땜빵해적단 키우기 샘플코드

1. 튜토리얼
2. 소환
3. 팝업
4. 아이템
5. 보상 및 재화
6. 레드닷

1. 튜토리얼

```
/// <summary>
/// 튜토리얼 손가락 UI를 활성화하고 지정된 버튼 위에 포인터를 이동시킵니다.
/// 커버 이미지의 셰이더 홀을 갱신하여 해당 버튼만 클릭 가능하도록 설정합니다.
/// </summary>
public void ActiveFinger(GameMng.HELPER helper)
{
    isReverse = false;

    // 팝업 닫기 및 튜토리얼 상황 정리
    closeAllPopupMoveHelper(helper);

    // 커버 이미지와 손가락 UI 활성화
    CoverImage.SetActive(true);
    fingerObjRect.gameObject.SetActive(true);

    // 튜토리얼 대상에 따라 UI 준비
    PrepareUIForHelper(helper);

    // 대상 RectTransform 가져오기
    RectTransform targetRect = GetHelperRectTransformOriginal(helper);
    if (targetRect != null)
    {
        // 손가락 위치 및 크기 지정
        Vector2 pos = GetHelperCanvasPositionModified(helper);
        fingerObjRect.anchoredPosition = pos;
        fingerObjRect.sizeDelta = targetRect.rect.size;

        // 셰이더 커버 이미지에서 홀 영역 갱신
        var coverRect = coverImage2.GetComponent<RectTransform>();
        var coverImage = coverImage2.GetComponent<Image>();
        if (coverImage?.material != null && coverRect != null)
        {
            Vector2 holeUV = ComputeHoleUV(targetRect, coverRect); // 클릭 가능한 중심
            Vector2 holeSize = ComputeHoleSizeRect(targetRect, coverRect); // 클릭 가능한 범위
            coverImage.material.SetVector("_HoleCenter", new Vector4(holeUV.x, holeUV.y, 0, 0));
            coverImage.material.SetVector("_HoleSize", new Vector4(holeSize.x, holeSize.y, 0, 0));
        }
    }
    else
    {
        // 타겟 없을 시 기본 위치로 초기화
        fingerObjRect.anchoredPosition = Vector2.zero;
    }

    // 손가락 애니메이션 및 자동 해제 코루틴 시작
    PlayAnimation(isReverse ? "idle_reverse" : "idle");
    if (autoDeactivateCoroutine != null) StopCoroutine(autoDeactivateCoroutine);
    autoDeactivateCoroutine = StartCoroutine(FingerAutoDeactivateCoroutine());
}
```

기능: 튜토리얼 단계에서 손가락 포인터를 활성화하고, 버튼 위에 위치시켜 클릭을 유도하며, 커버 이미지 셰이더를 통해 지정된 버튼만 클릭 가능하도록 마스킹 처리

특징: PrepareUIForHelper()와 연계되어 튜토리얼 흐름에 따라 자동 UI 동선 설정. ComputeHoleUV()를 통해 대상 UI 의 중심 좌표를 계산하고, Shader 의 _HoleCenter 및 _HoleSize 속성에 적용하여 마스킹

```
/// <summary>
/// 튜토리얼 손가락(FingerObj) 클릭 시 호출되는 함수.
/// 현재 가리키는 버튼을 클릭 처리하고, 다음 튜토리얼 단계로 넘어간다.
/// </summary>
참조 1개
public void OnHelperButtonClicked()
{
    // 다음 스텝 활성화 대기 중이면 무시
    if (isAwaitingNextStep)
        return;

    // 아직 처리할 스텝이 남아 있는 경우
    if (currentHelperIndex < currentHelperSteps.Count)
    {
        // ① 현재 단계의 헬퍼 enum 가져오기
        var helper = currentHelperSteps[currentHelperIndex];

        // ② 필요한 UI 세팅 (예: 탭 전환, 팝업 열기 등)
        PrepareUIForHelper(helper);

        // ③ 해당 헬퍼에 연결된 버튼 가져오기
        var btn = GetButtonFromHelper(helper);
        if (btn != null && btn.interactable)
        {
            // 버튼 클릭 이벤트 호출
            btn.onClick.Invoke();
        }
        else
        {
            Debug.LogWarning($"[Helper] 버튼이 없거나 비활성화됨: {helper}");
        }

        // ④ 다음 단계로 인덱스 증가 및 다음 스텝 활성화
        currentHelperIndex++;
        ActivateNextHelper();
    }
    else
    {
        // 모든 단계 완료 시 손가락 UI 비활성화
        DeActiveFinger();
    }
}
참조 1개
```

