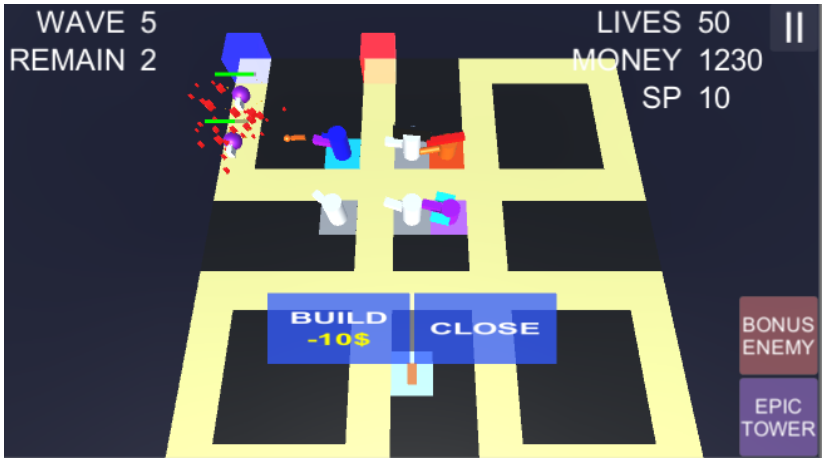
Random Tower Defense

남정웅

다운로드(GitHub)주소 : https://github.com/hehza90/-Unity-RandomTowerDefense.git



카메라 이동

1. 상하좌우
PC : 마우스 가장자리 위치 or 키보드 wasd
모바일 : 화면 드래그

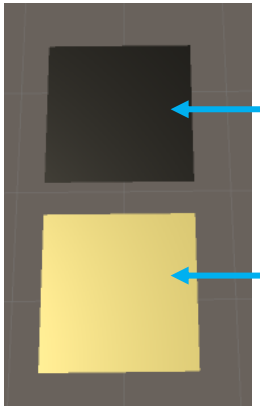
2. 줌 In& Out
PC : 마우스 휠
모바일 : 2개의 터치 거리 늘리기& 좁히기

