Team Project

Journey Of Hunter

Tel : 010-3902-6074

Email : nayeong0121@gmail.com

GitHub : https://github.com/jjontteok/JourneyOfHunter

CONTENTS

01 프로젝트 개요

◆ 장르 : 3D 방치형 RPG

◆ 사용 툴 : Unity, GitHub

◆ 개발 기간 : 2025.05.15 ~ 2025.07.15

◆ 개발 인원 : 프로그래머 (3명)

◆ 레퍼런스 게임 : 소울 스트라이크, 오늘도 우라라

◆ 담당 역할 : 필드 시스템, 성장 시스템, 네임드 몬스터 구현

◆ 제작 동기 : 협업 환경 경험을 위해 팀 프로젝트 수행
3D 환경에서 유니티 엔진을 활용해 새로운 기능을 구현

◆ 사용한 디자인 패턴 : 싱글톤 패턴, 오브젝트 풀 패턴, 이벤트 버스 패턴

02 게임 화면

메인 화면

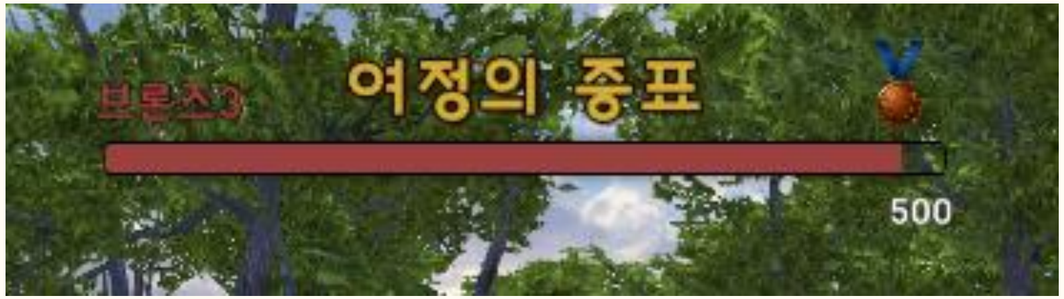
필드 화면

던전 화면(일반 몬스터)

던전 화면(네임드 몬스터)

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 구현



- 플레이어는 자동 및 수동 조작을 통해 앞으로 이동하며 일정 지점에 도달하면 스테이지가 변경되며 여정의 증표(경험치)가 증가합니다.
  스테이지가 변경될 때 일반 오브젝트, 보물상자, 떠돌이 상인, 던전 입장 포탈이 생성됩니다.

- 랭크별 경험치를 모으게 되면 다음 랭크로 올라가며, 랭크는 브론즈, 실버, 골드, 마스터까지 있습니다.

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 – 플레이어가 일정 지점에 도달했을 때 실행되는 함수

```
void OnPlayerCross()
{
    int rnd = UnityEngine.Random.Range(0, 100);
    if (_failedCount > 0)
        rnd = 50;

    _stageCount = ++StageCount;
    OnStageChanged?.Invoke(_stageCount, (rnd < 70));
    IsClear = false;

    if (_stageCount % 10 == 0)
        OnUpgradeMonsterStatus?.Invoke(2);

    Define.ItemValue rank = SetRank();
    _rewardSystem.ClearRewardList();

    if (rnd < 70)
    {
        _rewardSystem.SetReward(Define.RewardType.JourneyExp, 10 * _stageCount);
        _currentType = Define.JourneyType.Dungeon;
        rank = Define.ItemValue.Common;
    }
    else if (rnd < 90)
    {
        if (rnd < 80)
        {
            _currentType = Define.JourneyType.OtherObject;
            _rewardSystem.SetReward(Define.RewardType.JourneyExp, 25 * _stageCount * (int)(rank + 1));
        }
        else
        {
            _currentType = Define.JourneyType.TreasureBox;
            _rewardSystem.SetReward(Define.RewardType.JourneyExp, 50 * _stageCount * (int)(rank + 1));
            _rewardSystem.SetReward(Define.RewardType.Gem, 10 * _stageCount * (int)(rank + 1));
            _rewardSystem.SetReward(Define.RewardType.SilverCoin, 100 * _stageCount * (int)(rank + 1));
        }
    }
    else
    {
        _currentType = Define.JourneyType.Merchant;
        rank = Define.ItemValue.Common;
        OnMerchantAppeared?.Invoke();
    }
    OnJourneyEvent?.Invoke(_currentType, rank);
    PlayerManager.Instance.IsAutoMoving = false;
    PlayerManager.Instance.Player.ReleaseTarget();
}
```

