Unity – 드래곤플라이트 모방작

이승준

소개

- 유니티 게임개발 툴을 연습하기 위해 진행했던 원작 모방 게임입니다.
- 국내 스마트폰 게임 1세대라고 불릴 수 있는 종 스크롤 비행 슈팅게임 <드래곤 플라이트>의 게임플레이 씬의 게임시스템과 매니저 시스템을 구축해보았습니다.
- 코드 github : https://github.com/antivec/unity_project-practice/tree/master/Assets/Script

게임 진행방식

- 플레이어는 화면 하단의 캐릭터를 좌우로 움직이며 자동으로 발사되는 탄환을 사용
- 화면 상단에서 내려오는 드래곤 무리들을 격파해 나가며 오래 살아남아 점수를 얻는다
- 드래곤을 잡으면 일정확률로 게임진행에 도움이 되는 아이템을 얻거나, 플레이어 업그레이드에 필요한 코인을 얻을 수 있다.
- 드래곤과 부딪히면 플레이어는 패배한다.

필요 클래스 분석

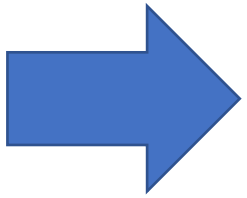
- 플레이어 움직임, 게임진행, 아이템, 몬스터 종류, 사운드 등.....
  - 게임이 실행되고 진행되기 위한 부품들..
  - 여러 개 생성 X
- 총알, 몬스터 웨이브, 아이템, 배경음악 & 효과음 등
  - 반복사용 , 재사용할 자원들..
  - → 반복적인 생성 / 삭제(instantiate, destroy())는 메모리 사용 낭비가 큼
- 용도에 따라 다른 클래스 구조가 필요함
  - 단 하나의 인스턴스 필요로 하고 데이터 전송이 많은 클래스
  - 여러 번 재활용이 필요한 오브젝트를 관리 할 클래스
  - 객체의 정보 및 유니티 속성을 초기화 및 관리

싱글톤
제너릭 클래스 선언

싱글톤이란 단 하나의 인스턴스를 미리 전역메모리에 선언하여 다양한 클래스로부터 데이터 공유에 용이해지거나 삭제/재생성으로 인한 메모리 낭비를 방지해준다

```
public class SingletonMonoBehaviour<T> : MonoBehaviour where T : SingletonMonoBehaviour<T> //제너릭 클래스로 다양한 매니
{
    static public T Instance { get; private set; }

    void Awake()
    {
        if (Instance == null)
        {
            Instance = (T)this;
            OnAwake();
        }
        else
        {
            Destroy(gameObject);
        }
    }
```

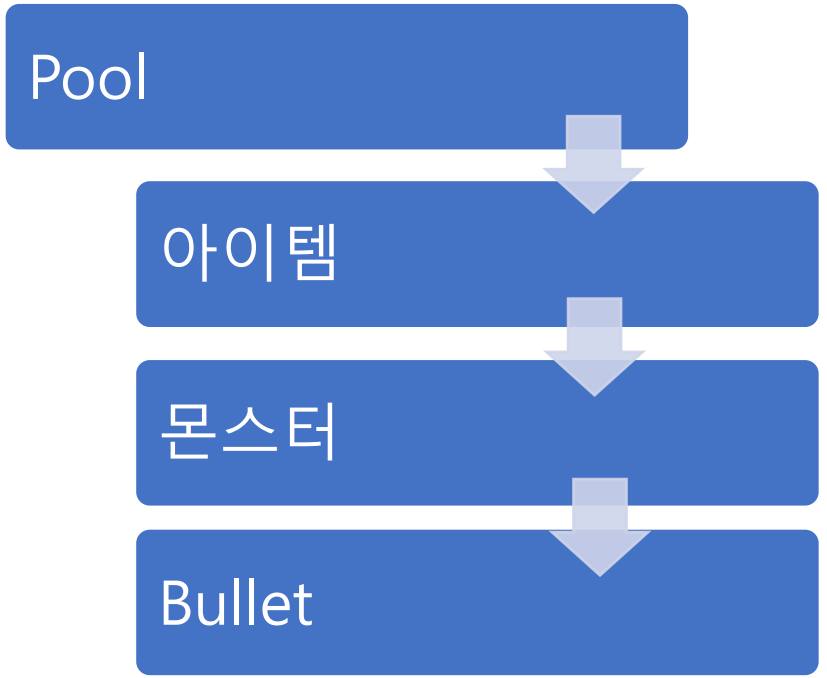


```
1  public class SoundManager : SingletonMonoBehaviour<SoundManager> { //사운드매니저 싱글턴 패턴 선언
2  ..
3  }
4
5
6  public class Item : MonoBehaviour {
7  ...
8      private void PlayItemGetSound() //아이템 클래스 내부에서 아이템 획득시 재생되는 효과음을 soundmanager.Instance 를 호출하는 예시
9      {
10         if (m_type == ItemType.Coin)
11             SoundManager.Instance.PlaySFX(SoundManager.SFX_CLIP.Get_Coin);
12         else if (m_type == ItemType.Gem_Red || m_type == ItemType.Gem_Green || m_type == ItemType.Gem_Blue)
13             SoundManager.Instance.PlaySFX(SoundManager.SFX_CLIP.Get_Gem);
14         else if (m_type == ItemType.Invincible)
15             SoundManager.Instance.PlaySFX(SoundManager.SFX_CLIP.Get_Invincible);
16         else if (m_type == ItemType.Magnet)
17             SoundManager.Instance.PlaySFX(SoundManager.SFX_CLIP.Get_Item);
18     }
19     // Update is called once per frame
20     void Update () {
21
22     }
23 }
24
```

