게임 개발하면서 사운드, 이펙트, 이미지, 텍스트 등 여러가지 리소스 관리할 일이 많은데, 이런 리소스 관리를 간단하게 할 수 있는 스크립트들을 만들어봤다.

원래 예전에는 Resources 폴더에 다 두고 CSV파일 등을 통해 파싱해서 리소스매니저를 쓰곤 했는데 매번 데이터 파일 고치기도 귀찮고… 그렇다고 리소스매니저 없이 사용하는 측 인스펙터에 그때그때 넣어주자니 뭐 수정 할 때 마다 어디서 썼는지 찾아가면서 수정하기도 귀찮고…

마침 이번에 애셋 할인하는 김에 OdinInspector를 구매했는데, Dictionary 를 Serialize 하는 기능이 있다고 해서 이 기능을 이용해서 리소스 매니저를 만들어 봤다.

아래 코드들은 OdinInspector랑 DOTween (사운드매니저용) 있어야 돌아가니 감안하고 봐주시기 바란다.

ResourceDataBase.cs

public class ResourceDatabase<T> : SerializedScriptableObject where T : class

{

public Dictionary<int, T> dictionary= new Dictionary<int, T>();

public virtual T GetItem(int id)

{

if (dictionary == null)

{

DebugLog.LogError(name + " DataBase is NULL ");

return null;

}

T retVal = null;

if (dictionary.ContainsKey(id))

{

retVal = dictionary[id];

}

else

{

DebugLog.LogError(name + " DataBase Wrong ID : " + id);

}

return retVal;

}

}

먼저 리소스데이터의 부모클래스. 관리할 리소스들을 제너릭으로 해뒀고 단순히 Dictionary를 들고 있는 ScriptableObject 임. 이걸 필요한 리소스마다 상속받아서 아래와 같이 사용한다.

[CreateAssetMenu(fileName = "PrefabDB", menuName = "ScriptableObjects/PrefabDB", order = 1)]

public class PrefabResourceDB : ResourceDatabase<GameObject>

{

}

[CreateAssetMenu(fileName = "AudioClipDB", menuName = "ScriptableObjects/AudioClipDB", order = 1)]

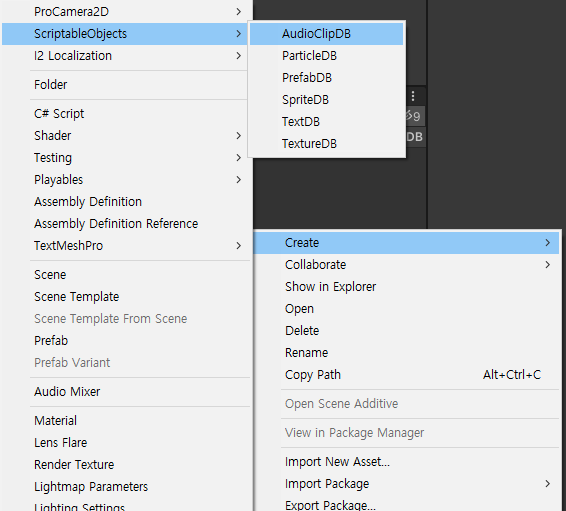
public class AudioClipResourceDB : ResourceDatabase<AudioClip>

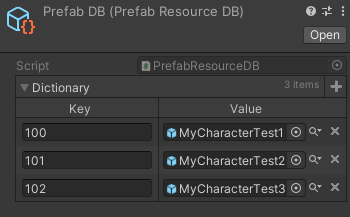
{

}

보면 알겠지만 구현부가 텅텅빈 단순히 Generic 상속 클래스랑 ScriptableObject를 만들어 줄 수 있는 메뉴 Attribute만 붙어있는 클래스들임. Prefab ,AudioClip, Texture, Particle 등 필요한 타입으로 만들어다가 쓰면 된다. 나 같은 경우엔 주로 Prefab, AudioClip, EffectPrefab 으로 사용하고, 혹시몰라서 Sprite, Texture2D, Text 등도 만들어 두긴 했다 .