역할: FingerObj 를 클릭했을 때 실제 UI 버튼을 클릭한 것처럼 처리하고, 다음 튜토리얼 단계로 자동 진행

보강 포인트: PrepareUIForHelper()를 활용해 사전 UI 상태 준비

클릭 불가능한 경우 디버그 로그 출력

마지막 단계에서는 DeActiveFinger()로 마무리 처리

```
/// <summary>
/// 대상 RectTransform의 중심을 기준으로 커버 이미지 상에서 셰이더에 전달할 UV 좌표(0~1)를 계산
/// </summary>
private Vector2 ComputeHoleUV(RectTransform targetRect, RectTransform coverRect)
{
    Vector3 worldCenter = targetRect.TransformPoint(targetRect.rect.center);
    Vector2 screenPoint = RectTransformUtility.WorldToScreenPoint(canvas.worldCamera, worldCenter);

    Vector2 localPoint;
    RectTransformUtility.ScreenPointToLocalPointInRectangle(coverRect, screenPoint,
        canvas.worldCamera, out localPoint);

    Vector2 uv;
    uv.x = (localPoint.x / coverRect.rect.width) + 0.5f;
    uv.y = (localPoint.y / coverRect.rect.height) + 0.5f;
    return uv;
}

/// <summary>
/// 대상 UI의 RectTransform 크기를 기준으로 셰이더 홀의 상대 크기를 계산합니다.
/// </summary>
private Vector2 ComputeHoleSizeRect(RectTransform targetRect, RectTransform coverRect)
{
    Vector3[] worldCorners = new Vector3[4];
    targetRect.GetWorldCorners(worldCorners);
    Vector2 min = new Vector2(float.MaxValue, float.MaxValue);
    Vector2 max = new Vector2(float.MinValue, float.MinValue);

    for (int i = 0; i < 4; i++)
    {
        Vector2 screenCorner = RectTransformUtility.WorldToScreenPoint(canvas.worldCamera, worldCorners[i]);
        Vector2 localCorner;
        RectTransformUtility.ScreenPointToLocalPointInRectangle(coverRect, screenCorner,
            canvas.worldCamera, out localCorner);
        min = Vector2.Min(min, localCorner);
        max = Vector2.Max(max, localCorner);
    }

    Vector2 size = max - min;
    Vector2 normalizedSize;
    normalizedSize.x = (size.x / coverRect.rect.width) * 0.5f;
    normalizedSize.y = (size.y / coverRect.rect.height) * 0.5f;
    return normalizedSize;
}
```

