TODO

어제 못했었던것

컬링, TMP 적용!

풀링

- 오브젝트를 미리 로드해서 비활성화 해놓기.
  - 어디에? - Root 만들어서 들고있기
  - 어떻게? - push, pop

이 오브젝트가 풀링해야하나?

Poolable 컴포넌트 만들어서 붙어있음 풀링하게!

그저 bool 변수 하나만 들고있음 풀링이 되었나? T/F

Push - 풀에넣기 — Poolable 받아서 확인

Pop - 풀에서 빼기. — original, parent - poolable 리턴

↳ pool? — original └ 원본, parent └ 부모

```
GameObject _root;          ← 풀을 들고있을 GO
public void Init()
{
    if(_root == null)
    {
        _root = new GameObject { name = "@Pool_Root" };
        Object.DontDestroyOnLoad(_root);
    }
}

public void Push(Poolable poolable)          ← 풀에 poolable 들고있는 오브젝트넣기
{
    string name = poolable.gameObject.name;
    if(name == null) { GameObject.Destroy(poolable.gameObject); return; }
}

public Poolable Pop(GameObject original, Transform parent = null)
{
    return null;          풀에서 original을 찾아보고 부모있으면 상속해서 보내기
}

public GameObject GetOriginal(string name)
{
    return null;          로드된 원본이있나 확인
}
```

문제점. 풀링해야하는 오브젝트가 2개이상이면?

@Pool_Root
└ Monster
└ Monster
└ Item
└ ⋮

해결법. 1. 꺼내올때마다 Root 순회
풀링이 많아지면..? 배보다 배꼽이 큼

2. 객체마다 풀 만들기.
관리가 귀찮을거 같음.

3. 하나의 풀에 객체별분류하기 (빈오브젝트로)

3번 해결법

Root에 오브젝트별로 분류하기.

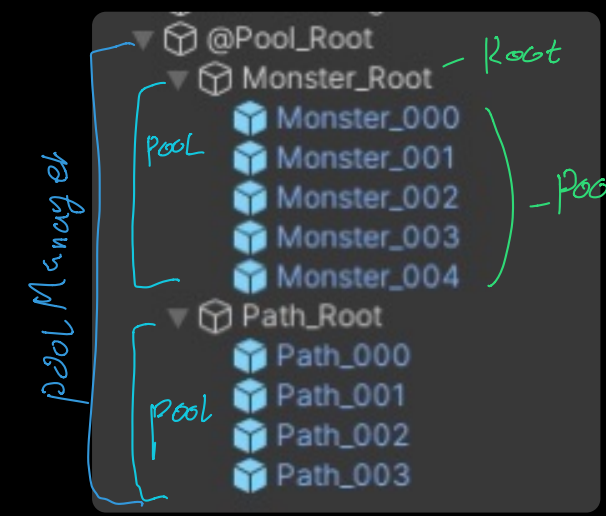
Root
└ Monster — 이걸 검색해서 참조
  └ Monster_1 — 딕셔너리!
  ⋮
└ Item — 진짜 Pool 클래스
  └ Item

```
class Pool { }
Dictionary<string, Pool> _pool = new Dictionary<string, Pool>();

GameObject _root;
public void Init()
{
    if (_root == null)
    {
        _root = new GameObject { name = "@Pool_Root" };
        Object.DontDestroyOnLoad(_root);
    }
}

public void Push(Poolable poolable)
{
    string name = poolable.gameObject.name;
    if (name == null) { GameObject.Destroy(poolable.gameObject); return; }
}

public Poolable Pop(GameObject original, Transform parent = null)
{
    return null;
}

public GameObject GetOriginal(string name)
{
    return null;
}
```

이런식으로 관리!

딕셔너리로접근 (pool)

```
class Pool {
    // 원본
    public GameObject Original { get; private set; }
    // 모아둘곳
    public Transform Root { get; set; }
    // POOL
    Stack<Poolable> _poolStack = new Stack<Poolable>();

    // 초기화
    public void Init(GameObject original, int count = 10)
    {
        Original = original;                                    // 원본을 저장하고
        Root = new GameObject().transform;                      // Pool_Root를 만들고
        Root.name = $"{original.name}_Root";

        for(int i=0;i< count;i++)                               // count 만큼 풀링
        {
            Push(Create());
        }
    }
    // 복제
    Poolable Create()
    {
        GameObject go = Object.Instantiate<GameObject>(Original);   // 원본 복사해서
        go.name = Original.name;                                    // 이름바꾸고
        return Util.GetOrAddComponent<Poolable>(go);                // Poolable 반환 없음 달아주기
    }

    // POOL에 넣기
    public void Push(Poolable poolable)
    {
        if (poolable == null) return;                           // 풀링 대상이 아님

        poolable.transform.parent = Root;                       // Root 상속받고
        poolable.gameObject.SetActive(false);                   // 숨기기
        poolable.IsUsing = false;                               // 사용안하는중! 표시


        _poolStack.Push(poolable);                              // 풀에 넣기
    }

    // POOL 에서 빼오기
    public Poolable Pop(Transform parent)
    {
        Poolable poolable;

        if (_poolStack.Count > 0)                               // 풀스택에 있으면
            poolable = _poolStack.Pop();                        // 빼오기
        else
            poolable = Create();                                // 없으면 만들기

        poolable.gameObject.SetActive(true);                    // 사용! 게임상에 나타남

        poolable.transform.parent = parent;                     // 사용할 곳 에 붙기
        poolable.IsUsing = true;                                // 사용중! 표시

        return poolable;
    }
}
```



