게임 제작에 앞서 프레임워크들부터 제작하기로 했다. 기본적인 싱글톤, 오브젝트풀, 리소스매니저들 관련 기능 등 어떤 게임을 만들더라도 쓸 수 있는 기능들만 만들었다. 본격적인 게임 프레임워크는 1인개발이라 안쓰지 않을까 싶다..

먼저 싱글톤. 싱글톤은 일반적인 제너릭 싱글톤 클래스에 첫 인스턴스 생성 시에 리소스폴더에서 프리팹 불러와서 생성하는 부분만 추가했다.

public class MonoBehaviourSingleton<T> : MonoBehaviour where T : MonoBehaviour

{

private static T instance;

public static T Instance

{

get

{

if (instance == null)

{

//이미 있는 싱글톤클래스 받아오기.

T[] objs = FindObjectsOfType<T>();

if (objs.Length > 1)

{

DebugLog.LogError("There are Singleotones more than 1" + typeof(T).Name);

}

if (objs.Length >= 1)

{

instance = objs[0];

}

else

{

//싱글톤 클래스 프리팹 있을경우 생성.

GameObject gObj = Resources.Load<GameObject>(rootPath + typeof(T).Name);

if (gObj != null)

{

instance = Instantiate(gObj).GetComponent<T>();

}

//없으면 새로운 오브젝트로 생성.

if (instance == null)

{

instance = new GameObject().AddComponent<T>();

instance.gameObject.name = typeof(T).Name;

}

}

}

return instance;

}

}

const string rootPath = "Singleton/";

protected virtual void Awake()

{

DontDestroyOnLoad(gameObject);

}

}

다음으로 오브젝트 풀. 오브젝트 풀 기본적인 구조는 <https://github.com/thefuntastic/unity-object-pool> 에서 가져와서 내가 필요한 기능들을 추가하는 방식으로 구현했다.

ObjectPool.cs

public class ObjectPool<T> where T : class

{

private List<ObjectPoolContainer<T>> list;

private Dictionary<T, ObjectPoolContainer<T>> lookup;

private Func<T> factoryFunc;

private Action<T> destroyFunc;

public T Original { get; private set; }

private int lastIndex = 0;

public ObjectPool(Func<T> factoryFunc, Action<T> destroyFunc, int initialSize, T origin = null)

{

this.factoryFunc = factoryFunc;

this.destroyFunc = destroyFunc;

this.Original = origin;

list = new List<ObjectPoolContainer<T>>(initialSize);

lookup = new Dictionary<T, ObjectPoolContainer<T>>(initialSize);

Warm(initialSize);

}

public void Warm(int capacity)

{

for (int i = 0; i < capacity; i++)

{

CreateContainer();

}

}

public void DestoryPool()

{

while (list.Count != 0)

{

var item = list[0].Item;

if (lookup.ContainsKey(item))

{

lookup.Remove(item);

}

destroyFunc?.Invoke(item);

list.RemoveAt(0);

}

}

private ObjectPoolContainer<T> CreateContainer()

{

var container = new ObjectPoolContainer<T>();

container.Item = factoryFunc();

list.Add(container);

return container;

}

public T GetItem()

{

ObjectPoolContainer<T> container = null;

for (int i = 0; i < list.Count; i++)

{

lastIndex++;

if (lastIndex > list.Count - 1) lastIndex = 0;

if (list[lastIndex].Used)

{

continue;

}

else

{

container = list[lastIndex];

break;

}

}

if (container == null)

{

container = CreateContainer();

}

container.Consume();

lookup.Add(container.Item, container);

return container.Item;

}

public void ReleaseItem(object item)

{

ReleaseItem((T)item);

}

public void ReleaseItem(T item)

{

if (lookup.ContainsKey(item))

{

var container = lookup[item];

container.Release();

lookup.Remove(item);

}

else

{

Debug.LogWarning("This object pool does not contain the item provided: " + item);

}

}

public int Count

{

get { return list.Count; }

}

public int CountUsedItems

{

get { return lookup.Count; }

}

}

ObjectPool.cs에서 내가 수정한 부분은 크게 보면 두가지

public T Original { get; private set; }

첫번째는 오브젝트 풀의 인스턴스를 만들기위한 원본 Original 변수 추가한 것 (프리팹을 지니는 용도로 사용)

public void DestoryPool()

{

while (list.Count != 0)

{

var item = list[0].Item;

if (lookup.ContainsKey(item))

{

lookup.Remove(item);

}

destroyFunc?.Invoke(item);

list.RemoveAt(0);

}

}

두번째는 이미 생성된 오브젝트 풀을 제거하는 기능을 추가. 관련되어 오브젝트 풀 생성자 부분에서 destroyFunc 도 받아오게 해놨음. 오브젝트 풀을 지우는 기능은 안쓰지 않을까 싶은데, 일단 어느 게임에서든 쓸 수 있는 프레임워크를 만들자가 목표라 혹시 몰라 넣어둠. 예를 들면 1스테이지 보스만 사용하는 총알을 ObjectPooling 한 후에, 다시는 그 스테이지에 들어갈 일이 없을거라 판단되면 생성된 총알 ObjectPool을 지운다던가 하는 식으로 사용하지 않을까 싶다.

