포트폴리오

김상현

목차

게임 소개

- 프로젝트명 : Project_A
- 장르 : 3D RPG
- 개발엔진 : UNITY
- 사용 유니티 버전 : 2021.3.8f1
- 플랫폼 : 모바일, PC

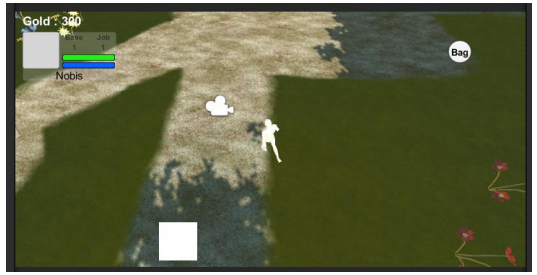
- 빌드파일 링크 : https://drive.google.com/file/d/1Uc6MK_jaj3UIVFw1TUl96vV4Z5SwY2_n/view?usp=share_link
- 깃허브 주소 : https://github.com/Sangchuuu/portfolio.git
- 실행 동영상 주소 : https://youtu.be/sshM2df9TQw

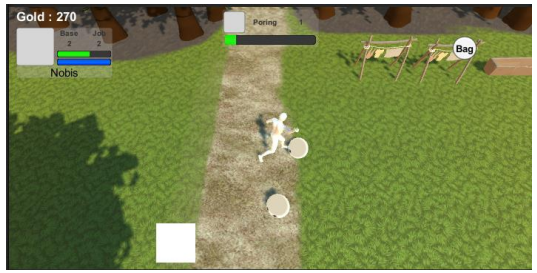
제작목표

- 현재 시장에서 가장 접근성이 좋은 PC, 모바일을 동시에 지원하도록 제작하여,
  플렛폼간의 연동에서 오류나 코드를 경험한다.
- 게임 장르에 맞는 몬스터 AI를 구현하며 로직에서 오는 문제를
  디버깅과 로깅을 통하여 경험하고 해결한다.
- 전투, 상점, 인벤토리, 플레이어 스테이터스 분배 등을 제작하면서
  상호작용에 관한 로직과 그에 따른 오류와 코드를 경험한다.
- RPG의 특성상 데이터를 저장하고 불러오는 부분을 통하여 데이터베이스를 경험한다.
- GitHub를 사용하여 작업한 데이터 백업 및 추후에 하게 될 협업에 관련된 툴을 익힌다.
- 다양한 자료구조와 알고리즘을 경험하고 제작해본다.

시스템(크로스 플렛폼)

Pc 버전





조작
플레이어 움직임 : 레이 캐스트 사용
공통 사용
버튼 클릭 or I 키 = 인벤토리 창 열기

Pc버전

모든 Npc 및 몬스터와의 상호 작용은 마우스로 이루어 진다.

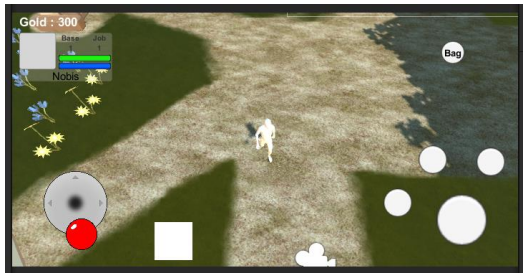
마우스 **좌** 클릭	=	해당 시점 바라보기
마우스 **우** 클릭	=	이동
NPC 클릭 (**좌우 상관 없음**)	=	해당 Ui 호출
몬스터 **좌** 클릭	=	몬스터 정보 확인
몬스터 **우** 클릭	=	몬스터 공격

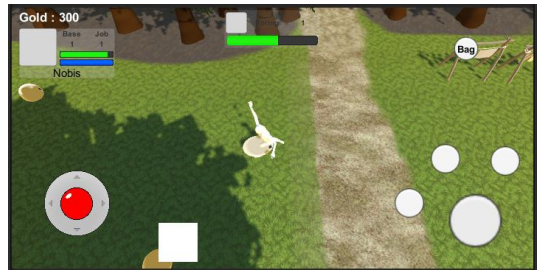
모바일 버전

조이스틱을 이용한 이동, 터치를 이용한 상호작용,
버튼을 이용한 실행

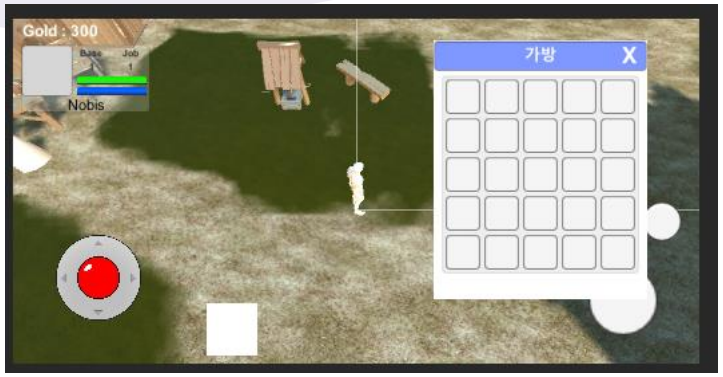
조이스틱	=	이동
터치	=	상호작용
실행	=	Ui버튼 클릭

모바일 버전





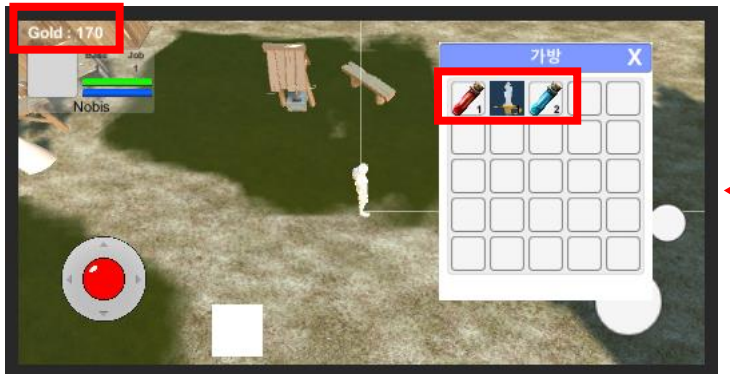
시스템(상점 및 인벤토리)