이렇게되면 애셋메뉴에서 이런식으로 각 DB들을 만들 수 있고, 각 DB들은 리소스의 ID값과 리소스를 지니게 된다.





실제 사용하는 쪽에선

[SerializeField]

AudioClipResourceDB audioClipDB;

와 같이 각 Resource DB를 선언한 후 만들어 둔 ScriptableObject를 로드해다가

audioClipDB.GetItem(id);

처럼 아이템을 받아올 수 있다.

이 기능들을 이용해서 실제로 가장 많이 쓰이게 될 EffectManager랑 SoundManager를 만들었다.

먼저 EffectManager.cs

public class EffectManager : MonoBehaviourSingleton<EffectManager>

{

[SerializeField]

PrefabResourceDB effectPrefabDB;

List<GameObject> activeEffectList = new List<GameObject>();

protected override void Awake()

{

base.Awake();

}

public void PlayOneShot(int id, Vector3 position)

{

PlayOneShot(id, null, position);

}

public void PlayOneShot(int id, Transform parent, Vector3 localPosition)

{

PlayEffect(id, parent, localPosition, -1f);

}

public void PlayTimer(int id, Vector3 position, float timer)

{

PlayTimer(id, null, position, timer);

}

public void PlayTimer(int id, Transform parent, Vector3 localPosition, float timer)

{

PlayEffect(id, parent, localPosition, timer);

}

void PlayEffect(int id, Transform parent, Vector3 localPosition, float timer = -1f)

{

var clone = SpawnEffect(id, parent, localPosition);

if (clone == null) { return; }

var particle = clone.GetComponent<ParticleSystem>();

if (particle == null)

{//파티클 없는 오브젝트면 GameObjectTimer로.

DebugLog.LogWarning("Effect Manager Play One Shot Not Particle System - ID : " + id);

StartCoroutine(PlayGameObjectTimerCoroutine(clone, timer < 0f ? 30f : timer));

}

else

{//파티클 있으면 파티클 관련 코루틴실행.

if (timer < 0f)

{

StartCoroutine(PlayOneShotCoroutine(particle));

}

else

{

StartCoroutine(PlayTimerCoroutine(particle, timer));

}

}

}

IEnumerator PlayOneShotCoroutine(ParticleSystem particle)

{

particle.Play(true);

while (particle.IsAlive(true))

{

yield return null;

}

ReleaseEffect(particle.gameObject);

}

IEnumerator PlayTimerCoroutine(ParticleSystem particle, float timer)

{

particle.Play();

while (particle.isPlaying && timer > 0f)

{

yield return null;

timer -= Time.deltaTime;

}

particle.Stop(true);

while (particle.IsAlive(true))

{

yield return null;

}

ReleaseEffect(particle.gameObject);

}

IEnumerator PlayGameObjectTimerCoroutine(GameObject obj, float timer = 30f)

{

while (obj.activeSelf && timer > 0f)

{

yield return null;

timer -= Time.deltaTime;

}

ReleaseEffect(obj);

}

public GameObject SpawnEffect(int id, Transform parent, Vector3 localPosition)

{

if (effectPrefabDB == null)

{

effectPrefabDB = Resources.Load<PrefabResourceDB>("ResourceDB/EffectPrefabDB");

if (effectPrefabDB == null)

{

DebugLog.LogError("Effect Prefab DB is NULL");

return null;

}

}

GameObject prefab = effectPrefabDB.GetItem(id);

if (prefab == null)

{

DebugLog.LogError("Effect Manager Spawn ID Error - ID : " + id);

return null;

}

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

activeEffectList.Add(clone);

clone.transform.parent = parent;

clone.transform.localPosition = localPosition;

return clone;

}

public void ReleaseEffect(ParticleSystem particle)

{

ReleaseEffect(particle.gameObject);

}

public void ReleaseEffect(GameObject clone)

{

ObjectPoolManager.Instance.ReleaseObject(clone);

activeEffectList.Remove(clone);

}

public void ReleaseEffectAll()

{

for (int i = activeEffectList.Count - 1; i > 0; i--)

{

ReleaseEffect(activeEffectList[i]);

}

}

}

이펙트 매니저의 경우 ParticleSystem으로 ObjectPool을 만들어서 쓸까 하다가.. 경험상 파티클을 쓰지 않는 이펙트들도 종종 있던 터라 그냥 Prefab기반의 풀링을 이용하기로 하고 ObjectPoolManager를 그대로 사용했다.

