양창근

팀 프로젝트

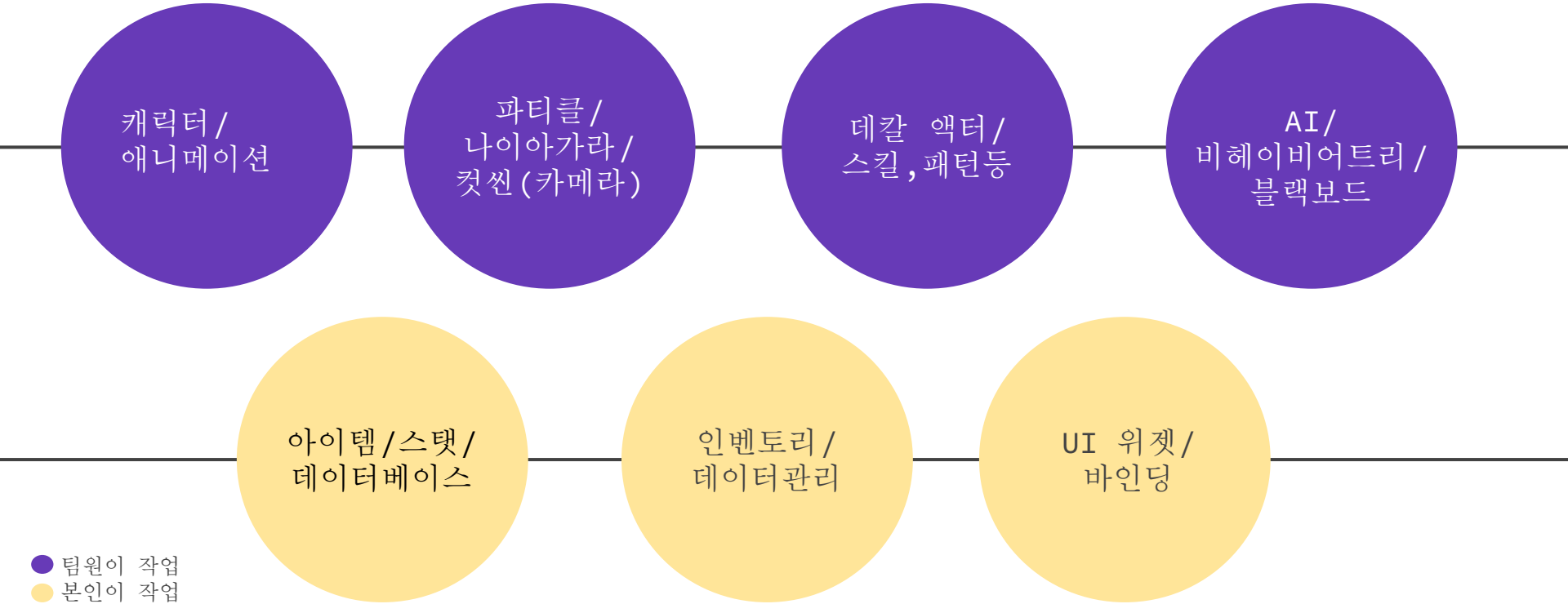
프로젝트 소개 : Tema TFT (The Forge Team)

-개발 인원 : 4명
-프로젝트 개발기간 : 2024-08-12 ~ 2024-09-02 (약 3주)

-프로젝트 목표 : C++ 로 각자 구현했엇던 TextRPG를 언리얼엔진으로 구현하며 팀 프로젝트로써 팀원들과 역할 분담등을 나누어 팀원끼리 역할을 분담하여 하나의 게임을 만들며 미리 다인 프로젝트 경험 생성
- 담당 업무 :