Monster Prefab을 Original에 저장

① Pool_Root

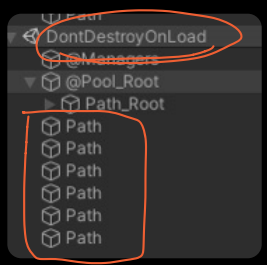
1. Create - Pool 만들어서 등록
2. Push - 해당 Pool에 넣기
3. Pop - 해당 Pool에서 빼기
4. Clear - 풀 상태 초기화
5. FindOriginal - 이름으로 원본찾기

```
    // POOL을 관리할 POOLMANAGER
    Dictionary<string, Pool> _pool = new Dictionary<string, Pool>(); // <Pool.Original.Name , Pool>

    // Pool을! 모아둘곳
    Transform _root;
    public void Init()
    {
        if (_root == null)
        {
            _root = new GameObject { name = "@Pool_Root" }.transform;   // Pool을 모아둘 Root 생성
            Object.DontDestroyOnLoad(_root);
        }
    }
    // 풀생성
    public void CreatePool(GameObject original, int count = 10) {
        Pool pool = new Pool();
        pool.Init(original, count); // Pool 만들고
        pool.Root.parent = _root;   // _root에 상속시키기

        _pool.Add(original.name, pool); // 딕셔너리 등록
    }

    // 해당 풀에 넣기
    public void Push(Poolable poolable)
    {
        string name = poolable.gameObject.name;                             // 키 가져오기
        if (_pool.ContainsKey(name) == false) { GameObject.Destroy(poolable.gameObject); return; }  // 풀이 없으면 삭제

        _pool[name].Push(poolable);     // 풀이 있으면 풀에 push
    }

    // 해당 풀에서 빼오기
    public Poolable Pop(GameObject original, Transform parent = null)
    {
        if (_pool.ContainsKey(original.name) == false)
            CreatePool(original);   // 풀없으면 만들기

        return _pool[original.name].Pop(parent); // 해당 풀에 접근해 pop호출하기
    }

    // 풀의 원본 찾기
    public GameObject GetOriginal(string name)
    {
        if (_pool.ContainsKey(name) == false)
            return null;
        return _pool[name].Original;
    }

    public void Clear()
    {
        foreach (Transform child in _root)
            GameObject.Destroy(child.gameObject);

        _pool.Clear();
    }
```

```
public class ResourceManager
{
    public T Load<T>(string path) where T : Object
    {
        // 원본 들고있으면 바로 사용
        if (typeof(T) == typeof(GameObject)) // 프리팹?
        {
            string name = path;
            int index = name.LastIndexOf('/');
            if (index >= 0)
                name = name.Substring(index + 1);

            GameObject go = Managers.Pool.GetOriginal(name);
            if (go != null)
                return go as T;
        }
        return Resources.Load<T>(path);
    }

    public GameObject Instantiate(string path, Transform parent = null)
    {
        GameObject original = Load<GameObject>($"Prefabs/{path}");
        if (original == null)
        {
            Debug.Log($"Failed to load prefab : {path}");
            return null;
        }

        // 풀에서 찾기
        if (original.GetComponent<Poolable>() != null)
            return Managers.Pool.Pop(original, parent).gameObject;

        GameObject go = Object.Instantiate(original, parent);
        go.name = original.name;
        return go;
    }

    public void Destroy(GameObject go)
    {
        if (go == null)
            return;

        // 풀에 보존하기
        Poolable poolable = go.GetComponent<Poolable>();
        if (poolable != null)
        {
            Managers.Pool.Push(poolable);
            return;
        }

        Object.Destroy(go);
    }
}
```

원하는 프리팹에 Poolable 달아준사용!

몰자!



삭제안됨!

해제방법.