Pooling System

- Instantiate(),Destroy() –> GameObject와 같은 자원들을 실시간 생성/제거
  - 하지만 상당한 메모리 자원이 사용됨.

- 이를 해결하기 위하여 미리 MonoBehaviour 클래스가 실행되었을때 한번만 실행되는 함수 Start() or Awake() 함수에 진입할 때 리소스들을 한번만 사용할 양 만큼 생성하도록 하고 이를 재활용하도록 함 (스택 형식)
(ex: 몬스터 사망시 → 오브젝트 파괴 x
→ 오브젝트 deactivate & push
→ 다시 필요시 pop & activate)



Pooling system 구현 & 예제


```
public class GameObjectPool<T> where T : class
{
    short count;
    public delegate T Func();
    Func create_fn;
    Stack<T> objects;
    public GameObjectPool(short count, Func fn)
    {
        this.count = count;
        this.create_fn = fn;        //(C# delegate) == (C++ 함수포인터)
        this.objects = new Stack<T>(this.count); // 풀링된 object를 저장할 stack 자료형
        allocate();
    }
    void allocate()
    {
        for (int i = 0; i < this.count; ++i)
        {
            this.objects.Push(this.create_fn()); //stack.push()
        }
    }
    public T pop()
    {
        if (this.objects.Count <= 0)
        {
            allocate();
        }
        return this.objects.Pop();
    }
    public void push(T obj)
    {
        this.objects.Push(obj);
    }
}

public class ParticleManager : SingletonMonoBehaviour<ParticleManager> {    // partical manager 에서의 예제
        m_particlePool = new GameObjectPool<ParticleSystem>(10, () =>       // gameobjectpool의 생성자로써 (생성 갯수, 대리자(delegate) 함수)를 매개변수로 받아 pooling 한다
        {
            GameObject obj = Instantiate(m_effectObject) as GameObject;
            obj.transform.parent = transform;
            obj.SetActive(false);
            ParticleSystem particle = obj.GetComponent<ParticleSystem>();
            return particle;
        });
        m_listParticle = new List<ParticleSystem>();
}
```

Manager 목록

GameManager – 게임의 상황/환경 관리

- 플레이어가 사망상태인가? 무적 아이템을 먹은 상태인가? Normal 상태인가?

PlayerManager - 플레이어의 제어

- 움직임, 양옆 벽 Collider 제어 , 플레이어 캐릭터 애니메이션 제어

BG_Scroll - 게임 배경화면 제어

- 상황에 따른 배경 움직임 속도 변경
- 배경 변경 제어 & 배경 변경 효과 추가 (Fade-in & Fade-out)

MonsterManager – 몬스터 제어

- 몬스터 종류 컨트롤, 몬스터 웨이브 속도 제어
- 몬스터 Pool 에서의 데이터 사용

ParticleManager – 효과 파티클 제어

- 이펙트 효과 풀 제어
- (아이템을 먹었을때, 적을 처치했을때)