Index

1. Xml을 이용한 맵 만들기

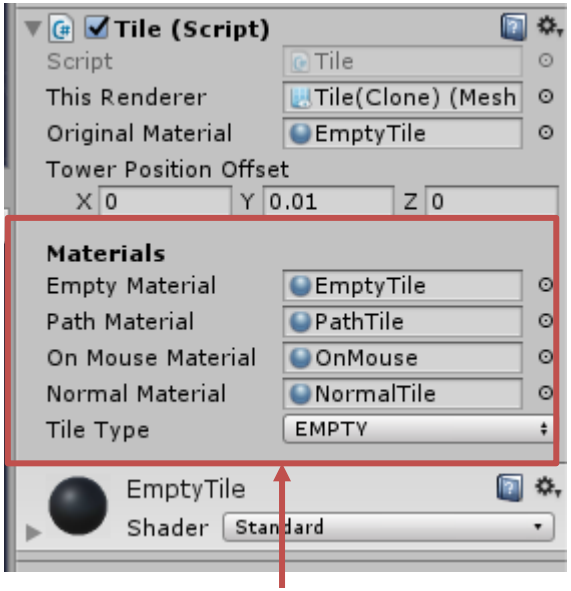
1) 타일

타일의 Type

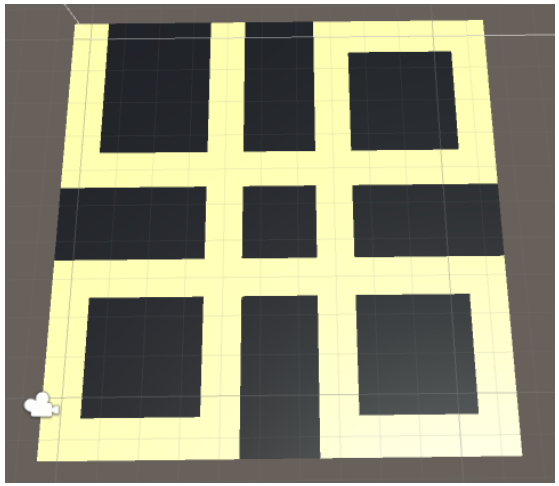


어두운 색 타일 –
타워를 지을 수 있는 타일

밝은 색 타일 –
적이 지나가는 경로



타일의 타입은 하나의 스크립트에서
enum으로 구별 타입에 따른 색상도 미리 할당

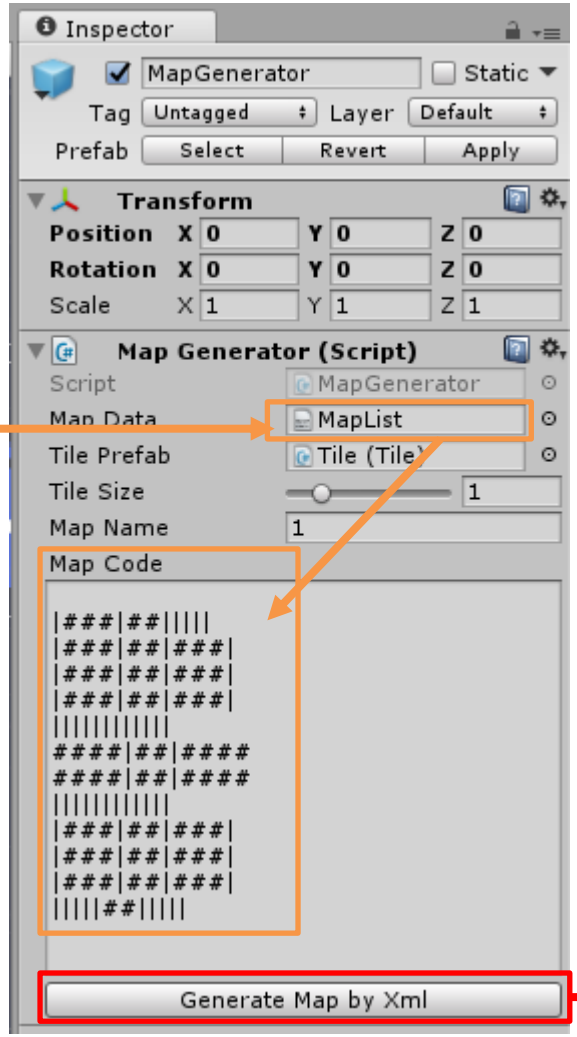


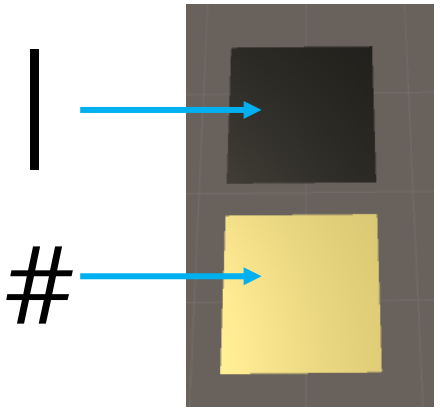
이 같은 맵을 만들기 위해서
타일의 프리팹을 Scene뷰에 직접 놓지 않고
다음 페이지에서 설명할 Map Generator을 이용하여 배치

2) class MapGenerator 활용한 타일 설정& 배치

Xml 데이터

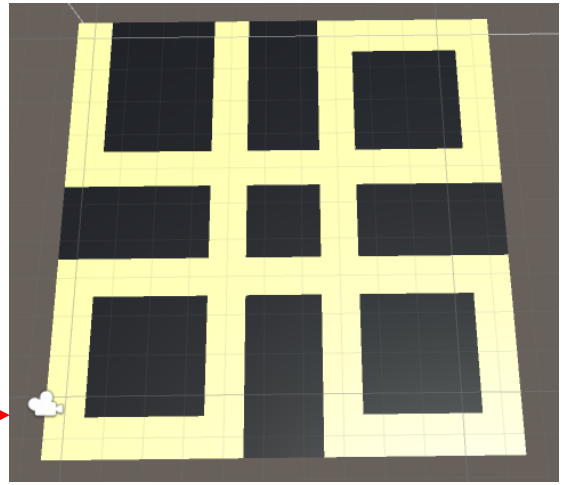
```
<MapList>
  <Map Name ="1">
    <MapCode>
|###|##|||||
|###|##|###|
|###|##|###|
|###|##|###|
||||||||||||
####|##|####
####|##|####
||||||||||||
|###|##|###|
|###|##|###|
|###|##|###|
|||||##|||||
    </MapCode>
  </Map>
```





생성된 맵

Click



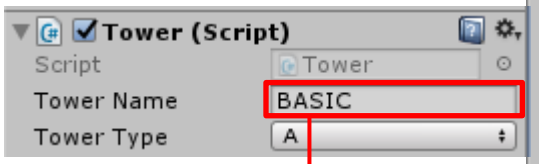
타일에 위치 값이나 타입 등을 일일이 지정하면 시간이 많이 걸리므로
에디터상에서(게임 실행 필요 없이) 버튼(Generate Map by Xml)을 누르면
Xml에서 값을 받아와 자동으로 위치, 타입, 타입에 따른 색상이 지정된 맵 생성

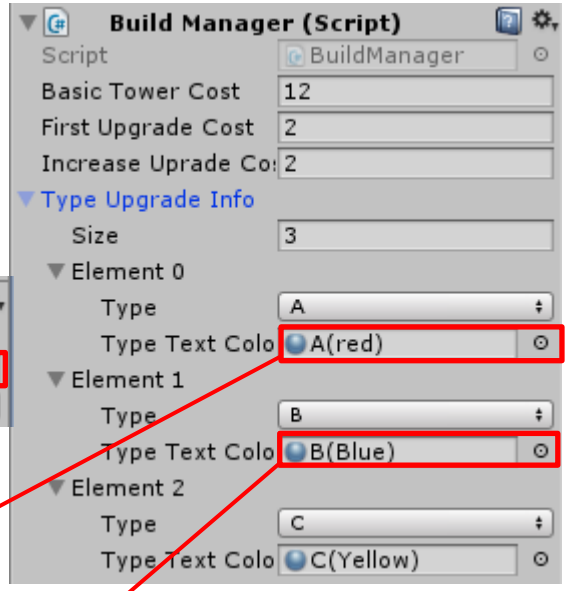
2. Class BuildManager

1) 타일 메뉴 UI & 기본 타워 짓기

타워의 이름 텍스트는 타입에 따라 자동으로 변경
(타워의 이름은 해당 타워 프리팹 한 곳, 타입색상은 BuildManager 한 곳에만 지정하면 됨)

