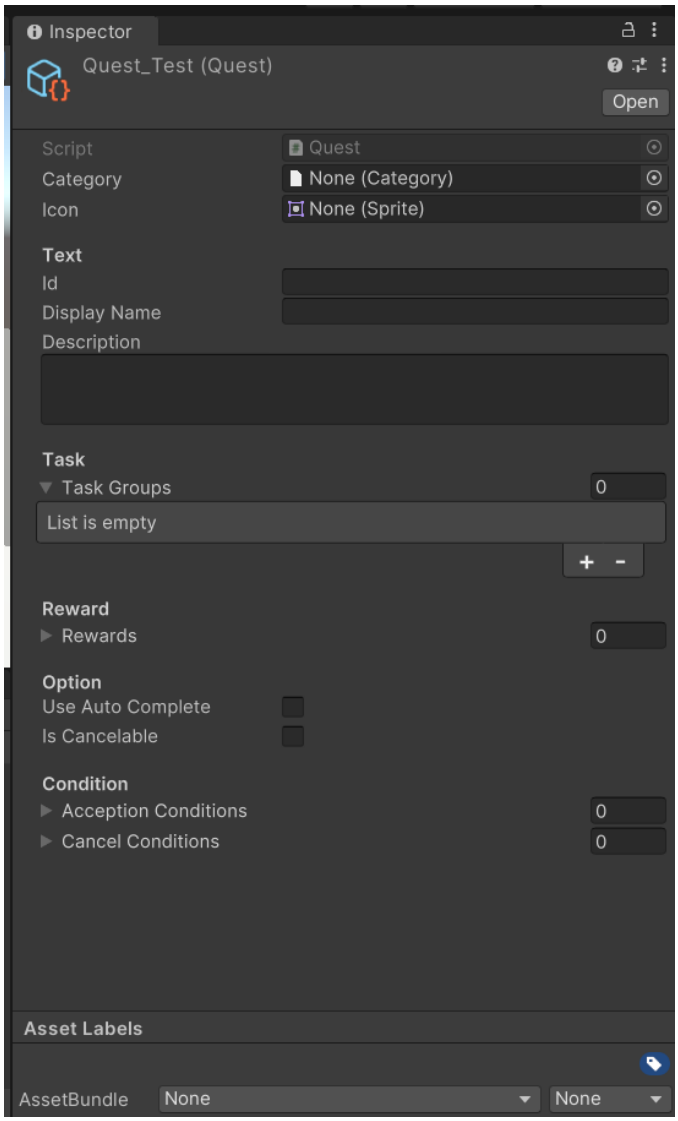
퀘스트 시스템 사용법

1. 설명 (별로 관심 없으면 2로 가기)

Quest



위에서부터

Category – 어떤 '종류'의 퀘스트인지 나타낸다.

Icon – 퀘스트의 아이콘

Id – 퀘스트의 id. 명명 규칙은 만든 스크립터블 오브젝트의 이름을 대문자로 치환한 것으로 한다.