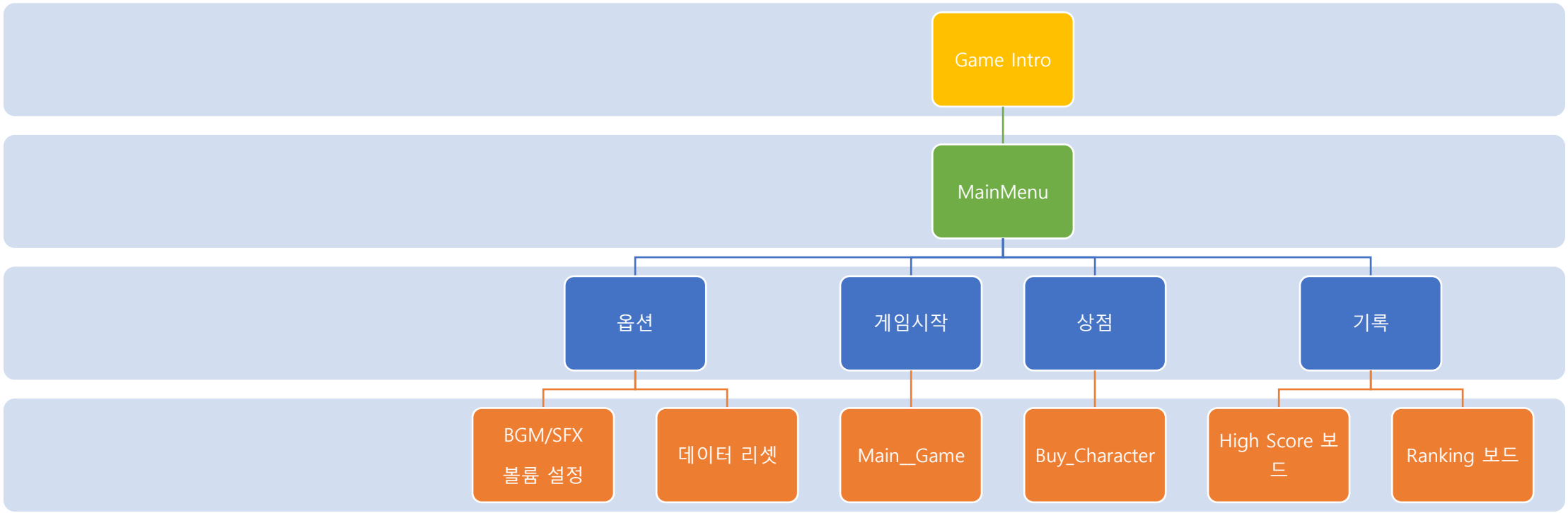
ItemManager – 아이템 생성, 확률 제어

- 아이템 생성 확률 제어, 아이템 생성 및 아이템 상태 처리
- (플레이어가 획득시 , 미획득 시)

SoundManager – 음향 효과 제어

- 각종 배경음악,효과음을 pool에 관리
- 배경음악을 재생 여부, 음량 조절

게임의 전체적 흐름도


Game Intro
MainMenu
옵션
게임시작
상점
기록
BGM/SFX
볼륨 설정
데이터 리셋
Main__Game
Buy_Character
High Score 보드
Ranking 보드

GameManager

-게임의 상황/환경 관리

- 무적 아이템을 먹었을 시 특수한 상태가 되고 Invincible State로 돌입한다
- 이때 플레이어 캐릭터가 빠르게 이동하는 것처럼 보이게 배경화면의 스크롤 효과를 빠르게 하고 몬스터 웨이브의 속도 또한 증가한다.
- 무적의 지속시간이 끝나면 Normal State로 돌아온다.

- 자석 아이템을 먹었을 시 주변의 아이템을 끌어오는 상태가 된다.
  - 이때 State는 Normal로 유지한다.

```
public enum GameState
{
    Normal = 0,
    Invincible
}
GameState m_state;
bool isSet;
float m_InvincibleMagPower = 5.0f;
public  void SetGameState(GameState state)
{
    if (state == m_state)
        return;
    m_state = state;
    isSet = true;
}
private void SetInvincibleMode() // 무적아이템을 먹을
{
    m_bgScroll.m_speed *= m_InvincibleMagPower;
    MonsterManager.Instance.SetMonsterSpeedInvincibl
    PlayerManager.Instance.SetInvincible();
    MonsterManager.Instance.ResetCreateMonster();
    isSet = false;
    Debug.Log("Invincible");
}
private void SetNormalMode()
{
```

GameManager
게임 제어

① 몬스터를 잡을시 score 증가
② 코인을 먹을 시 소지한 총 코인량 표시
③ 현재까지 진행한 거리 표시
일정 거리를 돌파하면 배경화면이 변경되면서 스테이지 상태가 변한다

④ 플레이어의 체력 상태

점수는 ScoreManager,
체력상태는 Heart script를 통해 관리 된다

하트가 다 사라지면
게임오버상태가 된다

게임오버 상태가 되면
gameover_UI가 활성화 되어 각종 점수가 표시 되고
'다시 하기' 버튼이 활성화 된다

Player Manager
플레이어의 제어

- 플레이어가 화면을 클릭(탭)을 유지할때, 캐릭터를 움직일 수 있다
- 화면 양옆에 collider를 추가해 게임화면에서 벗어나지 못하게 막는다
- 화면 위아래의 collider는 발사체와 적의 Setactive() 경계선이다.