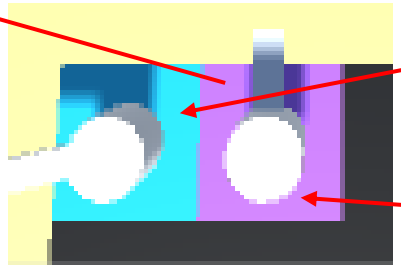
BUILD 클릭 시 1레벨 타워 중 랜덤으로 선정하여 생성

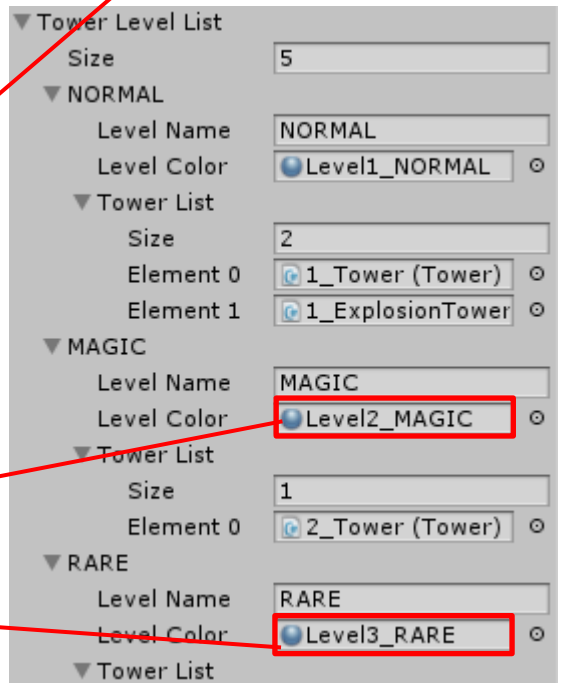




2) 타워 메뉴 UI

대상 타워 설정, 및 대상 타워의 정보를 바탕으로 UI의 텍스트 갱신

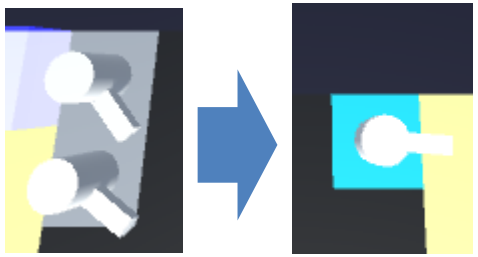




레벨 업 버튼과 레벨을 나타내는 색상 BuildManager에 지정된 색상으로 자동 설정
(타워의 레벨은 인스펙터뷰의 TowerLevelList의 인덱스로 자동 설정)