ex) Quest_Main 객체라면 QUEST_MAIN

Display Name – 퀘스트의 이름. 진행 상황에 표시할 이름이므로 한글 작성

Description – 퀘스트의 설명. 퀘스트 모음 UI에 표시할 것이므로 한글 작성

Task – 해당 퀘스트를 클리어하기 위해 진행해야할 '일'들

ex) 마야의 부탁 : 초록버섯 X 5, 주황버섯 X 3 에서 마야의 부탁이 Quest이고, 버섯들이 Task

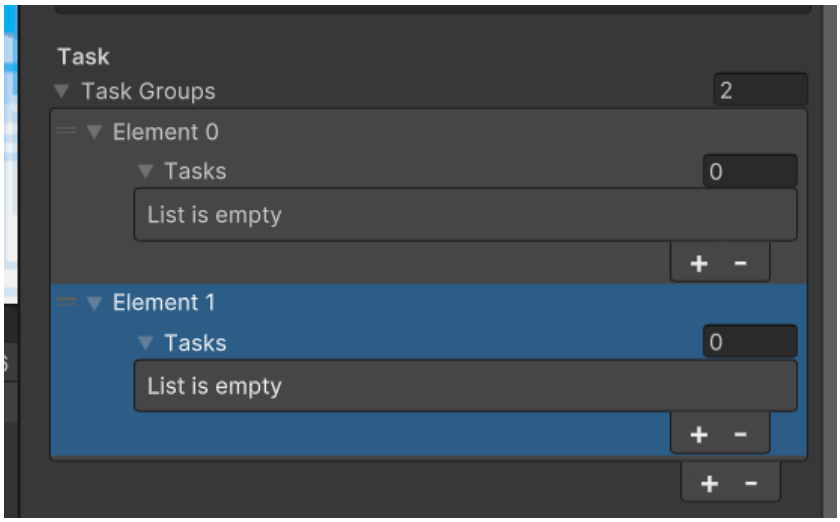
Reward – 퀘스트 클리어시 지급할 보상

Use Auto Complete – Task들을 전부 완료하면 자동으로 클리어시킬 것인지 유무

Is Cancelable – 퀘스트 중도 포기 가능한지 여부

Acception Conditions – 퀘스트 시작하기 위한 조건들

Cancel Conditions – 퀘스트 포기하기 위한 조건들



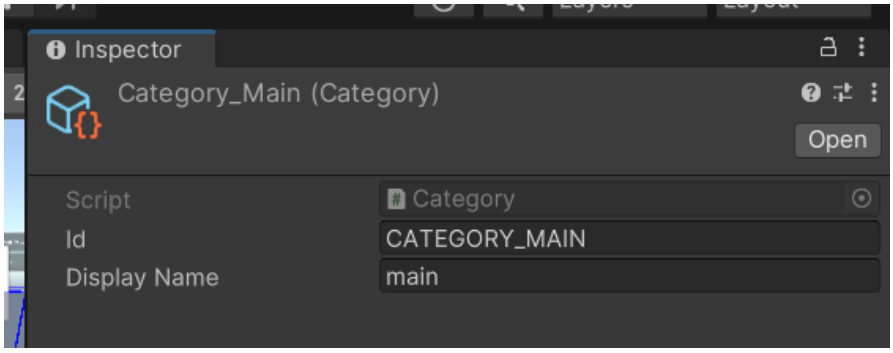
Task는 Task Group들을 리스트로 가지고 있다.

하나의 Task Group에서 여러 Task를 생성하는 것이 가능하다.

Task Group들은 Element0 -> Element1 -> .... 순서로 활성화가 되어서 순서대로 클리어하는 것이 가능하고 하나의 Element(Task Group) 내의 Task들은 순서 상관 없이 활성화가 되어있다.

만약 Element0의 모든 Task들을 클리어하면, Element1의 Task들이 활성화가 되고, Element1의 모든 Task들을 클리어하면 Element2의 Task들이 활성화되는 구조이다.

Category (Task들의 행동 구분도 담당하고, 퀘스트의 종류도 담당한다. - 범용)

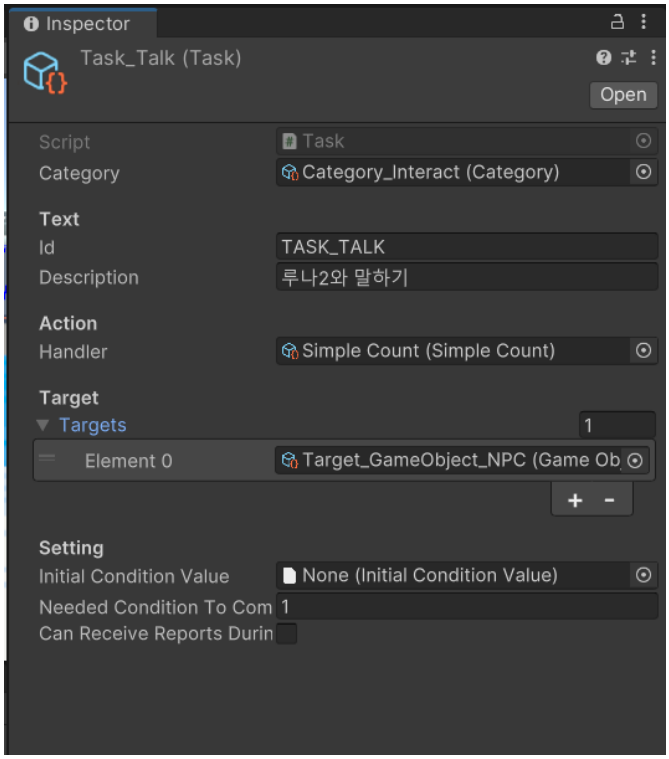


Id – 카테고리 id

Display Name – 카테고리가 실제로 UI에 표시되는 이름

여기에서 대괄호에 있는 것을 말한다.

Task



Category – 무슨 행동을 해야 클리어가 되는지 나타내기 위한 것(위에 언급했듯 퀘스트의 종류와 같이 Task의 종류를 나타내는 범용이다.)

Id – Task의 id

Description – UI에 나타낼 Task의 이름

Handler – 해당 Task에 대해 상호작용을 했다면, 어떤 방식으로 조건을 카운트해줄 것인지

Target – 해당 Task를 상호작용할 수 있는 타겟(해당 타겟도 역시 이것을 가지고 있다.)

여러 개인 이유는 Task가 초록버섯이나, 주황버섯을 아무거나 5마리 잡기. 이런 것일 경우를 대비한것

Initial Condition Value – 조건을 만족시키기 전 처음 Task를 받았을 때 초기값

예를 들어서 레벨 100 찍기는 레벨 10부터 시작할 경우 초기값이 0이 아니고 10이다. 만약 이부분을 비워놓으면, 초기값은 자동으로 0이다.

Needed Condition To Complete – Task 클리어하기 위한 목표값

Can Receive Reports During Completion – Task를 클리어 했음에도 조건 보고를 받을 것인가?