기능: 타겟 UI 의 위치와 크기를 UV 좌표로 변환하여 Shader 에 전달

특징: UI 중심 계산 및 정규화된 UV 반영으로 모든 화면 해상도 대응

2. 소환

```
/// <summary>
/// 가챠 결과 아이템들을 하나씩 애니메이션과 함께 표시합니다.
/// 각 셀은 일정한 딜레이 후 생성되며, 연출이 끝난 후 버튼들이 다시 활성화됩니다.
/// </summary>
private async UniTask DisplayItemsWithDelayAsync(List<KeyValuePair<int, int>> sortedItemCounts)
{
    StartCoroutine(CoDelayTopScroll());             // 스크롤 위치 맨 위로 이동
    float delay = 0.05f;                            // 각 셀 출력 간의 딜레이 시간
    SetInteractableBottomBtns(false);               // 수락/가챠 버튼 비활성화

    int siblingIndex = 0;                           // 정렬 순서를 위한 인덱스

    foreach (var kvp in sortedItemCounts)
    {
        int itemIndex = kvp.Key;
        int itemCount = kvp.Value;

        Item item = GameMng.m_Instance.GetItemInfo(itemIndex); // 아이템 정보 조회

        ItemCell cell = GetOrCreateItemCell();              // 셀 생성
        cell.ShowCountOnlyOnEquipItem(item, itemCount);     // 수량 표시
        cell.AnimateCellOnSummonResultPopUp();              // 등장 애니메이션
        cell.PlayRarityAnimation(item.GetRarity());         // 레어리티 연출
        cell.transform.SetSiblingIndex(siblingIndex++);     // 정렬 순서 지정

        itemCellDictionary[itemIndex] = cell;               // 딕셔너리에 저장

        await UniTask.Delay(TimeSpan.FromSeconds(delay),    // 딜레이
            cancellationToken: this.GetCancellationTokenOnDestroy(),
            delayType: DelayType.Realtime);
    }

    await UpdateScrollContentAsync();                       // 콘텐츠 높이 재계산
    SetInteractableBottomBtns(!GaChaManager.Instance.GetContinueGaCha()); // 버튼 재활성화
}
```

기능 요약: 가챠 결과 화면에서 **아이템이 한 개씩 등장하며 희귀도 애니메이션과 함께 출력**되는 연출 처리.

특징: UniTask 비동기 코루틴으로 애니메이션 연출

희귀도 기반 애니메이션, SetSiblingIndex() 등 UI 정렬 고려

버튼 비활성화 처리 포함 (연출 도중 클릭 방지)

```
/// <summary>
/// 지정된 가챠 타입에 따라 아이템 1개를 뽑아 지급하고, 경험치를 추가합니다.
/// </summary>
/// <param name="gachaType">가챠 타입 (1: 무기, 2: 망토/목걸이, 3: 룬)</param>
/// <returns>획득한 아이템 인덱스, 실패 시 -1</returns>
public int PerformGacha(int gachaType)
{
    int index = ConvertGachaTypeToIndex(gachaType);
    if (index < 0 || index >= currentGachaLevels.Count)
        return -1;

    InCreaseGaChaQuestProgress(gachaType, 1);            // 퀘스트 진척도 증가

    int clampedLevel = Mathf.Min(currentGachaLevels[index], maxGachaLevel); // 현재 가챠 레벨

    var levelData = gachaLevelDataList.Find(ld => ld.Level == clampedLevel && ld.GachaType == gachaType);
    if (levelData == null) return -1;

    List<float> gachaRates = GetGachaRatesForLevel(gachaType, clampedLevel); // 확률표 불러오기
    if (gachaRates == null || gachaRates.Count < 7) return -1;

    InventoryManager.RARITY selectedRarity = SelectRarity(gachaRates); // 레어리티 추출
    if (selectedRarity == InventoryManager.RARITY.RARITY_END) return -1;

    var filteredGroupIndices = levelData.GroupIndices.FindAll(gIdx =>
    {
        var group = gachaGroups.Find(g => g.GroupIndex == gIdx);
        return group != null && group.Rarity == selectedRarity;
    });

    if (filteredGroupIndices.Count == 0) return -1;

    int selectedItem = DrawItemFromGroups(filteredGroupIndices); // 그룹에서 아이템 추출
    if (selectedItem == -1) return -1;

    AddExperience(gachaType, 1);                    // 소환레벨 경험치 추가
    GameMng.m_Instance.AddItem(selectedItem, 1);    // 아이템 지급

    return selectedItem;
}
```

기능 요약: 가챠를 실행하고, **희귀도 확률에 따라 그룹을 선택하고 아이템을 추출**하는 전반 로직.

특징: PerformGacha() 함수는 가챠 타입별로 레벨과 그룹 구조를 참조하여 희귀도 기반의 그룹을 필터링한 후, 확률에 따라 아이템을 선택하고 지급하는 핵심 로직입니다.