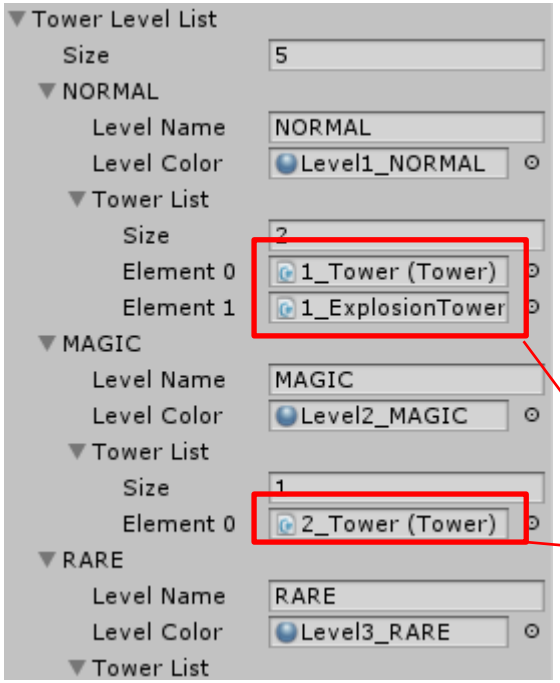
3) 타워 레벨 업 규칙

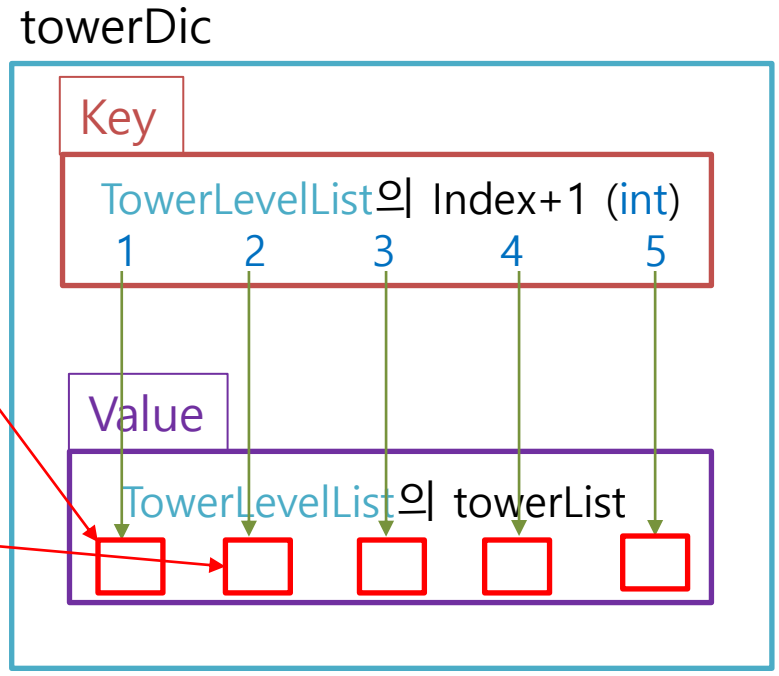
1. 같은 타워가 2개 있을 경우 레벨 업 가능



Awake()에서 다음 레벨의 타워 목록을 불러오기 위해
BuildManager에서 Dictionary을 활용하여 타워 목록 저장

```
//레벨로 타워 목록을 검색하기 위한 Dic
private Dictionary<int, Tower[]> towerDic = new Dictionary<int, Tower[]>();
```





클릭시 BuildManager에서
(1) 같은 타워 검색 (타워를 만들 때 마다 별도의 List에 참조& 타워 프리팹의 종류 마다 고유 ID 지정)
(2) 존재한다면 towerDic에서 다음 레벨의 타워 목록에 접근해 랜덤으로 하나 선택
(3) 같은 타워 2개를 삭제하고 선택된 타워 생성

2. 타입이 같다면 업그레이드 공유 (스타크래프트의 업그레이드 시스템과 유사)

같은 타입이라면 이름의 텍스트 색상이 같으며 하나를 업그레이드 수치도 공유

Q) 어느 타워에서 업그레이드를 하면 같은 타입의 다른 타워들에게 어떻게 알릴 것인가?

*버튼을 누르는 즉시 해당 타입의 모든 타워에 증가된 데미지가 적용 되어야 함

A) 콜백(아래의 빨간 밑줄)과 Dictionary활용

class BuildManager의 class TypeUpgradeInfo

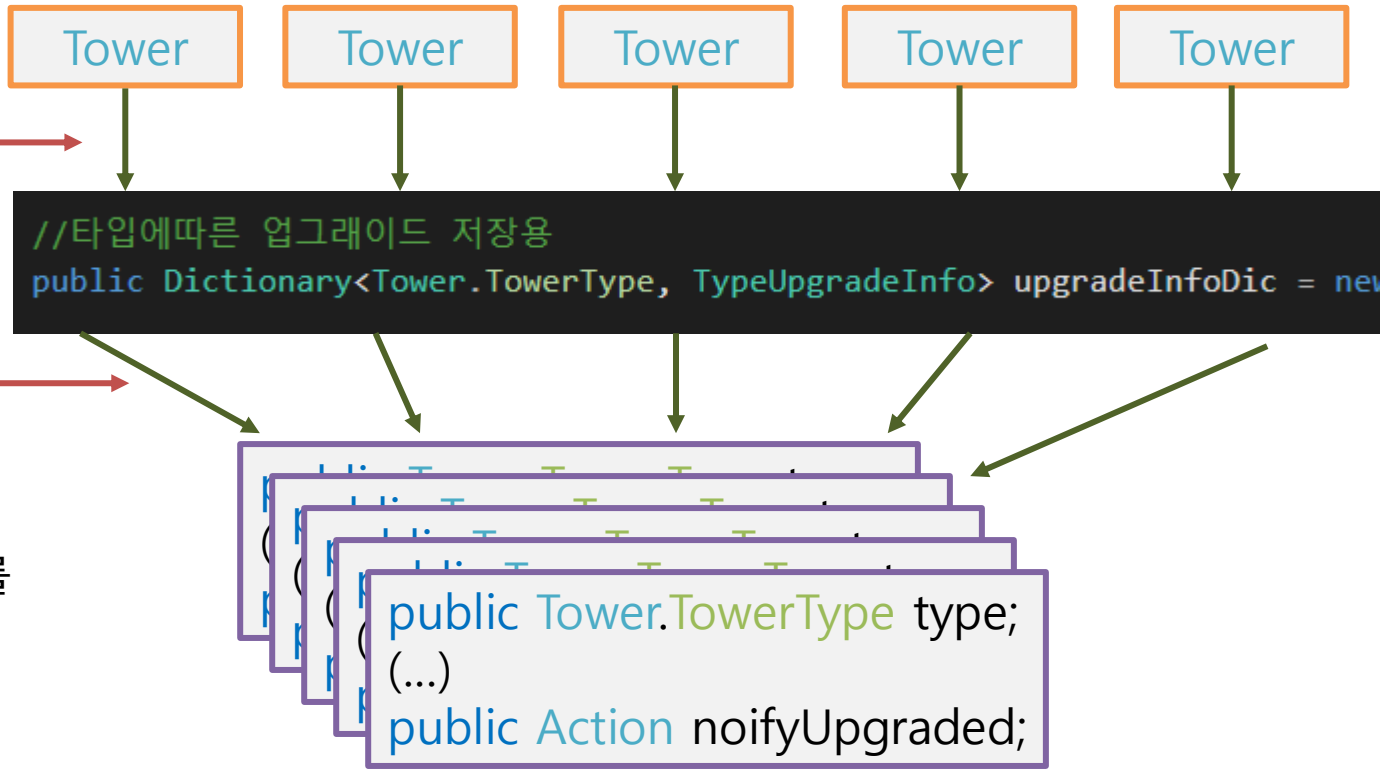
```
//타입에 따른 색상 및 업그레이드 정보  upgradeInfoDic 의 Value
[Serializable]
public class TypeUpgradeInfo
{
    public Tower.TowerType type;        //타입 지정  upgradeInfoDic의 Key값
    public Material typeTextColor;      //타입이름 텍스트 색상 변경용
    [NonSerialized]
    public int upgradeLevel=0;          //레벨
    [NonSerialized]
    public float nextUpgradeCost;       //다음 업그레이드 비용

    public Action noifyUpgraded;        //해당 타입의 타워들에게 알림
}
```