예를 들어서 아이템 100개를 모으는 Task가 있다면, 100개를 모았을 때 Task가 Complete가 될 텐데, 이때 퀘스트를 클리어하기 전, 플레이어가 아이템 50개를 버린다면 퀘스트 클리어를 못하게

해야하기에 필요한 조건이다. Handler중에서 Simple Set과 같이 쓰인다.(현재 프로젝트에서는 안써도 무방할 것으로 보인다.)

Reward

```
Unity 스크립트 | 참조 2개
public abstract class Reward : ScriptableObject
{
    [SerializeField] private Sprite icon;
    [SerializeField] private string description;
    [SerializeField] private int quantity;

    참조 1개
    public Sprite Icon => icon;
    참조 3개
    public string Description => description;
    참조 3개
    public int Quantity => quantity;

    /// <summary>
    /// Grant the specified reward.
    /// The `Quest` parameter indicates which quest is granting the reward.
    /// </summary>
    /// <param name="quest"></param>
    참조 1개
    public abstract void Give(Quest quest);
}
```

해당 스크립트를 상속 받아서, 원하는 보상 내용을 추상 메서드를 오버라이드해서 작성하면 된다. 그리고 스크립터블 오브젝트화 해서 보상 칸에 넣을 수 있다.

Icon – 보상 아이콘(아이템 등)

Description – 보상 내용

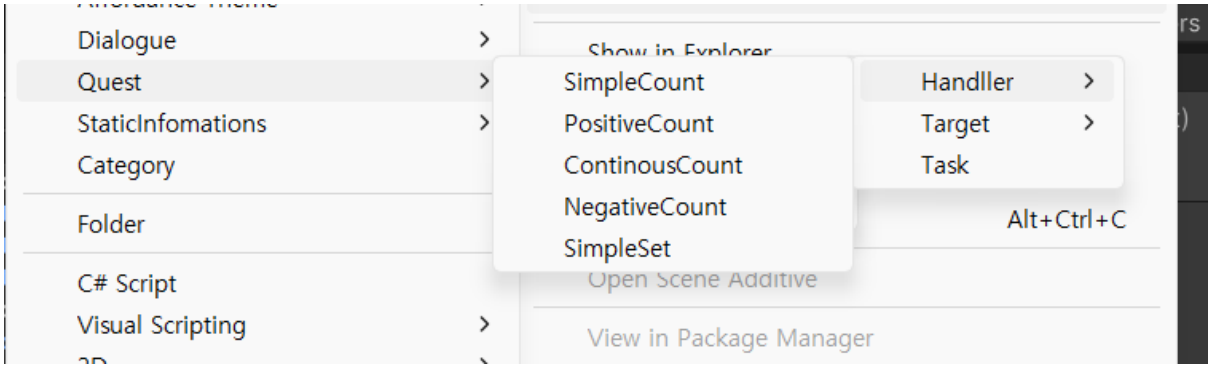
Quantity – 양

Condition

```
Unity 스크립트 | 참조 2개
public abstract class QuestAcceptionCondition : ScriptableObject
{
    [SerializeField] private string description;

    /// <summary>
    /// Prerequisites for starting or canceling a quest.
    /// Quest parameter indicates which quest is granting the condition.
    /// </summary>
    /// <param name="quest"></param>
    /// <returns></returns>
    참조 2개
    public abstract bool IsPass(Quest quest);
}
```

마찬가지로 퀘스트를 수주하기 위한 조건과, 취소하기 위한 조건을 해당 스크립트를 상속받아서 원하는 대로 작성해, 스크립터블 오브젝트화를 해서 넣어준다.

Handler



Task와 상호작용시, 적용해줄 Handler들이다.

Simple Count – 현재값에 정해준 값 +해주기

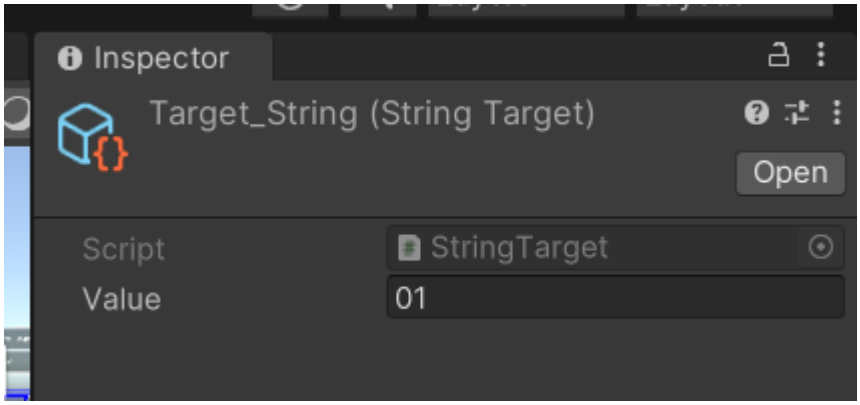
Positive Count – 정해준 값이 양수라면 현재값에 +, 그게 아니라면 현재값 그대로

