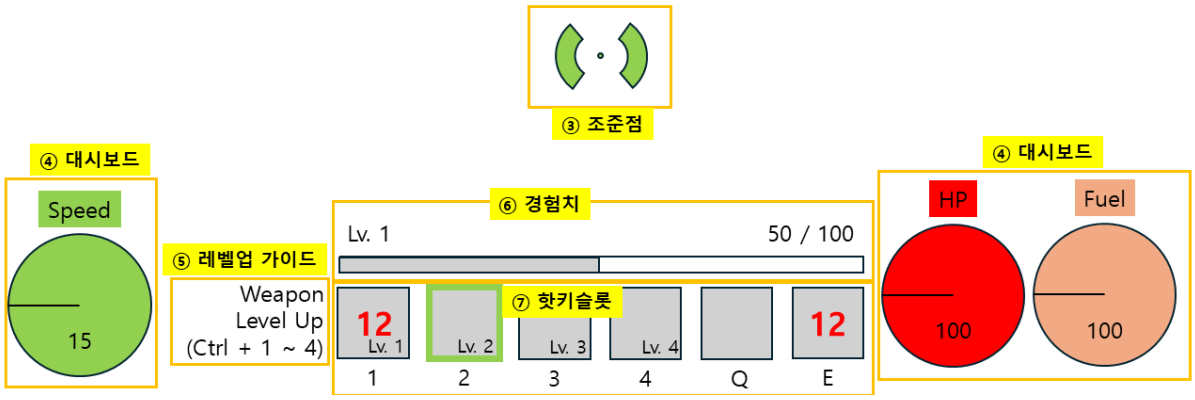
Retreat Protocol

기술 문서: UI

본 문서에서는 게임에 사용된 UI의 구현 과정에 대해 기술한다. 특별한 기술 없이 유니티에서 제공하는 버튼 컴포넌트만으로도 구현 가능한 타이틀 화면과 함선 선택 화면의 UI는 생략하고, 게임 중 화면 UI에 대해서만, 그 중에서도 특별한 기술을 사용한 UI의 구현 과정만 기술한다.

게임 중 화면





목표 게이지

현재 게임 진행도(플레이어가 목표에 가까운 정도)를 직관적으로 표시하기 위한 게이지이다.

게임 시작 시 플레이어와 목표 위치 사이의 거리 md를 계산하여 저장하고, 매 프레임마다 현재 플레이어와 목표 위치 사이의 거리 cd를 계산한다. 이후 md와 cd의 비를 계산하여 현재 게임 진행도(백분율)를 산출한다.

게이지는 이렇게 계산한 현재 진행도를 기반으로 표시된다. 0%일 경우 게이지가 완전히 비어 있고, 100%일 경우 완전히 차오른다. 그 외의 경우 현재 진행도에 맞춰 적절히 게이지를 채운다. 현재 게임 진행도를 보다 정확하게 알 수 있도록 게이지 하단에 텍스트(정수)로도 표시한다.

게이지는 두 개의 막대 이미지, 게이지의 배경이 될 이미지 BG와 게이지 내부에서 차오를 이미지 G로 구성된다. G의 x pivot을 0으로 설정하면 x축의 왼쪽 끝을 중점으로 삼게 되는데, 이 상태

이창민 (5702600)

에서 막대 이미지의 width 값을 조절할 경우, 중점을 기준으로 늘었다 작아지므로 게이지가 차는 연출을 구현할 수 있다.

```
void Update()
{
    게이지_출력 함수 호출
}

int 백분율_계산
{
    현재 플레이어와 골 오브젝트 사이의 거리를 계산(Vector3.Distnace 활용)
    현재 거리와 최대 거리의 백분율을 계산
    if( 현재 거리 값 < 최대 거리 값 ) 백분율 = 0
    이렇게 계산한 백분율을 반환
}

void 게이지_출력
{
    백분율_계산 함수 호출하여 백분율을 반환 받음
    백분율의 비율만큼 G의 width를 BG의 n%로 계산
    계산한 G의 width를 실제로 적용(Mathf.lerp 보간 활용)
    현재 백분율을 텍스트로도 출력(TextMeshProUGUI 활용)
}
```

대시보드

함선의 속도와 HP, 연료를 직관적으로 표시하는 대시보드다. 속도 대시보드는 화면의 좌측 하단에, HP와 연료 대시보드는 화면의 우측 하단에 표시한다. 대시보드는 바늘과 텍스트로 구성되어 있다. 바늘은 플레이어에게 현재 값을 추상적으로 보여주고, 텍스트는 정확한 값으로 보여준다.