콜백과 Dictionary활용 이해 해 보기

타워가 만들어질 때

(1) 만들어진 타워에서 noifyUpgraded의 Key인 TowerType의 값을 전달하여 upgradeInfoDic의 TypeUpgradeInfo(TowerType이 같은)에 접근



(2) 타워의 업그레이드 수치에 따른 데미지 계산 함수를 noifyUpgraded에 등록

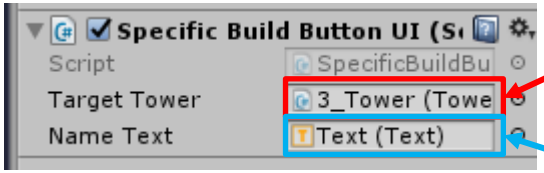
public TypeUpgradeInfo[] typeUpgradeInfo;

(3) 업그레이드 버튼을 누르면 대상 타워의 TowerType값이 같은 TypeUpgradeInfo에서 noifyUpgraded가 실행되어 이전 과정에서 (각 타워에서)등록된 데미지 계산 함수 실행

4) 특정 타워 짓기

1. 규칙 : 보스를 잡을 때 얻는 특별 재화(SP)로 구입

버튼UI의 스크립트



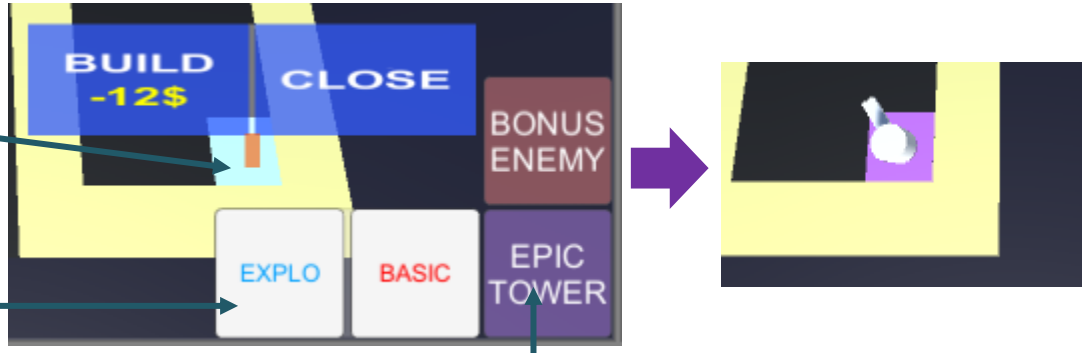
타워의 프리팹 할당

타워의 이름과 타입에 따라 텍스트 자동 변경 (전 페이지의 타워 메뉴 UI에서와 같은 원리)

2. 순서

(2) 대상 타일 클릭

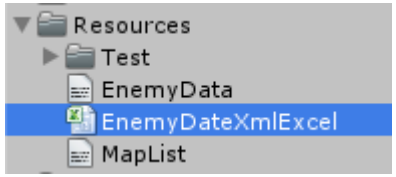
(3) 원하는 타워 클릭 시 타워 생성
(텍스트의 내용과 색상은 할당된 타워의 종류에 따라 자동으로 변경)



(1) 타워 메뉴 열기
(클릭 시 여닫기 가능)