상점 NPC클릭

상점 Ui 열림

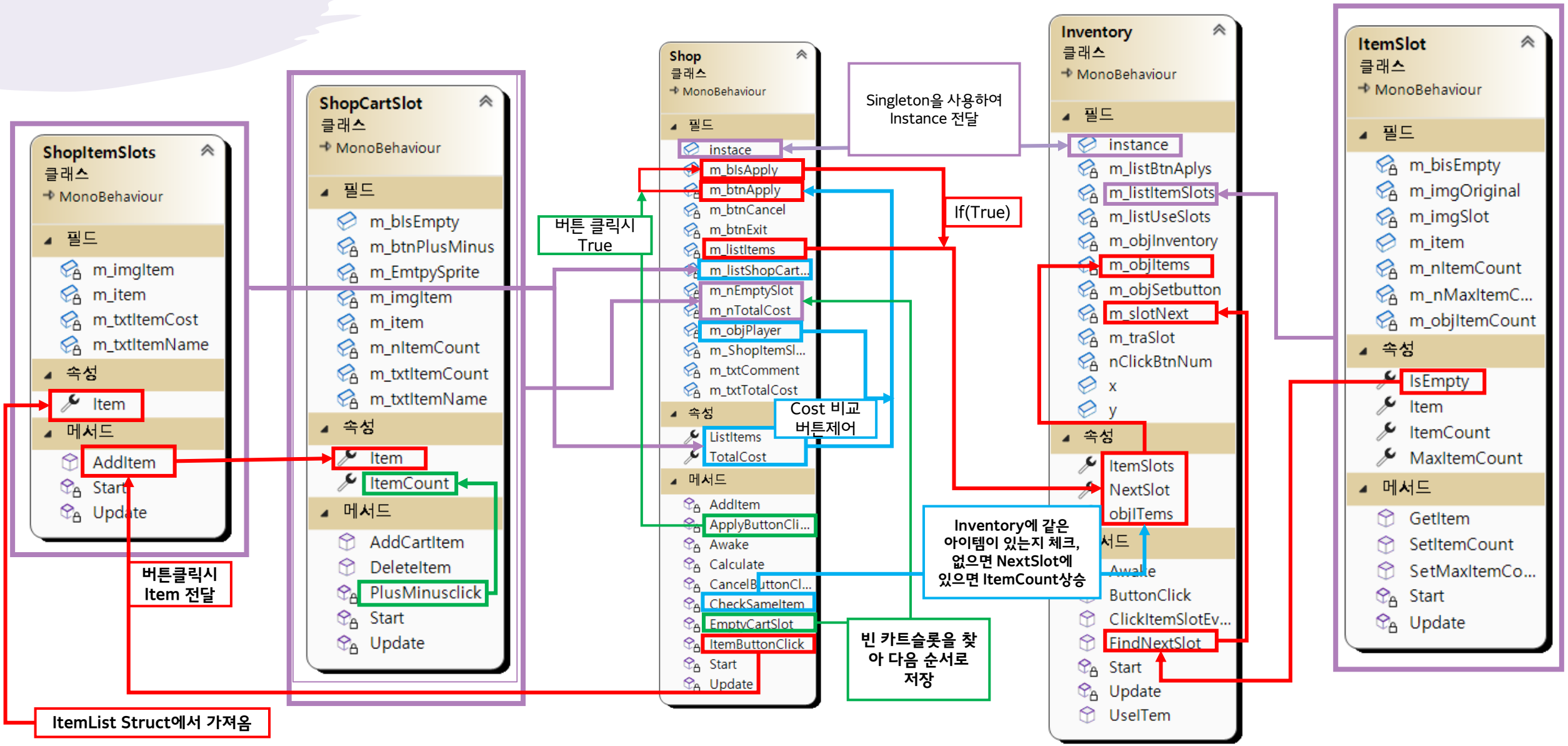
상점 오브젝트 클릭 후 적용버튼 클릭



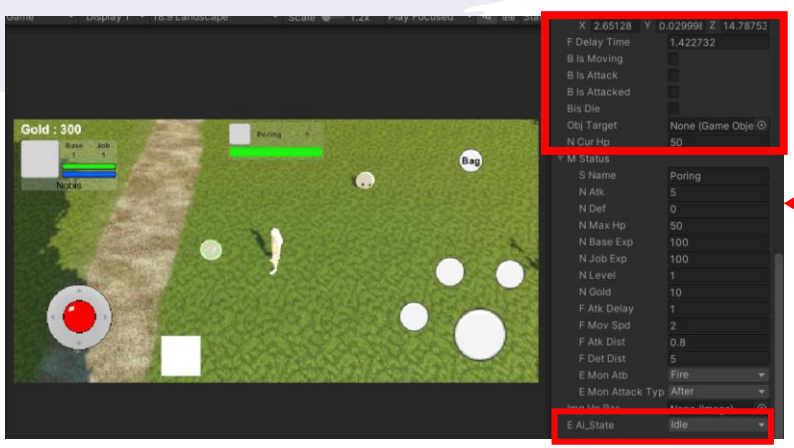
골드 감소 및 아이템 획득

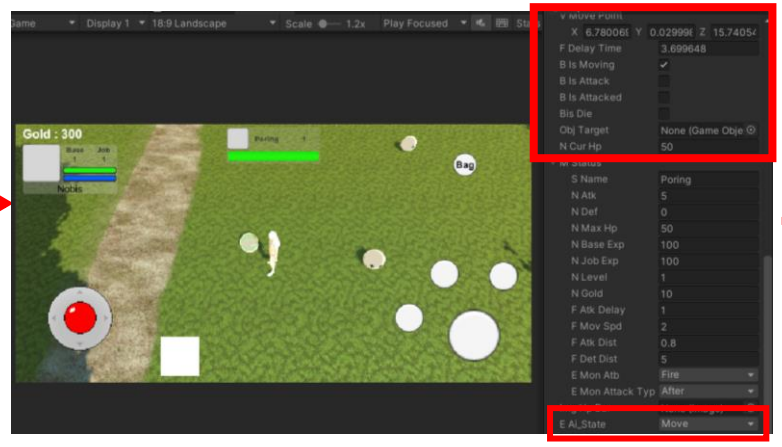
총 금액이 보유재화보다 많으면
적용버튼 비활성 및 재화부족 멘트 적용.

