**ObjectPooling 개요**

**ObjectPool.cs**

ObjectPool 클래스는 Unity 2021의 빌트인 ObjectPool의 기능을 직접 구현해보고자 그 아이디어 일부를 계승하여 작성했으며 다양한 타입의 오브젝트에 대응할 수 있도록 Generic 클래스로 작성되었다.

public ObjectPool(Func<T> \_instantiate\_func, Action<T> \_OnGetAction,

Action<T> \_OnReleaseAction, Action<T> \_OnDestoryAction, Action<Queue<T>> \_OnCleanUpAction, int \_maxSize)

{

readyPool = new Queue<T>(); //실제 Pool컨테이너에 해당하는 queue 초기화

instantiate\_func = \_instantiate\_func; //Object 생성함수 콜백 바인드

OnGetAction = \_OnGetAction; //GetObject 액션 콜백 바인드

OnReleaseAction = \_OnReleaseAction; //ReleaseObject 액션 콜백 바인드

OnDestoryAction = \_OnDestoryAction; //Object Destroy 액션 콜백 바인드

OnCleanUpAction = \_OnCleanUpAction; // Pool CleanUp 액션 콜백 바인드

maxSize = \_maxSize; //ObjectPool 최대 수용량 (현재 pool size는 readyPool.Count + activeCount로 계산)

}

ObjectPool 클래스 생성자

Instantiate\_func : 프리팹 생성 함수이며 Object 생성 동작시 어떤 프리팹을 생성하고 어떻게 초기화 시킬것인지를 외부 콜백에 일임한다.

OnGetAction : GetObject 동작 시 어떤 식으로 Object가 활용 / 재활용될 것인지를 외부 콜백에 일임한다.

OnReleaseAction : GetRelease 동작 시 어떤 식으로 Object가 readyPool에 다시

반환될 것인지를 외부 콜백에 일임한다.

OnDestroyAction : Object 파괴 동작 시 어떤 식으로 Object가 파괴될 것인지를 외부 콜백에 일임한다.

OnCleanUpAction : 현재 readyPool 컨테이너에 대한 CleanUp 동작 시 어떤 식으로 readyPool 을 비울것인지를 외부 콜백에 일임한다.

maxSize : readyPool 에서 수용 가능한 최대 용량을 설정한다.

GetObject():

public T GetObject()

{

//ObjectPool에서 생성된 Object의 수가 아직 최대 용량보다 적은 상황이거나 꺼내서 쓸 수 있는 Object가 더 이상 존재하지 않을때

if (readyPool.Count + activeCount < maxSize || readyPool.Count == 0)

{

//Object 생성 콜백 호출

var instance = instantiate\_func();

//GetObject 액션 콜백 호출

OnGetAction(instance);

++activeCount;

return instance;

}

else

{

//readyPool에 회수되어있는 Object를 Dequeue해서 재활용

var instance = readyPool.Dequeue();

OnGetAction(instance);

++activeCount;

return instance;

}

}

If(readyPool.Count + activeCount < maxSize || readyPool.Count == 0)

위 조건문이 True인 경우

위 조건문은 현재 비활성화된 readyPool 내의 Object와 활성화되어 아직 회수되지 않은 Object의 수의 합, 즉 ObjectPool에서 생성된 Object의 수가 아직 최대 용량보다 적은 상황이거나 readyPool에서 꺼내서 쓸 수 있는 Object가 더 이상 존재하지 않는다는 의미이다.

따라서 더 수용이 가능한 상황이면 Object를 더 생성해내야 하고, 재활용 가능한 Object가 더 이상 존재하지 않더라도 Object를 더 생성해내야 하므로

instantiate\_func()를 통해 외부에서 바인드되어 있는 Object 생성함수를 호출하고 OnGetAction(instance)를 통해 GetObject가 실행될 시 수행되어야 하는 콜백을 호출하고 생성된 Object를 리턴한다.