3. Class WaveSpawner

1) Xml과 Excel 연동으로 데이터 편집

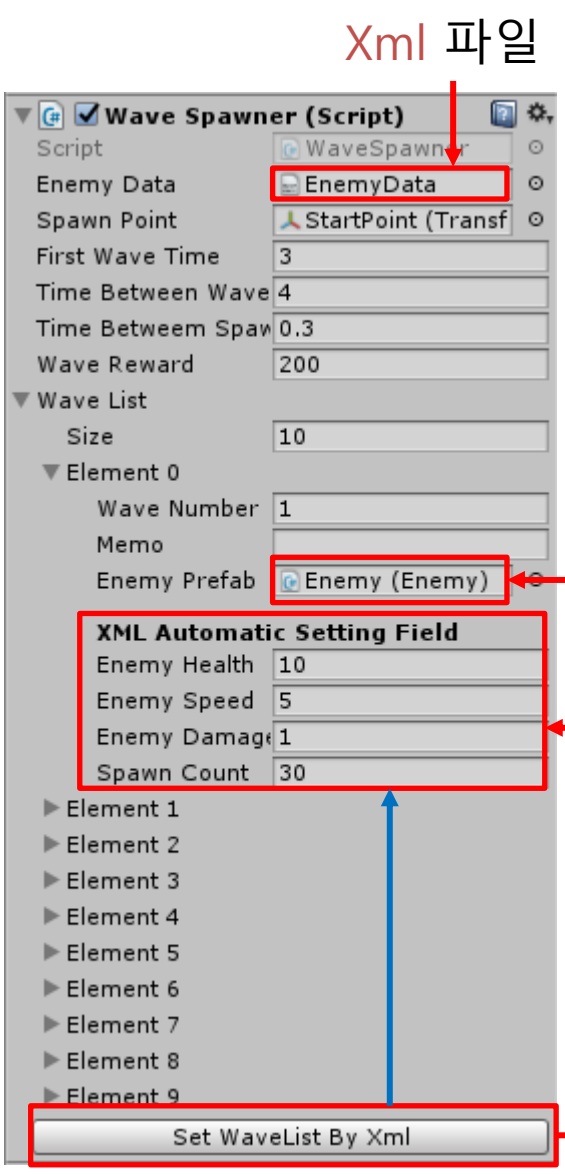


	A	B	C	D	E
1	WaveNumber	Health	Speed	Damage	SpawnNumber
2	1	10	5	1	30
3	2	15	5	1	30
4	3	20	5	1	30
5	4	30	5	1	30
6	5	40	5	1	30
7	6	50	5	1	30
8	7	60	5	1	30
9	8	70	5	1	30
10	9	80	5	1	30
11	10	2000	5	15	1

```
<EnemyStatList xmlns:xsi="http://www.
  <Wave WaveNumber="1">
    <Health>10</Health>
    <Speed>5</Speed>
    <Damage>1</Damage>
    <SpawnNumber>30</SpawnNumber>
  </Wave>
  <Wave WaveNumber="2">
    <Health>15</Health>
    <Speed>5</Speed>
    <Damage>1</Damage>
    <SpawnNumber>30</SpawnNumber>
  </Wave>
  <Wave WaveNumber="3">
    <Health>20</Health>
    <Speed>5</Speed>
    <Damage>1</Damage>
    <SpawnNumber>30</SpawnNumber>
  </Wave>
  <Wave WaveNumber="4">
    <Health>30</Health>
    <Speed>5</Speed>
    <Damage>1</Damage>
    <SpawnNumber>30</SpawnNumber>
  </Wave>
  <Wave WaveNumber="5">
    <Health>40</Health>
    <Speed>5</Speed>
    <Damage>1</Damage>
    <SpawnNumber>30</SpawnNumber>
```

(1) 메모장에서 Xml의 스키마(열)작성 후 Excel로 가져온 후 저장
(2) Excel에서 편집하면 변경사항이 Xml 데이터에 자동 적용
(3) Excel에서 추가 Element(행) 추가 가능
(위 그림에서는11번 이상의 WaveNumber)
(4) 필요하다면 추가 스키마도 생성 가능
(스키마 : Health, Speed등 외 위 그림에 없는 다른 필드)

2) Xml 데이터 적용



이러한 방법의 장점

(1) Inspector뷰 대신 Excel을 통해 여러 적의 능력치를 간편하게 편집가능
(2) 능력치만 다른 같은 적을 사용할 때 같은 프리팹 재사용 가능
(3) 그와 더불어 오브젝트 풀을 활용하기도 용이함

WaveSpawner의 Inspector뷰에서는 적의 프리팹만 직접 할당
(적 프리팹의 필드(능력치)도 따로 설정하지 않음)

적의 능력치는 적이 활성화 된 직후에 적용
(오브젝트 풀링 활용)

*class ObjectPoolingManager에 대한 설계는 개별 문서 참고

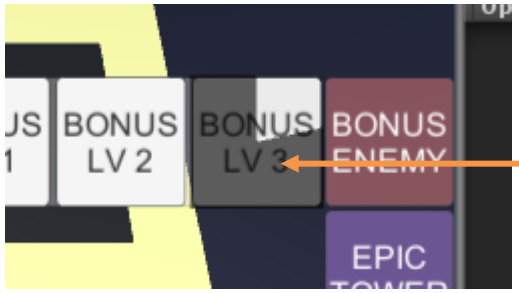
클릭 시 Xml의 데이터(Excel로 편집한)를 WaveList에 적용
(1개의 행 = WaveList의 1개의 Index)

