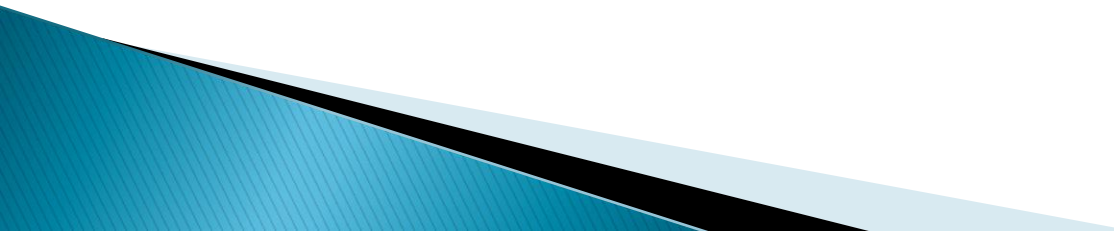
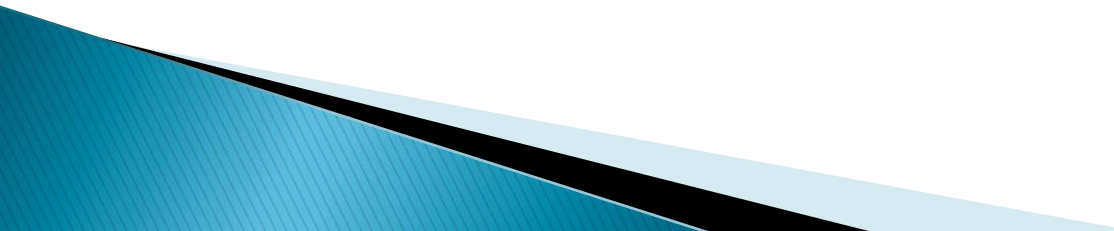


Unity 3D Dodge

목차

1. Package Manager · · · 4p
 2. 월드, 플레이어 배치 · · · 5p
 3. Game Manager · · · 6p
 4. Input System · · · 9p
 5. Player · · · 12p
- 

목차

- 6. 포대, 탄환 만들기 . . . 16p
 - 7. 탄환 생성(Spawn) 및 발사(Fire) . . . 17p
 - 8. 탄환의 충돌 확인 . . . 24p
 - 9. 타이머 . . . 25p
 - 10. 게임오버 . . . 29p
 - 11. 재시작 . . . 32p
- 

Package Manager

▶ Input System

- Unity에서 지원하는 새로운 인풋 시스템
- PackageManager에서 최신 버전의 Input System을 설치
- 설치가 완료되면 대화상자의 [Yes] 버튼을 선택하여 프로젝트를 재시작 한다

The image shows a sequence of steps for installing the Input System package in Unity. It includes the 'Window' menu, the Package Manager window, a warning dialog, and the final 'Install' button.

Window Menu: The 'Package Manager' option is highlighted in the 'Window' menu.

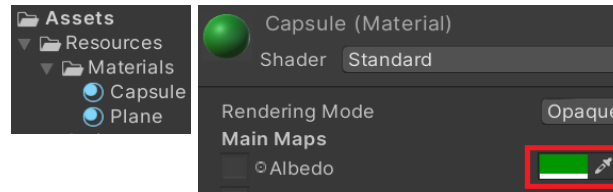
Package Manager Window: The 'Unity Registry' package source is selected. The 'Input System' package (version 1.4.1) is listed in the package list. The package details on the right show it is a 'Release' version by Unity Technologies, dated June 23, 2022.

Warning Dialog: A warning message states: "This project is using the new input system package but the native platform backends for the new input system are not enabled in the player settings. This means that no input from native devices will come through. Do you want to enable the backends? Doing so will *RESTART* the editor." The 'Yes' button is highlighted.

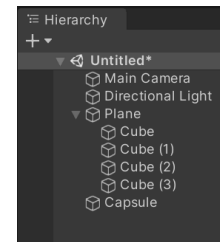
Package Manager Window (Bottom): The 'Input System' package is selected in the list. The 'Install' button is highlighted at the bottom right.

