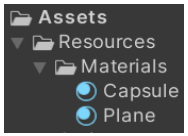
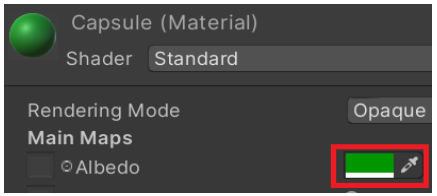
Unity
3D Dodge

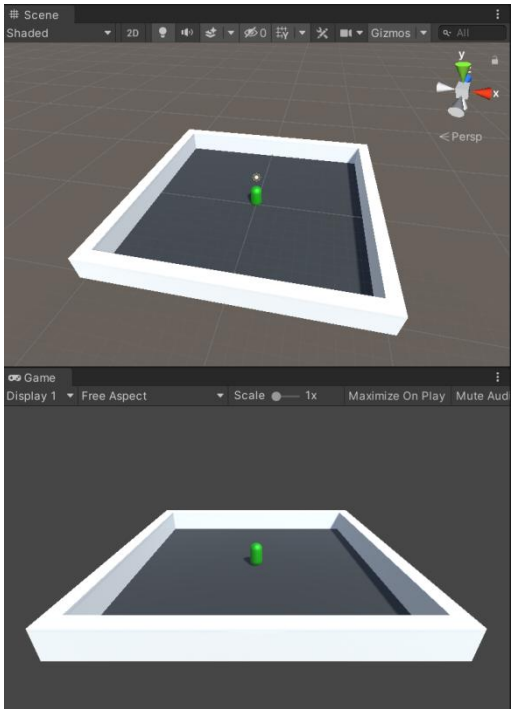
참고:레트로의 유니티 게임 프로그래밍 에센스

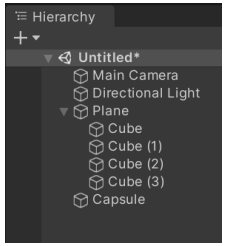
월드, 플레이어 배치

- Material
- Create=>Material x2
  - Name : Capsule
    - Color : RGB(0, 150, 0)
  - Name : Plane
    - Color : RGB(40, 40, 40)









- 메인 카메라
  - Transform
    - Position : 0, 20, -20
    - Rotation : 50, 0, 0
  - Camera
    - Clear Flag : Solid Color
    - Background : RGB(70, 70, 70)

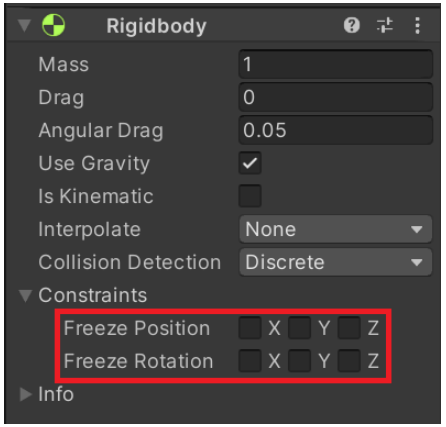
- Plane(필드)
- 3D Object=>Plane
  - Transform
    - Scale : 2, 1, 2
  - Mesh Renderer
    - Materials : Plane material로 변경

- Cube(외벽)
- 3D Object=>Cube x4
- Plane의 자식 계층으로
  - Transform
    - local Scale : 0.5, 2, 10.5
    - local Position : (5/-5, 1, 0)
      (0, 1, 5/-5)

- Capsule(플레이어)
- 3D Object=>Capsule
  - Transform
    - Position :0, 1, 0
  - Mesh Renderer
    - Materials : Capsule material로 변경

플레이어 이동

- **Player 스크립트**
- **플레이어의 움직임 제어**를 위하여 **Rigidbody 컴포넌트를 추가** 해야 한다
- 단순 위치 변경이 아닌 **오브젝트의 이동 처리**는 **Transform보다 Rigidbody를 이용**하는 것이 좋다
- RequireComponent : **특정 컴포넌트**가 반드시 필요할 경우 **자동으로 확인하여 추가**한다
- GetComponent()를 이용하여 컴포넌트의 유무를 확인 후 AddComponent()로 추가도 가능하다
- 물리 작용으로 **원하지 않는 이동, 회전이 발생**하는 것을 막기 위해 **Freeze Position, Rotation을 이용하여 제한**해 준다