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 – 여정의 증표를 업데이트 하는 함수들

```
// * 여정의 증표를 업데이트 하는 함수
참조 2개
void UpdateJourneyExp(float journeyExp)
{
    _currentJourneyExp += journeyExp;
    _journeyExpQueue.Enqueue(_currentJourneyExp);
    _journeyExpText.text = (_currentJourneyExp).ToString();
    _journeyExpText.color = Color.white;

    if(!_isJourneyCoroutineRun) //코루틴이 실행 중이 아니라면
        //EXP가 점진적으로 차오르도록
        StartCoroutine(ProcessJourneyExpQueue());
}

참조 1개
IEnumerator ProcessJourneyExpQueue()
{
    _isJourneyCoroutineRun = true;
    while (_journeyExpQueue.Count > 0)
    {
        float journeyExp = _journeyExpQueue.Dequeue();

        yield return StartCoroutine(StartJourneyExp(journeyExp));
    }
    _isJourneyCoroutineRun = false;
}
```

```
IEnumerator StartJourneyExp(float journeyExp)
{
    float t = 0;
    float end = (journeyExp - _journeyRankData.MinJourneyExp)
        / (_journeyRankData.MaxJourneyExp-_journeyRankData.MinJourneyExp);
    float start = _journeyGaugeBarImage.fillAmount;

    //증가량이 현재 메달의 최대게이지보다 높으면
    while (end >= 1)
    {
        //다 채우고
        while (_journeyGaugeBarImage.fillAmount < 1)
        {
            t += Time.deltaTime * (1 - start) * 50;
            _journeyGaugeBarImage.fillAmount = Mathf.Lerp(start, 1, t);
            yield return null;
        }
        t = 0;
        //다음 메달로 갱신
        UpdateJourneyRank(_currentJourneyRank+1);

        StartCoroutine(_rankText.GetComponent<UIEffectsManager>().PerformEffect(1));
        StartCoroutine(_rankImage.GetComponent<UIEffectsManager>().PerformEffect(1));
        UpdateJourneyUI();

        //현재의 fillAmount를 0으로 하고
        start = 0;
        //새로 갱신된 최대최소게이지 비율로 맞춤
        end = (journeyExp - _journeyRankData.MinJourneyExp)
            / (_journeyRankData.MaxJourneyExp - _journeyRankData.MinJourneyExp);
    }

    while (Mathf.Abs(end- _journeyGaugeBarImage.fillAmount) > 0.001f)
    {
        t += Time.deltaTime * Mathf.Abs(end - start) * 50;
        _journeyGaugeBarImage.fillAmount = Mathf.Lerp(start, end, t);
        yield return null;
    }
    t = 0;
}
```

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 구현 – 일반 오브젝트와 보물상자

- 일반 오브젝트는 등급별로 나눠져 있으며 플레이어가 오브젝트와 접촉하면 등급별 여정의 증표를 획득하게 됩니다.
- 일반 오브젝트의 사운드 크기는 플레이어와의 거리에 반비례합니다.

- 보물상자는 등급별로 나눠져 있으며 플레이어의 스킬을 통해 상자를 열 수 있습니다.
- 등급별 타격 횟수가 다르게 적용되고 타격 누적에 따라 이펙트가 변화합니다.
- 보물 상자가 열리면 여정의 증표, 젬, 은화를 획득할 수 있습니다.