월드, 플레이어 배치

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

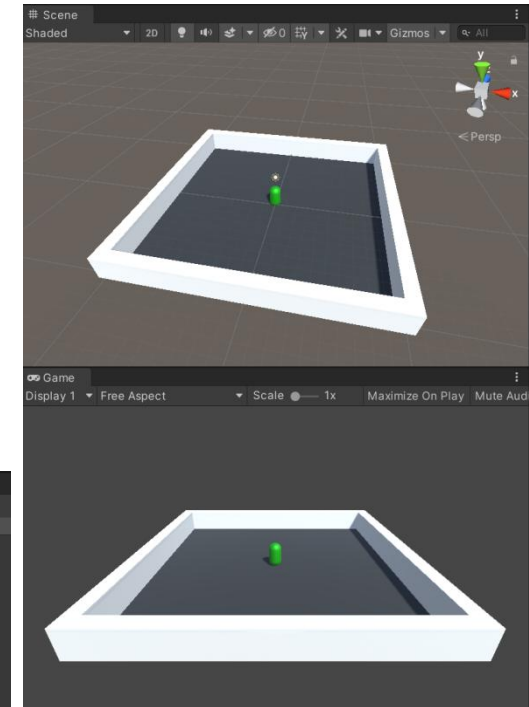


- **메인 카메라**
- > **Transform**
 - Position : 0, 20, -20
 - Rotation : 50, 0, 0
- > **Camera**
 - Clear Flag : Solid Color
 - Background : RGB(70, 70, 70)



- **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)
 - local Scale : 10.5, 2, 0.5
 - local Position : (0, 1, 5/-5)

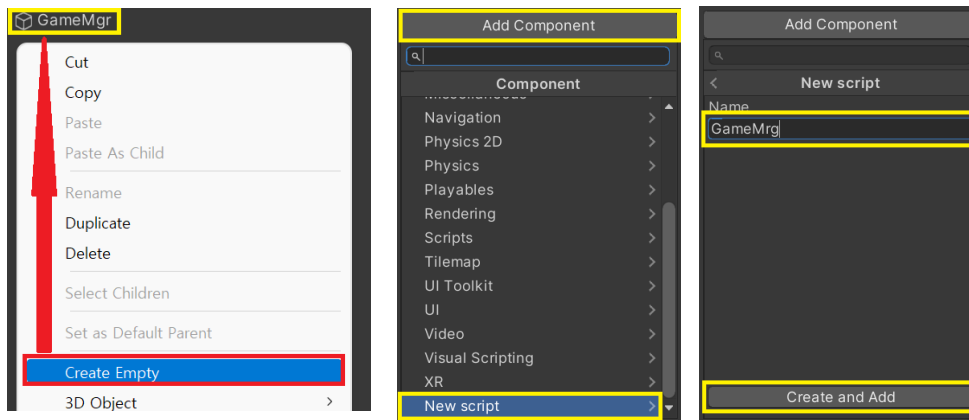


- **Capsule(플레이어)**
- 3D Object=>Capsule
- > **Transform**
 - Position : 0, 1, 0
- > **Mesh Renderer**
 - Materials : Capsule material로 변경