MovePlayer() & Wall_Collider

```
void MovePlayer()
{
    float dir = Input.GetAxis("Horizontal");
    if (Input.GetMouseButtonDown(0))
    {
        m_clickOn = true;
        startPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
    }
    if (Input.GetMouseButtonUp(0))
    {
        m_clickOn = false;
        startPos = Vector3.zero;
    }
    if (m_clickOn) //클릭 유지한 상태에서 움직임 적용
    {
        Vector3 mousePos = Input.mousePosition;
        targetPos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
        float reverse = 1.0f; // -1.0f;
        dir = (targetPos.x - startPos.x) * reverse;
        if (dir < 0 && m_col_left || dir > 0 && m_col_right) { dir = 0; } //양쪽끝 제어
        transform.Translate(new Vector3(dir, 0, 0));
        startPos = targetPos;
    }
```

```
private void OnTriggerEnter2D(Collider2D collision)
{
    if(collision.tag == "wall_left") {
        m_col_left = true;
        startPos = result;
    }
    if (collision.tag == "wall_right") {
        m_col_right = true;
        startPos = result;
    }
}
private void OnTriggerStay2D(Collider2D collision)
{
    if (collision.tag == "wall_left")
    {
        //m_col_left = true;
        startPos = result;
    }
    if (collision.tag == "wall_right")
    {
        //m_col_right = true;
        startPos = result;
    }
}
private void OnTriggerExit2D(Collider2D collision)
{
    if (collision.tag == "wall_left") {
        m_col_left = false;

    }
    if (collision.tag == "wall_right") {
        m_col_right = false;
    }
}
```

Monster Manager
몬스터 컨트롤

- 몬스터 웨이브는 화면상단에서 하단으로 내려온다
- 몬스터는 탄환에 맞아 체력이 0이 되거나 플레이어가 무적아이템을 먹어 부딪혔을때 사망한다
- 플레이어의 점수가 죽은 몬스터 타입에 따라 값이 오른다
- 몬스터 종류로는 일반 몬스터, 특수 몬스터가 있다
  - Dragon_bomb가 사망할때 폭발하여 같은 라인의 몬스터들을 제거
- 플레이어가 죽이지 못한 몬스터는 하단 collider를 만나 deactivate 한다


```
public int SetMonsterScore(MonsterType type)
  {
      int score = 0;
      switch (type)
      {
          case MonsterType.Normal_Yellow:
              score = 10;
              break;

          case MonsterType.Normal_Pink:
              score = 20;
              break;
          case MonsterType.Normal_White:
              score = 30;
              break;
          case MonsterType.Normal_Bomb:
              score = 5;
              break;
      }
      return score;
  }
  public void SetDieEffect() //몬스터 사망시 실행되는 함수들
  {
      PlayerManager.Instance.m_iScore += SetMonsterScore(m_type); //점수 증가
      ParticleManager.Instance.OnEffect(transform.position); //사망 이펙트
      ItemManager.Instance.CreateItem(transform.position); // 아이템 드롭
      SoundManager.Instance.PlaySFX(SoundManager.SFX_CLIP.Mon_Die);  //효과음 재생
  }
```

```
if (collision.tag == "bullet") m_life -= PlayerManager.Instance.m_power;
else m_life = 0;
if (m_life <= 0)
{
    if (m_type != MonsterType.Normal_Bomb)
        SetDie();
    else
        MonsterManager.Instance.DeleteMonsterLine(m_lineNum); // Normal_Bomb가 사망했을시 MonsterLine 삭제
}
```

Monster Manager – MonsterSetting

```
public void CreateMonsters()
{
    bool isBomb = false; //특수 몬스터(bomb)인가
    bool isSelect = false; // 특수 몬스터가 선택되어 할당되었는가
    #pragma region
    MonsterBehav.MonsterType type;
    for(int i = 0; i < m_count; i++ )
    {
        MonsterBehav mon = m_monsterPool.pop();
        do
        {
            type = (MonsterBehav.MonsterType)UnityEngine.Random.Range(0, (int)MonsterBehav.MonsterType.Max);
            if (type == MonsterBehav.MonsterType.Normal_Bomb && !isBomb) {
                isBomb = true;
            }
            else if (type == MonsterBehav.MonsterType.Normal_Bomb && isBomb) {
                isSelect = true;
            }
            else
                isSelect = false;
        } while (isSelect);
        mon.m_lineNum = i / m_count + 1;
        mon.InitMonster(0, type);
        mon.gameObject.SetActive(true);
        mon.gameObject.transform.parent = null;
        mon.gameObject.transform.position = new Vector3(m_startPos.x + (i * m_gap_x), m_startPos.y);

        m_monsterList.Add(mon);
    }
    //Debug.Log(m_monsterList.Count);
    Invoke("CreateMonsters", m_respawnInterval / m_curSpeed); //IEnumerable 함수 사용 --> respawnInterval 과 현재 게임상태(무적 or 일반)을 사용하여
}                                                             //몬스터 웨이브 생성 함수 시간 조절
public void ResetCreateMonster()
{
    CancelInvoke("CreateMonsters");
    Invoke("CreateMonsters", m_respawnInterval / m_curSpeed); //CreateMonster와 동일
}
public void RemoveMonster(MonsterBehav mon) // 몬스터 객체 제거
{
    m_monsterList.Remove(mon);
    ResetMonster(mon);
}
public void ResetMonster(MonsterBehav mon)
{
    mon.gameObject.transform.position = new Vector3(m_startPos.x, m_startPos.y);
    mon.gameObject.transform.parent = transform;
    mon.gameObject.SetActive(false);
    m_monsterPool.push(mon);
}
public void DeleteMonsterLine(int lineNum) //Monster_bomb 사망시 사용됨
{
    Debug.Log("DeleteMonsterLine: " +m_monsterList.Count + "LineNum : " + lineNum);
```