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 – 보물상자를 때릴 때 실행되는 함수들

```
// * 플레이어가 보물상자를 때릴 때 실행되는 함수
참조 1개
void UpHitCount()
{
    if (_isObtained) return;

    _hitCount++;                        //때린 횟수 증가
    if (_hitCount % 3 == 0)             //3번 때릴 때마다 상자 색 변경
    {
        if (_hitCount >= _count)        //때린 횟수가 현재 상자 랭크와 같다면
        {
            OpenTreasureBox();          //보물 상자 오픈
        }
        else
        {
            SetTreasureEffect(++_initEffectColor);
        }
    }
}
```

```
// * 보물상자 이펙트의 색을 설정하는 메서드
참조 2개
void SetTreasureEffect(Define.ItemValue treasureRank)
{
    foreach (var particle in _treasureEffectList)
    {
        ParticleSystem.MainModule main = particle.main;
        main.startColor = Define.EffectColorList[treasureRank];
    }
}
```

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 구현 – 떠돌이 상인

- 떠돌이 상인이 나타나면 떠돌이 상인의 말과 함께 필드 내 랜덤 위치에 떠돌이 상인이 등장합니다.
- 플레이어가 떠돌이 상인과 접촉하게 되면 떠돌이 상인의 아이템 구매 UI창이 활성화 되고, 아이템은 젬으로 구매 가능합니다.
- 구매한 아이템은 인벤토리의 소비 탭에서 확인할 수 있습니다.
- 플레이어가 자동으로 움직이고 있을 때 플레이어는 상인을 지나치게 되며, 화면 왼쪽에 떠돌이 상인 출현 알람 버튼이 뜨게 됩니다.

03 담당 역할 및 코드 – 주요 담당 기능

필드 시스템 – 떠돌이 상인 UI 관련 함수들

```
// 떠돌이 상인이 뜰 때마다 실행되는 보여줄 아이템 리스트
참조 1개
public void SetItemList() {
    _currentitemList.Clear();

    for (int i =0; i<4;)
    {
        ItemData item = _merchantAllItems.GetRandomItem();
        if (!_currentitemList.ContainsKey(item))
        {
            _currentitemList[item] = 1;
             _merchantItemSlots[i].SetMerchantItemSlot(item);
            i++;
        }
    }
    UpdateGemText();
}
```

```
// 구매 버튼을 누르면 실행되는 함수
참조 1개
void PurchaseItem()
{
    PlayerManager.Instance.Player.Inventory.UseGoods(Define.GoodsType.Gem, _item.Cost);
    _noticeText.enabled = true;
    _noticeText.text = "판매 완료";
    _purchaseButton.interactable = false;
    _isPurchased = true;
    OnPurchaseItem?.Invoke();

    PlayerManager.Instance.Player.Inventory.AddItem(_item, 1);
    PlayerManager.Instance.Player.Inventory.ApplyChangesToSO(
        PlayerManager.Instance.Player.PlayerInventoryData);
}
```

03 담당 역할 및 코드 - 주요 담당 기능

필드 시스템 구현 – 던전 포탈 생성

던전 포탈이 뜨게 되면 던전 등장 UI가 나타나며, 앞에 생성된 던전 포탈을 통해 플레이어는 던전에 입장하게 됩니다.

03 담당 역할 및 코드 - 주요 담당 기능

성장 시스템

- 게임 하단의 성장 버튼을 누르면 성장 UI가 활성화됩니다.
- 플레이어의 공격력, 체력, 체력 회복, 방어력을 높일 수 있으며, 은화로 성장이 가능합니다.
- 사용자 UI/UX를 고려하여 버튼을 길게 누르면 빠르게 플레이어를 성장시킬 수 있도록 구현하였습니다.

