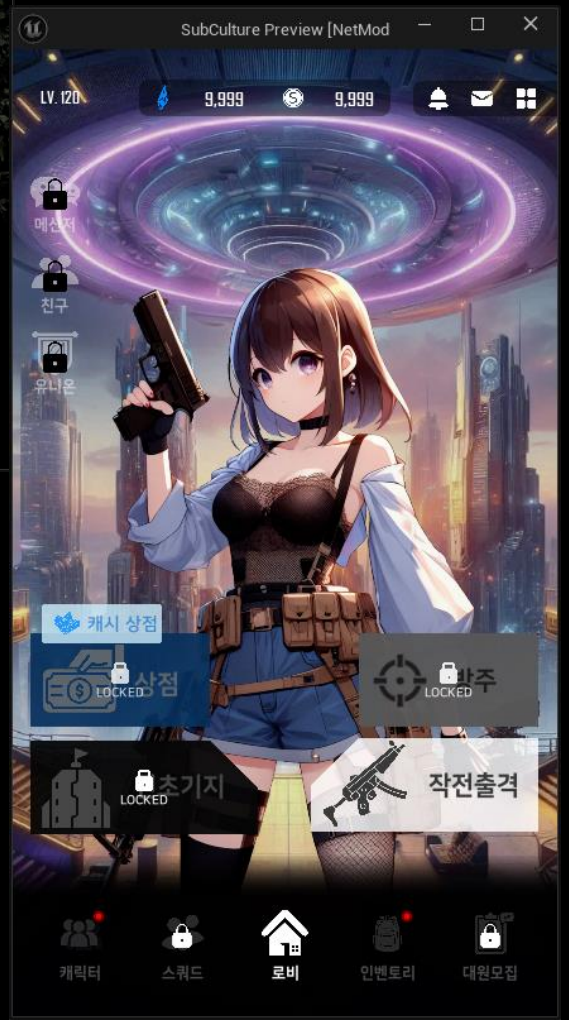
2025 포트폴리오

오재훈

1. MMO 스타일

2. 서브컬쳐 스타일



프로젝트 개요

프로젝트 설명

- 언리얼 엔진5 기반으로 개발된 개인 미니 UI 포트폴리오 프로젝트입니다. UMG 기반 위젯을 C++로 구조화하고 데이터 기반 UI 필터링 및 정렬, 강화, 제작 등의 컨텐츠들을 구현하였습니다.

개발 환경

- Unreal Engine 5.5
- Visual Studio 2022
- C++ / Blueprint

※본 프로젝트에 사용된 리소스 및 머테리얼(텍스처, 아이콘 등)은 모두 온라인에서 찾아가지고 오거나 AI 생성 도구를 통해 생성한 리소스를 사용하였습니다.