각종 UI(ex. HpBar, 인벤토리,캐릭터선택창등),Item(데이터테이블화) , InvenTory(내부함수), 게임모드베이스/인스턴스등

전체적인 구조

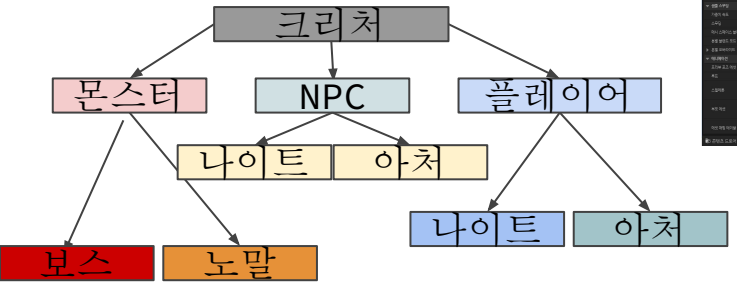


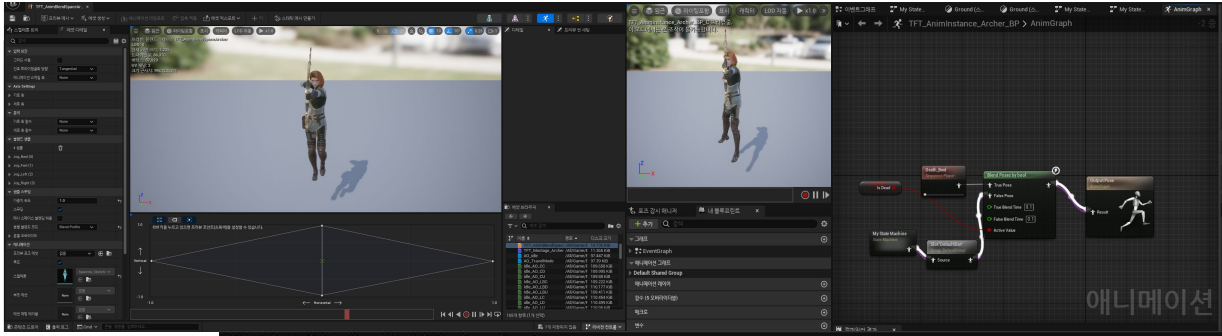
● 팀원이 작업
● 본인이 작업

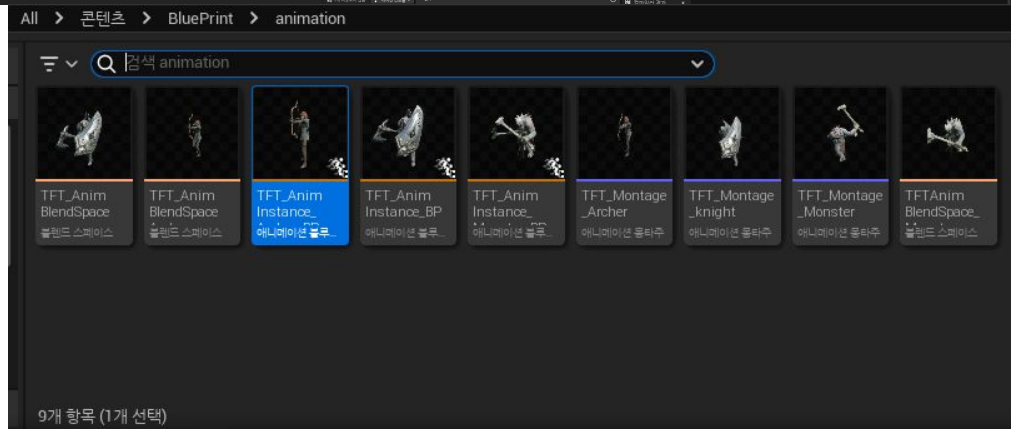
- 작업한 내용을 기준으로 소개

캐릭터 / 애니메이션 - 공용

기본적인 캐릭터의 전체구조







기본적으로 모든 캐릭터는 크리처를 상속하는 구조로 생성 하였고 모든 캐릭터가 가지는 공용함수등은 크리처에서 작성하고 이를 상속하고 각 개성에 맞는 함수를 추가해 구현.