레벨 기반 성장 구조와 경험치, 보상 구조까지 자연스럽게 통합해 설계했습니다.

```
/// <summary>
/// 해당 가챠 타입과 레벨에 따른 각 레어리티별 확률 정보를 반환합니다.
/// </summary>
/// <param name="gachaType">가챠 타입</param>
/// <param name="level">가챠 레벨</param>
/// <returns>레어리티별 확률 리스트 (C, UC, N, R, U, E, LR)</returns>
public List<float> GetGachaRatesForLevel(int gachaType, int level)
{
    if (level > maxGachaLevel || level <= 0)
        return null;

    var gachaRatesData = GameMng.m_Instance.GetCsvData("gacharate");
    int index = ConvertGachaTypeToIndex(gachaType);
    int dataIndex = (level - 1) + index * maxGachaLevel;

    if (dataIndex < 0 || dataIndex >= gachaRatesData.Count)
        return null;

    return new List<float>
    {
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Crate"),  // 일반
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Ucrate"), // 언커먼
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Nrate"),  // 노멀
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Rrate"),  // 레어
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Urate"),  // 유니크
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Erate"),  // 에픽
        GameMng.m_Instance.GetCsvColumnfloat(gachaRatesData, dataIndex, "Lrrate")  // 레전드
    };
}
```

기능 요약:

지정된 가챠 타입과 레벨에 대해 각 희귀도(C~LR)의 확률을 CSV 에서 로딩하여 반환.

특징:

GetGachaRatesForLevel() 함수는 각 가챠 타입과 레벨에 따라 희귀도별 확률을 읽어오며,
레벨이 오를수록 희귀 아이템 획득 확률이 상승하도록 설계되어 있습니다.
밸런스 조정이 용이한 데이터 기반 구조로 설계하여 운영 유연성도 확보했습니다.

3.팝업

```
public class PopUpManager : MonoBehaviour
{
    // 싱글턴 인스턴스
    private static PopUpManager _instance;

    [Header("팝업 인스턴스 목록")]
    [SerializeField] private List<PopUp> popUps = new List<PopUp>();

    // 등록된 모든 팝업을 저장하는 딕셔너리
    private Dictionary<POPUP_ID, PopUp> allPopups = new Dictionary<POPUP_ID, PopUp>();

    // 팝업 스택 관리
    private Stack<PopUp> popupStack = new Stack<PopUp>();

    public event Action<PopUp> OnPopUpOpened;
    public event Action<PopUp> OnPopUpClosed;
```

```
public class PopUp : MonoBehaviour
{
    [Header("팝업 기본 설정")]
    public POPUP_ID popUpID = POPUP_ID.None;
    public Button coverButton;
    public Button closeButton;
    public Transform popupContent;
    public CanvasGroup canvasGroup;

    [Header("옵션")]
    public bool notEsc = false;      // ESC 키로 닫기 허용 여부
    public bool playSound = true;    // 열기 시 효과음 재생

    private bool isInitialized;
    private bool isOpen;
```

기능요약:

공통 팝업 기반 클래스를 상속받는 구조로 팝업 UI 를 설계하였으며,
팝업 매니저는 Dictionary 와 Stack 을 활용해 중첩된 팝업을 안정적으로 처리할 수 있도록 구성했습니다.