포트폴리오 영상 링크
https://youtu.be/6dWpDaXHGjc

1. MMO 스타일

UI 구조

1. 메인 화면
   ↳ 강화 창
      ↳ 강화 성공 창 / 강화 실패 창

2. 메인 화면
   ↳ 제작 창
      ↳ 제작 결과 창

핵심 특징

1. 데이터 기반 UI 처리
- ListView와 DataTable, Enum을 활용하여 필터링 및 항목 출력

2. 재사용 및 확장을 고려한 위젯 구조 설계
- 아이템 슬롯, 성공 결과 창, 실패 결과 창 등 공통 UI 모듈화

1 – 1. 강화 창

기능 개요

– 장비를 강화할 수 있는 시스템으로 조건에 따라 확률 기반으로 성공, 실패 여부가 결정

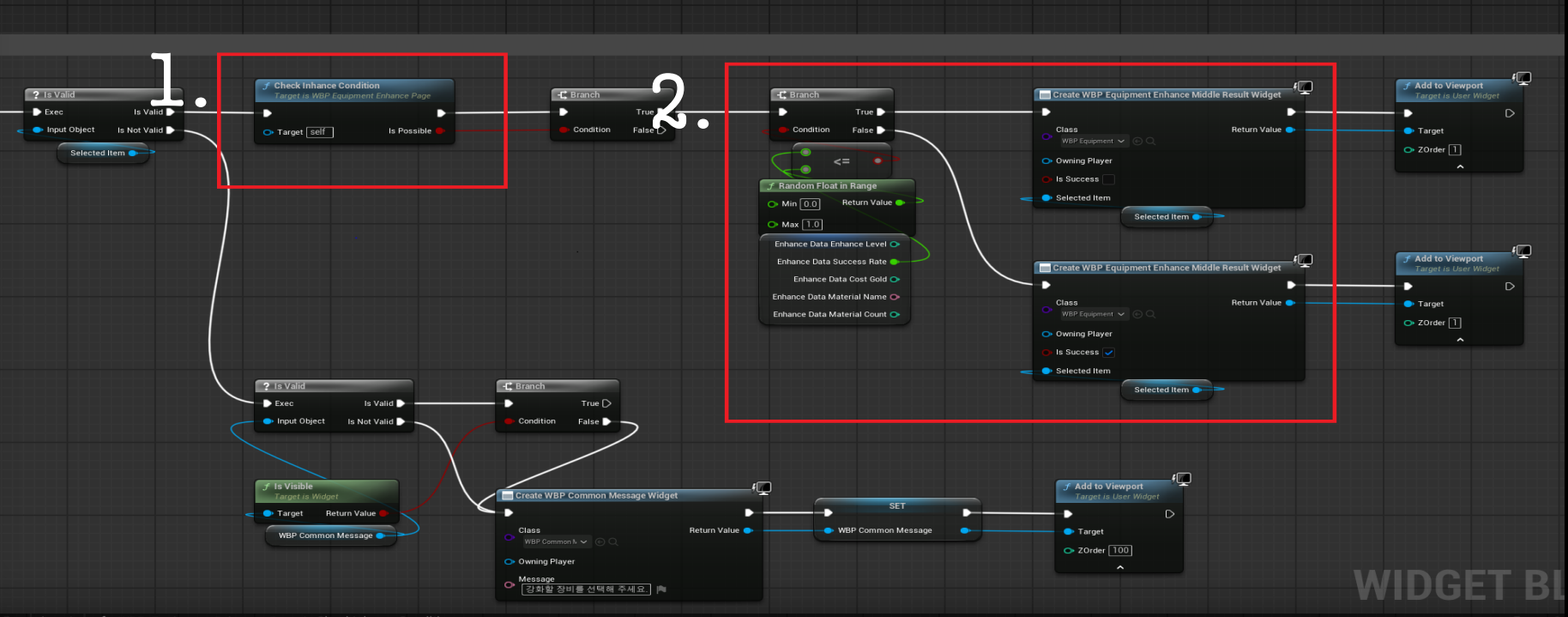
기능 요약

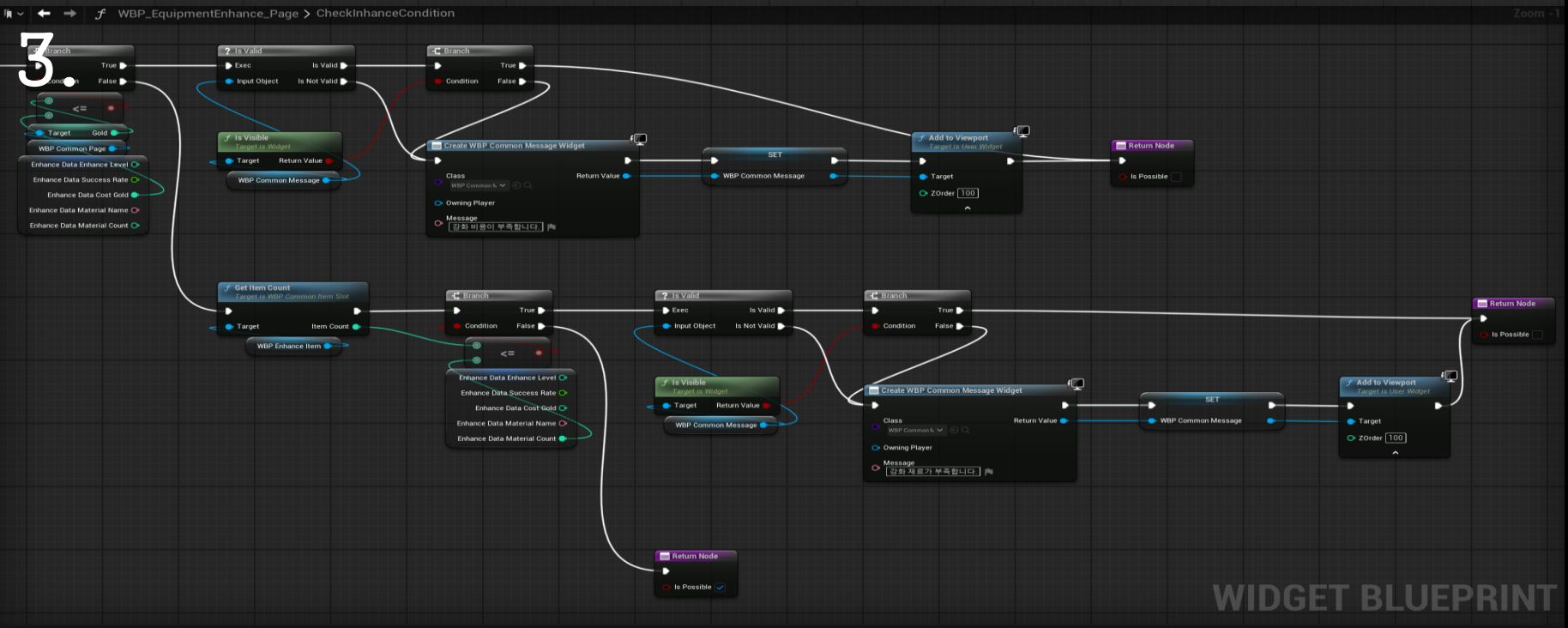
– 장비 선택 : 강화 대상 장비를 목록에서 선택 후 조건을 만족하는 경우만 세팅

