게임 클라이언트 포트폴리오

윤창민

‘UnityStory’ 포트폴리오 영상 링크

https://www.youtube.com/watch?v=WEp17_Rq-ws&t

Unity Story

Unity 2020.3.25f1

Git : https://github.com/dlwlxns4/StudyUnity3DRPG

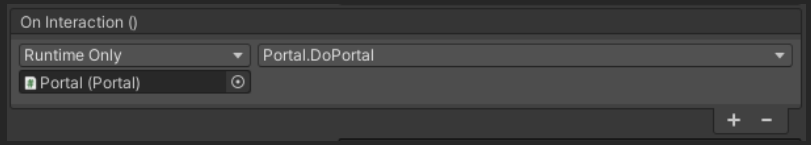
- 기간: 4주
- 인원: 1인

상호작용 오브젝트

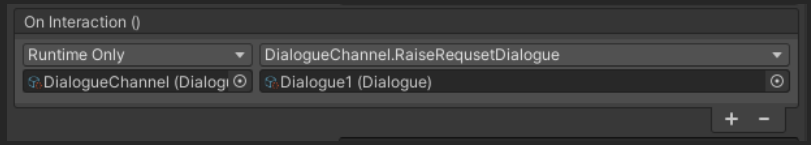
```
public class Interactable : MonoBehaviour
{
    [SerializeField]
    public UnityEvent OnInteraction;
    [SerializeField]
    private int objectID;

    public void DoInteraction()
    {
        OnInteraction.Invoke();
    }
}
```

UnityEvent의 OnInteraction에 이벤트를 추가하여 동작을 한다.

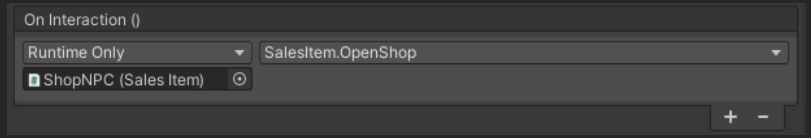
포탈



대화 NPC



상점



아이템

획득

```
public abstract void AcquireItem();

if(Vector3.Distance(this.transform.position, other.transform.position)<=0.8f)
{
    AcquireItem();
    Destroy(this.gameObject);
    break;
}
```

Abstract Class인 Item 클래스를 상속받아 아이템 획득 시 AcquireItem 함수를 호출

사용

```
public interface IUsable
{
    public void UseItem();
}
```

Interface IUsable을 통한 아이템 사용

아이템 획득

코인 아이템

```
public class Coin : Item
{
    [SerializeField]
    int minCoin;
    [SerializeField]
    int maxCoin;

    public override void AcquireItem()
    {
        int cash = Random.Range(minCoin, maxCoin+1);
        UIChannel.RaiseAccuireCoin(cash);
    }
}
```

기본 아이템

```
public class BasicItem : Item
{
    public override void AcquireItem()
    {
        UIChannel.RaiseGetUseItem(this, true);
    }
}
```

아이템 획득 시 서로 다른 기능의 AcquireItem 호출

아이템 사용

HP Potion

```
public class HpPotion : BasicItem, IUsable
{
    [SerializeField]
    private int increaseCount;

    public void UseItem()
    {
        PlayerChannel.RaiseUseItem(PlayerStatus.PlayerState.Hp, increaseCount);
        UIChannel.RaiseGetUseItem(this, false);
    }
}
```

MP Potion

```
public class MpPotion : BasicItem, IUsable
{
    [SerializeField]
    private int increaseCount;

    public void UseItem()
    {
        PlayerChannel.RaiseUseItem(PlayerStatus.PlayerState.Mp, increaseCount);
        UIChannel.RaiseGetUseItem(this, false);
    }
}
```

퀵 슬롯 - 아이템 등록

```
Vector2 size = new Vector2(45,35);
foreach(var quickSlotElement in quickSlotsList)
{
    Vector2 position = quickSlotElement.GetComponent<RectTransform>().position;
    if(position.x - size.x/2 < pointer.x && position.x + size.x/2 > pointer.x)
    {
        if(position.y - size.y/2 < pointer.y && position.y + size.y/2 > pointer.y)
        {
            quickSlotElement.SetData(itemSlot);
            break;
        }
    }
}
```

QuickSlot Class

인벤토리에서 드래그 이벤트가 끝날 시 위의 함수 실행

```
public void OnEndDrag(PointerEventData eventData)
{
    Destroy(shadowSlotImage.gameObject);
    UIChannel.RaiseSetQuickSlot(eventData.position, this);
}
```