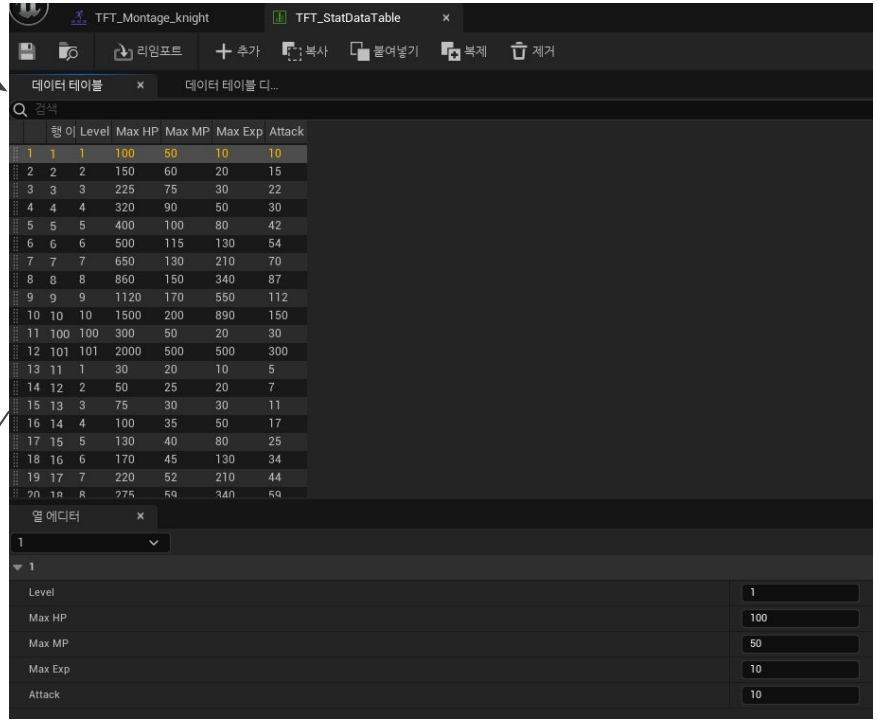
기본적으로 이동은 블랜드스페이스로 적용후 애님인스턴스로 각각 관리

데이터 베이스 - stat

```
TFT_Project_A                                                    (전역 범위)
10
11      DECLARE_MULTICAST_DELEGATE(DeathDelegate)
12      DECLARE_MULTICAST_DELEGATE_OneParam(HpChanged, float)
13      DECLARE_MULTICAST_DELEGATE_OneParam(ExpChanged, float)
14      DECLARE_MULTICAST_DELEGATE(LevelUp)
15
16      USTRUCT()
17  struct FTFT_StatData : public FTableRowBase
18      {
19          GENERATED_BODY()
20
21          UPROPERTY(EditAnywhere, BlueprintReadWrite)
22          int32 level = 0;
23          UPROPERTY(EditAnywhere, BlueprintReadWrite)
24          int32 maxHP = 0;
25          UPROPERTY(EditAnywhere, BlueprintReadWrite)
26          int32 maxMP = 0;
27          UPROPERTY(EditAnywhere, BlueprintReadWrite)
28          int32 maxExp = 0;
29          UPROPERTY(EditAnywhere, BlueprintReadWrite)
30          int32 attack = 0;
31
32      };
33
```

```
void UTFT_StatComponent::SetLevelAndInit(int32 level)
{
    auto myGameInstance = Cast<UTFT_GameInstance>(GetWorld()->GetGameInstance());
    if (myGameInstance)
    {
        FTFT_StatData* data = myGameInstance->GetStatDataByLevel(level);
        _curLevel = level;
        _maxHp = data->maxHP;
        _curHp = _maxHp;
        _maxMp = data->maxMP;
        _curMp = _maxMp;
        _maxExp = data->maxExp;
        _attackDamage = data->attack;

        if (AddsItem_Attack != 0) _attackDamage += AddsItem_Attack;

        UE_LOG(LogTemp, Warning, TEXT("%s..., Level : %d, hp : %d, attackDamage : %d"), *GetOwner()->GetName(), _curLevel, _maxHp, _attackDamage);
    }
}
```