– 강화 버튼 클릭 시 처리 흐름 : 강화 성공 시 강화 성공 창, 실패 시 실패 창 출력

동적 반응 처리

– 선택한 장비의 강화 수치가 최대일 시, 강화할 장비를 선택하지 않았을 시 메시지로 유저에게 안내





1 – 1. 강화 창

주요 동작 흐름

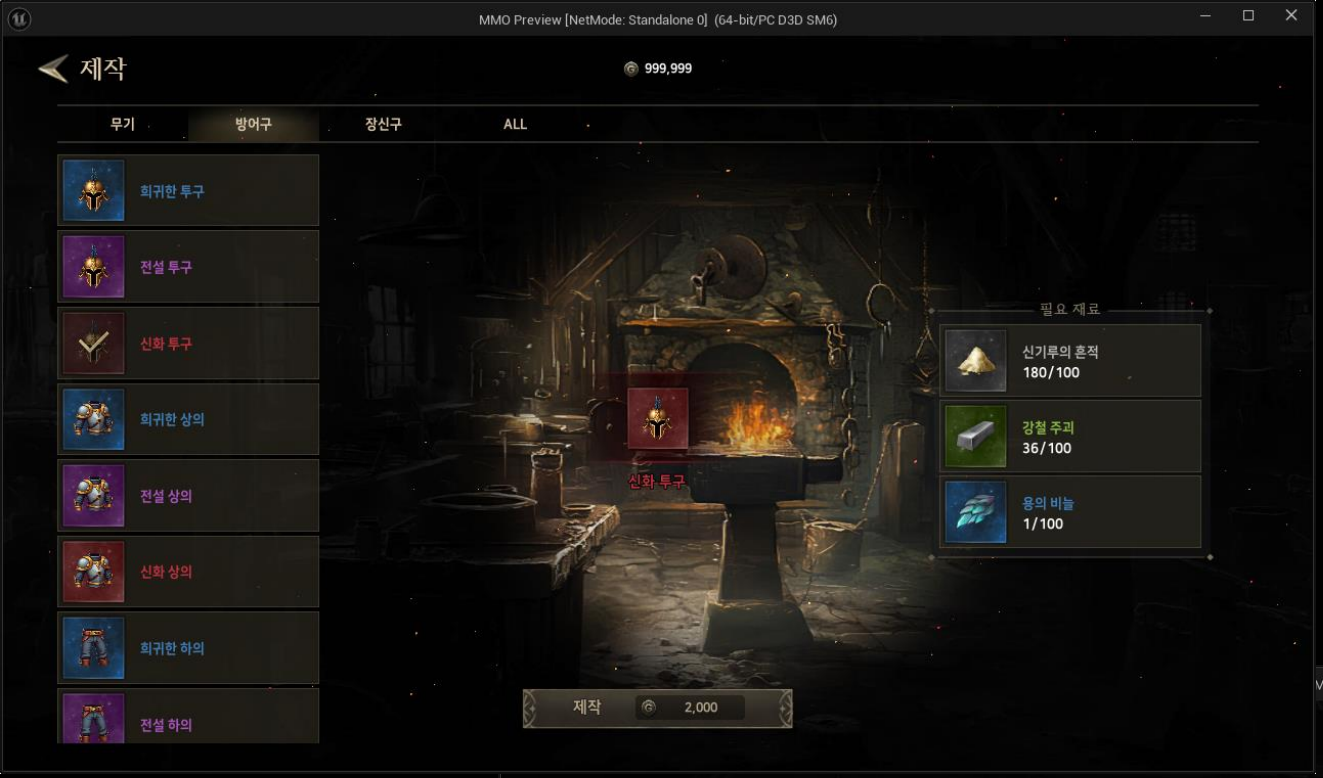
1. 강화 조건 체크
– 강화 가능한 상태인지 확인

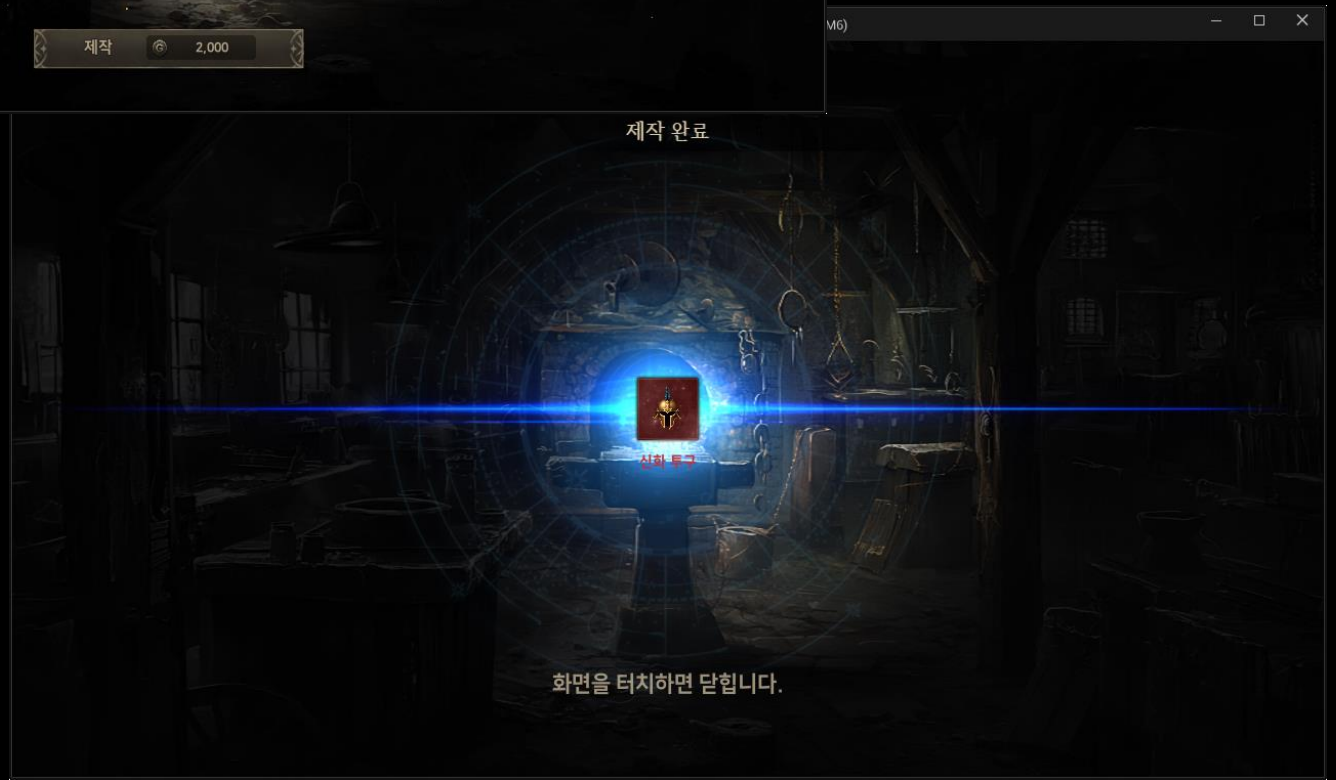
– 조건 불충분 시 오류 메시지 출력

2. 조건 충족 시 확률 계산