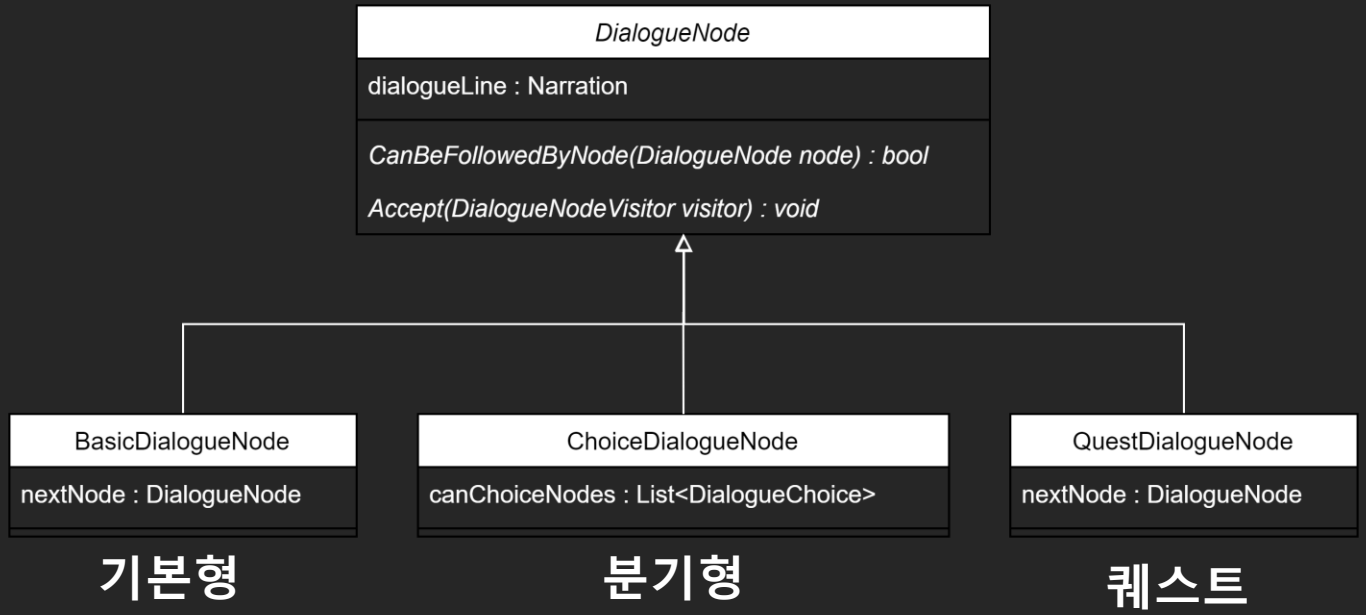
인벤토리

GetUseItem 메소드 일부

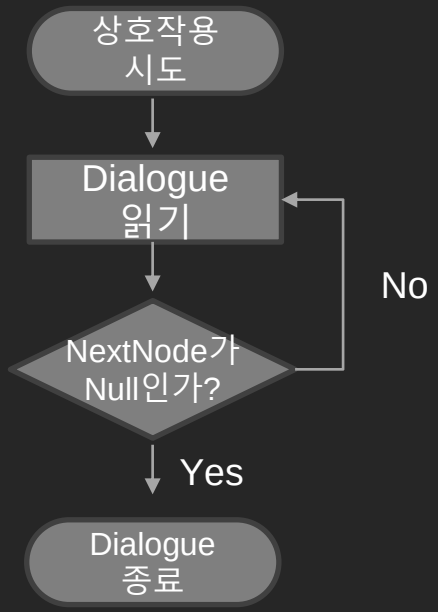
```
foreach(var itemSlot in itemList)
{
    if(itemSlot.itemData == null)
    {
        itemSlot.SetItemImage(itemData);
        return ;
    }
    else if(itemSlot.itemData.GetItemData.ItemCode
                ==itemData.GetItemData.ItemCode )
    {
        itemSlot.IncreaseCount();
        return ;
    }
}

UIChannel.OnGetUseItem += GetUseItem;
```