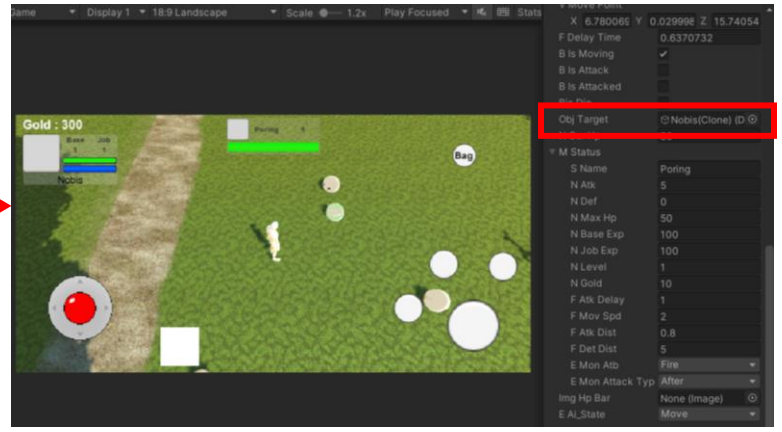
시스템(상점 및 인벤토리)


ShopItemSlots
클래스
MonoBehaviour
필드
m_imgItem
m_item
m_txtItemCost
m_txtItemName
속성
Item
메서드
AddItem
Start
Update
ShopCartSlot
클래스
MonoBehaviour
필드
m_bIsEmpty
m_btnPlusMinus
m_EmtpySprite
m_imgItem
m_item
m_nItemCount
m_txtItemCount
m_txtItemName
속성
Item
ItemCount
메서드
AddCartItem
DeleteItem
PlusMinusclick
Start
Update
Shop
클래스
MonoBehaviour
필드
instace
m_bIsApply
m_btnApply
m_btnCancel
m_btnExit
m_listItems
m_listShopCart...
m_nEmptySlot
m_nTotalCost
m_objPlayer
m_ShopItemSl...
m_txtComment
m_txtTotalCost
속성
ListItems
TotalCost
메서드
AddItem
ApplyButtonCli...
Awake
Calculate
CancelButtonCl...
CheckSameItem
EmptyCartSlot
ItemButtonClick
Start
Update
Inventory
클래스
MonoBehaviour
필드
instance
m_listBtnAplys
m_listItemSlots
m_listUseSlots
m_objInventory
m_objItems
m_objSetbutton
m_slotNext
m_traSlot
nClickBtnNum
x
y
속성
ItemSlots
NextSlot
objITems
서드
Awake
ButtonClick
ClickItemSlotEv...
FindNextSlot
Start
Update
UseITem
ItemSlot
클래스
MonoBehaviour
필드
m_bisEmpty
m_imgOriginal
m_imgSlot
m_item
m_nItemCount
m_nMaxItemC...
m_objItemCount
속성
IsEmpty
Item
ItemCount
MaxItemCount
메서드
GetItem
SetItemCount
SetMaxItemCo...
Start
Update
Singleton을 사용하여 Instance 전달
If(True)
버튼 클릭시 True
Cost 비교 버튼제어
Inventory에 같은 아이템이 있는지 체크, 없으면 NextSlot에 있으면 ItemCount상승
빈 카트슬롯을 찾아 다음 순서로 저장
버튼클릭시 Item 전달
ItemList Struct에서 가져옴

시스템(몬스터 Ai)

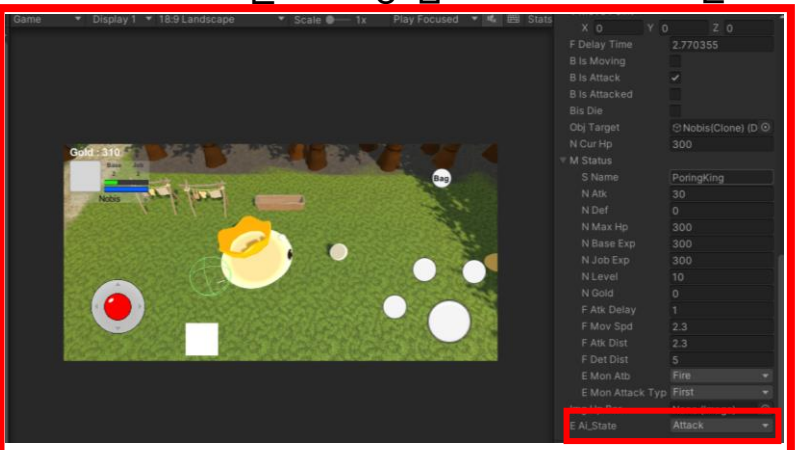






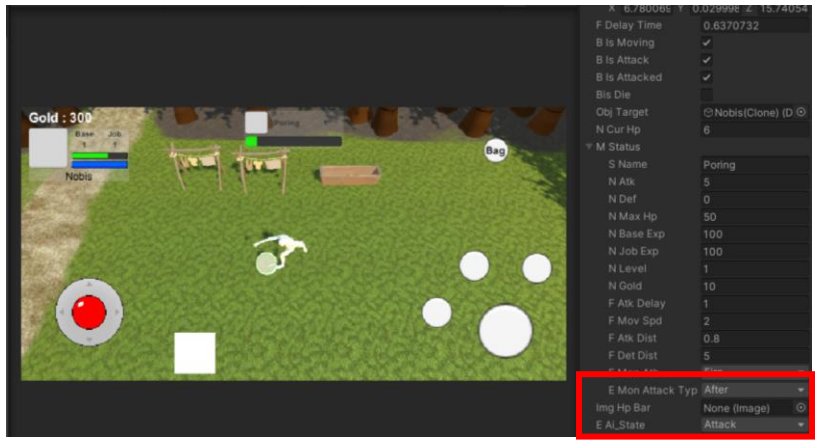
Idle과 Move상태는 DelayTime = 0시에 바뀐다.
DelayTime = 0 이면 DelayTime = 지정 범위 내 Random값으로 지정된다.
Move로 바뀔 때 지정 범위내의 Random한 좌표를 Vector3로 지정하여 다음 Movepoint로 지정된다.

Player가 몬스터를 클릭하면
해당 몬스터에 Player의 정보가 넘어가고,
DelayTime과 몬스터가 멈춘다.



선공 몬스터는 플레이어가 감지거리 내로 들어오면
Chase상태, 사정거리 내로 들어오면 Attack

플레이어가 도망갈 경우 지정거리 내에 있다면,
Chase로 바뀌고 플레이어를 쫓아간다. 플레이어가 지
정거리 밖으로 가면 다시 Idle상태로 전환한다.