– 강화 수치에 해당하는 확률을 데이터에서 불러와 성공 여부 계산

– 성공/실패에 따라 분기 처리

3. 강화 조건 체크 내부 로직
– 재료 소지 여부 확인





1 – 2. 제작 창

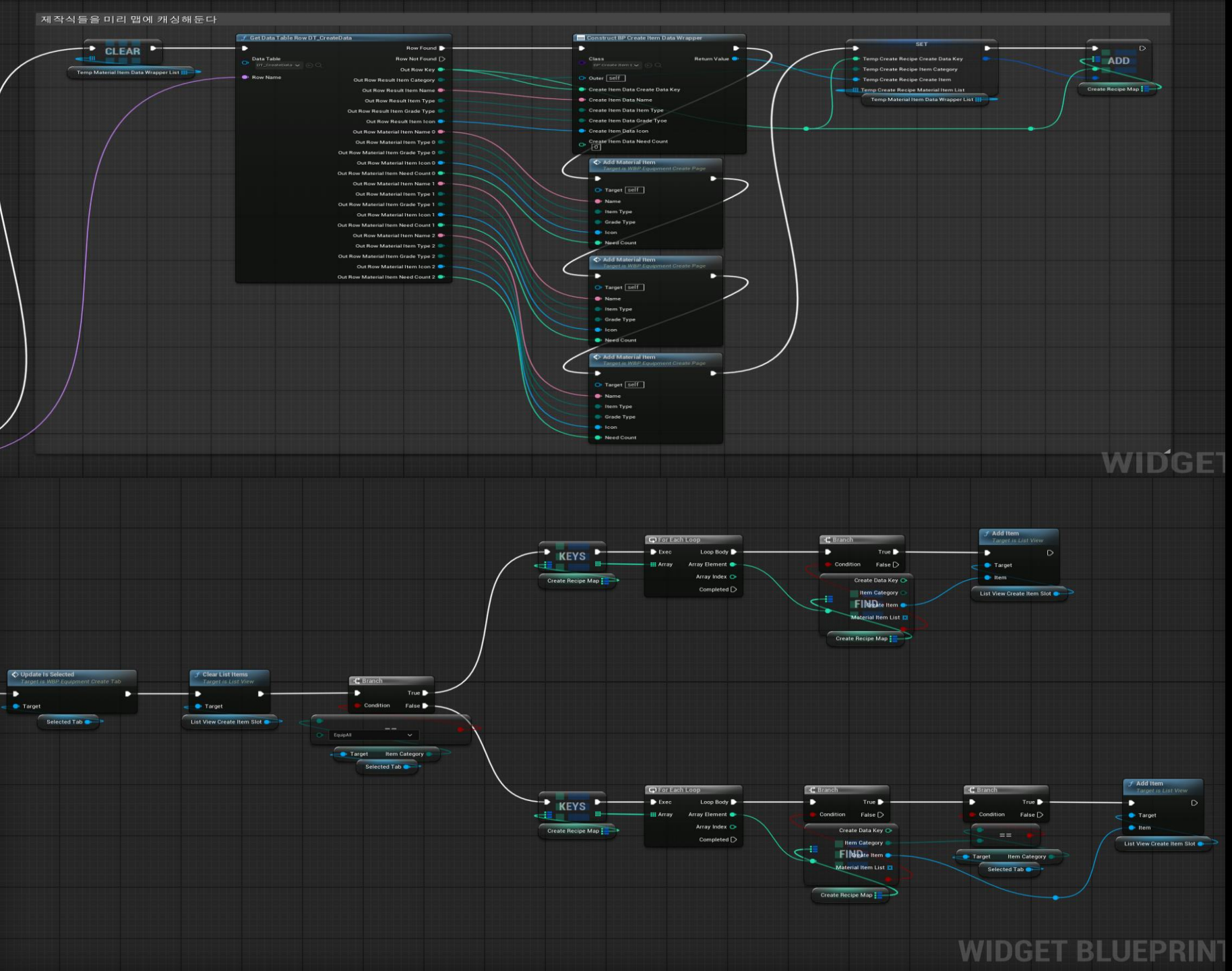
기능 개요

– 아이템을 제작할 수 있는 인터페이스로 제작 가능한 항목을 출력

기능 요약

– 제작 아이템 리스트 필터링 : Category 타입을 기준으로 탭 선택 시 해당 카테고리에 속한 아이템만 리스트에 출력

– 데이터 기반의 제작 재료 정보 표시 : 제작할 아이템을 선택하면 아이템에 필요한 재료 정보를 데이터 테이블 기반으로 UI에 표시