ESC 키 대응, 효과음 설정, 팝업 이벤트 전파 등 유연한 기능 커스터마이징이 가능하도록 설계하여 다양한 상황에서도 일관된 팝업 UX 를 제공할 수 있도록 했습니다.

```
/// <summary>
/// 팝업을 열고 애니메이션을 재생합니다.
/// 팝업 스택에 자신을 추가하며, 사운드도 함께 재생됩니다.
/// </summary>
참조 0개
public virtual void Open()
{
    isOpen = true;
    gameObject.SetActive(true);              // 팝업 활성화
    manager.PushPopup(this);                 // 팝업 스택에 등록
    AnimateOpen();                           // 열기 애니메이션 실행

    if (PlaySound)
        GameMng.m_Instance.PlaySoundOneShot(121); // 사운드 재생 (선택사항)
}

/// <summary>
/// 팝업이 열릴 때 애니메이션을 재생합니다.
/// 캔버스 페이드와 스케일 확장을 통해 부드러운 효과를 만듭니다.
/// </summary>
참조 0개
public virtual void AnimateOpen()
{
    if (canvasGroup != null)
    {
        canvasGroup.interactable = false;
        canvasGroup.DOKill();                        // 기존 애니메이션 제거
        canvasGroup.alpha = 0;
        canvasGroup.DOFade(1, 0.3f).SetUpdate(true); // 투명도 애니메이션
    }

    if (PopUpObj != null)
    {
        PopUpObj.DOKill();
        PopUpObj.localScale = Vector3.zero;
        PopUpObj.DOScale(1, 0.3f)
            .SetEase(Ease.OutBack)
            .SetUpdate(true)
            .OnComplete(() =>
            {
                canvasGroup.interactable = true;
                RefreshPopUp();                      // 팝업 열기 후 데이터 갱신
            });
    }
}
```

특징: 팝업 매니저에 자신을 푸시하고, 애니메이션과 사운드까지 포함해 팝업의 "전체적인 열림 흐름"을 담당합니다. 또한 DOTween 을 활용한 자연스러운 등장 연출을 처리하며, 사용자와의 상호작용은 애니메이션이 끝나고 활성화됩니다.

```
/// <summary>
/// 특정 POPUP_ID에 해당하는 팝업을 찾아 열고 스택에 등록합니다.
/// </summary>
public void OpenPopupByID(POPUP_ID popupID)
{
    if (popupID == POPUP_ID.None)
    {
        Debug.LogWarning($"{popupID}은 열 수 없습니다.");
        return;
    }

    if (allPopups.TryGetValue(popupID, out PopUp popup))
    {
        if (popup.gameObject.activeSelf) // 이미 열려 있으면 무시
        {
            return;
        }
        popup.Open();                    // 팝업 열기
        PushPopup(popup);                // 스택에 등록
    }
    else
    {
        Debug.LogWarning($"POPUP_ID {popupID}에 해당하는 팝업을 찾을 수 없습니다.");
    }
}
```

특징: 게임 내 대부분의 팝업 호출은 이 함수로 통합됩니다. POPUP_ID 로 접근이 가능하다는 점에서 **중앙 팝업 관리**에 매우 중요합니다.

4. 아이템

```
public abstract class Item
{
    참조 2개
    public enum ItemType { Weapon, Cloak, Neck, Badge, FishingRod, Consume, Ma

    protected int index;
    protected long count;
    protected string name;
    protected bool isUnlocked;
    protected ItemType type;
    protected List<Dictionary<string, object>> sheetData;
    protected int sheetIndex;

    참조 0개
    public int Index => index;
    참조 0개
    public long Count => count;
    참조 0개
    public string Name => name;
    참조 0개
    public bool IsUnlocked => isUnlocked;
    참조 0개
    public ItemType Type => type;

    /// <summary>
    /// 개수 설정: 최초 획득 시 잠금 해제
    /// </summary>
    참조 11개
    public virtual void SetCount(long newCount)
    {
        if (!isUnlocked && newCount > 0) isUnlocked = true;
        if (count > 0 && newCount <= 0) isUnlocked = false;
        count = newCount;
    }

    참조 0개
    public abstract int Grade { get; }
    참조 0개
    public abstract int Rarity { get; }
    참조 0개
    public abstract string IconName { get; }
}
```

모든 아이템은 공통 부모 클래스인 Item 을 상속받아 구현되며,

index, count, name, type 등 주요 속성을 기본으로 관리합니다.

각 아이템 타입은 별도 상속 클래스로 분리하여, 장비/소모품/재료 등

세부 기능과 처리를 유연하게 확장할 수 있도록 설계했습니다.

외부 시트 연동을 고려한 sheetData 구조와, UI 및 서버 저장에 필요한 속성만

유지하는 구조로

확장성과 최적화 모두를 고려한 설계를 적용했습니다.