MonsterSetting은 몬스터 생성 및 제거 사용되는 상세 설정을 담당한다

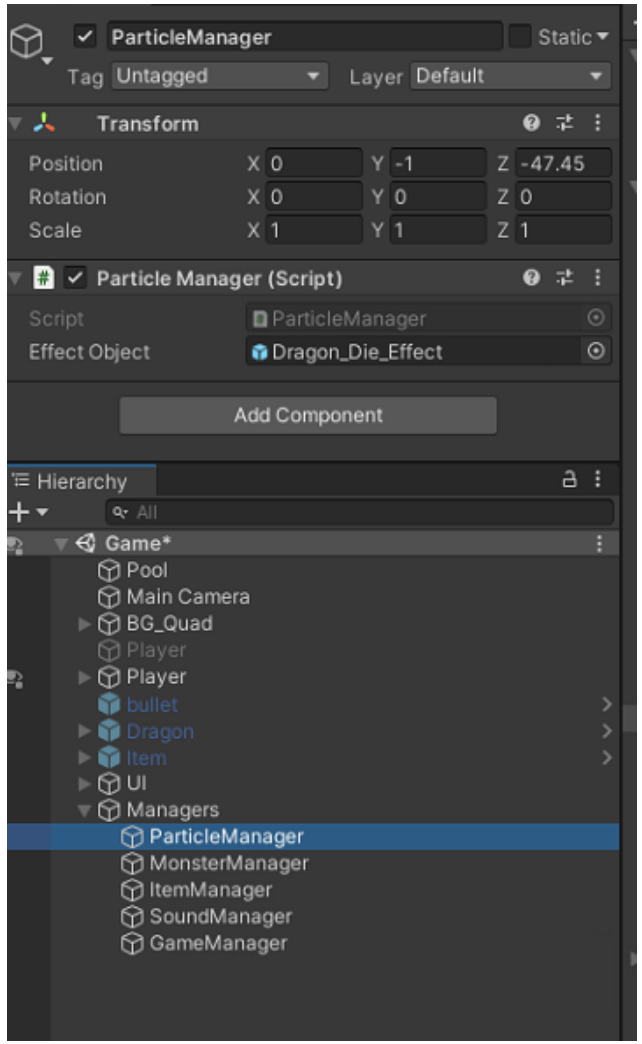
- 특수 몬스터 중복 생성 방지
- 특수 몬스터 사망시

라인 제거함수

```
void Start () {
    m_animator = GetComponent<Animator>();
    m_monster = transform.parent.GetComponent<MonsterSetting>();
}
public void SetStateIdle()
{
    m_monster.m_state = MonsterSetting.MonsterState.IDLE;
    m_animator.SetInteger("state", (int)m_monster.m_state);
    m_monster.SetIdleSprite();
}
```

MonsterAniControl은 몬스터의 움직임 Animator를 handle하는데 도와준다

Particle Manager 오브젝트의 시각효과 제어




```
public class ParticleManager : SingletonMonoBehaviour<ParticleManager> {
    GameObjectPool<ParticleSystem> m_particlePool;
    [SerializeField]
    GameObject m_effectObject;
    List<ParticleSystem> m_listParticle;
    protected override void OnStart()
    {
        base.OnStart();
        // .. particle object pool
    }

    public void OnEffect(Vector3 pos)
    {
        var effect = m_particlePool.pop();
        effect.transform.position = pos;
        effect.gameObject.SetActive(true);
        effect.transform.parent = null;
        effect.Play();
        //Invoke("OffEffect", 1f);
        StartCoroutine("OffEffect", effect);
    }

    public IEnumerator OffEffect(ParticleSystem effect)
    {
        yield return new WaitForSeconds(1f); // 이펙트 발생후 1초후 offeffect yield
        m_listParticle.Remove(effect);
        effect.transform.parent = transform;
        effect.gameObject.SetActive(false);
        m_particlePool.push(effect);
    }

}
```