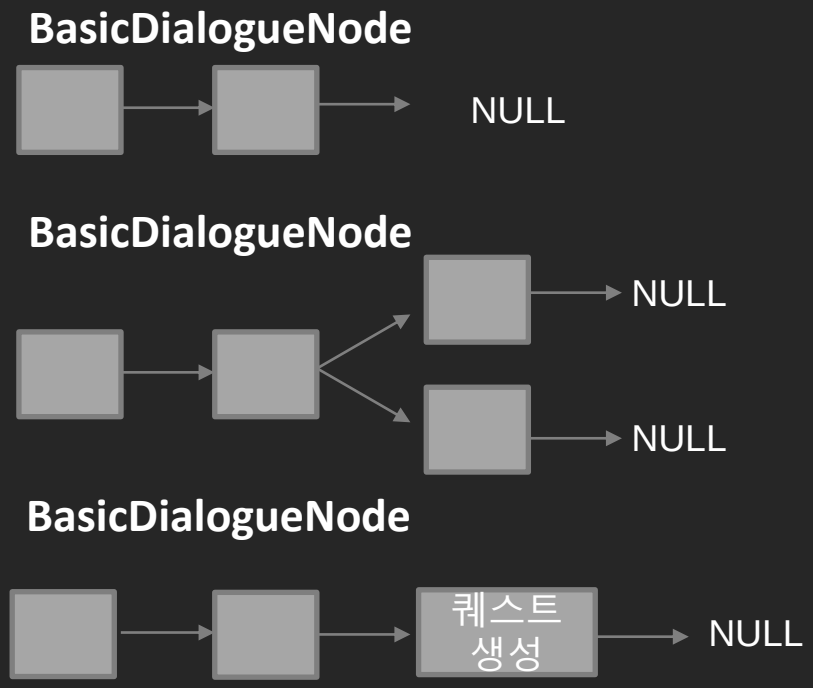
아이템 획득 시 콜백 함수 실행
사용시 제거 및 카운트 감소도 동일

대화 시스템 – DialogueNode 종류



대화 시스템 – 흐름


상호작용
시도
Dialogue
읽기
NextNode가
Null인가?
No
Yes
Dialogue
종료


BasicDialogueNode
NULL
BasicDialogueNode
NULL
NULL
BasicDialogueNode
퀘스트
생성
NULL

대화 시스템

```
public delegate void DialogueCallBack(Dialogue dialogue);
public DialogueCallBack OnDialogueRequsted; -> Dialogue체크
public DialogueCallBack OnDialogueStart; -> UI 활성화
public DialogueCallBack OnDialogueEnd;  -> UI 비활성화

public delegate void DialogueNodeCallBack(DialogueNode node);
public DialogueNodeCallBack OnDialogueNodeStart; -> DialogueNode 읽기
public DialogueNodeCallBack OnDialogueNodeEnd; -> DialogueNode 종료
public DialogueNodeCallBack OnDialogueNodeRequested; -> NextNode 요청
```

대화 시스템은 콜백 함수를 통하여 실행

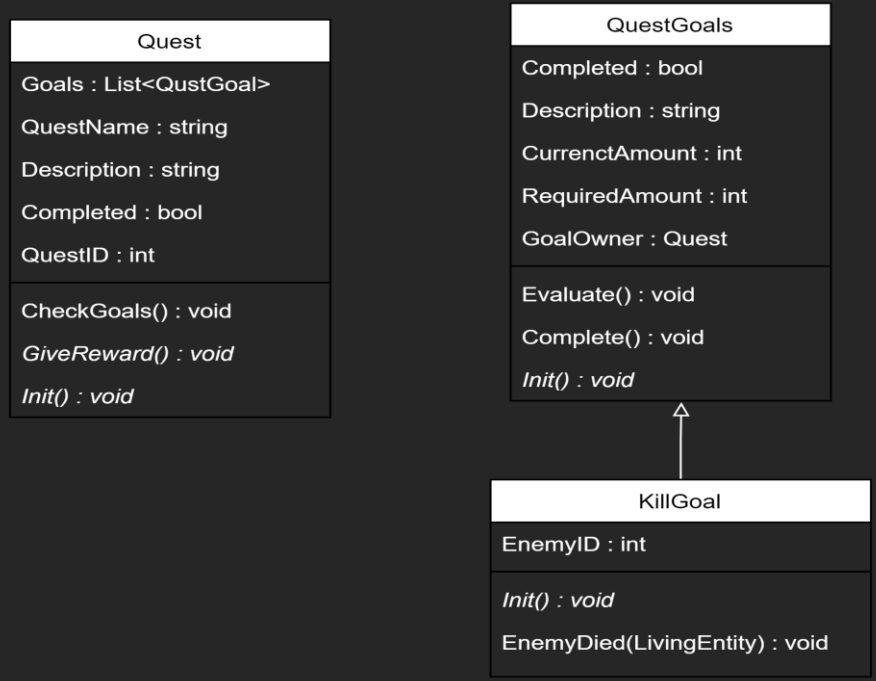
대화 시스템 – Scriptable Object Maker

```
for(int i=1; i<allLines.Length; ++i)
{
    string[] splitData = allLines[i].Split('\n');

    if(splitData.Length != 1)
    {
        return ;
    }

    NarrationCharacter character = ScriptableObject.CreateInstance<NarrationCharacter>();

    character.characterName = splitData[0]; // 이름

    AssetDatabase.CreateAsset(character,
        $"Assets/ScriptableObjects/Narration/NarrationCharacter_{character.characterName}.asset");
}
```