03 담당 역할 및 코드 – 주요 담당 기능

성장 시스템 – 버튼을 눌렀을 때 실행되는 함수, 스탯을 업그레이드 하는 함수

```
void UpgradeButton()
{
    if (_isButtonDown && !_isSilverCoinLack)
    {
        _time += Time.deltaTime;

        if (_time > _delay)
        {
            UpgradeStatus();
            _time = 0;
            _delay = Mathf.Max(_delay * 0.9f, 0.05f);
        }
    }
    else
        _delay = 0.3f;
}
```

```
//스탯 업그레이드
참조 2개
public void UpgradeStatus()
{
    _inventoryData.UseGoods(Define.GoodsType.SilverCoin, _statusSlotData.upgradeCost);
    _statusSlotData.level++;
    _statusSlotData.upgradeCost += 10 * _statusSlotData.level;

    _statusSlotData.currentStatusCount = (int)ApplyUpgrade(_statusSlotData.statusType, 5);
    _upgradeCostText.text = _statusSlotData.upgradeCost.ToString();
    _currentStatusText.text = _statusSlotData.currentStatusCount.ToString();
    AudioManager.Instance.PlayClickSound();
}
```

03 담당 역할 및 코드 - 주요 담당 기능

환경 시스템

- 실제 하루처럼 아침, 낮, 저녁, 밤을 구현하였습니다.
- 각 시기마다 스카이박스의 Material 색상과 회전을 조정하고, Direction Light의 회전과 세기를 조정해 실제 시간의 흐름처럼 느껴지도록 구현하였습니다.

03 담당 역할 및 코드 – 주요 담당 기능

환경 시스템 – 시간마다 스카이박스의 색을 변경하고, 스카이박스를 교체하는 함수

```
//현재의 스카이박스 색을 변경하는 함수
참조 1개
void LerpSkyBox(Define.TimeOfDayType type)
{
    string current = GetSkyBoxKey(type);
    _time += Time.deltaTime;

    float t = (_time / _duration) * 10 / _toKey;
    Color changeColor = Color.Lerp(Define.ColorList[_currentTime],
        Define.TargetColorList[_currentTime], t);
    _currentSkyBox.SetColor("_Tint", changeColor);

    LerpLight(current, t);
    LerpMountain(current, t);

    //색 변경이 다 끝나면 스카이박스 변경
    if (t >= 1f)
    {
        _time = 0;
        _changeType = Define.TimeOfDayType.None;
        if (current == Define.Noon) _betterColor = changeColor;

        ChangeSkyBox(Extension.GetNextType(type));
    }
}
```

```
//스카이박스를 변경하는 함수
참조 2개
void ChangeSkyBox(Define.TimeOfDayType skyBox)
{
    string skyBoxName = GetSkyBoxKey(skyBox);
    _currentSkyBox = _skyBoxList[skyBoxName];
    CurrentType =  SetCurrentProperty(skyBoxName);
    _currentTime = skyBoxName;

    //바꿀 스카이박스가 저녁이면
    if (skyBoxName == Define.Evening)
    {
        //초기 저녁 색을 낮의 마지막 색으로 변경
        Define.ColorList[Define.Evening] = _betterColor;

        //초기 저녁 회전값을 낮의 마지막 회전값으로 변경
        _currentSkyBox.SetFloat("_Rotation", _skyBoxRotateValue + Time.deltaTime);
        _currentSkyBox.SetColor("_Tint", Define.ColorList[skyBoxName]);
    }
    else if(skyBoxName == Define.Morning)
    {
        _isMorning = true;
        _skyBoxRotateValue = Define.RotateSkyBoxList[skyBoxName];
    }
    else
    {
        //바꿀 스카이박스의 시작 회전값으로 초기화
        _skyBoxRotateValue = Define.RotateSkyBoxList[skyBoxName];
        _currentSkyBox.SetColor("_Tint", Define.ColorList[skyBoxName]);
    }
    _duration = Define.SkyBoxDurationList[skyBoxName];

    //실제로 스카이박스 변경
    ChangeRenderSettings();
}
```

03 담당 역할 및 코드 - 주요 담당 기능

네임드 몬스터

- 네임드 몬스터가 등장할 때는 컷신을 만들어 플레이어가 게임에 더 몰입되게 구현하였습니다.
- 네임드 몬스터는 플레이어와의 거리에 따라 근거리 공격을 하거나, 플레이어를 향해 움직입니다.
- 또한 플레이어와 거리가 더 멀어지면, 원거리 공격을 하고 일정 범위를 벗어나면 원위치로 돌아갑니다.