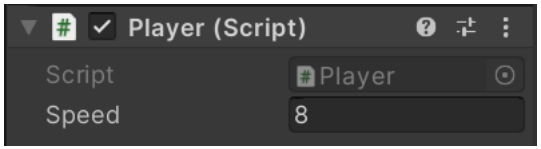



```
// 해당 컴포넌트가 없다면 자동으로 추가.
[RequireComponent(typeof(Rigidbody))]
public class Player : MonoBehaviour
{
    private Rigidbody rigid;

    void Start()
    {
        rigid = GetComponent<Rigidbody>();
        // position의 y값과 rotation의 x, z가 변경되지 않게 고정.
        rigid.constraints = RigidbodyConstraints.FreezePositionY | RigidbodyConstraints.FreezeRotationX
    | RigidbodyConstraints.FreezeRotationZ;
    }
}
```

플레이어 이동

- **Input 클래스**를 이용하여 플레이어의 **움직임을 제어**할 수 있다
- [SerializeField]를 이용하여 **private으로 지정한 변수**를 **Inspector 창에서 접근, 제어가 가능**하며 이때의 **실제 적용되는 값**은 스크립트에 지정된 값이 아닌 **Inspector에서 설정한 값**이다
- **물리를 이용한 이동** 등의 처리는 **Update()가 아닌 FixedUpdate()를 이용**하는 것이 좋으며, 이 경우 **speed에 Time.deltaTime을 곱하지 않아도 된다**
- speed와 Input 값을 계산, **velocity(속도)**를 구하여 rigid.velocity에 적용한다
- **Capsule** GameObject에 **Player 스크립트를 컴포넌트로 추가**

```
public class Player : MonoBehaviour
{
    [SerializeField] private float speed = 8f;
    private Vector3 velocity = Vector3.zero;

    void FixedUpdate()
    {
        velocity.x = Input.GetAxis("Horizontal") * speed;
        velocity.z = Input.GetAxis("Vertical") * speed;
        rigid.velocity = velocity;
    }
}
```



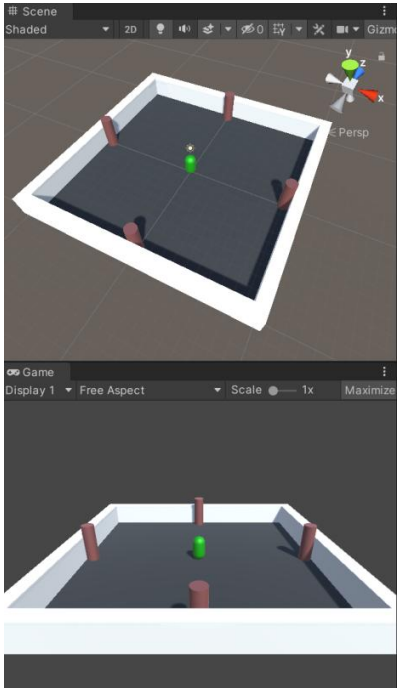
플레이어 피격

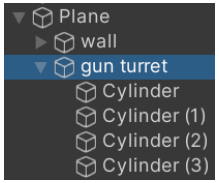
- **피격 확인**을 **탄환**에서 하여 **OnDamaged() 함수를 호출**하게 한다
- 플레이어가 죽으면 **게임 오버** 처리를 하기 위하여 **isLive 변수** 추가
- **플레이어는 게임 시작 시 반드시 살아있는 상태**가 되도록 Init()에서 **게임 오브젝트를 활성화** 한다

```
public class Player : MonoBehaviour
{
    public bool isLive { get { return gameObject.activeSelf; } }

    public void OnDamaged()
    {
        gameObject.SetActive(false);
    }

    public void Init()
    {
        velocity = Vector3.zero;
        gameObject.SetActive(true);
    }
}
```