	A	B	C	D
1	이름			
2	유니티 쨩			
3	담묘			

CSV파일 읽기를 통한 ScriptableObject Maker

퀘스트




```
public override void Init()
{
    base.Init();
    CombatChannel.OnEnemyDiedEvent += EnemyDied;
}

void EnemyDied(LivingEntity enemy)
{
    if(enemy.ObjectId == this.EnemyID)
    {
        this.CurrentAmount++;
        UIChannel.RaiseSetQuestInformation(GoalOwner);
        Evalutate();
    }
}
```

몬스터의 Die호출 시 EnemyDied가 호출이 되어 목표 조건 체크를 한다.

몬스터 - 애니메이션

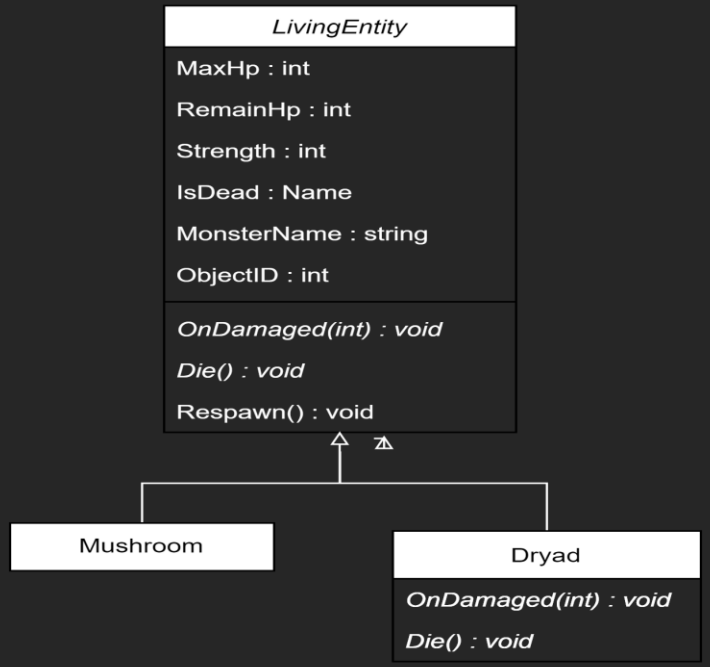
StateMachineBehaviour를 사용

모든 몬스터는 각 동작에 맞는 Interface호출하여 Interface만 정의해주면 된다.

```
public interface IAttackable
{
    public void AttackReady();
    public void Attack();
    public void AttackExit();
}
```

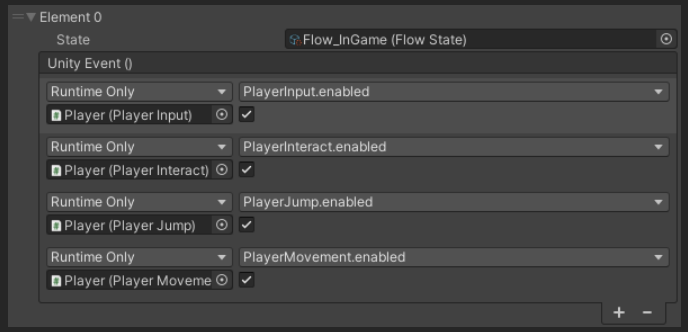
```
public interface IMovable
{

    public void Move();
    public void CanMove();
    public void Chase();
    public void GoHome(Vector3 homePos);
}
```