03 담당 역할 및 코드 – 주요 담당 기능

네임드 몬스터 – 근거리 공격, 이동, 원거리 공격, 원위치로 이동하는 함수들

```
//공격 범위 내에 있으면 계속 호출될 함수
참조 2개
public override void Attack()
{
    if (_isMoveToOrigin)
        StopMove();

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

    //0~3범위 내에서는 이동하지 않고 공격
    if (distance < _closeAttackLimit)
    {
        ActivateCloseAttack();
    }
    //타겟과의 거리가 MoveRange 범위 밖에 있고 AttackRange 안에 있을 때에는 공격
    else if (distance > _runtimeData.MoveRange && distance < _runtimeData.AttackRange)
    {
        _coolTime += Time.deltaTime;
        if (_coolTime > 3f)
        {
            _coolTime = 0;
            ActivateLongAttack(distance);
        }
    }
    _animator.SetBool(Define.IsAttacking, false);
}
```

```
//원래 자리로 갈지 정하는 함수
참조 1개
void CheckMoveToOrigin()
{
    //원래 자리로 가야 하면
    if (_isMoveToOrigin)
    {
        //원래 자리로 움직인다.
        MoveToTarget(_originPos);
        if ((transform.position - _originPos).sqrMagnitude < 0.1f ||
            Vector3.Distance(transform.position, _target.transform.position) < _runtimeData.MoveRange)
        {
            StopMove();
        }
    }

}

//moveRange 범위 안에 있을 때 실행되는 함수
//5~8범위 내에서는 이동
참조 1개
void OnMoveToTarget()
{
    if (_animator.GetBool(Define.IsAttacking))
        return;
    float distance = Vector3.Distance(transform.position, _target.transform.position);
    if (distance > _closeAttackLimit && distance <= _runtimeData.MoveRange)
        MoveToTarget(_target.transform.position);
    else
        StopMove();
}
```

03 담당 역할 및 코드 - 그 외 담당 기능

카메라 관리

- 메인 화면의 카메라, 플레이어를 따라다니는 카메라, 컷신 카메라에 Cinemachine Camera 컴포넌트를 붙여, 상황에 따라 Main Camera의 Live Camera를 각 카메라로 변경하여 부드러운 카메라 전환을 구현하였습니다.
- 컷신은 타임라인을 만들어 Playable Director 컴포넌트의 Playable에 해당 타임라인을 연결하여 구현하였습니다.