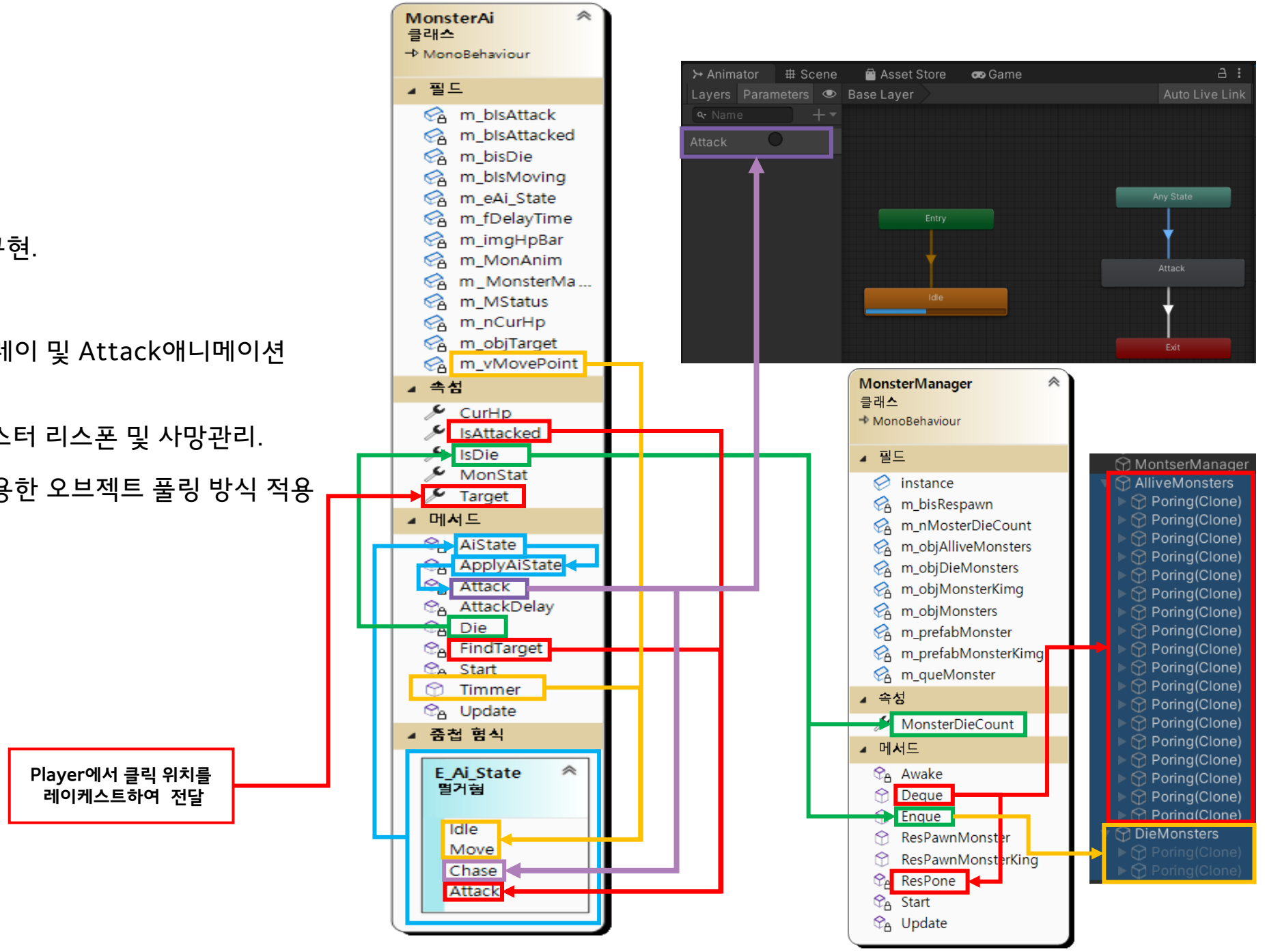


해당 몬스터는 후공 몬스터라 플레이어가 공격을 하면
상태가 Attack으로 플레이어를 공격한다.

시스템(몬스터 Ai)

구현 방식

- 유한상태머신을 사용하여 구현.
- 타겟팅 방식으로 공격 구현.
- 코루틴을 사용하여 공격 딜레이 및 Attack애니메이션 트리거 제어.
- MonsterManager에서 몬스터 리스폰 및 사망관리.
- 부활과 리스폰은 큐잉을 사용한 오브젝트 풀링 방식 적용



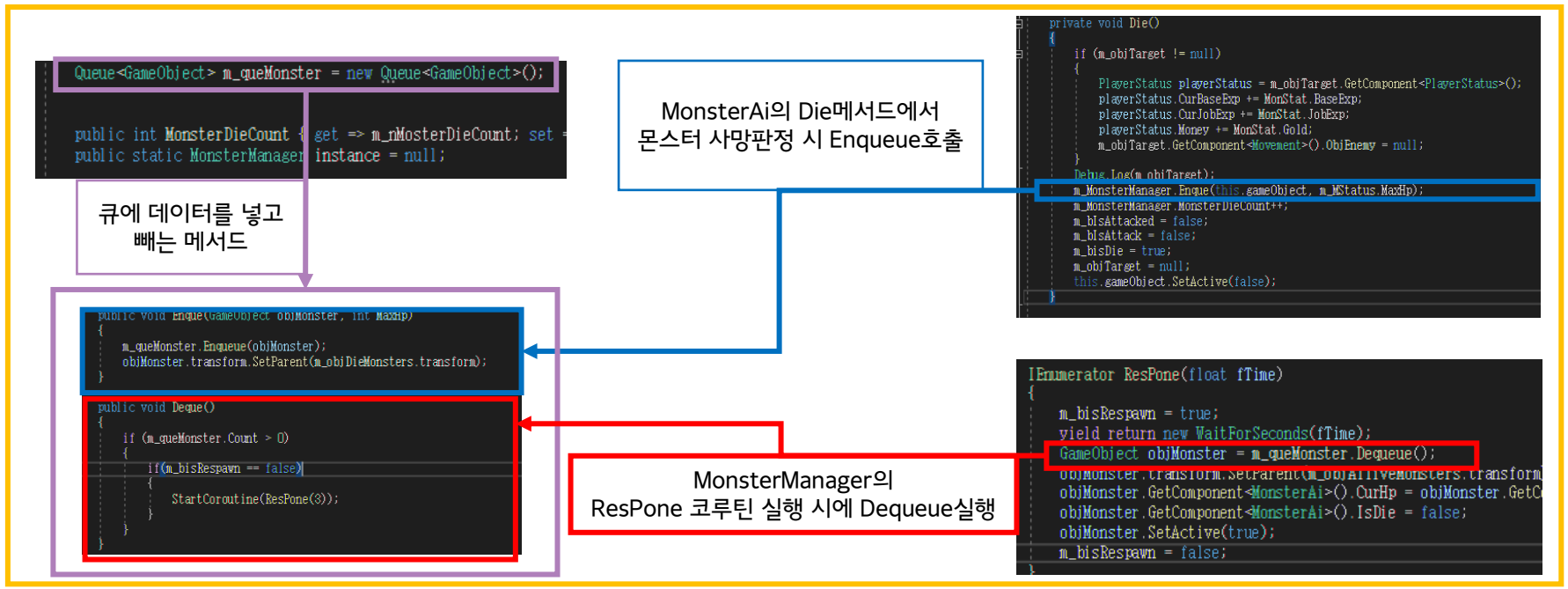
기술정보(오브젝트 풀링)