포대, 탄환 배치

- Material
- Create=>Material x2
  - Name : Cylinder
    - Color : RGB(120, 60, 60)
  - Name : Sphere
    - Color : RGB(200, 100, 20)





- Cylinder
- 3D Object=> Cylinder x4
- Plane의 자식 계층으로
  - Transform
    - Scale : 0.5, 1.5, 0.5
    - Position : (4.5/-4.5, 1.5, 0)
      (0, 1.5, 4.5/-4.5)
  - Mesh Renderer
    - Materials : Cylinder material로 변경

- Sphere
- 3D Object=> Sphere
  - Mesh Renderer
    - Materials : Sphere material로 변경
  - Sphere Collider
    - Is Trigger : true

- Resources 폴더에 Prefabs 폴더 생성
- Sphere 오브젝트를 Prefabs 폴더로 드래그&드롭 하여 Prefab으로 만든다
- Hierarchy의 Sphere 제거

탄환의 충돌

- **Bullet 스크립트**
- 탄환은 **오브젝트 풀링(pooling)**을 할것이기 때문에 **생성과 함께 오브젝트 비활성화**로 변경
- **Sphere 프리팹**에 **Bullet 스크립트 추가**
- 탄환 발사를 위한 Rigidbody가 필요

```
[RequireComponent(typeof(Rigidbody))]
public class Bullet : MonoBehaviour
{
    private void Start()
    {
        // 오브젝트 풀링(pooling)을 사용할 것이므로 생성 후 비활성화.
        gameObject.SetActive(false);
    }
}
```

탄환의 충돌

- **Bullet 스크립트**
  - **Capsule의 Tag**를 **Player**로 변경, **Cylinder(x4)의 Tag**를 **Respawn**으로 변경
  - **플레이어와 충돌하면 이벤트가 발생**
  - **포대 또는 탄환끼리 충돌한 경우를 제외**하고 어디든 **충돌하면 탄환 오브젝트를 비활성화**


```
using System;
public class Bullet : MonoBehaviour
{
    public event Action EventHadleOnCollisionPlayer;

    private void OnTriggerEnter(Collider other)
    {
        var tag = other.tag;
        if (tag.Equals("Player"))//"Player" == other.tag
        {
            if (null != EventHadleOnCollisionPlayer) EventHadleOnCollisionPlayer();
        }
        else if (tag.Equals("Respawn") || other.name.Equals(name)) return;

        gameObject.SetActive(false);
    }
}
```

탄환 생성

GameMgr 스크립트

- 새 **GameObject**(GameManager)에 **GameMgr 스크립트 추가**
- **탄환**을 미리 일정 개수를 만들어 List에 저장하여 두고 사용(**오브젝트 풀링**)한다
- **탄환의 생성 주기**를 **랜덤**하게 하기 위해서 최소, 최대 생성 주기를 지정하도록 한다

```
public class GameMgr : MonoBehaviour
{
    public static GameMgr Instance { get; private set; }

    private Player player;
    GameObject[] turrets;

    private Bullet bulletPrefab;
    private List<Bullet> listBullet;

    [SerializeField] private float spwanRateMin = 0.3f;
    [SerializeField] private float spwanRateMax = 0.8f;
    private float spawnRate = 1f;
    private float checkTime = 0;

    private void Awake()
    {
        if (null == Instance)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);

            return;
        }

        Destroy(gameObject);
    }
}
```