Particle Manager에 적 드래곤이 사망 시 재생에 필요한

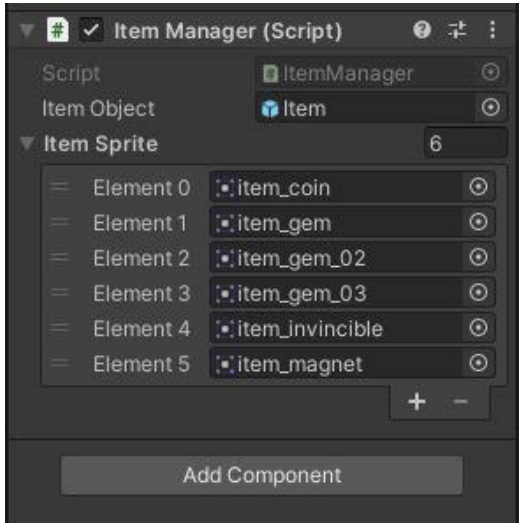
Dragon_Die_Effect 가 부착되어 있고 실제로 사망 시 아래와 같은 효과와 함께 파괴된다

Item Manager
아이템 제어 매니저

- 아이템 종류로는 코인, 무적물약, 마그넷이 있다

- 코인 → 스코어 산정 장치 중 하나
- 무적 물약 → 획득 할 경우 비행 속도가 빨라지며, 닿는 모든 적들을 라이프 소모 없이 파괴한다.
- 마그넷 → 일정 시간동안 플레이어 주변에 떨어지는 아이템을 끌어와 획득하게 한다

- 각각의 아이템에는 m_itemProbability 라는 int[]에 각각의 해당하는 확률 값을 가지고 있고, 이는 get_priority라는 함수의 계산을 통해 결과를 얻어 아이템을 생성한다.



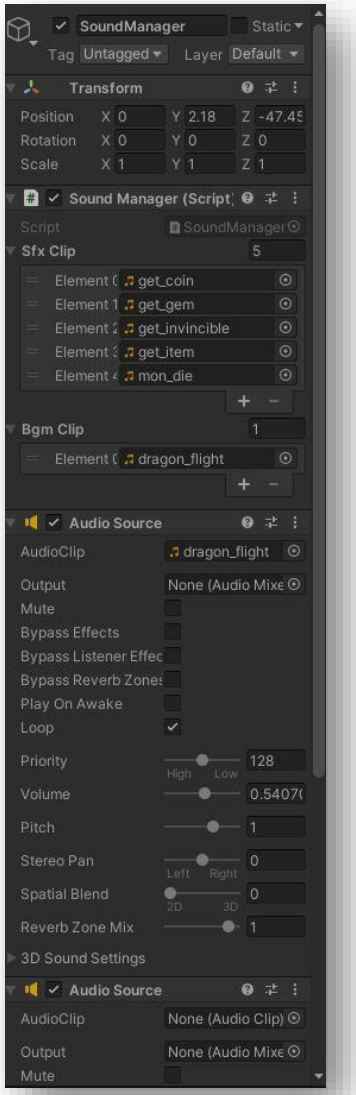
Item Manager
아이템 드랍 계산

```
    {
        base.OnStart();
        m_itemPool = new GameObjectPool<Item>(10, () => // item pool initialize
        {
            GameObject obj = Instantiate(m_itemObject) as GameObject;
            obj.SetActive(false);
            obj.transform.parent = transform;
            Item item = obj.GetComponent<Item>();
            return item;
        });
        m_itemList = new List<Item>();
        SelectItem();
    }
    public void CreateItem(Vector3 pos)
    {
        Item item = m_itemPool.pop();
        item.transform.position = pos;
        item.gameObject.SetActive(true);
        item.transform.parent = null;
        item.SetItem(SelectItem());
        m_itemList.Add(item);
    }
    public static float Rand(float min, float max)
    {
        return Random.Range(min, max);
    }
    public Item.ItemType SelectItem()
    {
        var result = Util.GetPriority(m_itemProbability);
        return (Item.ItemType)result;
    }
    public static int GetPriority(int[] priorities)
    {
        int sum = 0;
        for (int i = 0; i < priorities.Length; ++i)
        {
            sum += priorities[i];  //int[] m_itemProbability = new int[]{ 60, 6, 2, 1, 12, 19 }; == 100
        }
        if (sum <= 0) // illegal sum value
            return 0;

        int num = Rand(1, sum); // 60
        sum = 0;
        for (int i = 0; i < priorities.Length; ++i)
        {
            int start = sum;                //  sum ==> m_itemProbability 안의 값을 차례대로 합하여 num 과 비교한다.
            sum += priorities[i];           //  start ==>  0에서 시작 // (start < num && num <= sum) 조건이 실패했을때 sum 값을 저장.
            if (start < num && num <= sum)  //  num ==> 1 ~ sum 사이의 값, 아이템 생성의 기준값이 된다.
                                            //  m_itemProbability를 쉽게 풀어보자면 {60이하의 수 일 경우 일반 코인, 61이상 66이하의 값 일 경우 빨간 쥬얼이, ...., 82이상 100이하의 값이 num에 저장이 된다면 magn
            {                               // start = 0 , num = 60 , sum = 60  >> (start < num) && (num <= sum) 에 둘 다 만족하므로 item[0] 생성 ,
                return i;
            }
        }
        return 0;
```

SoundManager 배경음악, 효과음 매니저

- SoundManager 내의 sfx, bgm array에 사용할 bgm,sfx 파일을 unity editor에서 등록
- Audio Source 에서 음악 데이터의 볼륨,우선순위 등을 조정 할 수 있다.
- MainMenu - Option에서 유저가 소리크기를 조정 할 수 있다.
- 유저 정의 볼륨 크기는 PlayerPrefs를 통해 저장한다.