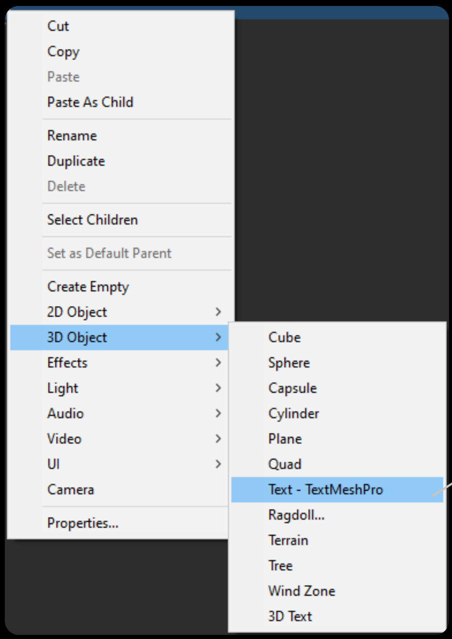
DontDestroyOnLoad 씬에 들어가기.

```
// DontDestroyOnLoad 해제
if (parent == null)
    poolable.transform.parent = Camera.main.transform;
```

항상 존재하는것.

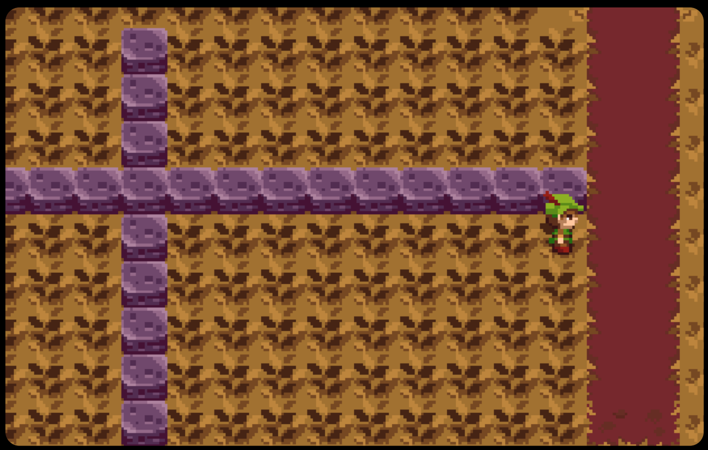
TMP(Text Mesh Pro)

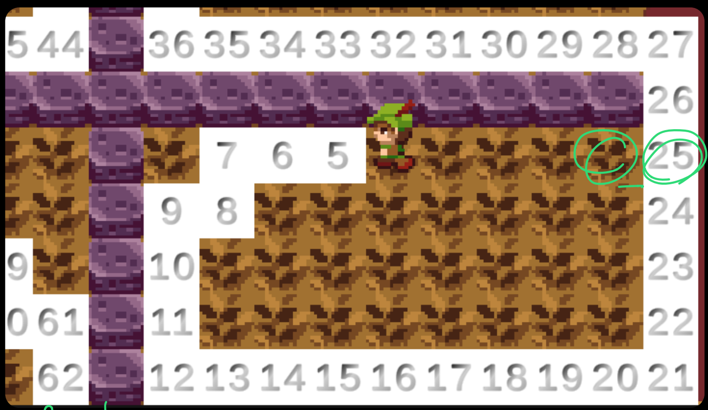
UI가 아닌 오브젝트에 Text 쓰기



추가.

Text Mesh Pro 컴포넌트에서 관리.





누적

저들을 잇는 획기적인 G+H 구하는 방법 없나?

```
214   -   open[dest.Y, dest.X] = g + h;
```

dest를 Next로

하는길 에

배열참조 → 딕셔너리!

3차원 → 2차원으로!

```
if (Util.TryAdd<Pos, int>(openList, next, g + h) == false)
    openList[next] = g + h;
```

```
        // 갈수있나 (충돌할 물체가 있나)
        public bool CanGo(Vector3Int cellPos)
+       {
+           return CanGo(new Vector2Int(cellPos.x, cellPos.y));
+       }
+       public bool CanGo(Vector2Int cellPos)
        {
            if (cellPos.x < MinX || cellPos.x + 1 > MaxX)
                return false;
@@ -97,6 +101,31 @@ public struct Pos
            public Pos(int y, int x) { Y = y; X = x; }
            public int Y;
            public int X;
+
+           public static bool operator ==(Pos l, Pos r)
+           {
+               return l.X == r.X && l.Y == r.Y;
+           }
+           public static bool operator !=(Pos l, Pos r)
+           {
+               return !(l == r);
+           }
+
+           public override bool Equals(object obj)
+           {
+               return (Pos)obj == this;
+           }
+
+           public override int GetHashCode()
+           {
+               long value = (Y << 32) | X;
+               return value.GetHashCode();
+           }
+
+           public override string ToString()
+           {
+               return base.ToString();
+           }
        }

        public struct PQNode : IComparable<PQNode>
@@ -115,28 +144,32 @@ public int CompareTo(PQNode other)
        }

        // cell에서 pos로
-       Pos Cell2Pos(Vector3Int cell)
```

연산자 오버로딩!