cpp 에서 데이터테이블을 생성하여
이를 상속받는 데이터테이블 파일을
에디터에서
관리하여 행이름으로 그값을 불러와서
적용 가능한 구조로 제작

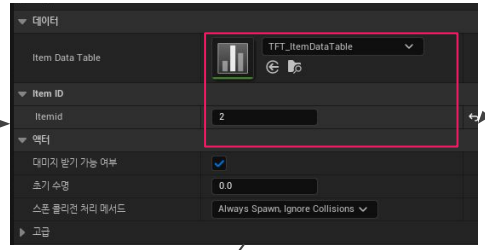


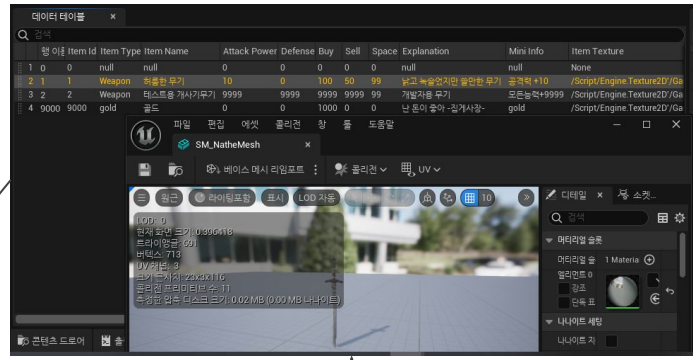
아이템 - 데이터테이블

```
USTRUCT(BlueprintType)
struct FTFT_ItemData : public FTableRowBase
{
    GENERATED_BODY()

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemId;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemType;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString ItemName;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 AttackPower;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Defense;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Buy;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Sell;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    int32 Space;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString Explanation;
    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    FString MiniInfo;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    UTexture2D* ItemTexture;

    UPROPERTY(EditAnywhere, BlueprintReadWrite)
    UStaticMesh* ItemMesh;
};
```

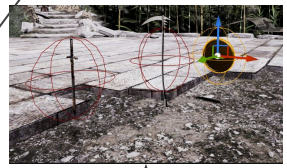
```
ATFT_Item::ATFT_Item()
{
    PrimaryActorTick.bCanEverTick = false;

    _meshComponent = CreateDefaultSubobject<UStaticMeshComponent>
    _trigger = CreateDefaultSubobject<USphereComponent>(TEXT("Tri

    _meshComponent->SetupAttachment(RootComponent);
    _trigger->SetupAttachment(_meshComponent);

    _meshComponent->SetCollisionProfileName(TEXT("TFT_Item"));
    _trigger->SetCollisionProfileName(TEXT("TFT_Item"));
    _trigger->SetSphereRadius(60.0f);

    ConstructorHelpers::FObjectFinder<UDataTable> DataTableObj(TE

    if (DataTableObj.Succeeded())
    {
        ItemDataTable = DataTableObj.Object;
    }
}
```

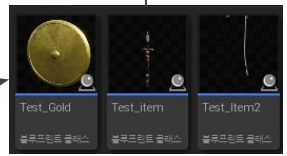
아이템id 변경 시 정보변경

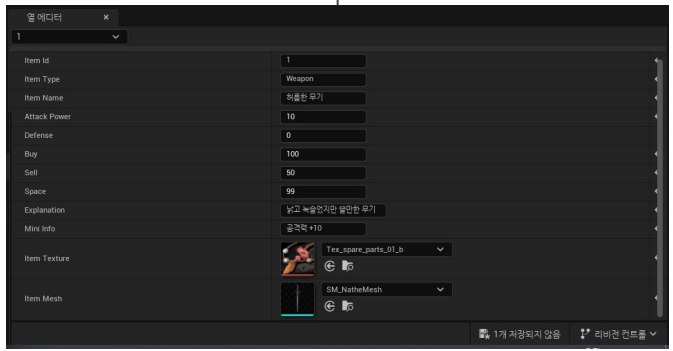





```
void ATFT_Item::PostEditChangeProperty(FPropertyChangedEvent& PropertyChangedEvent)
{
    Super::PostEditChangeProperty(PropertyChangedEvent);

    FName PropertyName = (PropertyChangedEvent.Property != nullptr) ? PropertyChangedEvent.Property->GetFName() : NAME_None;

    if (PropertyName == GET_MEMBER_NAME_CHECKED(ATFT_Item, _Itemid))
    {
        LoadItemData();
        UE_LOG(LogTemp, Warning, TEXT("PostEditChangeProperty called. ID changed to: %d"), _Itemid);
    }
}
```