탄환 생성

- **탄환과 플레이어 충돌 이벤트 처리**를 하기 위하여 **플레이어 스크립트를 찾는다**
- **Resources 폴더**에 있는 **Sphere(탄환)를 로드**하여 오브젝트를 생성(복제) 한다

```
public class GameMgr : MonoBehaviour
{
    void Start()
    {
        Init();
    }

    void Init()
    {
        spawnRate = 1f;
        spwanRateMax = 0.8f;

        player = FindObjectOfType<Player>();
        if (!player) player.Init();

        bulletPrefab = Resources.Load<Bullet>("Prefabs/Sphere");
    }

    Bullet MakeBullet()
    {
        if (bulletPrefab)
        {
            var bullet = Instantiate(bulletPrefab);
            if (bullet && player) bullet.EventHadleOnCollisionPlayer += player.OnDamaged;

            return bullet;
        }

        return null;
    }
}
```

탄환 생성(Pooling)

- **임의의 수만큼(ex:포대의 수) 미리 탄환을 만든다**
- **만들어진 탄환** 오브젝트를 **List에 저장**
- **필요한 경우 List에서** 상태를 확인하여 **사용**하게 된다

```
public class GameMgr : MonoBehaviour
{
    void Init()
    {
        turrets = GameObject.FindGameObjectsWithTag("Respawn");
        // 오브젝트 풀링.
        listBullet = new List<Bullet>();
        for (int i = 0; turrets.Length > i; i++)
        {
            var bullet = MakeBullet();
            if(bullet) listBullet.Add(bullet);
        }
    }

    void SpawnBullet()
    {
        if (0 >= turrets.Length) return;

        // 사용되고 있지 않은(비활성화 상태) 탄환을 찾는다.
        var bullet = listBullet.Find(b => !b.gameObject.activeSelf);
        // 사용되지 않는 탄환이 없다면 추가로 만든다.
        if (!bullet) bullet = MakeBullet();

        if (bullet)
        {
            // 탄환 발사.
        }
    }
}
```

탄환 생성(Respawn)

- **플레이어**의 상태를 확인하여 **살아있는 경에만 생성**
- 시작하자마자 바로 탄환이 발사되지 않도록 spawnRate의 값을 임의로 1초로 설정
- **탄환 발사 후** 몇 초의 주기로 생성할지 **랜덤**하게 **spawnRate의 값을 재설정** 한다

```
public class GameMgr : MonoBehaviour
{
    void Update()
    {
        if (player && player.isLive)
        {
            checkTime += Time.deltaTime;
            if (spawnRate <= checkTime)
            {
                checkTime = 0;
                spawnRate = Random.Range(spwanRateMin, spwanRateMax);

                SpawnBullet();
            }
        }
    }
}
```

탄환 발사

Player 스크립트

- **탄환을 플레이어가 있는 방향으로 발사**하기 위한 위치정보를 알려준다

```
public class Player : MonoBehaviour
{
    public Vector3 position { get { return transform.position; } }
}
```

Bullet 스크립트

- **발사위치**를 **변경**하기 위한 함수 추가
- 발사를 위해 Rigidbody의 AddForce() 함수를 이용
- **AddForce() 함수**는 오브젝트에 힘을 주어 특정 방향으로 밀어내는 물리 연산을 수행하여 준다
- AddForce()에 AddForce()를 하면 추가로 힘을 더하기 때문에 **이전의 속도를 제거하여 주고 다시 힘을 주도록 한다**(풀링하여 재사용하기 때문에)

```
public class Bullet : MonoBehaviour
{
    private void Start()
    {
        if (!rigid) rigid = GetComponent<Rigidbody>();
    }

    public void SetPosition(Vector3 pos)
    {
        transform.position = pos;
    }

    public void OnFire(Vector3 dir, float force)
    {
        gameObject.SetActive(true);

        rigid.velocity = Vector3.zero;
        rigid.AddForce(dir.normalized * force);
    }
}
```

