Super Rookie Challenge

게임 클라이언트 추가 구현 사항

Contents

추가 기능(레벨업)

```
public void GainExp(int amount)
{
    Exp += amount;

    // 경험치가 현재 레벨의 요구치를 충족하는지 확인
    while (Exp >= CurrentExpRequirement)
    {
        // 경험치가 충분하다면 레벨 업
        LevelUp();
    }
}

private void LevelUp()
{
    Exp -= CurrentExpRequirement; // 현재 레벨의 경험치 요구량만큼 차감
    Level++;
    MaxHealth+=Const.AddMaxHealthByLevelUp;
}

public void Upgrade()
{
    UpgradeCnt++;
    AttackPower++;
    MaxHealth+= Const.AddMaxHealthByUpgrade;
}
```

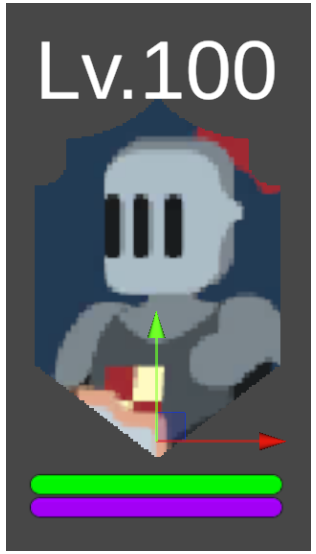
기능 설명 :

캐릭터가 몬스터를 처치할 때마다 경험치를 획득하고,

필요 경험치를 충족시키면 레벨이 상승하여 능력치가 상승합니다.

경험치를 얻는 기준은 몬스터가 죽기 전 마지막으로

때린 플레이어 유닛이 획득합니다.



경험치바

<OnFieldCharacterUI> 프리팹

체력게이지 UI

기능설명 :

유닛마다 별도의 UnitHpBar라는 스크립트가 존재하며,

해당 스크립트는 유닛의 체력에 비례해서 슬라이더를 조절하는 역할을 수행 합니다.

두트윈을 기능을 활용하여 유닛의 피가 자연스럽게 깎이게 하였고

유닛의 체력이 변화할때마다 이벤트가 호출되어 슬라이더의 UI를 업데이트 합니다.

```
public void SetHpbar()
{
    double ratio = thisUnit.CurrentHealth / (double)thisUnit.CurrentMaxHealth;  // 최대 체력 대비 현재 체력의 비율 계산

    ratio = System.Math.Clamp(ratio, 0f, 1f);  // 비율을 0과 1 사이로 제한

    DOTween.Kill(this);

    HpBarFront.transform.localScale = new Vector2((float)ratio * HpBarBackGround.transform.localScale.x, HpBarBackGround.transform.localScale.y);
    HpBarMiddle.transform.localScale = new Vector2((float)ratio * HpBarBackGround.transform.localScale.x, HpBarBackGround.transform.localScale.y);
}

private void WhenDamage()
{
    double ratio = thisUnit.CurrentHealth / thisUnit.CurrentMaxHealth;

    ratio = System.Math.Clamp(ratio, 0f, 1f);

    HpBarFront.transform.localScale = new Vector2((float)ratio * HpBarBackGround.transform.localScale.x, HpBarBackGround.transform.localScale.y);
    HpBarMiddle.transform.DOScaleX(HpBarFront.transform.localScale.x, 0.5f).OnComplete(() =>
    {
        if (thisUnit is not null && thisUnit.UNIT_STATE != UNIT_STATE.DEAD)
        {
            HpBarMiddle.transform.localScale = new Vector2(HpBarFront.transform.localScale.x, HpBarBackGround.transform.localScale.y);
        }
    });
}
```

능력치 성장 시스템

기능설명:

캐릭터들의 데이터를 읽어와 UI를 구성합니다.
또한 레벨업 버튼 클릭시 현재 가진 돈과 요구량을 비교해서
조건 만족시 재화를 소모해서 유닛의 업그레이드를 실행합니다.
해당 셀은 이벤트로 구독되어 유닛이 레벨업 하거나 골드를 획득할때 즉시 업데이트 됩니다

```
public void UpdateUI()
{
    if (characterData != null)
    {
        double requireUpgradeGold = Const.RequireUpgradeGoldPerCnt * (characterData.UpgradeCnt + 1);

        UpgradeBtn.interactable = CurrencyManager.Instance.IsEnoughCurrency(requireUpgradeGold);
        Portrait.sprite = characterData.Portrait;
        HpValue.text = characterData.MaxHealth.ToString();
        AtkValue.text = characterData.AttackPower.ToString();
        RequireGoldCnt.text = requireUpgradeGold.ToString();
    }
}

public void OnClickLevelUpBtn()
{
    double requireUpgradeGold = Const.RequireUpgradeGoldPerCnt * (characterData.UpgradeCnt + 1);

    if (CurrencyManager.Instance.IsEnoughCurrency(requireUpgradeGold))
    {
        CurrencyManager.Instance.UseGold(requireUpgradeGold);
        characterData.Upgrade();
        UpdateUI();
    }
}
```

몬스터 패트롤 및 추적

```
// 타겟 찾기 또는 업데이트
if (unit.AttackTarget == null || unit.AttackTarget.UNIT_STATE == UNIT_STATE.DEAD)
{
    unit.AttackTarget = unit.FindAttackTarget(); // 가장 가까운 플레이어를 새 타겟으로 설정
    if (unit.AttackTarget == null)
    {
        // 가까운 플레이어가 없다면 순찰 상태로 전환
        unit.UNIT_STATE = UNIT_STATE.PATROL;
        return;
    }
    unit.UnitFlip(unit);  // 새 타겟에 따라 방향 조정
}
```

```
public override BaseUnit FindAttackTarget()
{
    BaseUnit closest = null;
    float minDistance = EnemyData.ChaseRange;
    Vector3 currentPosition = transform.position;

    if (UnitManager.Instance.SummonCharacterList.Count > 0)
    {
        foreach (BaseUnit monster in UnitManager.Instance.SummonCharacterList)
        {
            float distance = Vector3.Distance(monster.transform.position, currentPosition);
            if (distance < minDistance)
            {
                closest = monster;
                minDistance = distance;
            }
        }
    }

    return closest;
}
```

기능설명:

몬스터는 ChaseRange내의 플레이어를 찾지 못하면
Patrol 상태에 들어갑니다.
Patrol 상태에서는 무작위 목적지로 이동하고,
이동하는 중에 플레이어를 탐지하면
Move상태로 들어가 플레이어를 추적합니다.
또한 Move상태에서 플레이어가 시야내에서 사라지면
다시 Patrol상태로 돌입합니다.

보스 몬스터 및 스테이지 진행 메커니즘

기능설명:

스테이지 시작시 일반 몬스터들은 플레이어 근처에서 소환되고,
몬스터 재생성 횟수가 최대치가 아닌경우, 설정한 최대갯수에 따라 개체수가 유지됩니다.
일반 몬스터를 일정 이상 처치한 경우 보스몬스터가 소환되고
보스 몬스터를 처치하면 모든 일반몬스터들이 처치되고 다음 스테이지로 이동합니다.

```
public void MonsterReSpwan()
{
    if (isStageClear == true)
    {
        return;
    }

    if (respawnCount > KillMonsterRequiredForBoss && IsBossSpawn == false)
    {
        IsBossSpawn = true;
        SpawnMonster("Boss");
    }
    else if (UnitManager.Instance.SummonMonsterList.Count < maxMonsters && respawnCount < respawnLimitPerStage)
    {
        SpawnMonster("Monster");
    }

}
```

스킬 이펙트 구현

```
public abstract class Skill : MonoBehaviour
{
    public float attackRange = 2f;
    public float skillCoefficient;

    public abstract void Activate(BaseUnit user, BaseUnit target);

    public abstract void ApplyEffect(BaseUnit target);

    public virtual void ActiveEffect()
    {

    }

    public virtual void DeActiveEffect()
    {

    }
}
```

기능설명:

해당 게임의 스킬들은 Skill이라는 클래스를 상속받으며,
각각 스크립트에 효과나 이펙트 활성화 타이밍 등을
오버라이드하여 사용하고 있습니다.
아쳐는 일반화살 대신 불화살이 나가거나
도적은 범위공격시 독가스가 나오고,
힐이나 스턴은 대상이 된 유닛의 콜라이더를 중심으로
알맞은 위치에 이펙트가 생성되도록 하였습니다.