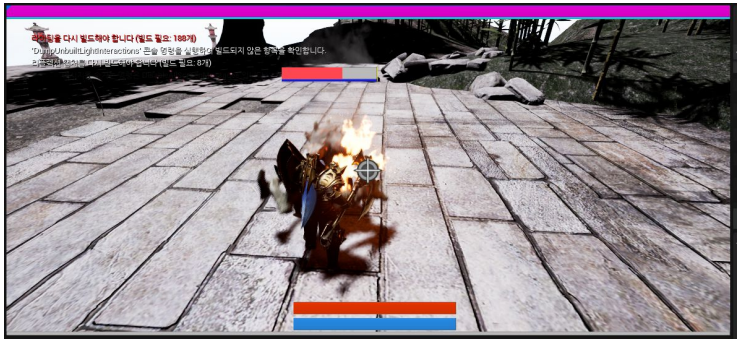



```
void ATFT_Item::LoadItemData()
{
    if (ItemDataTable)
    {
        static const FString ContextString(TEXT("ItemData"));
        FTFT_ItemData* ItemData = ItemDataTable->FindRow<FTFT_ItemData>(FName(*FString::FromInt(_Itemid)), ContextString, true);
        if (ItemData)
        {

            _ItemType = ItemData->ItemType;
            _Name = ItemData->ItemName;
            _AttackPower = ItemData->AttackPower;
            _Defense = ItemData->Defense;
            _Buy = ItemData->Buy;
            _Sell = ItemData->Sell;
            _Space = ItemData->Space;
            _Explanation = ItemData->Explanation;
            _MiniInfo = ItemData->MiniInfo;

            if (ItemData->ItemMesh)
            {
                _meshComponent->SetStaticMesh(ItemData->ItemMesh);

            }

            if (ItemData->ItemTexture)
            {
                _ItemTexture = ItemData->ItemTexture;
                UMaterialInstanceDynamic* MaterialInstance = _meshComponent->CreateAndSetMaterialInstanceDynamic(0);
                if (MaterialInstance)
                {
                    MaterialInstance->SetTextureParameterValue("ItemTexture", ItemData->ItemTexture);
                }
            }


            UE_LOG(LogTemp, Warning, TEXT("Item Name : %s"), *_Name);
        }
    }
```

스탯과 비슷하게 데이터 테이블로 관리했지만 아이템에 대한 각 정보(매쉬, 텍스쳐, 능력치등)를 저장할
아이템 클래스를 따로 만들어 아이템클래스를 상속하는 블루프린트클래스를 생성하여 `itemId` 를 변경하여 아이템에 대한 값과 정보를 관리할수 있게함