탄환 발사

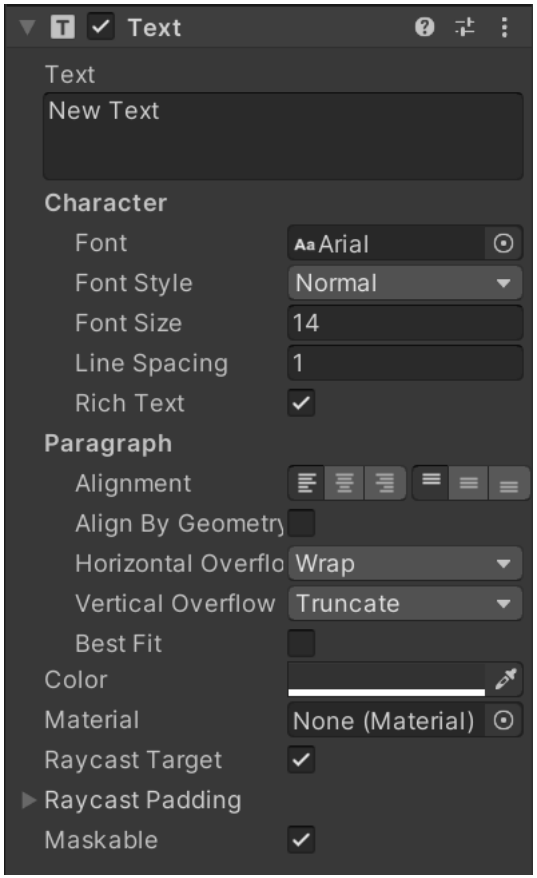
- **GameMgr 스크립트**
- 4개의 발사위치 중 **랜덤한 위치**를 선택
- 발사하기 위한 **탄환을 발사 위치로 이동**
- 현재 **플레이어가 있는 방향**을 구하여 **탄환을 발사**한다

```
public class GameMgr : MonoBehaviour
{
    void SpawnBullet()
    {
        ...
        if (bullet)
        {
            // 탄환 발사.
            var pos_index = Random.Range(0, turrets.Length);
            var pos = turrets[pos_index].transform.position + Vector3.up * 1.5f;
            bullet.SetPosition(pos);

            var dir = (player.position - pos).normalized;
            dir.y = 0.2f;

            var force = Random.Range(3, 8);
            bullet.OnFire(dir, force * 100);
        }
    }
}
```

UI(유저 인터페이스) – Text

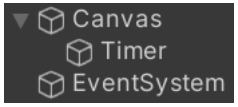
Text

- Text : 출력할 문자열
- Font : 문자를 출력하는데 사용되는 글꼴
- Font Style : 폰트 스타일
  - Normal : 원본
  - Bold : **굵게**
  - Italic : *기울어지게*
  - Bold And Italic : ***굵고 기울어지게***
- Font Size : 출력 글자 크기
- Line Spacing : 행간의 거리, 줄높이
- Rich Text : 서식 문자 사용
- Alignment : 문자 수평 및 수직 정렬 방법
- Align By Geomtry : 글리프 메트릭(glyph metric)이 아닌 글리프 지오메트리 범위를 이용하여 수평 정렬한다
  글리프 : 글자 하나의 모양 정보 / 글리프 메트릭 : 글자 하나가 차지하는 공간
- Horizontal/Vertical Overflow : 영역을 벗어날 경우 처리 방법
  - Warp : 줄 변경
  - Truncate : 보이지 않게 한다
  - Overflow : 무시하고 출력한다
- Best Fit : 영역의 크기에 맞게 Min Size에서 Max Size 범위의 값으로 Font Size를 알아서 지정
- Color : 출력할 문자의 색상