Game Manager

▶ GameMgr 스크립트

- 새 GameObject(GameMgr)에 GameMgr 스크립트 생성 및 추가

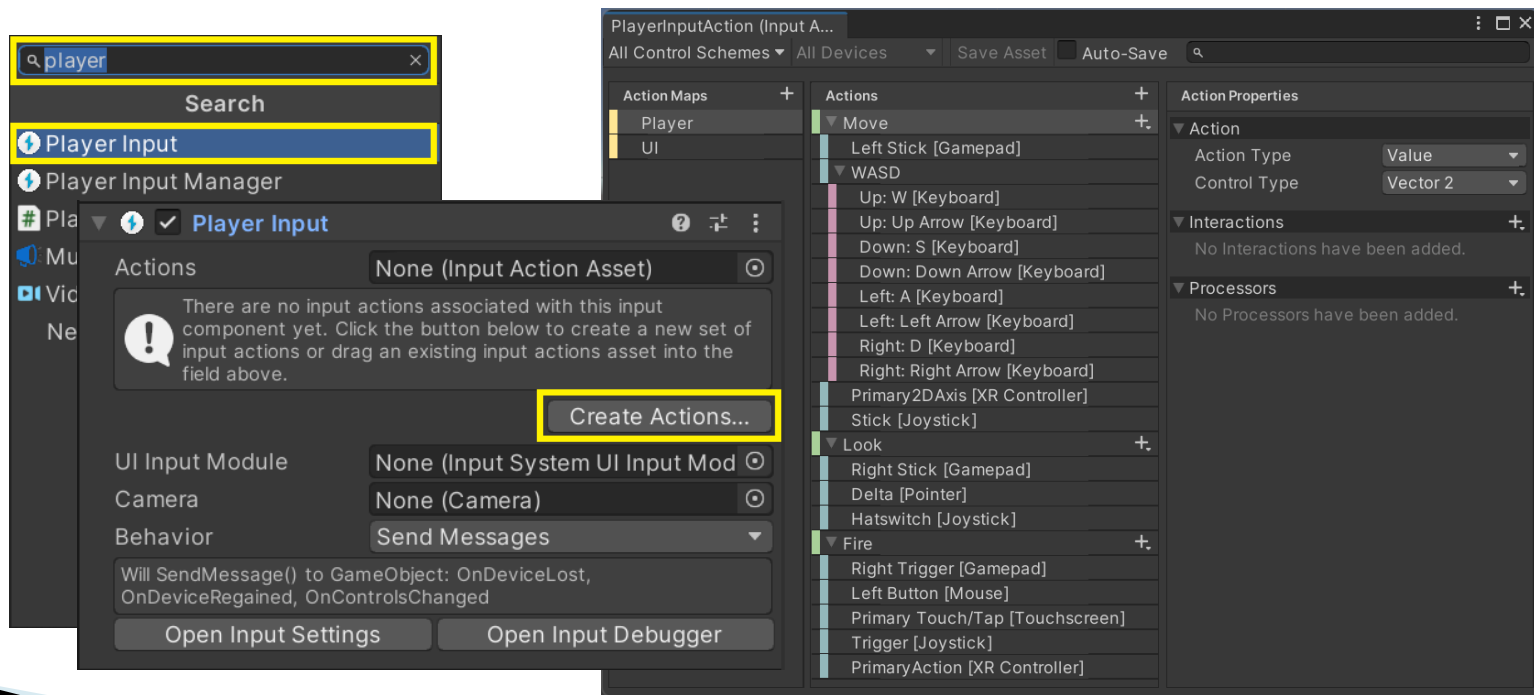


- 게임 전체를 통제 및 관리하는 객체
- 오직 하나만 존재해야 하기 때문에 Singleton 디자인 패턴으로 Instance를 만든다

Game Manager

▶ PlayerInput

- Input Controller를 GameManager에서 제어
- GameMgr(GameObject)에 [Add Component]에서 Player Input을 검색하여 **추가**
- Player Input 컴포넌트의 [Create Actions]으로 **InputAction(PlayerInputAction)** 에셋을 **생성**



Game Manager

```
using UnityEngine.InputSystem;
public class GameMgr : MonoBehaviour
{
    public static GameMgr Instance { get; private set; }

    [Header("Controller")]
    [SerializeField] PlayerInput input;

    void Awake()
    {
        if (null == Instance)
        {
            Instance = this;
            // DontDestroyOnLoad() : 씬(Scene)이 변경될 경우 GameObject의 파괴를 막기위한 함수.
            DontDestroyOnLoad(gameObject);

            return;
        }

        Destroy(gameObject);
    }
}
```

