

2018-2학기 세종창의학기제 주간학습보고서

이름	오민규	학과(전공)	컴퓨터공학과		
학번	128206	학년	4		
연락처	010-8843-4757	e-Mail	hotalsrb12@naver.com		
과목명	자기주도 창의전공 I, II, III, IV	신청학점	12	분반	3
학습기간	2018. 12. 03 ~ 2018. 12. 09	학습주차	13	학습시간	55
창의과제	네트워크를 통해 상대방과의 실감나는 대전형 슈팅 3D 모바일게임 개발				
금주 학습목표	<ul style="list-style-type: none"> - 오브젝트 풀 구현 - 풀 매니저 구현 - 프로토타입 맵 제작 				
학습내용	<p>1. 오브젝트 풀 구현</p> <p>오브젝트들을 생성할 때 객체 하나씩 생성하여 사용하였다. 캐릭터나 맵에 존재하는 오브젝트의 경우는 객체 하나하나 생성하여 사용해도 되지만 이펙트 또는 발사체 생성할 때 이전과 같은 방식을 사용한다면 엄청난 부하가 걸리게 된다. 발사체가 평균 3초 후에 사라진다면 6명의 플레이어가 초당 각자 10발씩 발사를 하였다면 총 180개의 객체가 생성된다. 카메라 컬링이 존재하지 않는다면 맵 + 캐릭터 + 총알 180발 + 이펙트 + 게임 UI를 렌더링 해야 하는데 모바일 기기에서는 절대로 정상 실행이 될 수 없는 상황이다. 따라서 오브젝트 풀링 방식을 이용하여 객체들을 관리해야한다. 지금까지 게임 개발을 하였을 때 풀링 방식을 사용하지 않았고 비슷한 기술인 인스턴싱을 사용하여 객체들을 관리했던 적이 있었다. 인스턴싱의 경우 하나의 객체를 가지고 위치, 회전, 크기, 색상 등 작은 속성 정보만 변경하여 이 속성에 맞춰 렌더링 하는 방식이었다. 이 기술과 풀링 방식은 비슷할 것이라고 생각되었다. 하지만 처음 구현을 해보는 것이기 때문에 학습이 필요하였다. 문헌과 구글링을 통해 학습을 하였다. 다행이도 인스턴싱과 크게 다르지는 않았지만 인스턴싱은 셰이더로 구현을 하는 방법이었고 이것은 응용 프로그램에서 사용되므로 구현하는 방식이 달랐다. 하지만 리스트 두 개로 해결됨을 알 수 있었기 때문에 학습을 마치고 바로 구현으로 넘어갔다. 풀 스크립트를 사용하는 게임 오브젝트를 저장하기 위해 게임 오브젝트 변수를 생성하였고 초당 10발씩 발사된다는 가정 하에 아이스크림 볼의 프리팹을 초기 생성 수를 10개로 할당하였다. 추후 레벨디자인을 생각하여 런타임 시 생성 플래그와 이 플래그가 활성화가 되면 추가 생성 인스턴스 수를 정하는 변수를 정의하였다. 그리고 가장 중요한 활성화된 인스턴스 리스트와 비활성화된 인스턴스 리스트를 관리하는 리스트들이 필요하여 정의하였다. 게임이 시작되었을 때 지정된 초기 생성 개수로부터 미리 생성해 놓고 게임 오브젝트들을 비활성화 리스트에 추가하였다. 플레이어가 공격을 시작하면 비활성화 리스트에 존재하는 풀을 꺼내어 활성화 시키고 이것을 사용하게끔 Spawn 메소드와 일정 시간이 지나거나 충돌되었거나 했을 때 비활성화 시키는 Despawn 메소드를 구현하였다. 그 외에도 예외처리 메소드와 오브젝트 명을 생성하는 메소드를 구현하였다.</p>				

```

public class Pool : MonoBehaviour {

    // 풀링하기 위한 프리랩 생성
    public GameObject prefab;

    // 게임 시작 시 생성할 인스턴스 양
    public int preLoad = 0;

    // 런타임에 새 인스턴스 생성을 제한할지 여부
    public bool limit = false;

    // limit 변수가 활성화된 경우 최대 인스턴스 양
    public int maxCount;

    // 풀에서 활성화된 프리랩 인스턴스 목록
    [HideInInspector]
    public List<GameObject> active = new List<GameObject>();

    // 풀에서 비활성화된 프리랩 인스턴스 목록
    [HideInInspector]
    public List<GameObject> inactive = new List<GameObject>();

    // 런타임 생성 풀에서 풀관리자가 호출하는 초기화
    public void Awake()
    {
        // 프리랩 없이는 초기화 불가능
        if (prefab == null)
            return;

        // 풀 관리자 사전에 이 풀 추가
        PoolManager.Add(this);

        PreLoad();
    }

    // 플레이 시간 전에 지정된 개체 양을 로드
    public void PreLoad()
    {
        if (prefab == null)
            return;

        public void PreLoad()
        {
            if (prefab == null)
            {
                Debug.LogWarning("풀 안에 프리랩이 비어 있습니다. 프리로드가 발생하지 않습니다. 참조를 확인하십시오.");
                return;
            }

            // 정의된 프리로드 양을 인스턴스화 하지만 최대 개체 양을 초과하지 않음
            for (int iCnt = totalCount; iCnt < preLoad; iCnt++)
            {
                // 프리랩의 새 인스턴스를 인스턴스화
                GameObject obj = (GameObject)Object.Instantiate(prefab, Vector3.zero, Quaternion.identity);
                // 이 위치의 새 인스턴스를 부모로 지정
                obj.transform.SetParent(transform);

                // 편집자 개요가 용이하도록 고유한 제목으로 변경
                Rename(obj, transform);
                // 자식 개체들을 포함한 개체 비활성화
                obj.SetActive(false);
                // 비활성 인스턴스 리스트에 개체 추가
                inactive.Add(obj);
            }

            // 이 풀의 새 인스턴스를 활성화(또는 인스턴스화)
            public GameObject Spawn(Vector3 position, Quaternion rotation)
            {
                // 변수 초기화
                GameObject obj;
                Transform transform;

                // 활성화할 비활성 개체가 존재
                if (inactive.Count > 0)
                {
                    // 리스트에서 첫 번째 비활성화 개체를 가져옴
                    obj = inactive[0];
                    // 가져온 개체를 활성화하고, 비활성화 리스트에서 제거
                    inactive.Remove(obj);
                }
            }
        }
    }
}