사용목적

- 몬스터가 사망할 경우 모두Destroy 처리를 하면 몬스터를 생성하고 삭제하는데 쓸데없는 연산이 많이 발생하여 메모리가 낭비가 예상됨.
- 추후에 서버연동과 범위 스킬이 적용되어 한번에 많은 양이 몬스터가 한번에 죽고 리스폰 된다면 가비지컬렉터에서 한번에 처리해야 할 데이터가 많아지고, 프레임 드랍과 성능저하가 예상됨.

기대효과

1. 몬스터를 생성하고 삭제하는 쓸데없는 연산이 줄어 메모리 절약.
2. 성능저하 개선



기술정보(싱글톤)

사용목적

- 게임 내에서 인스턴스가 여러 개 생성될 필요가 없는 스크립트를 싱글톤 패턴을 적용.

기대효과

1. 불필요한 인스턴스 생성을 줄여 메모리 절약
2. 다른 오브젝트와 상호작용시 접근성이 좋다.

사용처

GameManager

```
public ItemList ItemList { get => m_itemList; set => m_itemList = value; }
public GameObject Player { get => m_objPlayer; set => m_objPlayer = value; }
public GameObject PlayerCamera { get => m_objPCamera; set => m_objPCamera = value; }


public static GameManager instance = null;
private void Awake()
{

    if (instance != null)
    {
        Destroy(this.gameObject);
    }
    else
    {
        instance = this;
        DontDestroyOnLoad(this.gameObject);
    }
    m_prePlayer = Resources.Load("Prefabs/Nobis") as GameObject;
    m_preCamera = Resources.Load("Prefabs/PlayerCamera") as GameObject;
}
```

```
private void AddItem()
{
    ItemList itemList = GameManager.instance.GetComponent<ItemList>();
    for (int i =0 ; itemList.items.Count > i; i++)
    {
        m_ShopItemSlots[i].AddItem(itemList.items[i]);
    }
}
```

Inventory

```
public List<GameObject> objITems { get => m_ob
public ItemSlot[] ItemSlots { get => m_listIte
public ItemSlot NextSlot { get => m_slotNext;

public static Inventory instance = null;
private void Awake()
{
    if (instance != null)
    {
        Destroy(this);
    }
    else
        instance = this;
}
```

```
public void GetItem(Item item)
{
    if (m_nItemCount != 0)
    {
        IsEmpty = false;
        Inventory.instance.objITems.Add(this.gameObject);
    }
    m_imgSlot = this.gameObject.GetComponent<Image>();
    m_imgSlot.sprite = item.m_nItemImg;
    m_item = item;
}
```

SceneChange

```
Scene m_sceneNow;

static public SceneChange instance = null;
private void Awake()
{
    if (instance != null)
    {
        Destroy(this.gameObject);
    }
    else
        instance = this;
    DontDestroyOnLoad(this.gameObject);
}
```

씬 전환할 때 마다 생성되어 여러 개가 되는 것 방지목적

MonsterManager

```
public static MonsterManager instance = null;

private void Awake()
{
    if(instance != null)
    {
        Destroy(instance);
    }
    else if (instance == null)
    {
        instance = this;
    }
}
```

```
void Start()
{
    m_MonsterManager = MonsterManager.instance;
    m_fDelayTime = Random.Range(1f, 5f);
    m_MonAnim = this.gameObject.GetComponent<Animator>();
    m_bIsAttacked = false;
    m_nCurHp = m_MStatus.MaxHp;
}
```

기술정보(유한상태머신)

사용목적

- 몬스터 Ai를 제어하기 위한 목적으로 사용.

기대효과

1. 직관적으로 현재상태와 예상상태를 파악에 용이
2. 추후 기능추가가 쉬움

적용방식

1. AiState()메서드에서 몬스터의 현재상태에 따른 State 지정.
2. ApplayAiState()메서드에서 적용된 State에 따른 동작 구현

State에 따른 동작

```
private void ApplyAiState(E_Ai_State state)
{
    switch (state)
    {
        case E_Ai_State.Idle:
            break;

        case E_Ai_State.Move:
            float fDist = (this.transform.position - m_vMovePoint).magnitude;
            this.transform.LookAt(m_vMovePoint);
            if (fDist >= 0.3f)
            {
                this.transform.position += transform.forward * m_MStatus.MovSpd * Time.deltaTime;
            }
            break;

        case E_Ai_State.Chase:
            if (m_objTarget != null)
            {
                transform.LookAt(m_objTarget.transform);
                this.transform.position += transform.forward * m_MStatus.MovSpd * Time.deltaTime;
            }
            break;

        case E_Ai_State.Attack:
            Attack();
            break;
```