3) Wave 진행

(1) 타이머가 다 지나면 다음 웨이브

웨이브의 모든 적을 처리시 지정된
시간만큼 타이머 다시 작동

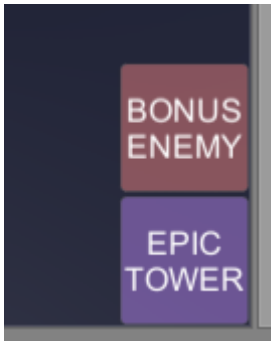
(2) 보너스 적은 대기시간 중에만 생성 가능



재사용 대기시간 타이머

재사용 대기시간 값은 BonusEnemy의 필드에서 가져옴
(버튼에 대해 따로 재사용 대기시간 값을 넣어 줄 필요 없음)

(3) 메뉴가 닫혀 있을 때



- Update()에서의 연산을 최소화 하기 위해 메뉴가 닫혀 있을 때는 시계 방향으로 줄어드는 검은색 이미지에 대한 처리를 생략 (메뉴가 닫혀서 버튼이 보이지 않아 불필요 하므로)
- InvokeRepeating() 함수로 남은 시간에 대한 처리만 함 (코루틴과 달리 InvokeRepeating()은 비활성화 상태에서도 작동)
- 메뉴를 열 때와 열려 있을 때만 계산된 시간 만큼 검은색 이미지 처리

4. Inteface를 이용한 피격 처리

inteface IDamageable

```
public interface IDamageable
{
    void TakeDamage(float damage);

    void Die();
}
```

if (target.tag == "Enemy"){...} 와 같이 문자열(tag의 string을 비교하여)로 대상을 판단하는 대신
대상 오브젝트에 IDamgeable의 유무로 데미지를 받을 수 있는 대상인지 판단)

class Projectile에서 데미지를 주는 함수

```
//데미지 주기
protected void Damage(GameObject target)
{
    //데미지를 줄수 있는 대상인지 검사
    IDamageable damegeable = target.GetComponent<IDamageable>();
    if (damegeable != null)
    {
        damegeable.TakeDamage(this.damage);
    }
}
```

타겟의 tag나 layer의 명칭으로 판단하는 것과 비교한 위 방법의 장점

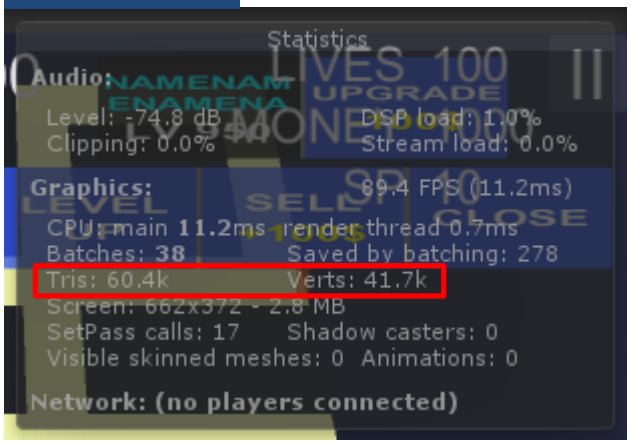
(1) string을 사용하지 않으므로 성능향상을 기대할 수 있고
(2) 타겟에 대해서는 Idamgeable의 유무만 판단하면 되므로 새로운 유형의 오브젝트를 추가하더라도 데미지를 주는 함수에서는 수정을 할 필요가 없으며 (tag나 layer 사용시 일일이 조건에 tag나 layer명칭을 추가해 줘야 한다)
(3) 새로운 유형의 오브젝트의 스크립트에 Idamgeable을 상속하고 이의 함수만 재정의 하면 된다