```

2. 풀 매니저 구현

오브젝트 풀을 구현하였지만 풀은 앞으로 여러 개(파티클, 이펙트, 환경 등)일 것이며 풀들을 생성하여 매니저 스크립트로 관리를 할 수 있어야 한다. 풀 스크립트의 Awake 함수에서 스크립트가 활성화 되면 풀 매니저에 등록해야한다. 풀 매니저는 풀을 저장하기 위해 풀에 존재하는 오브젝트와 풀을 저장해야한다. 따라서 풀 매니저에서 이들을 관리할 변수를 딕셔너리 컨테이너로 사용해야한다. 위치와 회전 그리고 게임오브젝트의 정보를 전달받아 딕셔너리에 전달 받은 게임오브젝트가 존재한다면 이전에 작업한 풀을 Spawn한다. 이와 같은 방식으로 Despawn 메소드도 게임오브젝트를 매개변수로 넘겨주어 해당 딕셔너리를 탐색하여 그 안에 존재하면 넘겨받은 풀의 오브젝트를 비활성화 한다. 지금까지 매니저를 구현한 경험이 많이 있었기 때문에 큰 어려움 없이 구현을 할 수 있었다.

```

public class PoolManager : MonoBehaviour
{
    //Prefab을 풀 컨테이너에 매핑하여 모든 인스턴스를 관리
    private static Dictionary<GameObject, Pool> Pools = new Dictionary<GameObject, Pool>();

    // 각 풀 자체에서 호출하면 사전에 추가
    public static void Add(Pool pool)
    {
        // 풀에 프리랩이 포함되어있지 않은지 확인
        if (pool.prefab == null)
        {
            Debug.LogError("풀의 프리랩: " + pool.gameObject.name + " Pools Dictionary에 이미 추가되었습니다.");
            return;
        }

        // 사전에 추가
        Pools.Add(pool.prefab, pool);
    }

    // 런타임에 새 풀을 생성
    // 어떤 연결되지 않은 프리랩들에 대해 호출
    // 에디터에서 빈의 풀로 연결되지만 그림에도 불구하고 Spawn()을 통해 호출
    public static void CreatePool(GameObject prefab, int preLoad, bool limit, int maxCount)
    {
        // 이미 풀이 추가된 경우 디버그 메시징 발생
        if (Pools.ContainsKey(prefab))
        {
            Debug.LogError("이미 풀 매니저에 프리랩 풀이 포함: " + prefab.gameObject.name);
            return;
        }

        // 새로운 풀 컴포넌트를 고정할 새로운 gameObject 생성
        GameObject newPoolGO = new GameObject("Pool " + prefab.name);
        // 새로운 게임오브젝트를 빈안에 올 컴포넌트를 추가
        Pool newPool = newPoolGO.AddComponent<Pool>();
        // 기본 매개변수 할당
        newPool.prefab = prefab;