1 – 2. 제작 창

주요 동작 흐름

1. 제작식 사전 캐싱

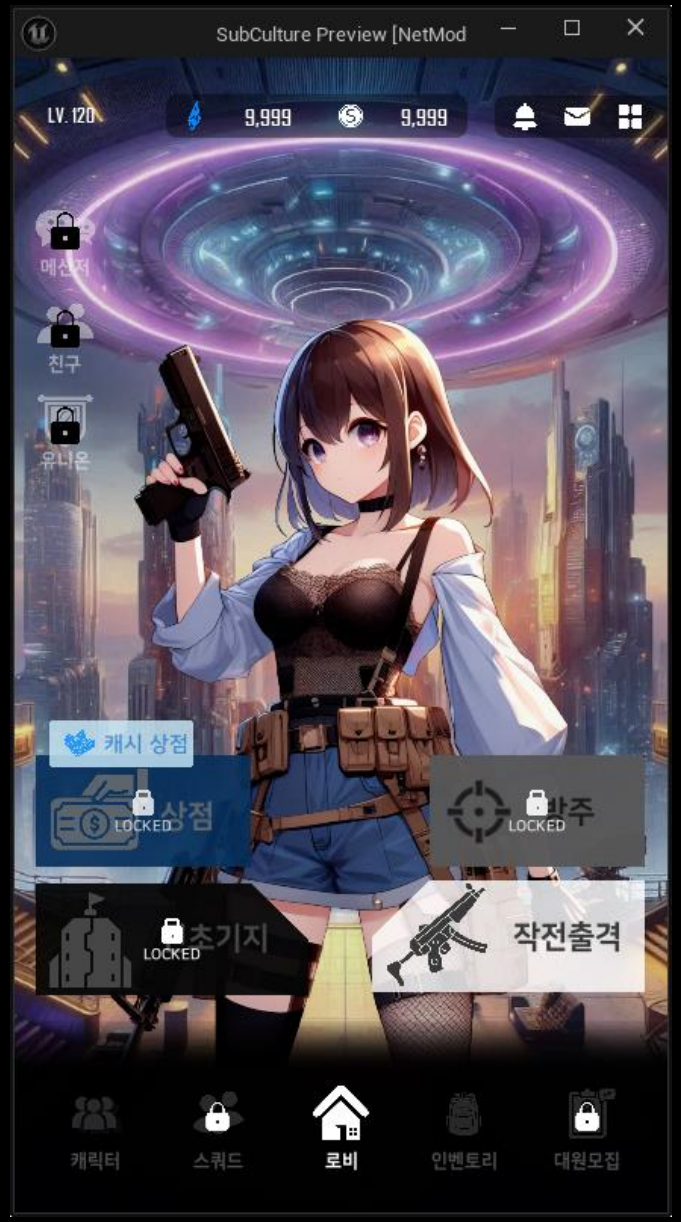
– Construct 노드에서 모든 제작 데이터를 Map<Key, Recipe> 형식으로 미리 캐싱해둡니다.

2. 카테고리 필터링 및 아이템 출력

– 탭 선택 시 해당 카테고리에 맞는 아이템을 캐싱된 맵에서 가져와 리스트에 출력합니다.

3. 아이템 선택 시 재료 세팅

– 선택한 아이템에 맞는 제작 재료 정보를 데이터에서 가져와 UI에 세팅합니다.