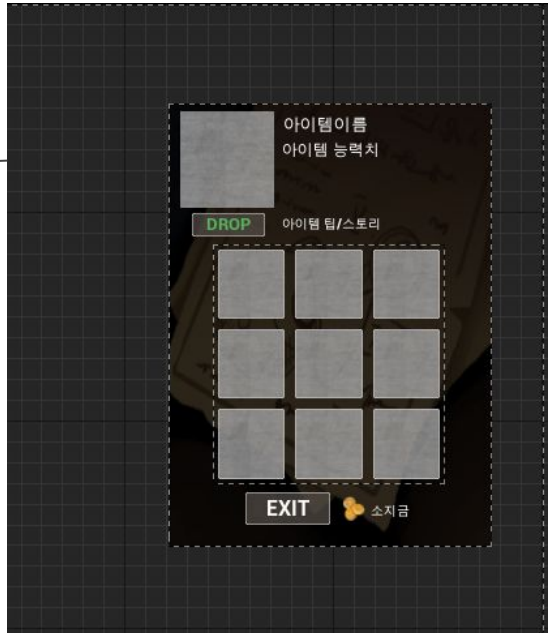
UI 위젯


```
UCLASS()
class TFT_PROJECT_A_API UTFT_HpBar : public UUserWidget
{
    GENERATED_BODY()

public:
    void SetHpBarValue(float ratio);
    void SetMpBarValue(float ratio);
    void SetExpBarValue(float ratio);

private:
    UPROPERTY(meta = (BindWidget))
        class UProgressBar* PB_HpBar;
    UPROPERTY(meta = (BindWidget))
        class UProgressBar* PB_MpBar;
    UPROPERTY(meta = (BindWidget))
        class UProgressBar* PB_ExpBar;
};
```



```
#define UIMANAGER Cast<UTFT_GameInstance>(GetGameInstance())->GetUIManager()
#define EffectManager Cast<UTFT_GameInstance>(GetGameInstance())->GetEffectManager()
#define SoundManager Cast<UTFT_GameInstance>(GetGameInstance())->GetSoundManager()

UCLASS()
class TFT_PROJECT_A_API UTFT_GameInstance : public UGameInstance
{
    GENERATED_BODY()

public:
    UTFT_GameInstance();

    virtual void Init() override;

    FTFT_StatData* GetStatDataByLevel(int32 level);

    ATFT_UIManager* GetUIManager() { return _uiManager; }
    ATFT_Effect_Manager* GetEffectManager() { return _effectManager; }
    ATFT_SoundManager* GetSoundManager() { return _soundManager; }

private:
    UPROPERTY()
    class UDataTable* _statTable;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    ATFT_UIManager* _uiManager;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    ATFT_Effect_Manager* _effectManager;

    UPROPERTY(VisibleAnywhere, BlueprintReadWrite, meta = (AllowPrivateAccess = "true"))
    ATFT_SoundManager* _soundManager;
};
```

```
    if (dataTable.Succeeded())
    {
        _statTable = dataTable.Object;
        UE_LOG(LogTemp, Error, TEXT("StatTable Load Complete"));
    }


}

void UTFT_GameInstance::Init()
{
    Super::Init();

    auto statData = GetStatDataByLevel(10);

    UE_LOG(LogTemp, Warning, TEXT("Level : %d , MaxHp : %d , MaxMp : %d , Attack : %d")
        , statData->level, statData->maxHP, statData->maxMP, statData->attack);

    FActorSpawnParameters params;
    params.Name = TEXT("UIManager");
    _uiManager = GetWorld()->SpawnActor<ATFT_UIManager>(FVector::ZeroVector, FRotator::ZeroRotator, params);

    params.Name = TEXT("EffectManager");
    _effectManager = GetWorld()->SpawnActor<ATFT_Effect_Manager>(FVector::ZeroVector, FRotator::ZeroRotator, params);

    params.Name = TEXT("SoundManager");
    _soundManager = GetWorld()->SpawnActor<ATFT_SoundManager>(FVector::ZeroVector, FRotator::ZeroRotator, params);
}

FTFT_StatData* UTFT_GameInstance::GetStatDataByLevel(int32 level)
{
    auto statData = _statTable->FindRow<FTFT_StatData>(*FString::FromInt(level), TEXT(""));
    return statData;
```



같은 값을 공유하는 경우는 하나의 클래스를 상속받는 위젯을 설계한 대로 객체에 맞게 바인딩 특정 UI 경우 각 객체가 소유하는게 관리하기 유용한 경우가있는경우는(HpBar..) 객체가 각각 관리.

하나만 존재해도 되고 언제 어디서나 글로벌적으로 게임에서 전역으로 사용되는 경우
지속적으로 데이터를 관리하기 좋은 UI(인터페이스, 인벤토리...)등은 게임인스턴스에서 관리한다.