        newPool.preLoad = preLoad;
        newPool.limit = limit;
        newPool.maxCount = maxCount;

        // 풀하는 위치에서 전달된 프리랩에 대한 사전 인스턴스를 활성화
        public static GameObject Spawn(GameObject prefab, Vector3 position, Quaternion rotation)
        {
            // 풀에서 prefab을 찾을 수 없는 경우 로그 출력 제거
            // 실행 시, 새로운 풀을 생성하기 때문에 이는 중요하지 않음
            if (Pools.ContainsKey(prefab) == false)
            {
                Debug.Log("기존 풀에서 프리랩을 찾을 수 없음: " + prefab.name + " 새 풀이 생성되었습니다.");
                CreatePool(prefab, 0, false, 0);
            }

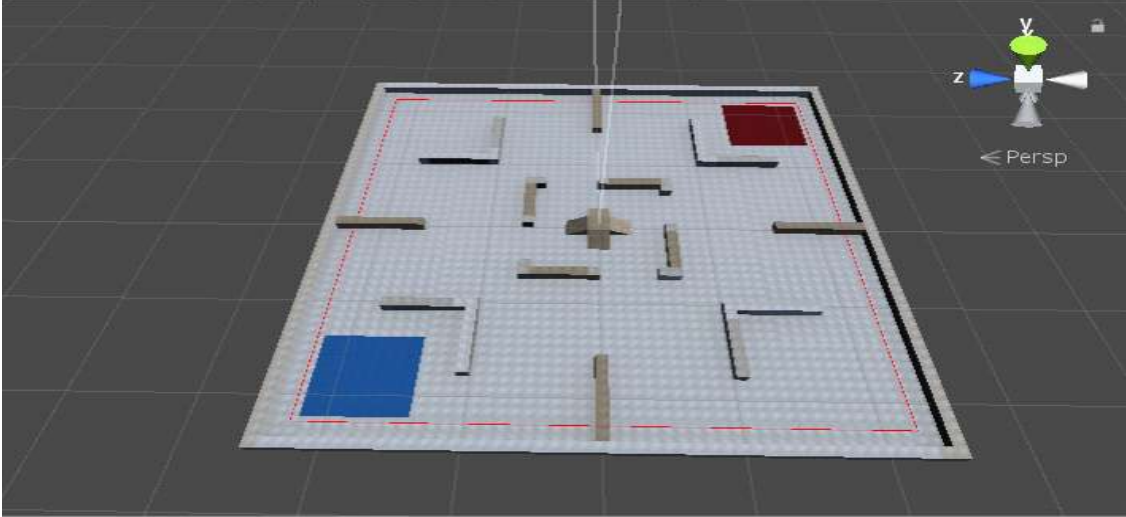
            // 해당 풀에서 인스턴스 생성
            return Pools[prefab].Spawn(position, rotation);
        }

        // 나중에 사용하기 위해 이전에 생성된 인스턴스를 비활성화
        // despawn 후만큼 지연시키기 위해 선택적으로 시간 값을 사용
        public static void Despawn(GameObject instance, float time = 0.0f)
        {
            if (time > 0)
            {
                GetPool(instance).Despawn(instance, time);
            }
            else
            {
                GetPool(instance).Despawn(instance);
            }
        }

        // 풀링된 객체를 빠르게 조회할 수 있는 편리한 방법
        // 인스턴스가 발견된 풀 구성 요소를 반환
        public static Pool GetPool(GameObject instance)
        {
            // 풀로 이용하여 인스턴스 찾기
            foreach (GameObject prefab in Pools.Keys)
            {
                if (Pools[prefab].active.Contains(instance))
                {
                    return Pools[prefab];
                }
            }

            // 풀에서 인스턴스를 찾을 수 없습니다
            Debug.LogError("PoolManager 풀에서 인스턴스를 찾을 수 없음: " + instance.name);
            return null;
        }
    }
}