2. 서브컬쳐 스타일

UI 구조

로비 화면

↳ 캐릭터 창
↳ 캐시 상점
↳ 인벤토리
↳ 작전출격

핵심 특징

1. 코드 및 데이터 기반 UI 처리

- ListView와 DataTable, Enum을 활용하여 UI 항목의 출력과 필터링을 데이터 기반으로 구현

2. 재사용 및 확장을 고려한 위젯 구조 설계

- 공통 슬롯 UI를 캐릭터, 아이템 등에서 모듈화하여 재사용 가능하게 설계

- 컨텐츠 확장 시에도 기존 위젯 수정 없이 재활용 가능

3. 컨텐츠 이용 여부에 따른 잠금 처리

- 컨텐츠 조건을 만족하지 못한 항목은 비활성화 상태로 표기

2 – 1. 로비 화면

기능 개요

– 게임의 주요 컨텐츠 진입점 역할, 다양한 메뉴 버튼을 통해 유저가 컨텐츠를 선택할 수 있도록 구성

기능 요약

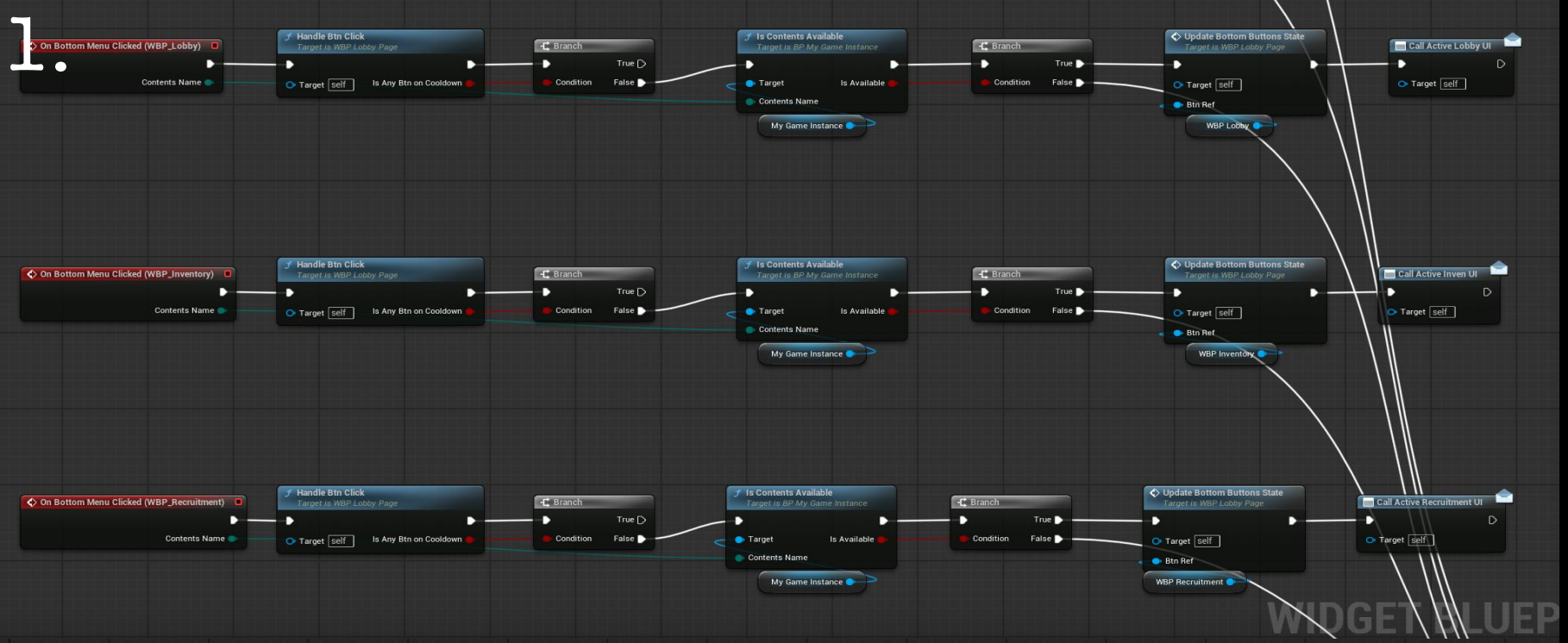
– 컨텐츠 진입 조건 검사 : 각 메뉴 버튼 클릭 시 ContentsAvailable 데이터의 조건을 확인

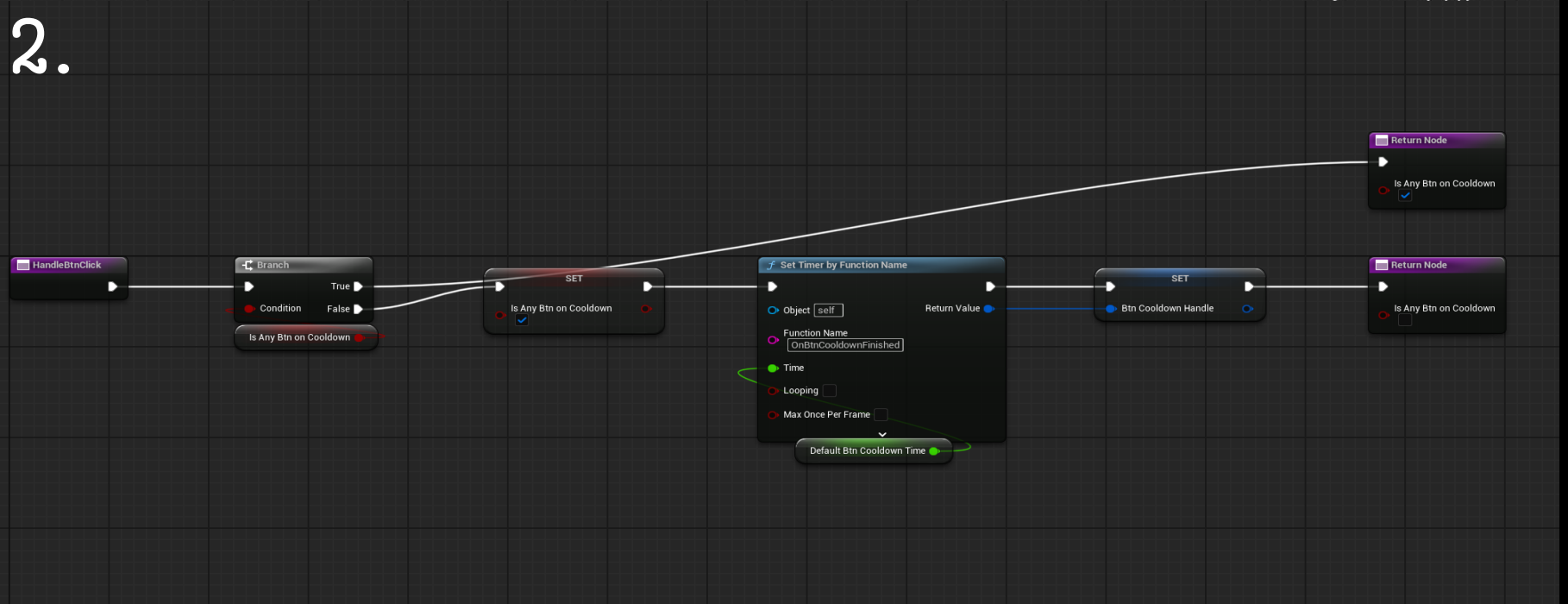
– 버튼 다중 클릭 방지 : IsAnyBtnOnCooldown 변수를 통해 다중 입력을 제한하여 의도하지 않은 중복 진입 방지