03 담당 역할 및 코드 – 그 외 담당 기능

카메라 관리 – 카메라 교체하는 함수, 컷신 실행하는 함수

```
public void SetStartCam()
{
    _startCam.Priority = highPriority;
    _cutSceneCam.Priority = lowPriority;
    _followCam.Priority = lowPriority;
}

참조 2개
public void SetFollowPlayerCam()
{
    _startCam.Priority = lowPriority;
    _cutSceneCam.Priority = lowPriority;
    _followCam.Priority = highPriority;
}

참조 1개
public void SetCutSceneCam()
{
    Transform monster =
        FindAnyObjectByType<NamedMonsterController>().transform;
    _cutSceneCam.Follow = monster;
    _cutSceneCam.LookAt = monster;
    CinemachineCamera cim =
        _cutSceneCam.GetComponentInChildren<CinemachineCamera>(true);
    cim.Priority = highPriority;
    _followCam.Priority = lowPriority;
}
```

```
void SetBinding()
{
    GameObject namedMonster = FindAnyObjectByType<NamedMonsterController>().gameObject;
    _monsterTransform = namedMonster.transform;

    //타임라인 복제
    var timeline = _playableDirector.playableAsset as TimelineAsset;

    var activateTrack = timeline.GetOutputTrack(0);
    var animationTrack = timeline.GetOutputTracks().OfType<AnimationTrack>().FirstOrDefault();

    _playableDirector.SetGenericBinding(activateTrack, namedMonster);
    _playableDirector.SetGenericBinding(animationTrack, namedMonster.GetComponent<Animator>());
}

참조 1개
void SetAnimStartPos()
{
    GameObject player = PlayerManager.Instance.Player.gameObject;
    transform.position = player.transform.position;
}

// * CutScene 실행 함수
참조 1개
public void PlayCutScene()
{
    //시네머신 카메라들의 우선순위 설정
    CameraManager.Instance.SetCutSceneCam();

    //컷신 카메라의 위치 설정
    SetAnimStartPos();

    //트랙 연결
    SetBinding();

    _monsterAppearEffect.transform.position = _monsterTransform.position - Vector3.up * 2.5f;
    _monsterAppearEffect.SetActive(true);
    _playableDirector.Play();

    UIManager.Instance.DeactivateUIGame();
    PlayerManager.Instance.IsCutSceneOn = true;
}
```

03 담당 역할 및 코드 – 그 외 담당 기능

인벤토리 – 소비 아이템 사용

- 게임 하단의 인벤토리 버튼을 누르고, 소비 탭을 눌러 떠돌이 상인한테서 구매한 아이템들이 뜨게 됩니다.
- 아이템은 회복 아이템, 이동 속도 증가 아이템, 버프 아이템이 있습니다.
- 이동 속도 증가 아이템과 버프 아이템은 일정 시간 동안만 이동 속도가 증가되거나 버프 효과가 부여됩니다.
- 사용할 아이템을 눌러 사용 버튼을 누르게 되면 해당 아이템을 사용할 수 있습니다.

03 담당 역할 및 코드 - 그 외 담당 기능

소비 아이템 사용 시 실행되는 함수

```
public void UseItem(ItemSlot _itemSlot)
{
    PlayerManager.Instance.Player.Inventory.RemoveItem(_itemSlot.ItemData);

    ConsumerItemData data = (ConsumerItemData)_itemSlot.ItemData;

    Define.StatusType statusType;

    if (data.ConsumerType == Define.ConsumeTarget.Atk ||
        data.ConsumerType == Define.ConsumeTarget.HP ||
        data.ConsumerType == Define.ConsumeTarget.Speed)
    {
        float amount = (float)_itemSlot.ItemData.Value * 10;
        if (data.ConsumerType == Define.ConsumeTarget.Atk)
            statusType = Define.StatusType.Atk;
        else if (data.ConsumerType == Define.ConsumeTarget.HP)
            statusType = Define.StatusType.HP;
        else
        {
            statusType = Define.StatusType.Speed;
            amount = 5f;
        }
        PlayerManager.Instance.Player.UseItem(statusType, amount, data.SustainmentTime, data.IconImage);
        UIManager.Instance.UI_Game.StatusEffect.UpdateStatusEffect(data.IconImage, true);
    }
    UI_StatusEffect statusEffect = UIManager.Instance.UI_Game.StatusEffect.GetComponent<UI_StatusEffect>();

    int count = _itemSlot.ItemData.Count; //해당 아이템 카운트 감소

    //해당 아이템 슬롯 삭제 및 Inventory Data에서 해당 아이템 없애기
    if (count <= 0)
    {
        _itemSlots[_itemSlot.ItemData.Type].Remove(_itemSlot);
        _itemSlot.gameObject.SetActive(false);
        return;
    }
    _itemSlot.SetItemCount(count);
}
```

감사합니다.