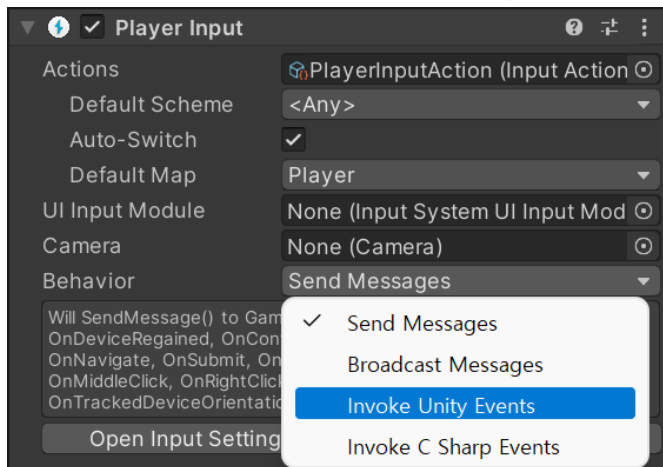

Input System

▶ InputSystem 연동

- GameMgr(GameObject)가 가지고 있는 PlayerInput 컴포넌트 연결



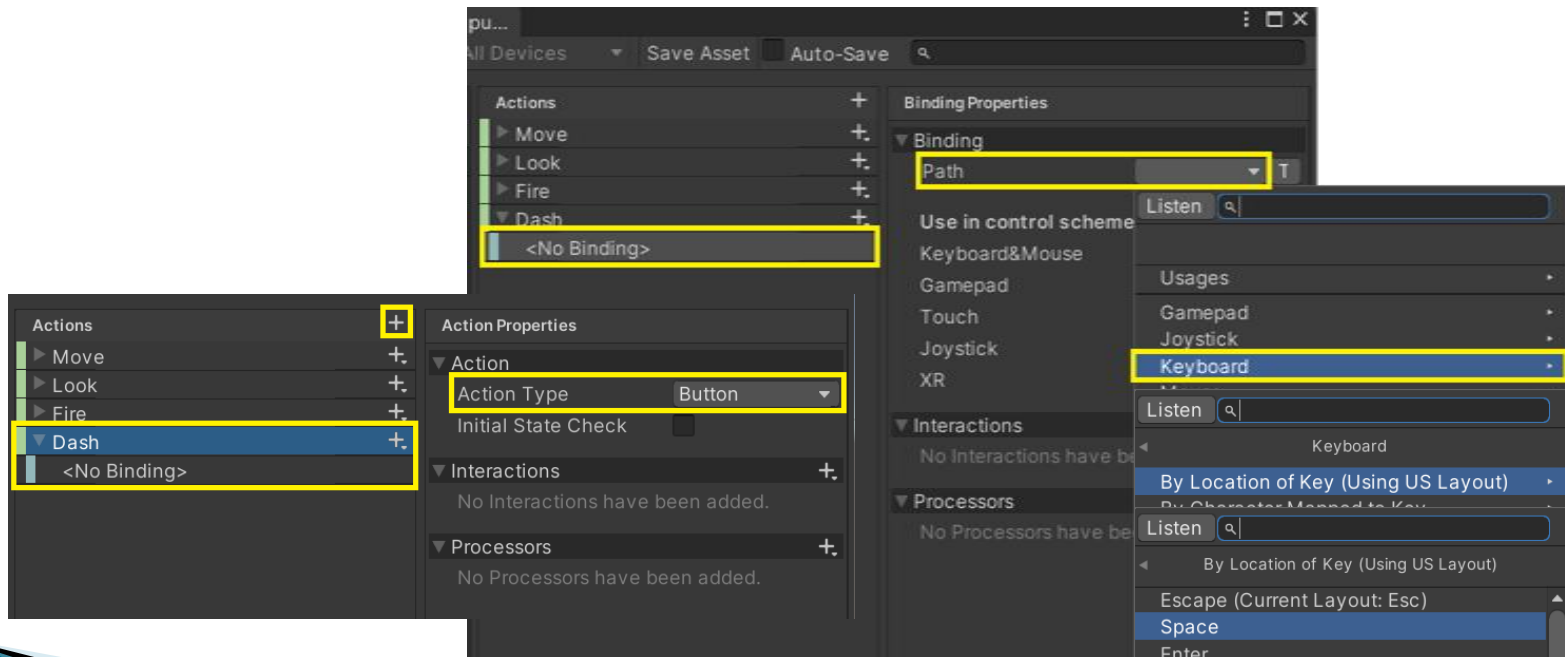
- Behavior의 값을 **Invoke Unity Events**로 변경