Negative Count – 정해준 값이 음수라면 현재값에 -, 그게 아니라면 현재값 그대로

Continuous Count – 정해준 값이 양수라면 현재값에 +, 그게 아니라면 현재값 0으로 초기화

Simple Set – 정해준 값으로 현재값 덮어쓰기

Target



Task가 가지고 있는 Target Object와, Target이 가지고 있는 Target Object가 같으면 Task에 Handler가 작동

InitialConditionValue

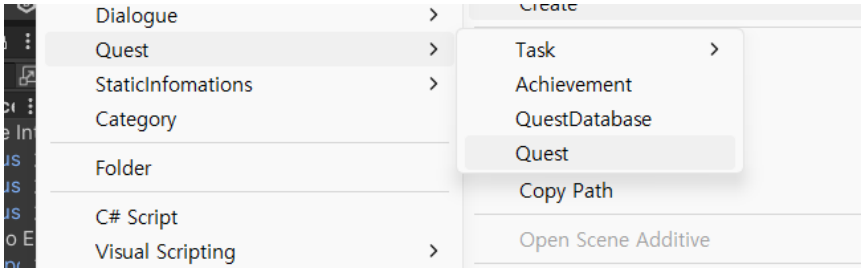
```
Unity 스크립트 | 참조 1개
public abstract class InitialConditionValue : ScriptableObject
{
    /// <summary>
    /// The initial value for the condition to clear a task.
    /// </summary>
    /// <param name="task"></param>
    /// <returns></returns>
    참조 1개
    public abstract int GetValue(Task task);
}
```

해당 스크립트를 상속받아 원하는 초기값이 리턴되도록 작성후 넣어주기.

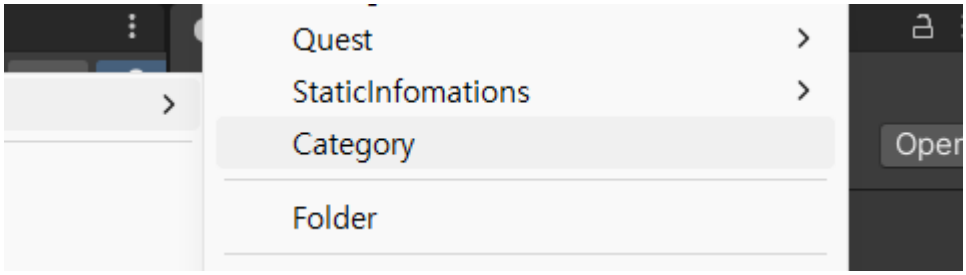
그냥 만들지 않고 비워두면 자동으로 0이다.

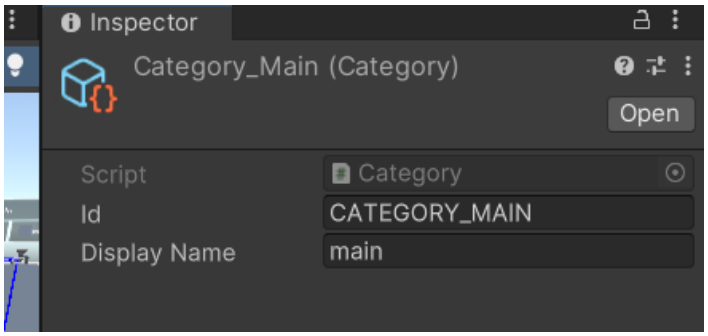
2. 사용법

(1) Quest 객체 만들기

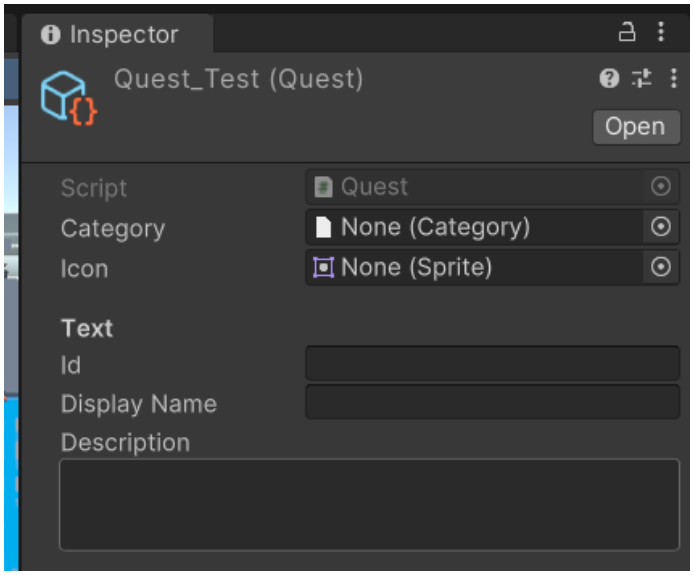


(2) 원하는 Category객체 만들고 넣기.(현 프로젝트에는 Main 퀘스트만 있을 예정이니 만들어놓은 Quest_Main을 넣으면 된다.)