4.1 아이템 기능 일부

```
public override void SetInfo(int idx, long cnt, int t)
{
    index = idx;
    count = cnt;
    type = (ITEMTYPE)t;
    level = 0;

    sheetData = GameMng.m_Instance.GetCsvData(GameMng.m_Instance.GetCsvDataName((int)type));

    for (int i = 0; i < sheetData.Count; i++)
    {
        if ((int)sheetData[i]["Index"] == index)
        {
            sheetIndex = i;
            break;
        }
    }

    name = GameMng.m_Instance.GetCsvColumnstring(sheetData, sheetIndex, "Name" + GameMng.m_Instance.GetCountry());
    GameMng.m_Instance.SaveInventory(this);
}
```

기능: CSV 에서 정보 로드 → 내부 상태값 셋팅

```
public override bool Merge(long count)
{
    long evolValue = GameMng.m_Instance.GetCsvColumnlong(m_SheetData, m_SheetIndex, "EvolItemValue");
    if (evolValue <= 0) return false;

    if (count == 0)
        count = m_Count / evolValue;

    long totalCost = evolValue * count;
    if (totalCost <= 0 || !GameMng.m_Instance.IsEnoughMoney(m_Index, totalCost))
        return false;

    int resultIndex = GameMng.m_Instance.GetCsvColumnint(m_SheetData, m_SheetIndex, "EvolItem");

    GameMng.m_Instance.AddItem(resultIndex, count);
    GameMng.m_Instance.DelItem(m_Index, totalCost);
    GameMng.m_Instance.SaveInventory(this);

    IncreaseMergeQuestProgress(count);
    return true;
}
```

기능: 아이템 수량이 진화 조건을 만족하면 상위 아이템으로 전환

퀘스트 진척도까지 자동 반영

5. 보상 및 재화

```
/// <summary>
/// 지연 후 보상 부여 (팝업 연출용)
/// </summary>
public async UniTaskVoid GrantRewardsWithDelay(
    List<RewardData> rewards,
    float delaySeconds = 1f,
    POPUP_SOUND sound = POPUP_SOUND.NORMAL)
{
    if (rewards == null || rewards.Count == 0) return;

    rewards.RemoveAll(r => r.ItemIndex == 0);
    if (rewards.Count == 0) return;

    AddToInventory(rewards);
    await UniTask.Delay(TimeSpan.FromSeconds(delaySeconds), DelayType.Realtime);
    PlayPopUpSound(sound);
    OnRewardsGranted?.Invoke(rewards);
}
```

RewardManager.GrantRewardsWithDelay()

용도: 비동기 처리를 통해 연출 타이밍을 주고, 보상을 UI 에 자연스럽게 연결

특징: UniTask 활용, null-safety 처리, 이벤트 콜백 구조

```
private List<RewardData> MergeRewards(List<RewardData> list)
{
    var dict = new Dictionary<int, RewardData>();
    foreach (var r in list)
    {
        if (r.ItemIndex == 0) continue;
        if (dict.TryGetValue(r.ItemIndex, out var existing))
            existing.AddCount(r.Count);
        else
            dict[r.ItemIndex] = new RewardData(r.ItemIndex, r.Count);
    }
    return new List<RewardData>(dict.Values);
}
```

RewardPopUp.MergeRewards(List<RewardData> list)

용도: 같은 아이템이 여러 번 보상에 중복될 때 하나로 합쳐서 UI 깔끔하게 구성

특징: Dictionary 활용한 중복 처리의 기본 예제

```
private async UniTask PopulateRewards()
{
    scrollRect.horizontalNormalizedPosition = 0.5f;
    for (int i = 0; i < rewards.Count; i++)
    {
        var r = rewards[i];
        var item = GameMng.m_Instance.GetItemInfo(r.ItemIndex);
        if (item == null) continue;

        var cell = CellFactoryManager.Instance.CreateCell(cellPrefab, content, "RewardPopUp");
        itemCells.Add(cell);
        cell.ShowCountOnlyAndSetDeActiveLevel(item, r.Count);
        cell.AnimateCellOnRewardPopUp();
        cell.transform.SetSiblingIndex(i);

        scrollTween?.Kill();
        scrollTween = DOTween.To(
            () => scrollRect.horizontalNormalizedPosition,
            x => scrollRect.horizontalNormalizedPosition = x,
            1f, 0.3f)
            .SetEase(Ease.Linear)
            .SetUpdate(true);

        await UniTask.Delay(TimeSpan.FromSeconds(0.15f), cancellationToken: cts.Token);
    }
}
```