조건문이 False인 경우

ObjectPool에서 생성한 Object의 수가 최대 용량을 채운 상태이며 readyPool에 현재 재활용 가능한 Object 여분이 존재한다는 의미이므로 readyPool에서 비활성화 되어있는 Object를 dequeue하고 GetObject가 실행될 시 수행되어야 하는 콜백을 호출한 후 Object를 리턴한다.

프리팹 생성 함수, 오브젝트 Get 이벤트 콜백, 오브젝트 Release 이벤트 콜백, 오브젝트 Destroy 이벤트 콜백, Clean 이벤트 콜백을 외부에서 바인드 받아 오브젝트 풀의 GetObject, ReleaseObject 등의 각 동작마다의 작동 세부 로직을 외부에서 받도록 구현했습니다.

Generic 클래스로 구현되어 다양한 타입에 대응할 수 있는 ObjectPool로, 사용자는PoolingManager를 통해  pool id를 부여받아 해당 id에 해당하는 ObjectPool을 사용할 수 있다.

ReleaseObject():

public void ReleaseObject(T instance)

{

//ObjectPool에서 생성된 Object의 수가 maxSize를 초과하지 않았을 경우

if (readyPool.Count + activeCount <= maxSize)

{

//ReleaseObject 액션 콜백 호출

OnReleaseAction(instance);

//해당 instance를 다시 readyPool에 회수

readyPool.Enqueue(instance);

--activeCount;

}

else

{

//생성된 Obejct 수가 maxSize를 초과했을 경우

/\*

\* TODO: 외부 요인으로 destroy 되었을때 예외처리 필요

\*/

//해당 instance 삭제

OnDestoryAction(instance);

--activeCount;

}

}

If(readyPool.Count + activeCount <= maxSize)

조건문이 True인 경우

생성된 Object의 수가 최대 용량을 초과하지 않았으므로 ReleaseObject시 실행되어야 할 콜백을 호출한 후 해당 Object를 다시 readyPool에 회수한다.

조건문이 False인 경우

생성된 Object의 수가 최대 용량을 초과했으므로 해당 Obejct에 대한 Destroy 콜백을 호출한다.

**PoolingManager.cs**

GetPoolID():

public string GetPoolID(string id)

{

//같은 스크립트로 부터 요청되는 ObjectPool의 ID 중복을 방지하기 위한 대책

//요청된 ID가 idMap에 존재하지 않을경우

if (idMap.ContainsKey(id) == false)

{

//ID에 번호 0번을 부여 후 반환

string newKey = id + "0";

//해당 ID를 idMap에 등록하고 그 value값은 다음에 같은 ID로 요청이 들어올 시에 할당해야할 번호를 의미함

idMap[id] = 1;

return newKey;

}

else

{

//요청된 ID가 idMap에 존재할 경우 현재 value 번호를 부여해서 반환

string newKey = id + idMap[id].ToString();

//다음에 부여할 번호 value 증가

idMap[id]++;

return newKey;

}

}

서로 다른 오브젝트의 같은 스크립트에서 poolID를 요청할 경우 ID의 중복을 막기 위한 대책으로써 idMap을 활용하여 인자로 요청된 id가 idMap에 존재 하는지 여부를 분기로 동작한다.

1. 요청된 id가 idMap에 존재하지 않을 경우idMap에 id : 1을 등록한 후 ( key : val, 여기서 1은 다음에 같은 id로 요청이 들어올 시에 할당할 번호를 의미함) id + “0”을 반환한다.
2. 요청된 id가 idMap에 존재할 경우에는 해당 id에 해당하는 value가 현재 할당해줘야 할 번호를 의미하므로 id + idMap[id]를 반환하고 idMap++을 수행한다.