Input System

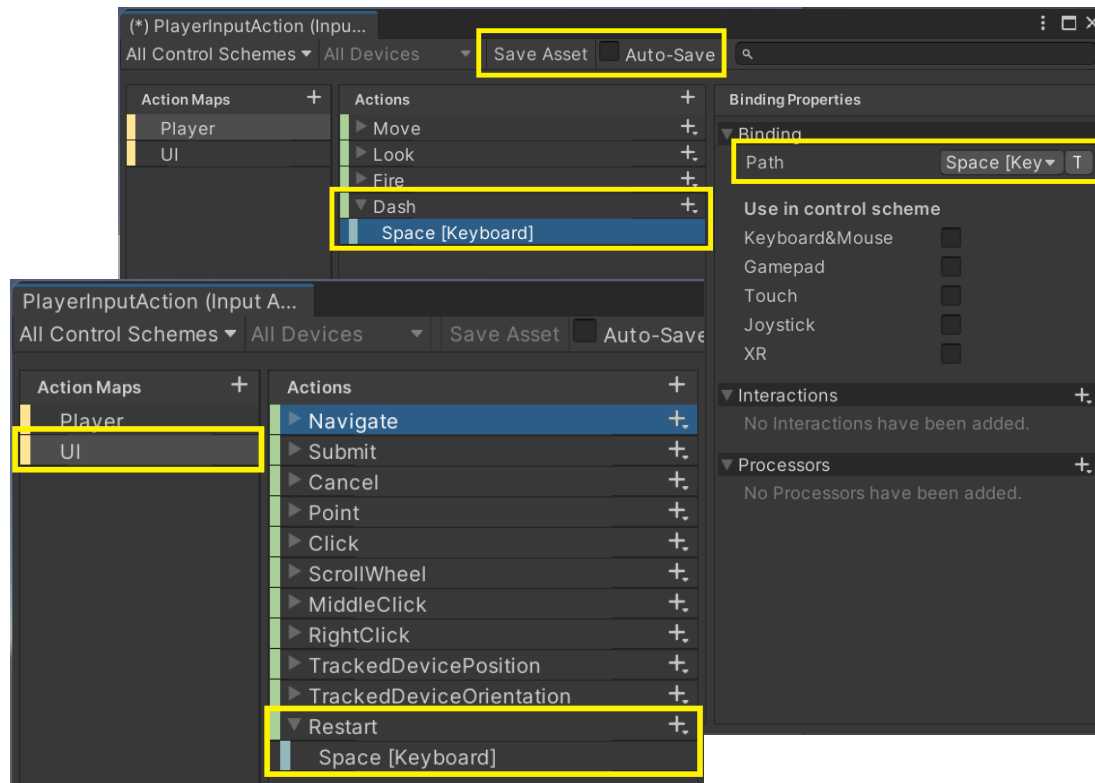
▶ Action 추가

- InputAction에 **[+]**를 눌러 **Action(Dash)**를 추가
- **Action Type**을 **Button**으로 설정(기본 Type이 Button이다)
- Action(Dash)하위의 **<No Binding>**을 선택
- **Path=>Keyboard=>By Location of Key=>Space**로 매핑



Input System

- 내용을 적용하려면 반드시 [Save Asset]을 선택하여 저장해야 한다
- [Auto-Save]를 체크하여 두면 자동으로 수정한 내용이 저장된다
- 같은 방식으로 UI(Map)에 Restart(Action)을 추가



Player

▶ Player 스크립트

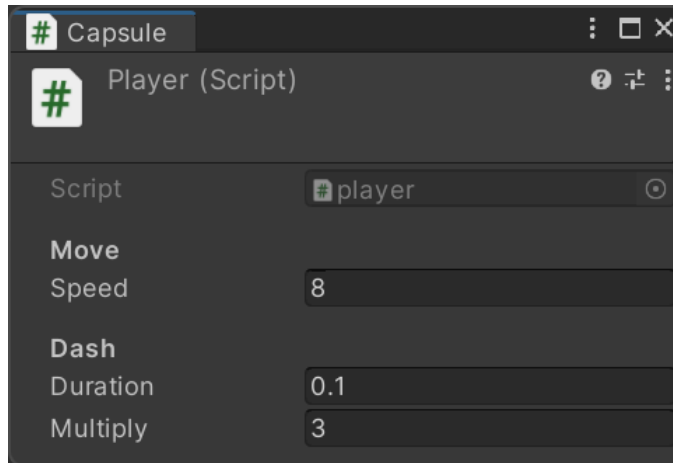
- 코드 작성 후 **Capsule**(GameObject)에 Player 스크립트를 컴포넌트로 **추가**
- 플레이어의 **움직임 제어**를 위하여 **Rigidbody** 컴포넌트를 추가 해야 한다
- **[SerializeField]**를 이용하여 **public** 변수가 아니라도 **Inspector** 창에서 접근, 수정 가능