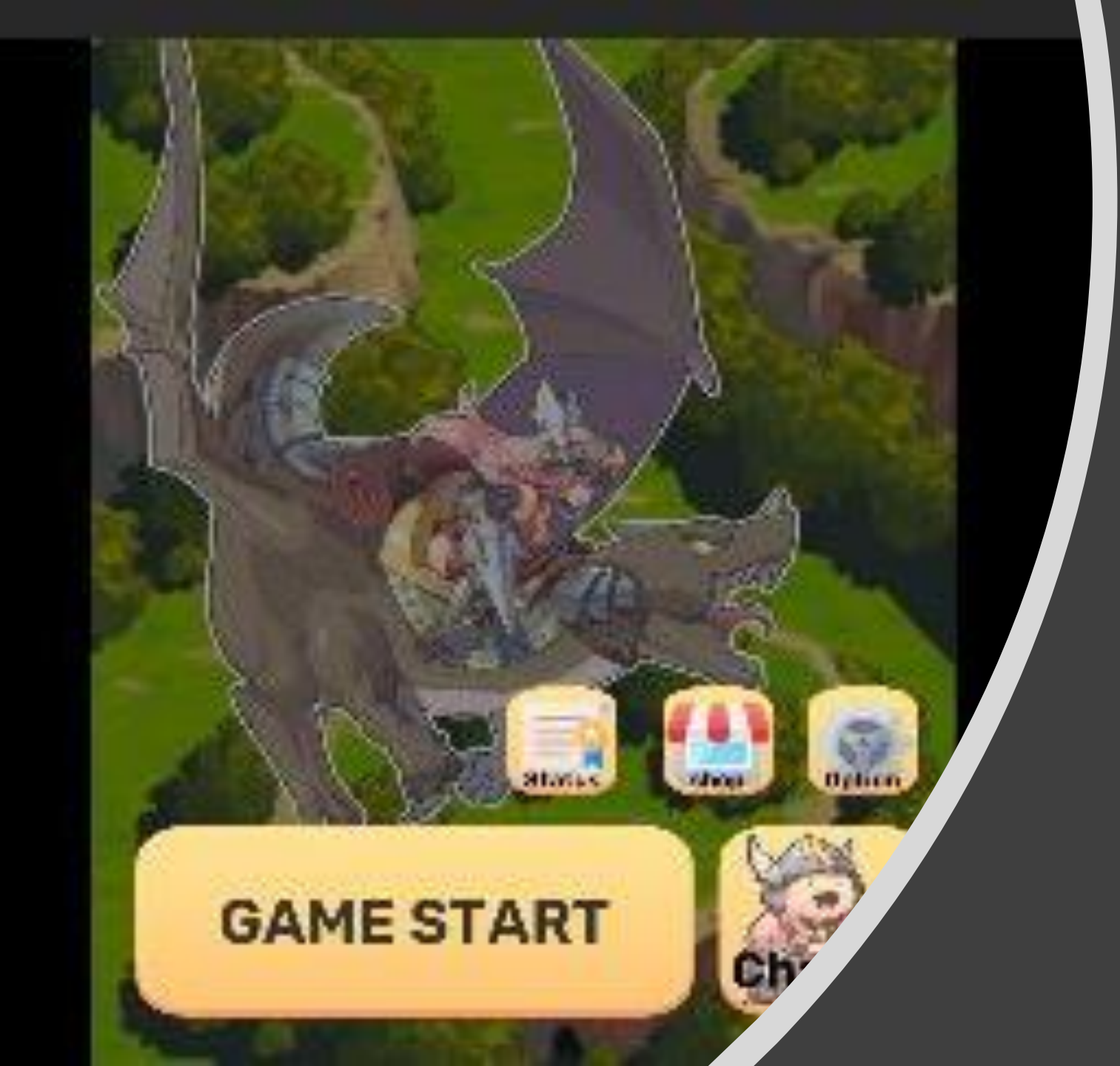
PlayerPrefs Editor
Asset으로 확인한 결과

```
public class SoundManager : SingletonMonoBehaviour<SoundManager> {
    AudioSource [] m_audio = new AudioSource[2];
    float m_fBGM_Volume;
    protected override void OnStart()
    {
        // Pool Initiate
    }
    public void SetVolume(float volume)
    {

        m_audio[(int)AudioType.SFX].volume = volume; // 효과음 음량 조정
        m_audio[(int)AudioType.BGM].volume = volume; // 배경음악 음량 조정
        for(int i = 0; i < m_audio.Length; i++)
        {
            m_audio[i].volume = volume;
        }
    }
    public void SetMute()
    {
        for (int i = 0; i < m_audio.Length; i++)
        {
            m_audio[i].mute = true; // 음소거
        }
        for (int i = 0; i < m_audio.Length; i++)
        {
            m_audio[i].mute = false; // 음소거 해제
        }
    }
    public void PlayBGM(BGM_CLIP bgm)
    {
        m_audio[(int)AudioType.BGM].clip = m_bgmClip[(int)bgm];
        m_audio[(int)AudioType.BGM].Play();

        m_audio[(int)AudioType.SFX].clip = m_sfxClip[(int)sfx];
        m_audio[(int)AudioType.SFX].Play(); // 효과음 플레이
    }
    public void Stop(AudioType type)
    {
        m_audio[(int)type].Stop();
    }
}
```