상태에 따른 State 지정

```
private void AiState()
{
    if (m_objTarget != null)
    {
        float fDist = (m_objTarget.transform.position - this.transform.position).magnitude;
        if (m_MStatus.EMonAttackType == MonsterStatus.E_Mon_AttackType.First)
        {
            if (m_bIsAttacked == true)
            {
                if (fDist > m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Chase;
                else if (fDist <= m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Attack;
                if (fDist >= m_MStatus.DetDist)
                    m_objTarget = null;
            }
            else
            {
                if (fDist > m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Chase;
                else if (fDist <= m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Attack;
                if (fDist >= m_MStatus.DetDist)
                {
                    m_eAi_State = E_Ai_State.Idle;
                    m_objTarget = null;
                }
            }
        }
        if (m_bIsAttacked == true)
        {
            if (m_MStatus.EMonAttackType == MonsterStatus.E_Mon_AttackType.After)
            {
                if (fDist > m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Chase;
                else if (fDist <= m_MStatus.AtkDist)
                    m_eAi_State = E_Ai_State.Attack;
                if (fDist >= m_MStatus.DetDist)
                    m_objTarget = null;
            }
        }
        else
        {
            m_bIsAttacked = false;
            Timmer();
            if (m_bIsMoving == true)
            {
                m_eAi_State = E_Ai_State.Move;
            }
            else
                m_eAi_State = E_Ai_State.Idle;
        }
        ApplyAiState(m_eAi_State);
        if (m_nCurHp <= 0)
            Die();
    }
```

기술정보(데이터베이스)

사용목적

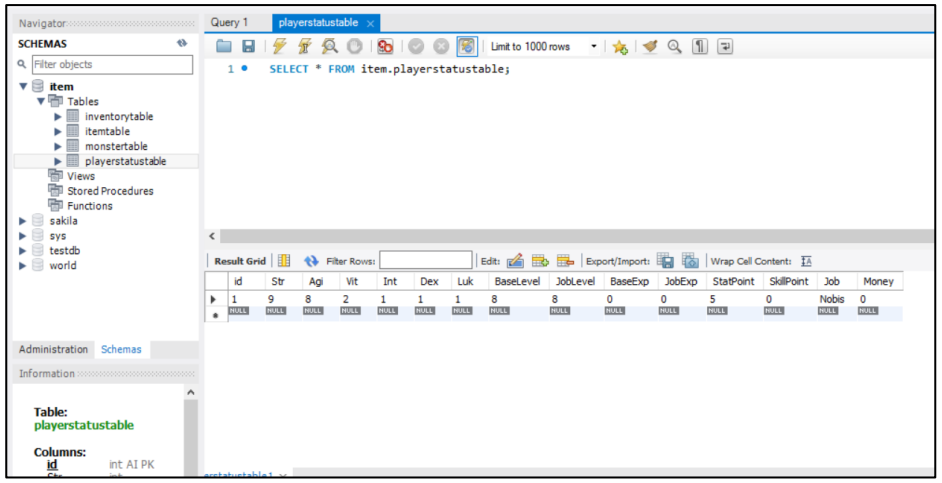
- 게임에서 플레이어의 정보저장을 저장을 위하여 사용

기대효과

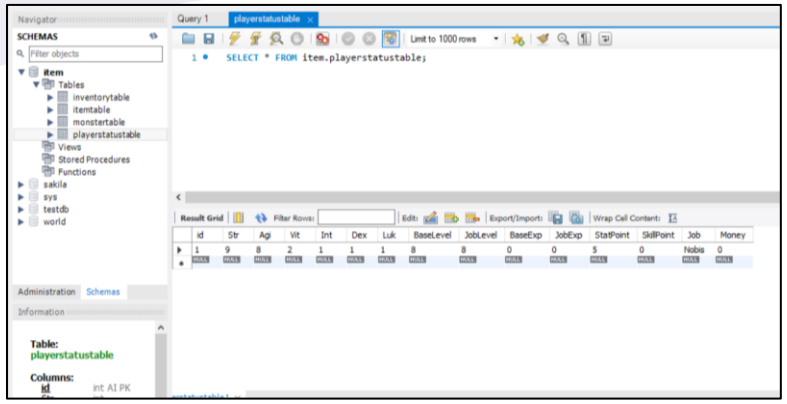
1. 게임을 새로 시작할 때 플레이어의 정보가 남아있어 게임의 기획에 좀 더 잘 맞다.

적용 방식

- 스크립트에서 쿼리문을 String으로 저장하여 링커에 전달하는 방식으로 구현.
- MySQL사용하여 플레이어 레벨, 스테이터스 저장.



기술정보(데이터베이스)



게임시작시 Start버튼 클릭 시 링커를 통하여
데이터베이스 접근 하여
데이터베이스에서 레벨과 status를 Select로
해당 값을 가져와 Playerstatus에 초기화.

게임 시작

```
if (PlayerUi.instance.m_bIsStart == true)
{
    mysqlLinker.Connect(m_strServerIP, m_strPort, m_strDB, m_strUser, m_strPSWD);
    m_strTable = "`playerstatustable`";
    PlayerStatus playerstatus = GameManager.instance.Player.GetComponent<PlayerStatus>();
    string strQuery = MakeQuery.Select("Str", m_strTable);
    playerstatus.Str = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("Agi", m_strTable);
    playerstatus.Agi = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("Vit", m_strTable);
    playerstatus.Vit = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("Dex", m_strTable);
    playerstatus.Dex = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("Luk", m_strTable);
    playerstatus.Luk = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("StatPoint", m_strTable);
    playerstatus.StatPoint = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("BaseLevel", m_strTable);
    playerstatus.CurBaseLevel = mysqlLinker.SelectToInt(strQuery);
    strQuery = MakeQuery.Select("JobLevel", m_strTable);
    playerstatus.CurJobLevel = mysqlLinker.SelectToInt(strQuery);
    PlayerUi.instance.m_bIsStart = false;
}
```

플레이어가 레벨업을 하면
각 레벨의 칼럼에 Udate를 적용

레벨 업

```
if (playerstatus.IsBaseLevelUp == true)
{
    m_strTable = "`playerstatustable`";
    string strQuery = MakeQuery.Update(m_strTable,
        "`BaseLevel`= " + " ' " + playerstatus.CurBaseLevel.ToString() + " ' "
        , "`id` <= " + " ' " + playerstatus.Agi.ToString() + " ' ");
        int nResult = mysqlLinker.CheckMode(strQuery);
    Debug.Log("Update[" + nResult + "]:" + strQuery);
    playerstatus.IsBaseLevelUp = false;
}
if (playerstatus.IsJobLevelUp == true)
{
    m_strTable = "`playerstatustable`";
    string strQuery = MakeQuery.Update(m_strTable,
        "`JobLevel`= " + " ' " + playerstatus.CurJobLevel.ToString() + " ' "
        , "`id` <= " + " ' " + playerstatus.Agi.ToString() + " ' ");
        int nResult = mysqlLinker.CheckMode(strQuery);
    Debug.Log("Update[" + nResult + "]:" + strQuery);
    playerstatus.IsBaseLevelUp = false;
}
```