– 모듈화된 컨텐츠 진입 처리 : 각 버튼은 컨텐츠 타입을 전달, 내부에서 공통 진입 함수를 통해 분기 처리하여 유지 보수가 용이하게 구성

동적 반응 처리

– 컨텐츠 진입 조건을 만족하지 못하는 경우 팝업 메시지로 유저에게 안내





2 – 1. 로비 화면

주요 동작 흐름

1. 각 메뉴 버튼 클릭 시

– 버튼에 매핑된 컨텐츠 이름을 전달

– 전달받은 이름을 기반으로 ContentsAvailable 데이터를 조회

– 조건 만족 시 해당 컨텐츠로 진입

– 만족하지 않을 경우 사용자에게 안내 메시지 출력

2. 중복 클릭 방지 처리

– 클릭 시 IsAnyBtnOnCooldown 플래그를 True로 설정

– 일정 시간 이후 자동으로 False로 변경되어 클릭 가능 상태로 복귀

– 이를 통해 UI 오작동 및 중복 진입 방지

2 – 2. 캐릭터 창

기능 개요

– 보유 중인 캐릭터들을 한눈에 확인하고 필터, 정렬 등을 통해 선택, 관리할 수 있는 인터페이스 제공

기능 요약

– TileView를 활용해 캐릭터 리스트 시각화

– 필터 기능 : 무기 타입을 선택적으로 적용 가능

– 정렬 기능 : 등급, 레벨, 전투력을 기준으로 오름/내림차순 정렬

데이터 연동 구조

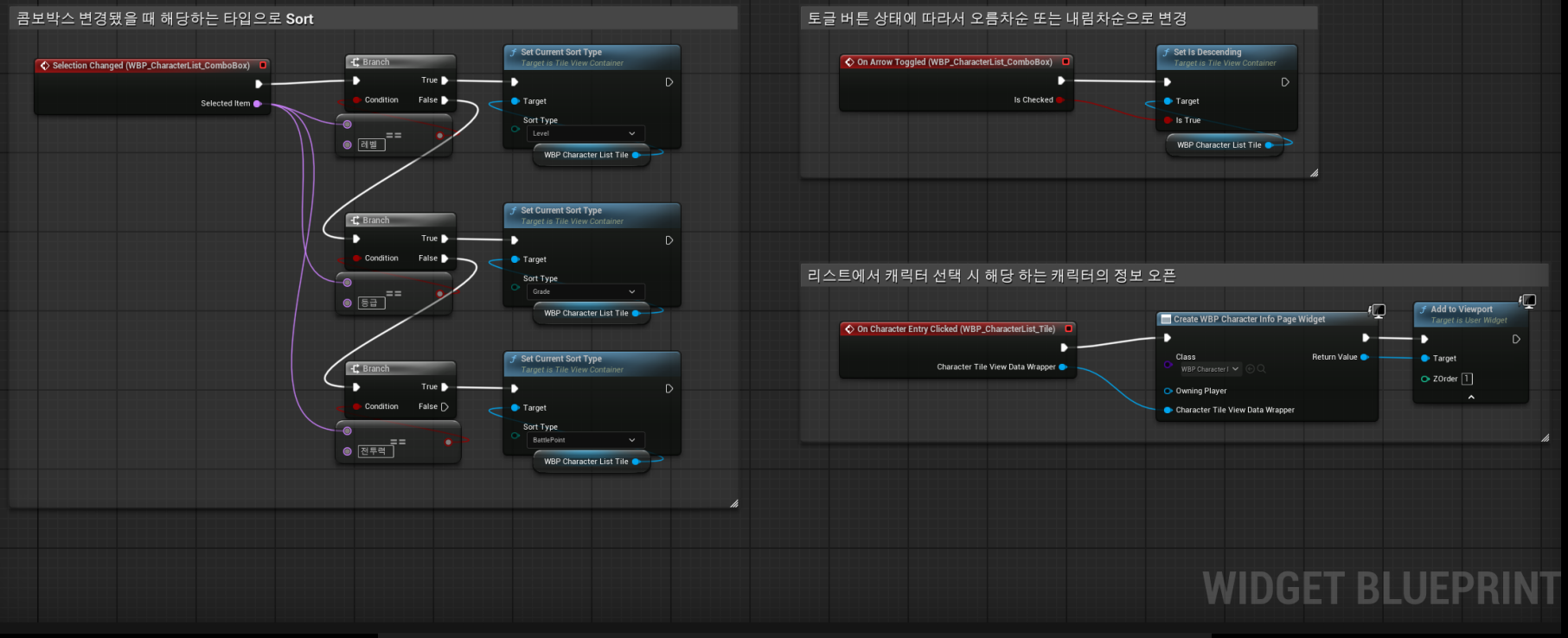
– FCharacterTileData 구조체를 기반으로 캐릭터 정보를 UCharacterTileViewDataWrapper에 감싸 전달