(3) Quest 객체 일부 채우기



Category 넣기

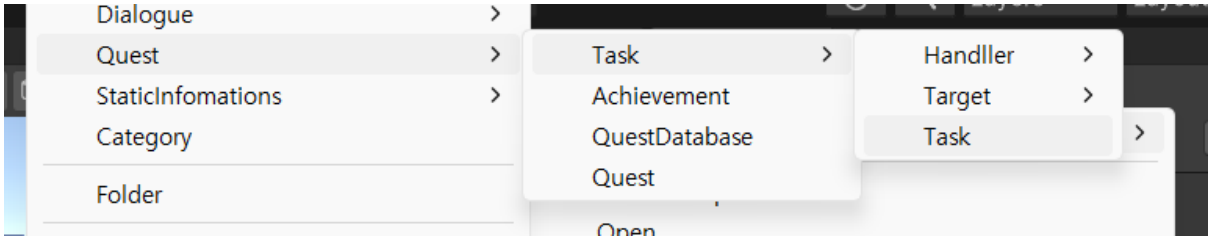
Icon 넣기(원한다면)

Id는 객체의 이름을 전부 대문자로 바꾸는 명명 규칙 따르기 (QUEST_TEST)

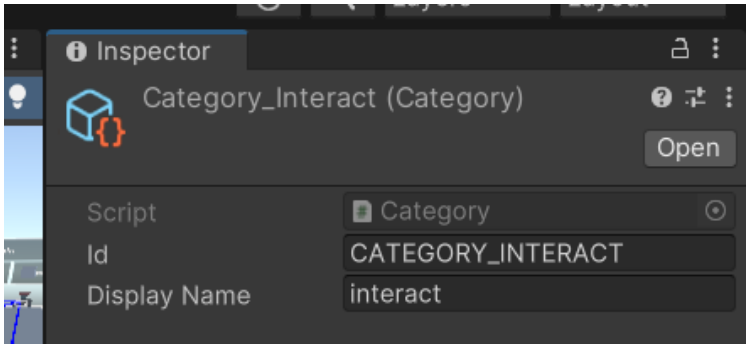
Display Name은 UI에 표시할 퀘스트의 이름 (한글)

Description은 퀘스트의 내용 (한글)

(4) Task 만들기

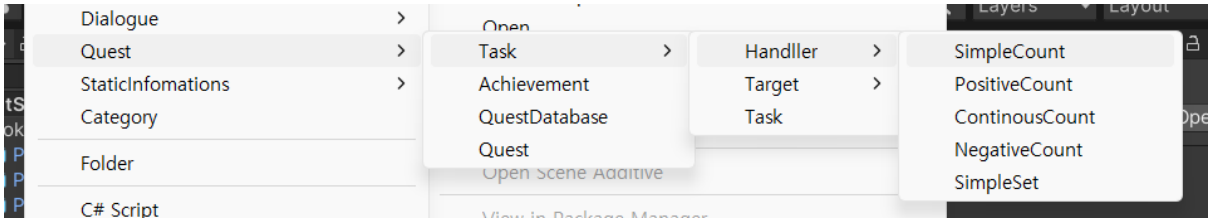


(5) Task를 클리어하기 위한 행동을 분류해주는 Category 추가(Quest를 분류해주는 용도와 같이 범용으로 사용)



ex) 상호작용으로 Task 처리

(6) Handler 생성 (상호작용한 Task 의 값을 어떤 방식으로 처리할 것인지)



Simple Count – 현재값에 정해준 값 더하기

Positive Count – 정해준 값이 양수면 현재 값에 더하기

Negative Count – 정해준 값이 음수면 현재 값에서 빼주기

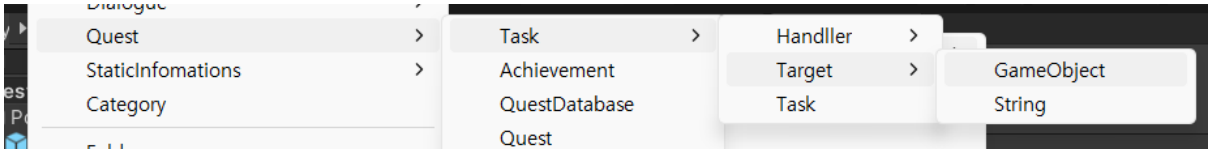
Continuous Count – 정해준 값이 양수면 현재 값에 더해주고, 아니라면 현재 값을 0으로