플레이어가 스텟 적용 후 적용버튼을 클릭하면
플레이어의 스텟을 각 칼럼에 Update

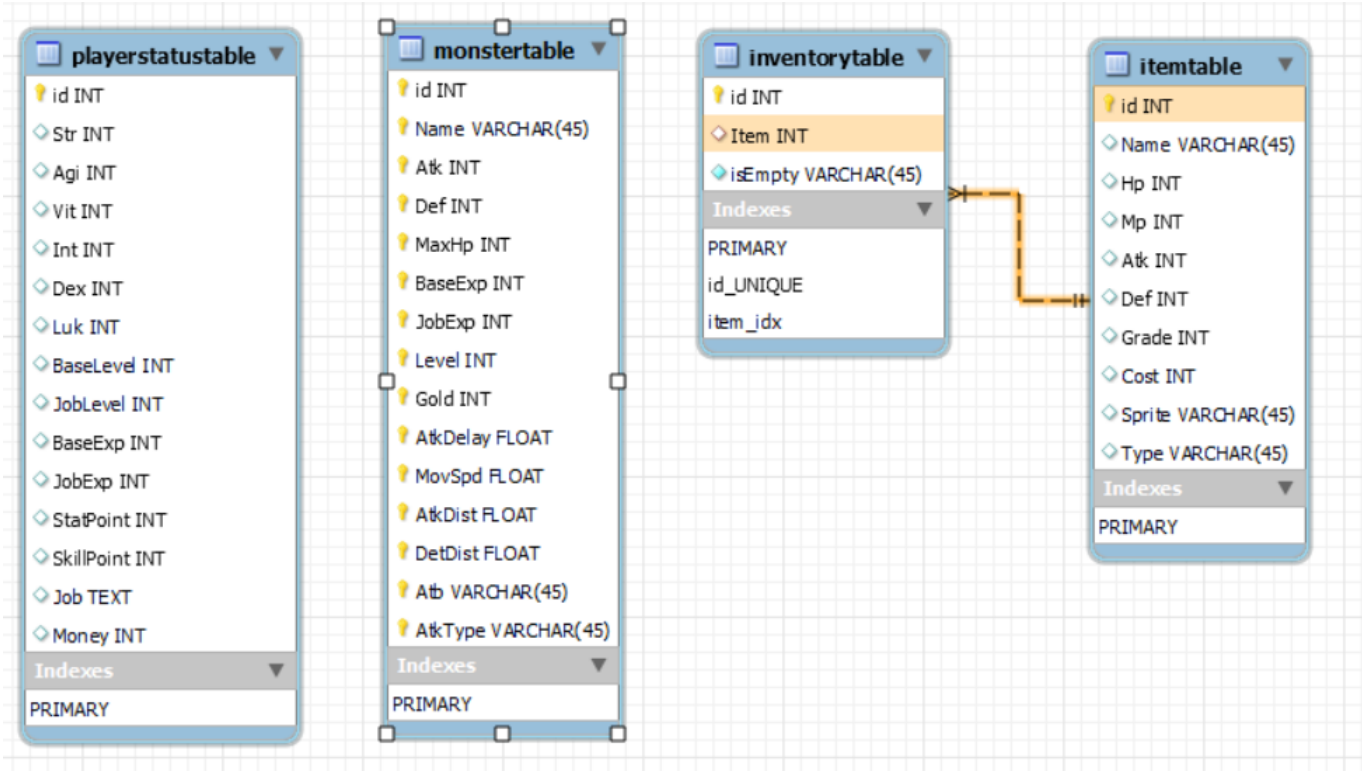
스텟 적용

```
if (PlayerUi.instance.m_bIsSetButtonClick == true)
{
    m_strTable = "`playerstatustable`";
    string strQuery = MakeQuery.Update(m_strTable,
    "`Str`= " + " ' " + playerstatus.Str.ToString() + " ' " + "," +
    "`Agi`= " + " ' " + playerstatus.Agi.ToString() + " ' " + "," +
    "`Vit`= " + " ' " + playerstatus.Vit.ToString() + " ' " + "," +
    "`Dex`= " + " ' " + playerstatus.Dex.ToString() + " ' " + "," +
    "`Luk`= " + " ' " + playerstatus.Luk.ToString() + " ' " + "," +
    "`StatPoint`= " + " ' " + playerstatus.StatPoint.ToString() + " ' " + "," +
    "`BaseLevel`= " + " ' " + playerstatus.CurBaseLevel.ToString()
    "`JobLevel`= " + " ' " + playerstatus.CurJobLevel.ToString() +
    , "`id` <= " +  " ' " + playerstatus.Agi.ToString() + " ' ");
    int nResult = mysqlLinker.CheckMode(strQuery);
    Debug.Log("Update[" + nResult + "]:" + strQuery);
    PlayerUi.instance.m_bIsSetButtonClick = false;
}
```

기술정보(데이터베이스)