5. 컴파일링 및 최적화

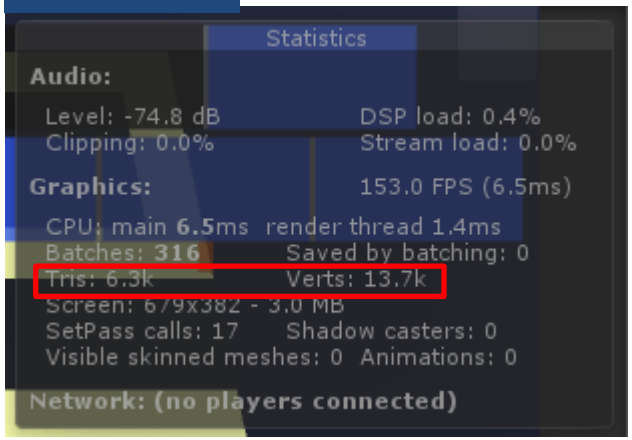
1) Triangle와 Vertex 줄이기

타일의 Mesh를 Plane에서 cube로 수정

수정 전



수정 후

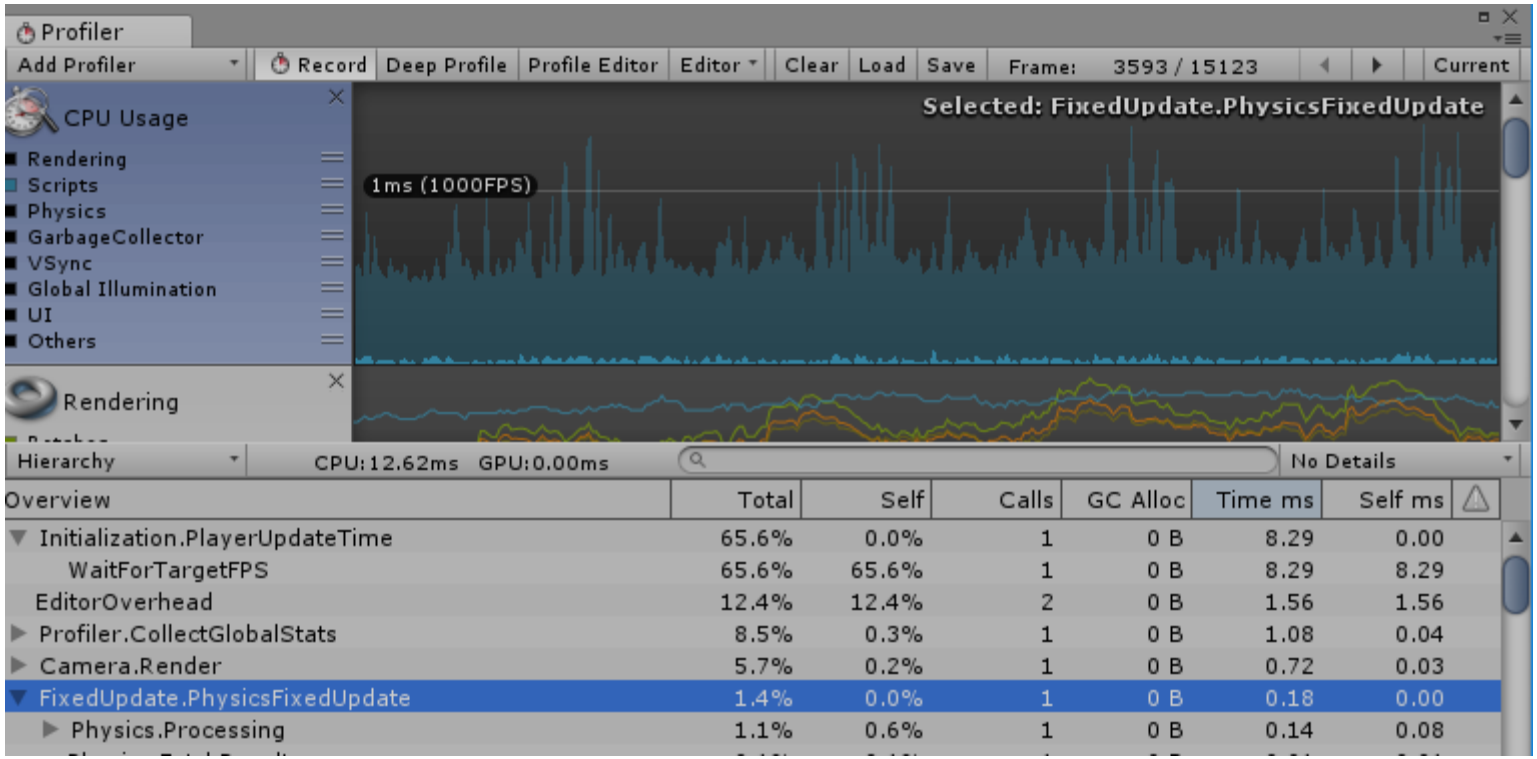


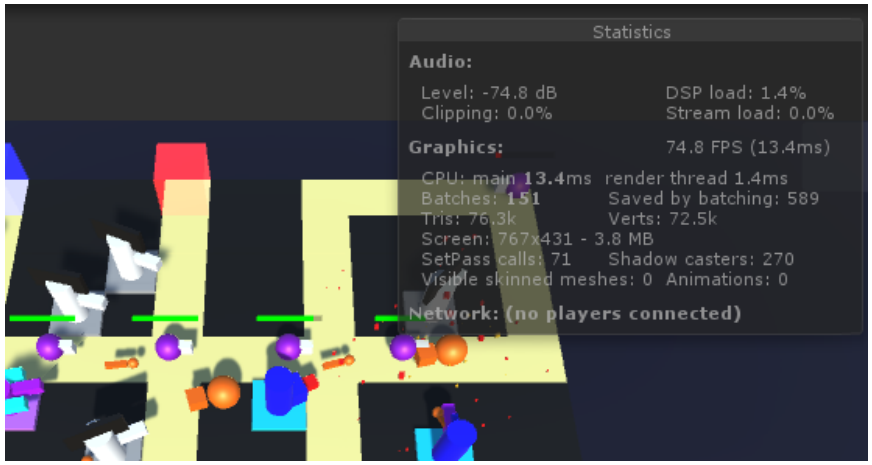
Triangle의 수 : 60.4k → 6.3k

Vertex의 수 : 41.7k → 13.7k로

GPU 최적화 및 전반적인 성능 향상

2) 성능 측정





스크립트 최대 실행 시간 : 1.5ms 이하
프레임 : 70~80 FPS

퍼포먼스의 문제는 발생하지 않음