주요 기능은 당연히 Effect Play하는 부분인데.. PlayOneShot이랑 PlayTimer 로 나눠놨다.

PlayOneShot은 말그대로 Looping 없는 이펙트를 한번만 실행한 후 다시 돌려놓는 기능이고, PlayTimer는 루핑이 있거나, 게임오브젝트 기반 이펙트를 생성한 후 특정 time 후에 제거하는 기능임. 루핑관련으로 사용한다면 예를들면 드래곤이 브레스를 3~5초간 발사한 후 제거해야한다던가..? 로 생각해봤다.

실제 구현하는 과정에선 생각없이 쓰려면

EffectManager.Instance.PlayOneShot(id,position); 식으로 함수만 호출해서 쓰면 되는데, 간혹 이펙트를 직접 관리하고 싶다거나 하면 SpawnEffect 해서 직접 다뤄도 될 것 같다. 대신 이 경우엔 반드시 Release해주는 것 잊으면 안된다…

다음은 SoundManager.cs

public class SoundManager : MonoBehaviourSingleton<SoundManager>

{

[SerializeField]

AudioClipResourceDB audioClipDB;

Transform root;

ObjectPool<AudioSource> sfxSourcePool;

ObjectPool<AudioSource> bgmSourcePool;

List<AudioSource> activeSfxSourceList = new List<AudioSource>();

List<AudioSource> activeBgmSourceList = new List<AudioSource>();

protected override void Awake()

{

base.Awake();

//Warm Pool;

root = transform;

sfxSourcePool = new ObjectPool<AudioSource>(CreateAudioSource, null, 20);

bgmSourcePool = new ObjectPool<AudioSource>(CreateAudioSource, null, 5);

}

AudioSource CreateAudioSource()

{

GameObject newObject = new GameObject();

newObject.transform.parent = root;

newObject.transform.name = "AudioSource";

return newObject.AddComponent<AudioSource>();

}

AudioClip GetClip(int id)

{

if (audioClipDB == null)

{

audioClipDB = Resources.Load<AudioClipResourceDB>("ResourceDB/AudioClipDB");

if (audioClipDB == null)

{

DebugLog.LogError("Audio DB is NULL");

return null;

}

}

return audioClipDB.GetItem(id);

}

bool PlaySfx(int id, float volume, out AudioSource source)

{

var audioClip = GetClip(id);

source = sfxSourcePool.GetItem();

if (audioClip != null)

{

source.clip = audioClip;

source.volume = volume;

source.Play();

activeSfxSourceList.Add(source);

return true;

}

return false;

}

void ReleaseSource(AudioSource source, List<AudioSource> sourceList, ObjectPool<AudioSource> pool)

{

if (source == null) { return; }

source.clip = null;

source.Stop();

sourceList.Remove(source);

pool.ReleaseItem(source);

}

void ReleaseSfxSource(AudioSource source)

{

ReleaseSource(source, activeSfxSourceList, sfxSourcePool);

}

void ReleaseBgmSource(AudioSource source)

{

ReleaseSource(source, activeBgmSourceList, bgmSourcePool);

}

public AudioSource PlaySfxOneShot(int id, float volume = 1f)

{

if (PlaySfx(id, volume, out AudioSource source))

{

source.loop = false;

StartCoroutine(PlaySfxOneShotCoroutine(source));

return source;

}

return null;

}

public AudioSource PlaySfxTimer(int id, float timer, float volume = 1f)

{

if (PlaySfx(id, volume, out AudioSource source))

{

source.loop = true;

StartCoroutine(PlaySfxTimerCoroutine(source, timer));

return source;