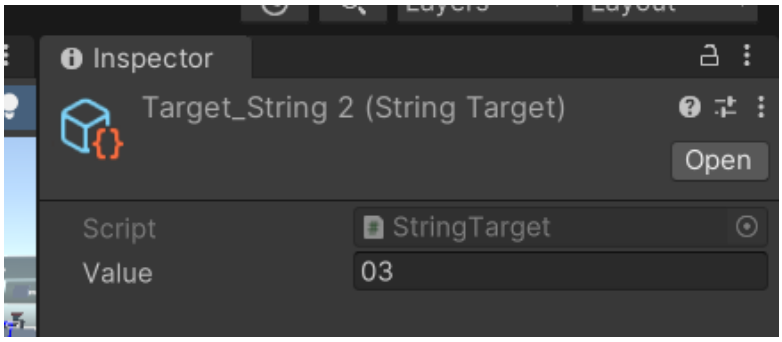
즉 연속적으로 양수여야 클리어 가능

Simple Set – 정해준 값으로 현재값을 덮어씌우기

만약 원하는게 없다면 상속받아서 새로 만들기

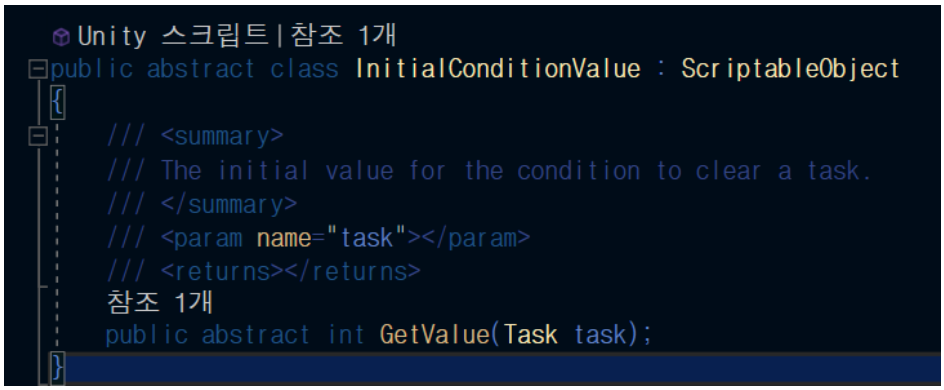
(7) 타겟 스크립터블 오브젝트 만들기





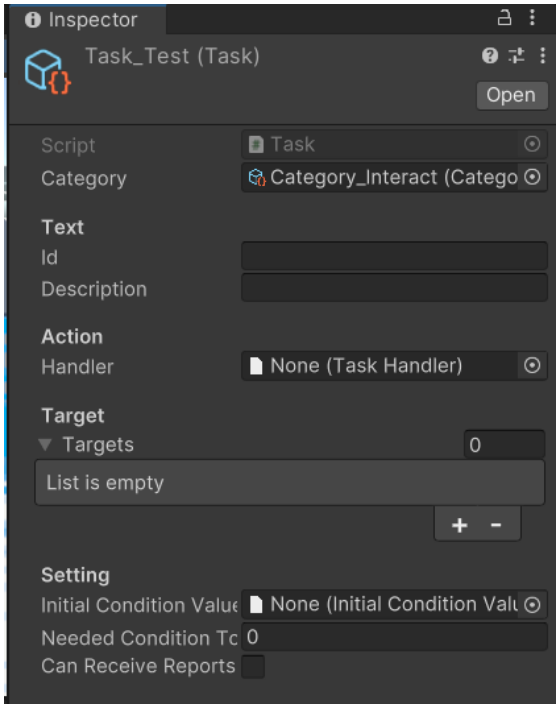
원하는 값 채워주기

(8) Initial Condition Value 생성해주기 (필수 아님. 비워놓을 시 0으로 초기화)



해당 스크립트 상속해서 원하는 Value 리턴해주도록 짜기

(9) Task 완성하기



만든 스크립터블 오브젝트들 등록해주고, 빈칸 채워주기

Id – 해당 Task id (Task객체 이름을 대문자로 치환한 명명 규칙)

Description – 해당 태스크의 내용 적기(한글)

Target은 여러 개 등록 가능 (ex - 주황버섯이나 초록버섯을 아무거나 5마리 잡아주세요)

Needed Condition To Complete – 클리어하기 위한 목표값

Can Receive Reports During Completion – 해당 태스크를 완료했음에도 조건 검사를 할 것인지.

(ex – 아이템 100개를 모으는 것이 목표일 때, 100개를 모았으면 저절로 Task가 완료되지만, 퀘스트 완료를 하지 않고 아이템 50개를 버리는 경우가 있음. 이때 Task를 완료했음에도 보고를 받도록 해서 현재 값을 50으로 바꾸면 Task가 다시 완료 전으로 돌아감. 따라서 해당 옵션은 Simple Set Handler와 같이 사용해줘야함. 보통 쓸 경우 없을 것임)

(10) Reward 만들기

```
Unity 스크립트 | 참조 2개
public abstract class Reward : ScriptableObject
{
    [SerializeField] private Sprite icon;
    [SerializeField] private string description;
    [SerializeField] private int quantity;

    참조 1개
    public Sprite Icon => icon;
    참조 3개
    public string Description => description;
    참조 3개
    public int Quantity => quantity;

    /// <summary>
    /// Grant the specified reward.
    /// The `Quest` parameter indicates which quest is granting the reward.
    /// </summary>
    /// <param name="quest"></param>
    참조 1개
    public abstract void Give(Quest quest);
}
```