Sound Manager summary code



메인 시작 메뉴

옵션 팝업 메뉴

- 플레이어는 이 옵션 창을 통해 BGM과 효과음의 볼륨을 조절
- 취소 옆에 있는 빨간 휴지통 버튼을 누르면 초기화 확인창이 뜬다
- 이를 통해 플레이어는 언제든 진행상황과 점수 데이터를 초기화할 수 있다.

확인 경고 창 구현

- 리소스 크기가 적은 부속 경고창을 prefab으로 만들어 항상 unity hierarchy에 로드 하는것이 아닌 유저가 필요할때에만 instantiate 하여 사용 하도록 설계하였다.

```
1  public class WarningPopup : MonoBehaviour
2  {
3      GameObject popup_obj;
4      public void PopupCall( )
5      {
6          popup_obj = Instantiate(Resources.Load("Prefab/Warning_Canvas", typeof(GameObject)) as GameObject);
7          popup_obj.gameObject.transform.SetParent(transform, false);
8          popup_obj.gameObject.SetActive(true);
9
10         this.Proceed_btn = GameObject.Find("Warning_Proceed_Btn").GetComponent<Button>();
11         this.Cancel_btn = GameObject.Find("Warning_Cancel_Btn").GetComponent<Button>();
12         Cancel_btn.onClick.AddListener(CancelAction);
13         Proceed_btn.onClick.AddListener(ConfirmAction);
14     }
```

후기 / 느낀 점

- 게임이라는 프로그램 자체가 메모리를 자주 사용하고 항상 부족한 프로그램이기 때문에 코딩 할때에도 상황에 따라 함수의 처리 속도와 메모리 사용에 대한 고민을 하게 되는 계기가 되었다.
- Coroutine 과 같은 사용하기 편한 비동기식 호출 함수 같은 경우:
  - 반환 시 사용 되는 IEnumerator Method들은 var 변수에 저장하는 것이 좋다

    Ex) var wait = new WaitForSeconds(0.01f)
  - 반복적이고 빈번하게 coroutine을 호출 할 때 마다 Garbage가 생성되고 이는 C#의 Garbage Collector의 대상이 된다. 힙 메모리의 불필요한 확장을 야기할 수 있고, 운영체제가 사용해야되는 주소 메모리가 적어지게 되면 Application을 강제 종료시킬 수 있다.

```
public IEnumerator MapFadeIn() // 실제 사용한 예 ) 플레이어가 일정 거리를 지나 배경화면을 변경할 때 사용한 fadein / out 함수이다
{
    //Debug.Log("FadeIn called!");
        // 배경의 색을 color.lerp를 통해 fadein/out 기능을 구현했는데 이는 while문을 사용하여 매 프레임마다 호출해야 하는데 매번 new를 통해
        // 힙메모리에 할당하게 되면 걷잡을수 없이 확장하게 된다는 점을 깨닳았고, 대규모 크기의 게임같은 경우 메모리 사용을 중요시 해야하기때문에 변경할 필요를 느꼈다.
      Color lerp_Color;
      float m_lerpTime = 0.0f;
      var yield_Fixed = new WaitForFixedUpdate(); // 때문에 이와 같이 var 변수를 할당하였고 한번만 new를 호출하여 할당한다.
    if(m_material.GetColor("_TintColor") == Color.gray)
    {
        m_material.SetColor("_TintColor", Color.black);
    }
      while (m_lerpTime < 1.0f)
       {
            m_lerpTime += (Time.deltaTime * m_speed);
            lerp_Color = Color.Lerp(Color.black, Color.gray, m_lerpTime);
            m_material.SetColor("_TintColor", lerp_Color);

            yield return yield_Fixed; //
       //Debug.Log("End of Coroutine ");
        }
    yield return null;
```