AddObjectPool():

public void AddObjectPool<T>(string id, Func<T> \_instantiate\_func, Action<T> \_OnGetAction,

Action<T> \_OnReleaseAction, Action<T> \_OnDestoryAction, Action<Queue<T>> \_OnCleanUpAction, int \_maxSize)

{

//해당 ID에 해당하는 ObjectPool이 poolMap에 존재하지 않을 경우

if (idMap.ContainsKey(id) == false)

{

//인자로 전달받은 Func, Action을 통해 새로운 ObjectPool 생성자를 호출하고 poolMap에 등록

poolMap.Add(id, new ObjectPool<T>(\_instantiate\_func, \_OnGetAction, \_OnReleaseAction,

\_OnDestoryAction, \_OnCleanUpAction ,\_maxSize));

}

}

인자로 요청된 id가 poolMap에 존재하지 않을 경우에만 동작하며 인자로 전달받은 Func, Action 콜백을 생성자로 전달하여 ObjectPool을 생성하고 요청된 id를 key값으로, 생성된 ObjectPool을 value값으로 poolMap에 새로 등록한다.

GetObjectPool():

public ObjectPool<T> GetObjectPool<T>(string id)

{

//해당 ID에 해당하는 ObjectPool 반환

if (poolMap.ContainsKey(id))

return (ObjectPool <T>)poolMap[id];

//존재하지 않을경우 null반환

return null;

}

poolMap에서 id에 매치되는 ObjectPool를 반환하며 해당 id가 존재하지 않을 경우 null을 반환한다.

**ObjectPool 활용**

**Gun.cs**

private void InitBulletPool()

{

bulletPoolKey = PoolingManager.GetInstance().GetPoolID("bulletPool");

PoolingManager.GetInstance().AddObjectPool<Projectile>(

//PoolingManager로부터 할당받은 ObjectPool ID

bulletPoolKey,

//Projectile 생성 콜백 함수

\_instantiate\_func: () =>

{

//Projectile Object 생성

var instance = Instantiate<Projectile>(currBulletPrefab, firePos.position, firePos.rotation);

//Projectile Object이 ReleaseObject 및 GetObject 목적으로 사용할 ObjectPool의 ID 인계

instance.bulletPoolKey = this.bulletPoolKey;

instance.hitParticlePoolKey = hitParticlePoolKey;

instance.damage = damage;

//Projectile의 소유자를 식별할 수 있는 Player SessionID 인계

if (playerSessionID == BackEnd.Tcp.SessionId.None)

Debug.LogError("PlayerSessionID was not Initialized (Gun)");

instance.shooterSessionID = playerSessionID;

return instance;

},

//GetObject 콜백 함수

\_OnGetAction: (bullet) =>

{

//비활성화 된 Object 재활성화

bullet.gameObject.SetActive(true);

//위치를 총구 발사 위치로 재조정

bullet.ResetPos(firePos.position, currTargetPos);

},

//ReleaseObject 콜백 함수

\_OnReleaseAction: (bullet) =>

{

//현재 활성화된 Object 비활성화 (재활용 목적)

bullet.gameObject.SetActive(false);

},

\_OnDestoryAction: (bullet) =>

{

//현재 활성화된 Object 삭제 (삭제 목적)

Destroy(bullet.gameObject);

},

\_OnCleanUpAction: (pool) =>

{

foreach (Projectile p in pool)

Destroy(p.gameObject);

},

maxPoolSize //maxSize

);

}

ObjectPool을 사용하고자 하는 스크립트에서 PoolingManager에 GetPoolID를 통해 희망하는 PoolID에 추가로 인덱스 넘버를 부여받고 이렇게 부여받은 새 id가 해당 스크립트에서 사용할 ObjectPool의 ID가 된다.

할당받은 id와 함께 각각의 콜백 함수를 작성하여 PoolingManager에게 AddObjectPool을 요청하여 ObjectPool을 생성하며 해당 ObjectPool을 사용하려면 PoolingManager에게 GetObjectPool(id)를 요청하여 ObjectPool의 인스턴스를 반환 받게된다.