몬스터 - LivingEntity

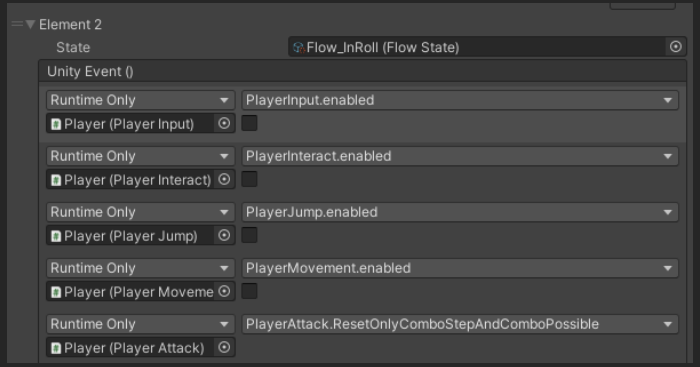


몬스터는 기본 상태를 정의하는 LivingEntity의 추상 클래스를 상속하여 구현

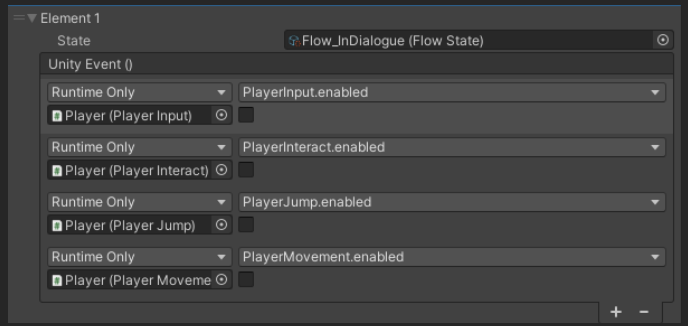
플레이어 – 동작 상태



게임 상태



구르기 상태



대화 상태

상태 변화시 UnityEvent를 통한
Script Enable을 관리하여 동작 관리

연출

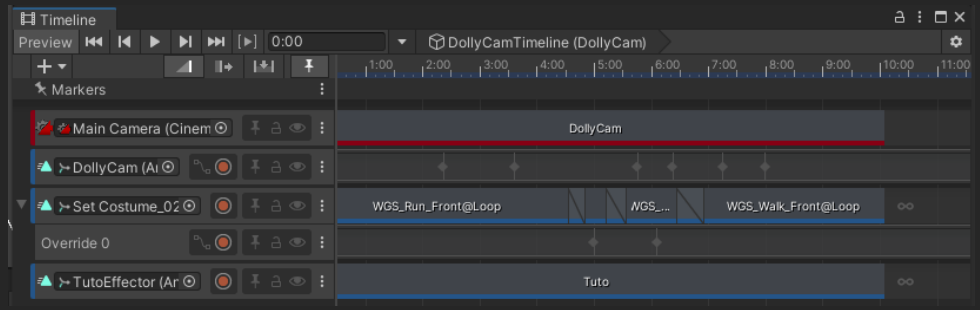
```
public abstract class ConditionBoundary : MonoBehaviour
{

    public abstract void ActionEvent();

    void OnTriggerEnter(Collider other)
    {
        ActionEvent();
    }
}
```

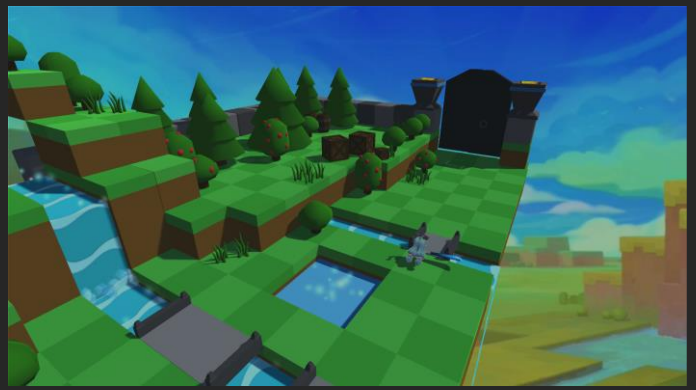
Trigger시 활성화 되도록 하는
오브젝트를 통하여 연출 구현

마을

보스

연출 – 게임 시작 연출





시네머신 카메라의 돌리트랙과
타임라인을 통한 연출 씬 구현

몬스터 스폰 - 오브젝트 풀링


```
void InitMonster()
{
    for(int i=0; i<spawnMonsterCount; ++i)
    {
        GameObject monster = Instantiate(monsterPrefab, Vector3.zero, Quaternion.identity);
        monster.SetActive(false);
        monsterList.Add(monster);
    }
}
```

몬스터를 미리 생성한 후 관리

감사합니다