설계

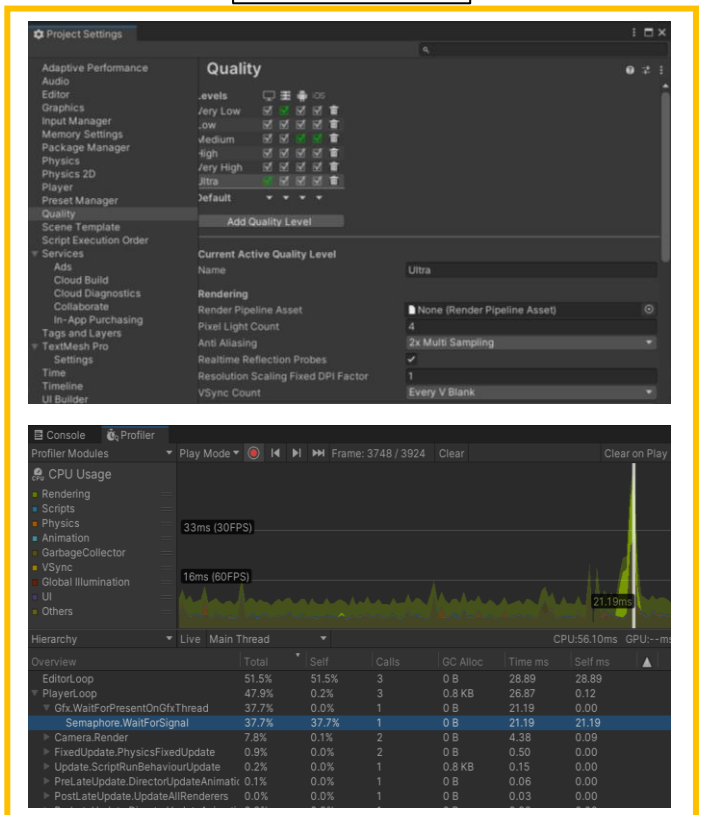
- PlayerStatusTable, MonsterStatusTable, InventoryTable, Itemtable존재.
- 상점에서 아이템을 구매하여 아이템을 인벤토리에 가져올 때 외래키를 설정하여 해당하는 아이템을 참조형식으로 아이템데이터를 불러오게 설계.



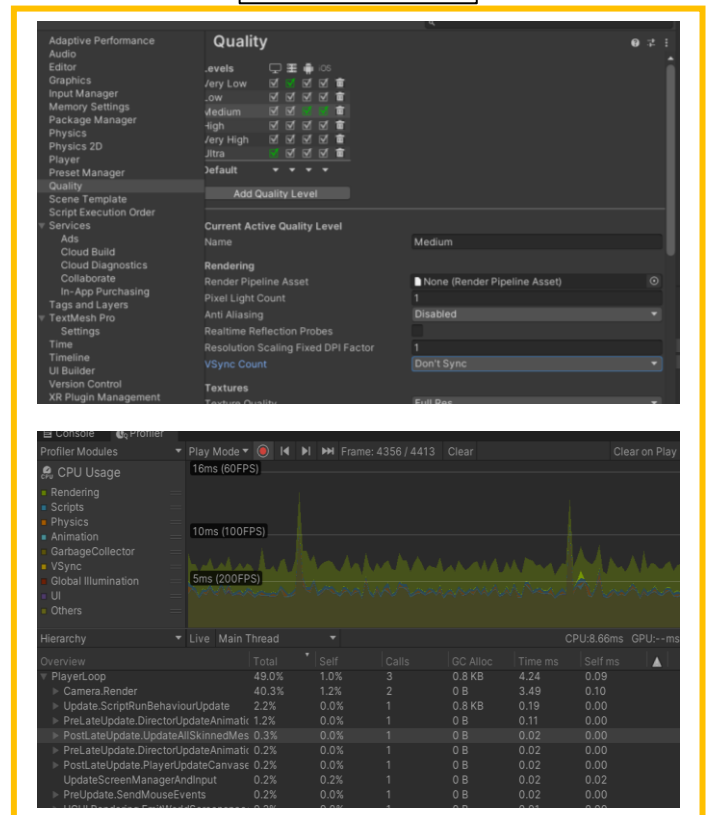
분석(프로파일링)

확인 : 기기에서 플레이를 하던 중 최고사양 모바일 임에도 한번씩 프레임 드랍이 발생하는 것을 발견.
원인 : semaphore.waitforsignal라는 Cpu가 Gpu의 작업이 끝나길 대기하는 상태가 발생하는 것을 확인.
작업 : 해당 내용이 Cpu와 Gpu의 작업을 싱크 시켜주는 Vsync부분 때문에 발생하는 것을 확인하였고, 해제
더불어 모바일 사양에 맞지 않게 지나치게 높은 사양으로 설정이 되어 있는 것을 확인하고, 모바일에 맞게 사양을 재설정.
결과 : 안정적으로 60프레임이 확보 됨.

변경 전



변경 후



제작을 하며(기초공부)

몬스터와 플레이어의 Status를 적용할 때 Struct를 사용 하였습니다.
여기서 Hp같은 자주 변경되는 Status의 값은 Struct는 값이 참조가 아닌 복사형식으로 들고 오기 때문에
Class에 지정 되었어야 하는데, Struct에 지정을 해버려 한번 수정할 때 마다 다시 덮어 씌워야 했습니다.
그에 따른 쓸데없는 연산과 메모리가 사용되는 것을 발견 하였습니다.
이 부분을 수정하기 위해 Hp를 참조하는 모든 스크립트를 수정해야 했고, 시간이 많이 소요 되었습니다.

여기서 무언가를 사용할 때 명확하게 알고 사용하는 것과 어설프게 알고 사용하는 것의 차이를
이 프로젝트를 진행하면서 다방면에서 느끼게 되었고, 다시한번 '기초적인 공부를 더 열심히 해야 겠다' 생각을 했습니다.
취직을 하고 업무에 어느정도 적응을 하게 되면 현재 제에게 모자란 전공공부를 편입을 하여 공부해서 3년내 에 학사학위를 취득할 것입니다.

제작을 하며(일정관리)

이번 프로젝트는 데드라인이 없고 혼자 하는 프로젝트라서 시간이 많다고 생각하였습니다.
저런 생각 때문에 프로젝트 초반에 속도가 잘 나지 않았습니다.
현업에서 일하는 친구가 회사에서의 업무가 어떻게 이루어 지는 지와 일정관리를 왜, 어떻게 하는지에 대해 이야기 해 주었고,
일정관리의 중요성을 느껴 시도를 해 보았습니다.

처음에는 일정을 관리하고 작업에 걸릴 시간을 예측하여 데드라인을 지정하고 작업을 하는게 어려웠지만,
관리를 하니 작업한 내역과 걸린 시간이 명확하게 보여 스스로의 수준을 파악하고, 작업에 속도를 내는데 도움이 되었습니다.
또, 데드라인을 지정할 때 제작을 해야 하는 것에 대해서 어떻게 제작을 해야 할 지에 대해 미리 한번 생각하게 되어서,
추후에 작업을 하는데 속도가 좀더 붙었던 것 같습니다.

프로그래밍 뿐만 아니라 앞으로의 일에도 일정관리를 하는 습관을 들이도록 노력하는 중 입니다.

감사합니다.