```

	<p>3. 프로토타입 맵 제작</p> <p>실제 게임에서 사용될 맵을 구현하는 것은 기획과 레벨디자인이 필요하기 때문에 신중하게 설계하고 제작하여 사용자들에게 서비스를 해야 한다. 맵이 없으면 테스트 또는 프로토타입을 제작하기에 어려움이 있다. 그래서 기본적으로 대전에 어울리는 맵을 제작해야한다. 실제 사용할 맵이 아니기 때문에 기존 대전 게임들의 맵을 참고로 하여 제작한다. 유튜브를 통하여 게임 들을 찾아보았다. 생각보다 게임을 찾는데 오랜 시간이 걸렸다. 금방 찾을 줄 알았지만 규모가 작은 맵 보다는 규모가 어느 정도 되는 맵들만 있었다. 방향을 바꾸어 게임에 있는 맵을 찾지 않고 서바이벌 게임장의 맵이나 놀이터를 위주로 찾아보았다. 서바이벌 게임장의 구조 중 괜찮은 것이 있어서 그것을 토대로 프로토타입 맵을 제작하였다.</p> 
<p>참고자료 및 문헌</p>	<ul style="list-style-type: none"> - 절대강좌! 유니티 - https://www.slideshare.net/agebreak/ss-75060410 (유니티 오브젝트 풀링) - 구글링 및 유튜브
<p>학습방법</p>	<p>처음으로 오브젝트 풀을 제작하는 것이므로 구현하기 이전에 학습을 해야 한다. 절대강좌! 유니티 문헌에도 오브젝트 풀은 존재하지만 오브젝트 풀 매니저를 어떻게 효율적으로 설계하는 방법에 대해서는 없으므로 구글링을 통해 많은 풀 매니저 설계 방법을 찾아 자료를 수집하고 학습하여 구현에 도움을 받았다.</p>
<p>학습성과 및 목표달성도</p>	<ul style="list-style-type: none"> - 오브젝트 풀 구현 완료(100%) - 풀 매니저 구현 완료(100%) - 프로토타입 맵 제작 완료(100%)
<p>내주 계획</p>	<p>다음 주에 창의설계경진대회가 진행된다. 따라서 실제 게임이 다 구현되어 있지 않기 때문에 프로토타입을 개발하기 위해 프로토타입용 이펙트를 제작하고 대회에 전시할 패널과 게임 시연 동영상을 제작해야한다.</p>

지도교수

(인)