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

    Vector3 dir = Vector3.zero;
    Vector3 velocity = Vector3.zero;

    [Header("Move")]
    [SerializeField] float speed = 8.0f;

    [Header("Dash")]
    [SerializeField] float duration = 0.1f;
    [SerializeField] float multiply = 3f;
    bool isInvincibility = false;
    Coroutine coroutine = null;
```



```
public Vector3 position { get { return rigid.position; } } // transform.position보다 rigidbody.position의 비용이 적다.
public bool isLive { get { return gameObject.activeSelf; } } // GameObject 활성화, 비활성화로 생존 여부 파악.
```

Player

```
void Start()
{
    rigid = GetComponent<Rigidbody>();
    // position의 y값과 rotation이 변경되지 않게 고정.
    rigid.constraints =
        RigidbodyConstraints.FreezePositionY | RigidbodyConstraints.FreezeRotation;
}
```

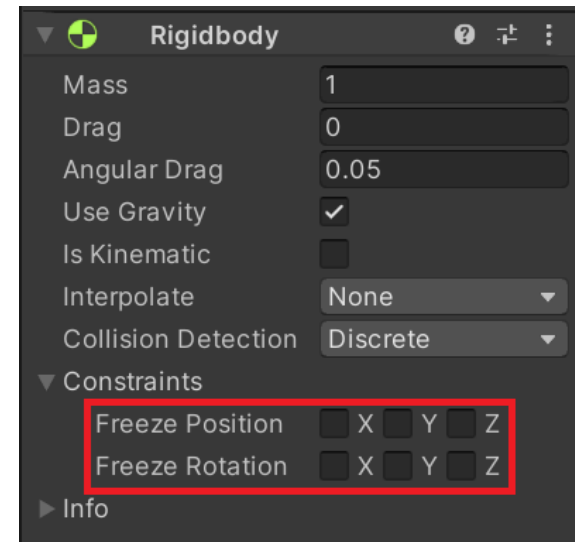
// Physics를 이용한 이동일 경우 Update()보다 FixedUpdate()를 이용하자.

```
void FixedUpdate() { rigid.velocity = velocity; }
```

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

// 데미지를 입어도, 무적 상태인 동안은 죽지 않는다.

```
public void OnDamaged() { gameObject.SetActive(isInvincibility); }
```



Player

```
public void Move(InputAction.CallbackContext context) {  
    // Move(Action)의 Control Type이 Vector2이다.  
    Vector2 v = context.ReadValue<Vector2>();  
    dir = new Vector3(v.x, 0, v.y);  
    velocity = dir * speed; // 방향과 속력을 이용하여 속도(velocity)를 구한다.  
}  
  
public void Dash(InputAction.CallbackContext context) {  
    // Action Type이 Button일 경우 System.Single(float)로 키 입력을 확인할 수 있다.  
    float value = context.ReadValue<System.Single>();  
    if (0 < value) {  
        if (null != coroutine) StopCoroutine(coroutine);  
        coroutine = StartCoroutine(StopDash());  
  
        velocity = dir * speed * multiply;  
        isInvincibility = true; // Dash가 지속되는 동안 무적 상태 적용.  
    }  
}  
  
IEnumerator StopDash() {  
    yield return new WaitForSeconds(duration);  
  
    velocity = dir * speed;  
    isInvincibility = false; // Dash가 끝나면 무적 상태 해제.  
    coroutine = null;  
}  
} // public class player : MonoBehaviour
```

Player

▶ Player Input

- GameMgr에서 Player의 Move(), Dash()를 PlayerInput과 연결

```
public class GameMgr : MonoBehaviour
{
    ...
    Player player;