대시보드가 가리키는 변수의 현재값과 최대값의 비를 백분율로 계산하고, 계산한 백분율에 맞춰 적절한 값을 가리키도록 바늘의 각도를 조정한다. 바늘의 각도가 변경될 때, 보간을 사용하여 부드럽게 변경되도록 한다.

대시보드의 바늘은 긴 막대 이미지를 사용하는데, 막대 이미지의 x pivot을 1로 설정하면 x축의 오른쪽 끝을 중심으로 삼을 수 있다. 막대를 회전시키면 중점을 중심으로 회전하기 때문에, 대시보드의 바늘이 움직이는 연출을 구현할 수 있다.

하나의 코드로 서로 다른 값을 가지는 세 가지 대시보드를 출력하기 위해, 오브젝트의 이름을 기반으로, 해당 대시보드가 보여야 하는 변수를 따로 참조하여 출력하게 한다.

이창민 (5702600)

```
void Start()
{
    gameObejct.name을 기반으로 자신이 담당할 변수의 이름을 string 형으로 저장
    string 값을 기준으로 변수를 PlayerCtrl 클래스에서 검색 (typeof().GetField() 활용)
    검색한 변수의 최대값, 최소값 변수도 함께 검색
    검색한 변수들을 float 형으로 변환하여 저장
}

void Update()
{
    바늘_각도_적용 함수 호출
}

int 바늘_각도_계산
{
    대시보드로 출력할 변수의 최대값과 현재 값의 비를 백분율로 계산
    if( 현재값 < 최소값 ) 백분율 = 0
}

void 바늘_각도_적용
{
    바늘_각도_계산 함수를 호출하여 백분율을 반환 받음
    백분율을 기반으로 0에서 180 사이의 각도 값을 계산
    계산한 각도 값을 바늘 이미지의 rotation에 적용(Mathf.lerp 보간 활용)
}
```

핫 키 슬롯

플레이어가 사용 가능한 무기와 스킬의 아이콘, 재사용 대기시간, 레벨, 단축키를 표시한다.

무기와 스킬의 레벨이 1미만인 경우, 아이콘에 (1,1,1,0.5f) 색상을 적용하여 흐리게 표시한다. 레벨이 1이상인 경우, 아이콘에 (1,1,1,1) 색상을 적용하여 또렷하게 표시한다.

무기는 4레벨로 분화되기 때문에, 아이콘의 우측 하단 모서리에 현재 레벨을 텍스트로 표시한다. 스킬은 레벨이 분화되지 않기 때문에, 따로 레벨을 표시하지 않는다.

현재 선택된 무기의 경우, 아이콘 테두리의 색상을 변경하여 강조한다.

무기 또는 스킬이 재사용 대기상태(쿨타임)인 경우, 남은 대기시간을 아이콘의 중앙에 텍스트(정수)로 표시한다. 대기시간은 1초마다 1씩 줄어든다.

이창민 (5702600)

```
void Update()
{
    if( 플레이어의 무기 선택 감지 시 )
    {
        if( 지금 선택한 무기가 이미 선택된 무기와 다른가? )
        {
            현재 선택된 무기 = 지금 막 선택한 무기
            현재 선택된 무기의 아이콘 테두리를 강조
        }
        else 아무것도 하지 않음
    }

    무기_쿨타임_표시 함수 호출
    무기_레벨_표시 함수 호출
}

void 무기_쿨타임_표시
{
    if( 현재 쿨타임 > 0 ) 현재 쿨타임을 텍스트로 출력 (TextMeshProUGUI 활용)
    else 현재 쿨타임을 공백으로 출력 (TextMeshProUGUI)
}

void 무기_레벨_표시
{
     if( 현재 레벨 < 0 ) 아이콘 색상을 new Color(1,1,1,0.5f)로 변경
     else 아이콘 색상을 new Color(1,1,1,1)로 변경

     현재 레벨을 텍스트로 출력 (TextMeshProUGUI)
}
```