동적 반응 처리

– 캐릭터 레벨이 상승하면 OnLevelChanged 델리게이트를 통해 Entry에서 UI 업데이트

– 최대 레벨 도달 시 MaxLevel 이벤트 트리거 가능

2 – 2. 캐릭터 창




```cpp
void UTileViewContainer::SetCurrentSortType(ETileSortType sortType)
{
    if (currentSortType == sortType)
    {
        return;
    }

    currentSortType = sortType;
    ApplyFilter();

}

void UTileViewContainer::SetIsDescending(bool isTrue)
{
    isDescending = isTrue;
    ApplyFilter();
}
```

주요 동작 흐름

1. 필터 조건 변경 시

- 사용자가 Gun타입 또는 정렬 방식을 선택하면 해당 정보가 UI 이벤트로 전달됨

- 전달된 Enum값을 기반으로 클라이언트 코드에서 데이터 테이블 필터링 및 정렬 로직 실행

- 필터링된 결과를 ListView에 바인딩하여 리스트 갱신

2. 캐릭터 선택 시

- ListView에서 항목 선택 시 해당 캐릭터의 데이터가 바인딩되어 상세 정보 UI에 적용

2 – 2. 캐릭터 창

주요 동작 흐름

1. 공통 Tile UI Container 구조

– 필터 및 정렬 로직을 기반 클래스로 구현해 재사용 가능하도록 분리

– 전달된 Enum값을 기반으로 클라이언트 코드에서 데이터 테이블 필터링 및 정렬 로직 실행

– 필터링된 결과를 ListView에 바인딩하여 리스트 갱신

2. 다중 필터링 허용

– TSet<Enum>으로 관리

– All 선택 시 나머지 필터 자동 해제

3. 설정된 SortType 기반으로 정렬

1.
```cpp
virtual bool CompareTileData(const UTileViewDataWrapper* first, const UTileViewDataWrapper* second) const PURE_VIRTUAL(UTileViewContainer::CompareTileData, return false;);
virtual void ApplyFilter() PURE_VIRTUAL(UTileViewContainer::ApplyFilter, );
```

2.
```cpp
void UTileCharacterView::ApplyFilter()
{
    if (!TileView)
    {
        return;
    }

    TArray<UTileViewDataWrapper*> filteredTileDataList;
    if (activeFilters.Num() == 0 || activeFilters.Contains(EGunCategory::All) == true)
    {
        filteredTileDataList.Append(tileDataList);
    }
    else
    {
        for (UTileViewDataWrapper* data : tileDataList)
        {
            if (data)
            {
                if (UCharacterTileViewDataWrapper* characterWrapper = Cast<UCharacterTileViewDataWrapper>(data))
                {
                    if (activeFilters.Contains(characterWrapper->characterData.category) == true)
                    {
                        filteredTileDataList.Add(characterWrapper);
                    }
                }
            }
        }
    }

    filteredTileDataList.Sort([this](const UTileViewDataWrapper& first, const UTileViewDataWrapper& second)
        {
            return CompareTileData(&first, &second);
        });

    SetAndRefreshTileView(filteredTileDataList);
}
```

3.
```cpp
bool UTileCharacterView::CompareTileData(const UTileViewDataWrapper* first, const UTileViewDataWrapper* second) const
{
    const UCharacterTileViewDataWrapper* firstCharacterTileWrapper = Cast<UCharacterTileViewDataWrapper>(first);
    const UCharacterTileViewDataWrapper* secondCharacterTileWrapper = Cast<UCharacterTileViewDataWrapper>(second);
    if (!firstCharacterTileWrapper || !secondCharacterTileWrapper)
    {
        return false;
    }

    const FCharacterTileData& firstCharacterData = firstCharacterTileWrapper->characterData;
    const FCharacterTileData& secondCharacterData = secondCharacterTileWrapper->characterData;
    switch (currentSortType)
    {
    case ETileSortType::Level:
        return isDescending ? firstCharacterData.level < secondCharacterData.level : secondCharacterData.level < firstCharacterData.level;
    case ETileSortType::Grade:
        if (firstCharacterData.characterGrade == secondCharacterData.characterGrade)
        {
            return secondCharacterData.level < firstCharacterData.level;
        }

        return isDescending ? firstCharacterData.characterGrade < secondCharacterData.characterGrade : secondCharacterData.characterGrade < firstCharacterData.characterGrade;
    case ETileSortType::BattlePoint:
        return isDescending ? firstCharacterData.level < secondCharacterData.level : secondCharacterData.level < firstCharacterData.level;
    default:
        return false;
    }
}
```