타이머

- **생존 시간 확인을 위한 Timer**
  - UI=>Text

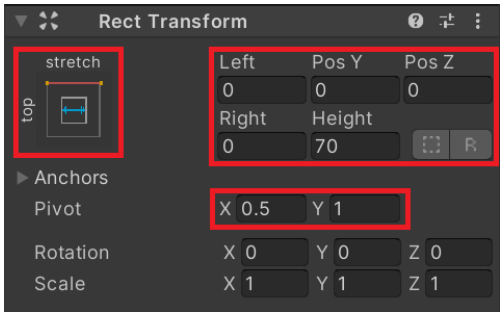


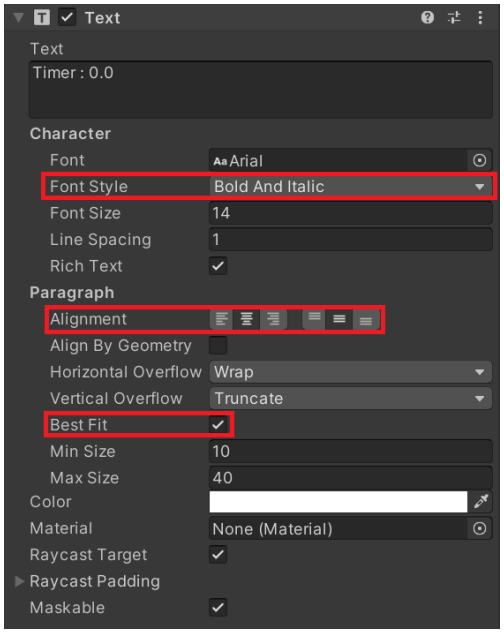
- **Rect Transform**
  - Anchor : (top, stretch)
  - Pivot : (0.5, 1)
  - Pos Y : 0
  - Left, Right : 0
  - Height : 70

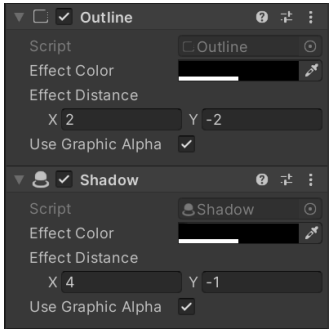
- **Text**
  - Font Style : Bold And Italic
  - Alignment : 가운데 맞춤
  - Best Fit : true

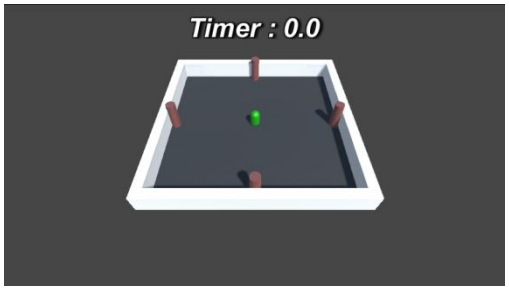
- Add Component : UI=>Effects=>Outline
  - Effect Distance : (2, -2)

- Add Component : UI=>Effects=>Shadow
  - Effect Distance : (4, -1)









타이머

- **UIMgr 스크립트**
  - **Canvas**에 **UIMgr 스크립트 추가**
  - 외부에서 **시간(초)을** 알아와 **Timer(Text)에서 출력**

```
public class UIMgr : MonoBehaviour
{
    public static UIMgr Instance { get; private set; }

    [SerializeField] private Text timer;
    public float Timer
    {
        set { if (timer /*or null != timer*/) timer.text = string.Format("Timer : {0:N2}", value); }
    }

    private void Awake()
    {
        if (null == Instance)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);
            return;
        }
        Destroy(gameObject);
    }
}
```

타이머

- **GameMgr 스크립트**
  - 시간을 계산하여 UIMgr에 알려 준다

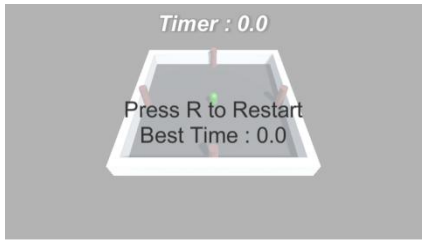
```
public class GameMgr : MonoBehaviour
{
    private float timer = 0;

    void Init()
    {
        timer = 0;

        ...
    }

    void Update()
    {
        if (player && player.isLive)
        {
            ...

            timer += Time.deltaTime;
            UIMgr.Instance.Timer = timer;
        }
    }
}
```

게임오버

- **GameOver 화면 구성**
  - UI=>Image(GameOver)
  - UI=>Text(Text)
  - Text Parent = GameOver

- **Rect Transform(GameOver)**
  - Anchor : (stretch, stretch)
  - Left, Top, Right, Bottom : 0