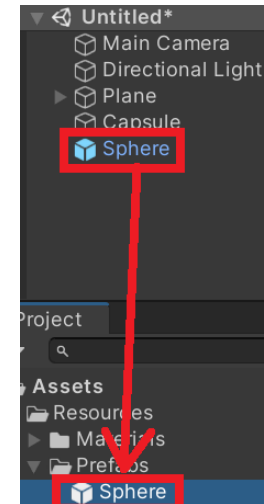
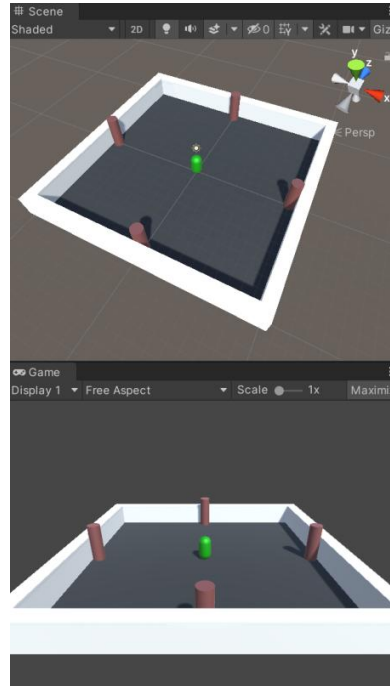
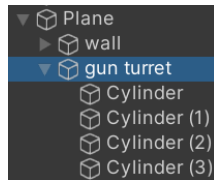
    void Start() { Initialize(); }

    void Initialize()
    {
        player = FindObjectOfType<Player>();
        player.Initialize();

        input.SwitchCurrentActionMap("Player");
        input.actions["Dash"].started += player.Dash;    // started : Input이 시작될 때.
        input.actions["Move"].performed += player.Move;  // performed : Input이 되었을 때.
        input.actions["Move"].canceled += player.Move;  // canceled : Input이 종료될 때.
    }
}
```

포대, 탄환 만들기

- **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

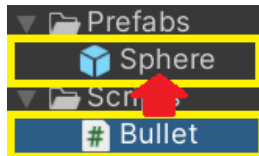


- **Sphere**
- 3D Object=> Sphere
- > **Mesh Renderer**
 - Materials : Sphere material로 변경
- > **Sphere Collider**
 - Is Trigger : true
- **Resources** 폴더에 **Prefabs** 폴더 생성
- **Sphere** 오브젝트를 Prefabs 폴더로 드래그&드롭하여 **Prefab**으로 만든다
- **Hierarchy**의 **Sphere** 제거

탄환 생성(Spawn) 및 발사(Fire)

▶ Bullet 스크립트

- Bullet은 **오브젝트 풀링(pooling)**을 하여 자원을 관리
- **Bullet 스크립트 생성**
- **코드 작성 후 Sphere 프리팹에 추가**



```
using UnityEngine.Events;
[RequireComponent(typeof(Rigidbody))] // 총알을 방사할 때 사용하기 위하여 Rigidbody를 추가.
public class Bullet : MonoBehaviour
{
    // 플레이어와 탄환이 충돌할 경우 사용될 event delegate
    public UnityEvent OnCollisionPlayerEvent = new();
    Rigidbody rigid;

    void Awake()
    {
        rigid = GetComponent<Rigidbody>();
        // 오브젝트 풀링을 하여 필요할 경우 활성화하여 사용.
        gameObject.SetActive(false);
    }
}
```

탄환 생성(Spawn) 및 발사(Fire)

```
public void SetPosition(Vector3 pos)
{
    // 총알을 발사할 경우 위치를 변경하기 위해 사용.
    transform.position = pos;
}
```

```
public void OnFire(Vector3 dir, float force)
{
    // 함수가 호출된 경우, 이 오브젝트를 사용한다는 뜻이므로,
    // 총알 오브젝트를 활성화 한다.
    gameObject.SetActive(true);

    // 이전에 사용되어 속도가 남아 있을 경우가 있기 때문에,
    // 물체의 속도를 0으로 변경.
    rigid.velocity = Vector3.zero;
    rigid.AddForce(dir.normalized * force);
}
```

```
} // class Bullet
```

탄환 생성(Spawn) 및 발사(Fire)