해당 스크립트 상속해서 스크립터블 오브젝트로 만들고, Give를 오버라이딩 해서 원하는 보상 넣을 수 있도록 함

(11) 퀘스트 수주 조건 만들기(필수는 아님)

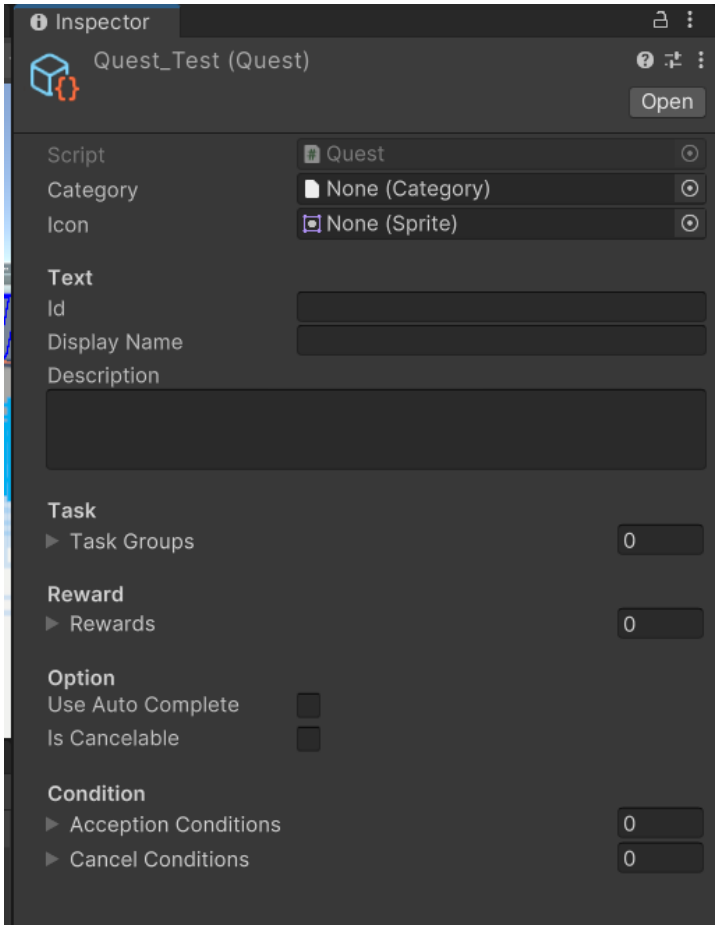
```
Unity 스크립트 | 참조 2개
public abstract class QuestAcceptionCondition : ScriptableObject
{
    [SerializeField] private string description;

    /// <summary>
    /// Prerequisites for starting or canceling a quest.
    /// Quest parameter indicates which quest is granting the condition.
    /// </summary>
    /// <param name="quest"></param>
    /// <returns></returns>
    참조 2개
    public abstract bool IsPass(Quest quest);
}
```

해당 스크립트 상속받아서 원하는 조건을 만들고, 스크립터블 오브젝트로 만들기

(12) 퀘스트 포기 조건 만들기(필수는 아님)