인벤토리

```
private:
    UPROPERTY(VisibleAnywhere, BlueprintReadOnly, Category = Items, meta = (AllowPrivateAcce
    TArray<ATFT_Item*> _items;

    int32 _inventoryMaxSize = 9;
    int32 _playerGold = 100;
public:
    // Delegate
    InvenUIOpen _invenOpenDelegate;

    ItemAdded _itemAddedEvent;
    ItemAdded _itemSlectEvent;

    InvenGold _GoldChangeEvnet;
};
```

```
UTFT_InvenComponent::UTFT_InvenComponent()
{

    PrimaryComponentTick.bCanEverTick = false;

    _items.SetNum(_inventoryMaxSize);

}
```

```
void UTFT_InvenComponent::AddItem(ATFT_Item* item)
{
    if (item->GetItemType() == "gold")
    {
        UE_LOG(LogTemp, Log, TEXT("%d gold Get~too ~"), item->GetItemGold());

        AddPlayerGold(item->GetItemGold());
        item->Disable();

        return;
    }
    else
    {
        for (int32 i = 0; i < _inventoryMaxSize; ++i)
        {
            if (_items.IsValidIndex(i) && _items[i] == nullptr)
            {
                _items[i] = item;
                UE_LOG(LogTemp, Log, TEXT("Added item to inventory at index: %d"), i)
                _itemAddedEvent.Broadcast(item, i);
                item->Disable();
                return;
            }
        }
        UE_LOG(LogTemp, Warning, TEXT("Inventory is full, unable to add item."));
    }

}
```

```
void ATFT_Player::BeginPlay()
{
    Super::BeginPlay();

    _invenCom->_itemAddedEvent.AddUObject(this, &ATFT_Player::AddItemHendle);
    _invenCom->_GoldChangeEvnet.AddUObject(this, &ATFT_Player::UIGold);
    UIMANAGER->GetInvenUI()->_SlotItemEvent.AddUObject(this, &ATFT_Player::DropItemPlayer);
}
```

```
void ATFT_Player::AddItemPlayer(ATFT_Item* item)
{
    _invenCom->AddItem(item);

    _statCom->AddAttackDamage(item->GetItemAttackDamage());
}

void ATFT_Player::AddItemHendle(ATFT_Item* item, int32 index)
{
    UIMANAGER->GetInvenUI()->AddUiItem(item, index);
}

void ATFT_Player::DropItemPlayer(ATFT_Item* item, int32 index)
{
    _invenCom->DropItem(index);

    _statCom->AddAttackDamage(-item->GetItemAttackDamage());
}

void ATFT_Player::UIGold(int32 gold)
{
    UIMANAGER->GetInvenUI()->UIGold(gold);
}
```

```
void UTFT_InvenUI::AddUiItem(ATFT_Item* item, int32 index)
{
    UE_LOG(LogTemp, Log, TEXT("AddUIItem In"));
    if (!item)
    {
        UE_LOG(LogTemp, Log, TEXT("NO ITem"));
    }
    SetItemSlot(item->GetMyTexture(), index);
    _UIsaveiteminfo[index] = item;
}
```

제작한 아이템을 관리할 컴포넌트를 생성 기본적으로 중요한 데이터는 전부 은닉하여서 관리함 현재 인벤토리는 배열로 생성했고 크기를 미리 지정해 인덱스 값으로 아이템을 관리함 아이템을 관리해야하는 상황이 되면 기본적인 연산은 컴포넌트가 처리하며 델리게이트를 이용해 연산이 끝난 정보를 쏴주고 이를 관리하도록 만듬

바인딩

최종적으로 위젯, 컴포넌트, 인스턴스등 모든 구역에 작성했던 코드등을 델리게이트 바인딩등을 통해서 정상적으로 동작하는지 체크

예시동작 : 체력바의 체력이 줄어듬 / 인벤토리에 아이템이 등록/드랍

End - 마무리 하며..

비주얼 스튜디오에서 C++ 코드를 Text로만 만들던 RPG등을 실제로 구현하면서 여러가지 경험해볼수 있었다.

개인개인이 따로 작업을 하는것이아닌 팀프로젝트로써 역할을 나눠서 각자 맡은 분야에서 코딩을 하면서 소통하며 맞춰가고 서로의 코드병합을 하며 처음 마주했던 오류와 문제들을 해결하면서 코드적인 부분 이외에도 깃허브등 개발에 필요한 여러가지 지식등을 얻을수있어서 후에 실제로 팀 프로젝트를 하더라도 이런 사소한 경험이 쌓이면서 실무에서도 도움이 될거같아 좋았습니다

앞으로도 이런식으로 팀프로젝트를 진행하면서 이번에는 하지못했던 작업등을 시도하고 좀더 많은것을 적용하고 사용할수 있도록 정진할 것입니다.

여기까지 봐주셔서 감사합니다.