- **Image**
  - Color : (255, 255, 255, 150)



- **Rect Transform(Text)**
  - Anchor : (stretch, stretch)
  - Top, Bottom : 150
  - Left, Right : 0

- **Text**
  - Alignment : 가운데 맞춤
  - Best Fit : true

게임오버

- **UIMgr 스크립트**
  - 플레이어가 죽게 되면 **저장된 BestTime**을 가져와 **현재의 생존 시간**과 **비교하여 큰 값을 출력**하고 **새로운 BestTime으로 갱신**하도록 한다
  - **'R' 문자를 강조**하기 위하여 서식문자를 이용하여 **붉게 출력**
  - **게임 시작 시**에 **GameOver**가 출력되면 안되기 때문에 **비활성화** 처리를 한다


```
public class UIMgr : MonoBehaviour
{
   [SerializeField] private Text gameOverText;
    private GameObject gameOverPanel;

    private void Start() {if(gameOverText) gameOverPanel = gameOverText.transform.parent.gameObject; }

    public void GameOver(float time)
    {
        if (gameOverText && gameOverPanel)
        {
            var bestTime = PlayerPrefs.GetFloat("BestTime", 0);
            bestTime = Mathf.Max(bestTime, time);

            gameOverText.text = string.Format("<b>Press <color=red>R</color> to Restart</b>\n<i>Best Time : {0:N2}</i>", bestTime);

            PlayerPrefs.SetFloat("BestTime", bestTime);

            gameOverPanel.SetActive(true);
        }
    }

    public void OnPlay() {if (gameOverPanel) gameOverPanel.SetActive(false); }
}
```

게임오버

- **GameMgr 스크립트**
  - 플레이어가 죽을 경우 **UIMgr의 GameOver()가 호출**되도록 내용을 추가

```
public class GameMgr : MonoBehaviour
{
    void Init()
    {
        UIMgr.Instance.OnPlay();

        ...
    }

    Bullet MakeBullet()
    {
        if (bulletPrefab)
        {
            var bullet = Instantiate(bulletPrefab);
            if (bullet && player)
            {
                bullet.EventHadleOnCollisionPlayer += player.OnDamaged;
                bullet.EventHadleOnCollisionPlayer += () => { UIMgr.Instance.GameOver(timer); };
            }

            return bullet;
        }

        return null;
    }
}
```

재시작

- **GameMgr 스크립트**
  - UnityEngine.SceneManagement의 SceneManager.LoadScene()으로 **원하는 씬(Scene)을 전환**할 수 있다
  - 씬이 로드 되면 **SceneManager.sceneLoaded**로 **특정 이벤트를 처리**할 수 있다
  - 해당 **씬(Scene) 이름**을 **'Dodge'**로 저장
  - 게임오버를 당했을 경우 'Dodge' 씬을 다시 Load하여 게임을 재시작 한다

```
using UnityEngine.SceneManagement;
public class GameMgr : MonoBehaviour
{
    private void Awake()
    {
        if (null == Instance)
        {
            Instance = this;
            DontDestroyOnLoad(gameObject);

            SceneManager.sceneLoaded += (scene, mode) => { Init(); };
            return;
        }

        Destroy(gameObject);
    }

    void Update()
    {
        if (player)
        {
            if (player.isLive)
            {
                ...
            }
            else if (Input.GetKeyDown(KeyCode.R)) SceneManager.LoadScene("Dodge");
        }
    }
}
```

etc

- **씬 전환할 때 라이트가 어두워지는 문제**
  - **유니티의 고질적인 문제**로 씬을 전환(LoadScene)하면 해당 씬의 **빛이 어둡게 변하는 문제**가 있다
  - **Window=>Rendering=>Lighting에서 Generate Lighting을 실행**
  - 이 후 씬을 전환하더라도 해당 씬의 라이트가 어두워지는 문제는 해결된다
  - 각 씬 별로 모두 Generate Lighting을 해줘야 하는 번거로움이 있다