똑같이 위 스크립트를 상속받고 포기 조건 만들기

(13) 퀘스트 완성하기

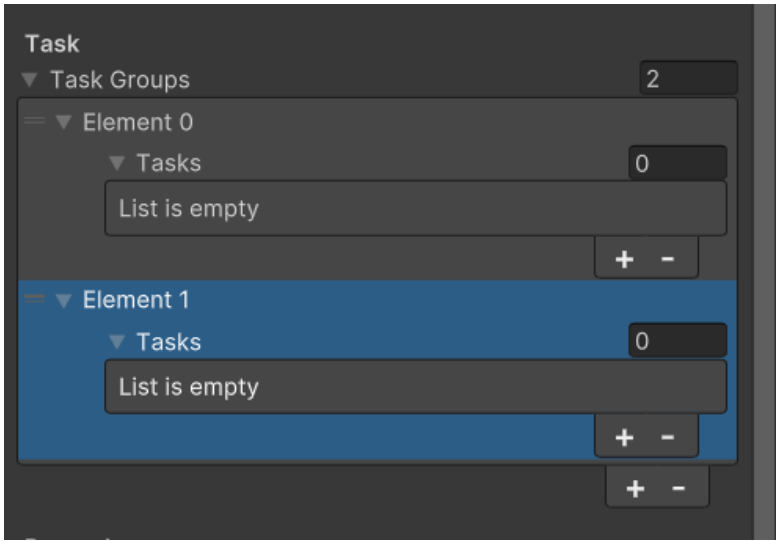


Use Auto Complete - 모든 태스크 완료시 자동으로 클리어 시킬 것인지

Is Cancelable – 취소가 가능한 퀘스트인지

Quest는 Task Group의 리스트를 가지고 있고,

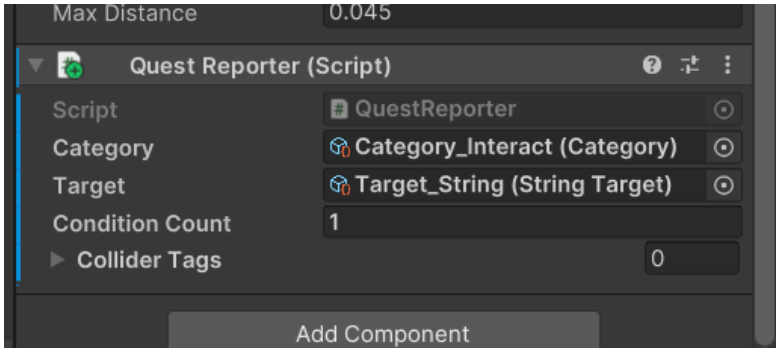
Task Group은 Task의 리스트를 가지고 있음



하나의 Task Group에 존재하는 모든 Task를 클리어하면, 다음 Task Group을 수행할 수 있게됨.

즉 Task Group은 순서대로 처리되고, 그 안의 Task들에는 순서가 없음

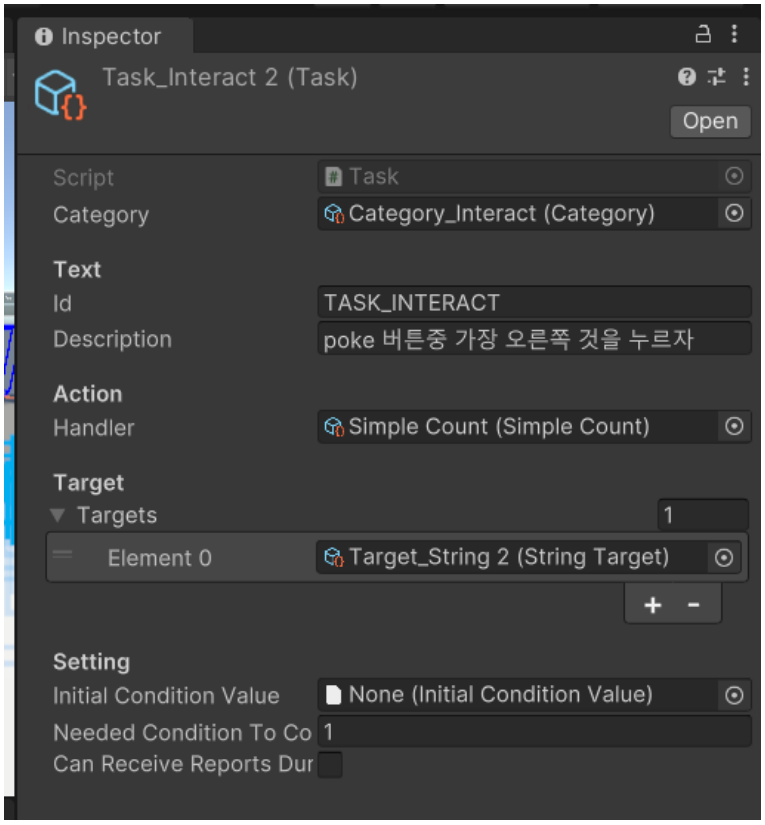
(14) Task의 타겟 설정하기



해당 스크립트를 타겟으로 하는 객체에다가 붙이기.

원하는 Category(어떤 행동으로 Task를 처리할 것인지)와 Target을 설정.

이때 Target은



내가 설정한 Task의 Target들 중에서 적어도 하나는 동일해야 해당 객체와 상호작용시, Task가 처리됨

Condition Count는 해당 타겟으로 인해 Task의 Handler가 작동했을 때, 얼마의 값을 기반으로 처리해줄지의 값임. 예를 들어서 Simple Count Handler이면 Condition Count = 3일 때

현재값 = 현재값 + 3 이 되는 것

Collider Tag는 해당 Tag에 맞는 콜라이더와 Trigger 처리시 Task가 처리된다는 소리. 만약 콜라이더 기능을 안쓸거면 무시해도 된다.

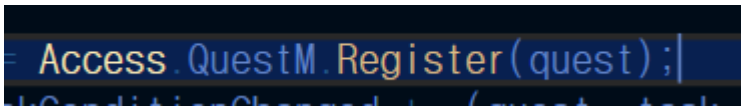
이후 해당 스크립트 내의

```
참조 1개
public void Report()
{
    Access.QuestM.ReceiveReport(category, target, conditionCount);
}
```

위 함수를 호출하면 Task가 처리됨.

만약 더 복잡한 방법으로 Task를 처리하고 싶다면 따로 스크립트를 작성해서 만들면 된다.

(15) Quest 제공하기



결론적으로는 위 방법으로 Quest 스크립터블 오브젝트를 등록시켜주면 된다.

인자는 Quest 스크립터블 오브젝트이다.

그저 상호작용으로 제공하고싶다면, Interactable 들의 이벤트에서 위 함수를 만들어서 호출해주면 된다.

만약 대화가 끝나면 넘겨주고싶다면, 대화 오브젝트에 퀘스트 수주 노드를 추가해주면 되는데, 이건 대화 문서에서 따로 작성하겠다.