}

return null;

}

IEnumerator PlaySfxOneShotCoroutine(AudioSource source)

{

while (source.isPlaying)

{

yield return null;

}

ReleaseSfxSource(source);

}

IEnumerator PlaySfxTimerCoroutine(AudioSource source, float timer)

{

while (source.isPlaying && timer > 0f)

{

yield return null;

timer -= Time.deltaTime;

}

ReleaseSfxSource(source);

}

AudioSource FindActiveBgmSource(int id)

{

var clip = GetClip(id);

return FindActiveBgmSource(clip);

}

AudioSource FindActiveBgmSource(AudioClip clip)

{

if (clip == null) { return null; }

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

{

if (activeBgmSourceList[i].clip == clip)

{

return activeBgmSourceList[i];

}

}

return null;

}

public void PlayBgm(int id, float volume = 1f, float fadeTime = -1f)

{

AudioClip clip = GetClip(id);

if (clip == null) { return; }

var source = FindActiveBgmSource(clip);

if (source == null)

{

source = bgmSourcePool.GetItem();

}

source.clip = clip;

source.volume = volume;

source.loop = true;

activeBgmSourceList.Add(source);

if (source.isPlaying == false)

{

source.Play();

if (fadeTime > 0f)

{

source.volume = 0f;

source.DOFade(volume, fadeTime);

}

}

}

public void StopBgm(int id, float fadeTime = -1f)

{

var source = FindActiveBgmSource(id);

if (source != null)

{

StopBgm(source, fadeTime);

}

}

public void StopBgm(AudioSource source, float fadeTime = -1f)

{

StopSource(source, ReleaseBgmSource, fadeTime);

}

public void StopSfx(AudioSource source, float fadeTime = -1f)

{

StopSource(source, ReleaseSfxSource, fadeTime);

}

void StopSource(AudioSource source, UnityEngine.Events.UnityAction<AudioSource> releaseAction, float fadeTime = -1f)

{

if (source != null)

{

source.DOKill();

if (fadeTime > 0f)

{

source.DOFade(0f, fadeTime).onComplete = () => { releaseAction(source); };

}

else

{

releaseAction(source);

}

}

}

public void StopSoundAll(float fadeTime = -1f)

{

StopSfxAll(fadeTime);

StopBgmAll(fadeTime);

}

public void StopSfxAll(float fadeTime = -1f)

{

for (int i = activeSfxSourceList.Count - 1; i >= 0; i--)

{

StopSfx(activeSfxSourceList[i], fadeTime);

}

}

public void StopBgmAll(float fadeTime = -1f)

{

for (int i = activeBgmSourceList.Count - 1; i >= 0; i--)

{

StopBgm(activeBgmSourceList[i], fadeTime);

}

}

public void ChangeBgm(int to, float volume = 1f, float fadeTime = -1f)

{

StopBgmAll(fadeTime);

PlayBgm(to, volume, fadeTime);

}

public void ChangeBgm(int from, int to, float volume = 1f, float fadeTime = -1f)

{

StopBgm(from, fadeTime);

PlayBgm(to, volume, fadeTime);

}

}

사운드매니저는 AudioSource를 ObjectPool로 관리하는 싱글톤클래스로 만들었고, 각종 효과음이나 BGM을 재생하기 편하게 만들어두었다. 여기다가 추가적으로 BGM전환시에 DOTween 기능 이용해서 블렌딩도 가능하게끔 해봤디.

지금 BGMSource랑 SFXSource를 분리시켜놨는데 이 부분은 StopBgmAll이나 StopSfxAll 하려는 의도도 있었지만, 더 큰 이유로는 나중에 SFX Volume이랑 BGM Volume을 따로 컨트롤 할 수 있게끔 하려고 나눠놓았다. 아직 InGame Data쪽 프레임워크를 어떻게 가져갈지 결정을 안해서 해당 기능을 구현을 못해놨다…

기타 플레이 관련된 부분은 EffectManager랑 동일하니 뭐 넘어가도록 하자.