RewardPopUp.PopulateRewards()

용도: 보상 아이템을 하나씩 순차적으로 UI 에 추가하며 애니메이션 적용

특징: DOTween, UniTask, 셀 재사용, 병합 처리 등 다양한 기술이 응축

```
public class RewardData
{
    참조 14개
    public int ItemIndex { get; private set; }
    참조 10개
    public long Count { get; private set; }

    참조 46개
    public RewardData(int index, long count)
    {
        this.ItemIndex = index;
        this.Count = count;
    }

    참조 1개
    public void AddCount(long count)
    {
        Count += count;
    }
}
```

6.레드닷

```
/// <summary>
/// 특정 키의 reddot 상태를 업데이트합니다.
/// </summary>
public void UpdateReddot(RED_DOT key, bool isActive)
{
    if (reddotReceivers.TryGetValue(key, out var receiver))
    {
        if (receiver != null)
            receiver.ReceiveNoti(isActive);
    }
}
```

용도: 특정 레드닷(RED_DOT) 키에 대해 UI 에 표시할지 여부를 갱신합니다.

핵심 기능:

Dictionary 기반의 레지스터된 대상 조회

null-safe 처리

UI 요소(ReddotReceiver)에 상태 알림 전송

```
public void EvaluateEquipReddot()
{
    bool canEquipBetter = GameMng.m_Instance.CanEquipBetterEquipment();
    UpdateReddot(RED_DOT.EQUIPMENT, canEquipBetter);
    UpdateReddot(RED_DOT.TRAINING, canEquipBetter);
    UpdateReddot(RED_DOT.GROWTH, canEquipBetter);
}
```

용도: 장비 관련 레드닷들을 한 번에 평가하여 UI 상태를 갱신합니다.

핵심 기능: **비즈니스 로직(장비 강화 가능 여부)** 판단

관련된 여러 레드닷에 동시에 상태 적용

```
public void EvaluateAllPasses()
{
    bool anyAttendance = PassManager.Instance.HasGrantableRewardsForPassType1((int)PassManager.PASS_TYPE.PASSTYPE_ATTENDANCE);
    bool anyMonster = PassManager.Instance.HasGrantableRewardsForPassType1((int)PassManager.PASS_TYPE.PASSTYPE_MONSTER);
    bool anyStage = PassManager.Instance.HasGrantableRewardsForPassType1((int)PassManager.PASS_TYPE.PASSTYPE_STAGE);
    bool anyMainLevel = PassManager.Instance.HasGrantableRewardsForPassType1((int)PassManager.PASS_TYPE.PASSTYPE_MAINLEVEL);

    bool anyPass = anyAttendance || anyMonster || anyStage || anyMainLevel;
    UpdateReddot(RED_DOT.PASS, anyPass);
    UpdateReddot(RED_DOT.PASS_ATTENDANCE, anyAttendance);
    UpdateReddot(RED_DOT.PASS_KILL_MONSTER, anyMonster);
    UpdateReddot(RED_DOT.PASS_STAGE, anyStage);
    UpdateReddot(RED_DOT.PASS_MAINPLAYER_LEVEL, anyMainLevel);
}
```

용도:

출석/몬스터/스테이지/레벨 패스 보상 여부를 평가하고, 그에 따른 레드닷 UI 업데이트

핵심 기능:

내부 조건들을 논리적으로 묶고, 전체 평가 결과를 함께 반영

각 세부 타입의 개별 레드닷도 동시에 관리