ObjectPoolContainer.cs

public class ObjectPoolContainer<T>

{

private T item;

public T Item

{

get

{

return item;

}

set

{

item = value;

}

}

public bool Used { get; private set; }

public void Consume()

{

Used = true;

}

public void Release()

{

Used = false;

}

}

여기는 크게 바뀌는 부분 없이 단순히 item 들고있는 ObjectPool 용 Container 클래스

ObjectPoolManager.cs

public class ObjectPoolManager : MonoBehaviourSingleton<ObjectPoolManager>

{

private Dictionary<GameObject, ObjectPool<GameObject>> prefabLookup;

private Dictionary<GameObject, ObjectPool<GameObject>> instanceLookup;

private Dictionary<GameObject, Transform> parentLookup;

protected override void Awake()

{

base.Awake();

prefabLookup = new Dictionary<GameObject, ObjectPool<GameObject>>();

instanceLookup = new Dictionary<GameObject, ObjectPool<GameObject>>();

parentLookup = new Dictionary<GameObject, Transform>();

}

public void WarmPool(GameObject prefab, int size)

{

ObjectPool<GameObject> pool;

if (prefabLookup.ContainsKey(prefab))

{

pool = prefabLookup[prefab];

if (pool.Count < size)

{

pool.Warm(pool.Count - size);

}

DebugLog.Log("Pool for prefab " + prefab.name + " has already been created");

}

else

{

pool = new ObjectPool<GameObject>(() => { return InstantiatePrefab(prefab); }, DestroyObject, size, prefab);

}

prefabLookup[prefab] = pool;

}

public void DestroyPool(GameObject prefab)

{

if (prefabLookup.ContainsKey(prefab))

{

var pool = prefabLookup[prefab];

pool.DestoryPool();

prefabLookup.Remove(prefab);

}

//부모오브젝트 제거.

if (parentLookup.ContainsKey(prefab))

{

Destroy(parentLookup[prefab]);

parentLookup.Remove(prefab);

}

}

public GameObject SpawnObject(GameObject prefab)

{

return SpawnObject(prefab, Vector3.zero, Quaternion.identity);

}

public GameObject SpawnObject(GameObject prefab, Vector3 position, Quaternion rotation)

{

if (!prefabLookup.ContainsKey(prefab))

{

WarmPool(prefab, 1);

}

var pool = prefabLookup[prefab];

var clone = pool.GetItem();

clone.transform.SetPositionAndRotation(position, rotation);

clone.SetActive(true);

instanceLookup.Add(clone, pool);

return clone;

}

public void ReleaseObject(GameObject clone)

{

clone.SetActive(false);

if (instanceLookup.ContainsKey(clone))

{

var pool = instanceLookup[clone];

//반환시 오브젝트 풀 부모로 재설정.

if (parentLookup.ContainsKey(pool.Original))

{

clone.transform.parent = parentLookup[pool.Original];

}

//스케일 원본으로 변경.

if (pool.Original != null)

{

clone.transform.localScale = pool.Original.transform.localScale;

}

pool.ReleaseItem(clone);

instanceLookup.Remove(clone);

}

else

{

Debug.LogWarning("No pool contains the object: " + clone.name);

}

}

private GameObject InstantiatePrefab(GameObject prefab)

{

var go = Instantiate(prefab) as GameObject;

//부모 찾기.

Transform parent = transform;

if (parentLookup.ContainsKey(prefab))

{

parent = parentLookup[prefab];

}

else

{//부모없으면 새로생성.

parent = new GameObject().transform;

parent.parent = transform;

parent.name = prefab.name;

parentLookup.Add(prefab, parent);

}

//부모설정 후 Deactive.

go.transform.parent = parent;

go.SetActive(false);

return go;

}

private void DestroyObject(GameObject clone)

{

if (instanceLookup.ContainsKey(clone))

{

instanceLookup.Remove(clone);

}

Destroy(clone);

}

}

실질적으로 게임오브젝트 프리팹들을 제공하여 ObjectPooling을 할 수 있는 싱글톤 매니저 클래스. 대다수의 게임오브젝트들이 이 매니저를 통해 풀링될 듯 하다. 수정된 부분은 ObjectPool.cs 와 마찬가지로 Pool과 Object Destory 되는 부분 및 원본 Prefab 넣어주는 부분이다

그리고 각 프리팹별로 Parent 만들어서 지금 어떤 풀이 생성됐고, 얼마나 사용중인지 대략적으로라도 파악하기 위해 parentLookup 관련 기능 추가했다.

실제 사용하는 부분에선

var clone = ObjectPoolManager.Instance.SpawnObject(prefab);

activeEffectList.Add(clone);

clone.transform.parent = parent;

clone.transform.localPosition = localPosition;

처럼 ObjectPoolManager에서 오브젝트를 Spawn 하고, 다 썼으면

ObjectPoolManager.Instance.ReleaseObject(clone);

처럼 반환하면 된다.

혹은 게임오브젝트 풀이 아닌 별도의 풀을 사용하고 싶으면,

ObjectPool<AudioSource> sfxSourcePool;

이나

ObjectPool<Bullet> bulletPool;

처럼 오브젝트 풀을 별도로 선언해서 활용 가능하다.

솔직히 어디에나 쓸 수 있게끔 최대한 확장한답시고 해봤는데, 어차피 각 프로젝트마다 맞춰서 조금조금씩 수정해가면서 쓸 것 같다.