<힐 스킬>

<스턴 상태이상>

<힐러 평타>

<범위 공격>

추가 개발기능(재화)

```
// 골드를 획득 및 이벤트로 ui업데이트
public void AddGold(double goldCnt)
{
    Gold+= goldCnt;

    CurrencyInfUpdateAction.Invoke();
}

// 골드를 사용 및 이벤트로 ui업데이트
public void UseGold(double goldCnt)
{
    if (Gold >= goldCnt)
    {
        Gold -= goldCnt;
    }

    CurrencyInfUpdateAction.Invoke();
}

// 현재 가지고 있는 골드량 확인
public double GetGoldCnt()
{
    return Gold;
}

// 소지 골드가 요구량만큼 있는지 확인
public bool IsEnoughCurrency(double requireCnt)
{
    if (requireCnt <= Gold)
    {
        return true;
    }

    return false;
}
```

기능 설명:
재화를 관리하는 매니저 클래스 입니다.
골드 획득, 소모시 액션 이벤트가 실행되며
이벤트에 구독해놓은 함수들(UI업데이트등)이 자동으로 실행됩니다.

추가 개발기능(데미지 폰트)

```
public void Init(string _s, BaseUnit _unit, FONT_TYPE _Type)
{
    text.text = _s;

    gameObject.transform.localScale = Vector3.one * 0.8f;
    gameObject.transform.SetParent(_unit.transform);

    gameObject.transform.localPosition = new Vector3(0, 0, 0.5f);
    gameObject.SetActive(true);

    Color topColor = Color.white;
    Color bottomColor = Color.white;
```

기능설명:

유닛의 체력에 변동이 생길때마다
데미지 폰트가 생성됩니다.
인자로는 데미지나 힐량, 대상유닛, 폰트타입을 받습니다.
폰트타입에 따라 데미지폰트의 색깔이 달라지며,
폰트가 생성될때는 유닛마다 존재하는 콜라이더의 중심을
기준으로 유닛이 크던 작던 일정한 위치에 생성되도록 합니다.

```
private void PlayNumberTweening(Collider collider)
{
    // 캐릭터의 렌더러 또는 콜라이더로부터 높이를 얻습니다.
    float characterHeight = collider.bounds.size.y;

    // 트윈이 시작할 위치를 캐릭터의 현재 위치 + 캐릭터의 높이로 설정합니다.
    float startHeight = transform.position.y + characterHeight;

    // 모든 데미지 폰트가 같은 높이만큼 올라가도록 설정합니다
    Vector3 endPosition = new Vector3(transform.position.x, startHeight+0.3f, transform.position.z);

    // DOTween을 사용하여 데미지 폰트를 원하는 위치로 이동시킵니다.
    transform.DOMoveY(endPosition.y, 0.5f).OnComplete(() =>
    {
        ObjectPool.Instance.ReStoreDamageFont(this);
    });
}
```

추가 개발기능
(오브젝트풀링)

```
public GameObject GetObject(string tag)
{
    if (!poolDictionary.ContainsKey(tag) || poolDictionary[tag].Count == 0)
    {
        return null;
    }

    GameObject obj = poolDictionary[tag].Dequeue();

    obj.SetActive(true);
    return obj;
}


public void ReturnObject(GameObject obj, string tag)
{
    if (!poolDictionary.ContainsKey(tag))
    {
        Debug.LogError("Pool with tag " + tag + " doesn't exist.");
        return;
    }

    obj.SetActive(false);
    poolDictionary[tag].Enqueue(obj);
}
```

기능설명:

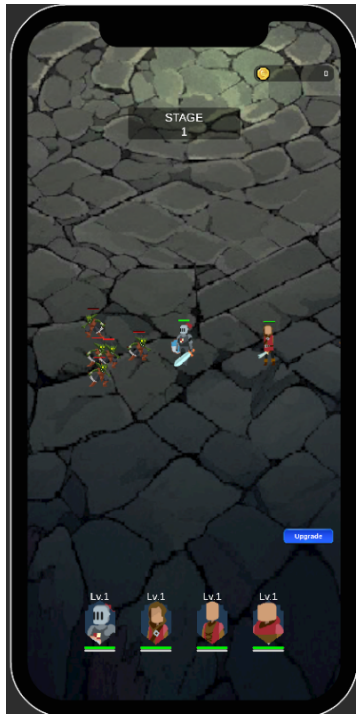
오브젝트의 생성과 파괴가 자주 일어나는
장르의 게임이다보니 성능향상을 위해
오브젝트 풀링을 사용했습니다.
유닛이 죽거나 이펙트등이 사용되면
파괴하는게 아닌 풀에 저장함으로써
다음번에 사용할때도 풀에 있는 오브젝트들을
가져올수 있도록하여 성능상 이점을 더했습니다.
모든 오브젝트들은 poolDictionary에 저장되며
회수나 반납도 전부 한곳에서 관리하게 됩니다.

추가 개발기능
(캔버스 조정)

```
    private void SetResolution()
    {
        if (isResolutioned)
        {
            return;
        }

        if (canvasScaler == null)
        {
            canvasScaler = gameObject.GetComponent<CanvasScaler>();
        }

        safeArea = Screen.safeArea;
        minAnchor = safeArea.position;
        maxAnchor = safeArea.position + safeArea.size;

        minAnchor.x /= Screen.width;
        minAnchor.y /= Screen.height;
        maxAnchor.x /= Screen.width;
        maxAnchor.y /= Screen.height;

        AnchoredObject.anchorMin = minAnchor;
        AnchoredObject.anchorMax = maxAnchor;

        float ratio = Screen.height / (float)Screen.width;
        canvasScaler.matchWidthOrHeight = (ratio >= 1.77f) ? 0.0f : 1f;

#if UNITY_EDITOR
        isResolutioned = false;
#else
    isResolutioned = true;
#endif
```

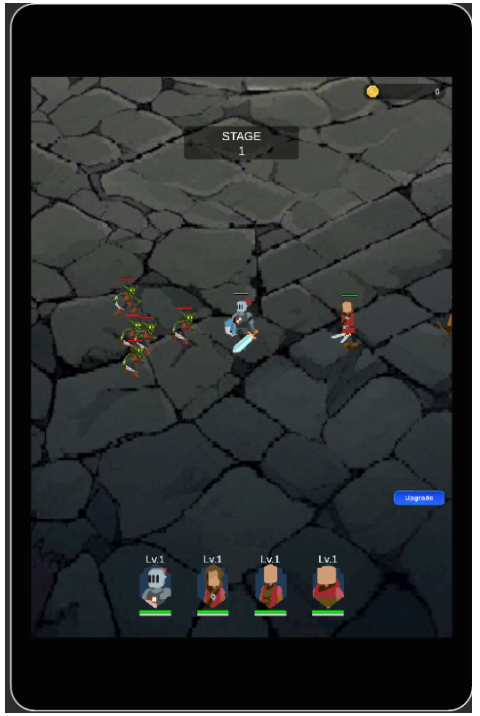
기능설명:

해당 메서드는 UI 화면 해상도와 SafeArea를 설정하는 역할을 합니다.
디바이스의 화면 SafeArea를 기반으로 UI의 최소 및 최대 앵커 포인트를 설정하고, 화면 비율에 따라 캔버스의 크기를 설정합니다.
이러한 역할로 인해 다양한 디바이스에서 UI가 일관되게 보이도록 합니다.



<아이폰12>

<갤럭시 폴드>



<아이패드>

감사합니다.