```
public class GameMgr : MonoBehaviour  
{
```

```
...
```

```
Vector3[] turretsPos;
```

```
Bullet bulletPrefab;
```

```
// 오브젝트를 수시로 생성(Instantiate), 삭제(Destroy)할 경우 많은 비용이 발생하기 때문에,  
// 오브젝트 풀링(Object Pooling)을 하여 Bullet(자원)을 관리한다.
```

```
// 미리 생성된 오브젝트를 담고 있을 오브젝트 풀 컨테이너.
```

```
Queue<Bullet> objectPool = new Queue<Bullet>();
```

```
[Header("Bullet Fire Interval")]
```

```
[SerializeField] float spawnTimeMin = 0.3f;
```

```
[SerializeField] float spawnTimeMax = 0.8f;
```

```
float spawnInterval = 1f;
```

```
float checkTime = 0;
```

```
public void PushObjectPool(Bullet bullet) { objectPool.Enqueue(bullet); }
```

탄환 생성(Spawn) 및 발사(Fire)

```
void Initialize()
{
    ...
    spawnInterval = 1f;
    spawnTimeMax = 0.8f;

    bulletPrefab = Resources.Load<Bullet>("Prefabs/Sphere");
    GameObject[] arrObj = GameObject.FindGameObjectsWithTag("Respawn");
    int length = arrObj.Length;
    turretsPos = new Vector3[length];
    for (int i = 0; length > i; i++) {
        turretsPos[i] = arrObj[i].transform.position + Vector3.up * 1.5f;
    }

    MakeBullet();
}
```

탄환 생성(Spawn) 및 발사(Fire)

```
void MakeBullet()
{
    if (bulletPrefab)
    {
        int count = turretsPos.Length;
        for (int i = 0; count > i; i++)
        {
            Bullet bullet = Instantiate(bulletPrefab);
            bullet.OnCollisionPlayerEvent.AddListener(player.OnDamaged);
            bullet.OnCollisionPlayerEvent.AddListener(() =>
            {
                if (!player.isLive)
                {
                    input.actions["Move"].performed -= player.Move;
                    input.actions["Move"].canceled -= player.Move;
                    input.actions["Dash"].started -= player.Dash;
                    input.SwitchCurrentActionMap("UI");
                }
            });

            objectPool.Enqueue(bullet);
        } // for (int i = 0; count > i; i++)
    } // if (bulletPrefab)
}
```

탄환 생성(Spawn) 및 발사(Fire)

```
void SpawnBullet()
{
    // Pool에 Bullet이 없다면 추가로 만든다.
    if (1 > objectPool.Count) MakeBullet();

    // Pool에서 Bullet 하나를 꺼낸다.
    Bullet bullet = objectPool.Dequeue();
    if (bullet)
    {
        // 월드에 배치되어 있는 terret 중 한 곳을 찾아,
        int pos_index = Random.Range(0, turretsPos.Length);
        Vector3 pos = turretsPos[pos_index];
        // 탄환을 배치.
        bullet.SetPosition(pos);

        // 플레이어를 향해 쏘기 위하여 방향을 구한다.
        Vector3 dir = (player.position - pos).normalized;
        dir.y = 0.2f;

        int force = Random.Range(3, 8);
        // 탄환 발사.
        bullet.OnFire(dir, force * 100);
    }
}
```

탄환 생성(Spawn) 및 발사(Fire)

```
void Update()
{
    if (player && player.isLive)
    {
        checkTime += Time.deltaTime;
        if (spawnInterval <= checkTime)
        {
            checkTime = 0;
            // 탄환 발사 주기를 갱신.
            spawnInterval = Random.Range(spwanTimeMin, spwanTimeMax);

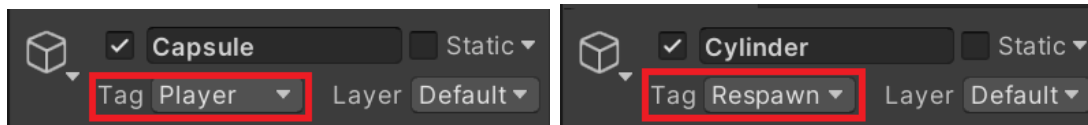
            // 탄환 생성 및 발사.
            SpawnBullet();
        } // if(spawnInterval <= checkTime)
    } // if(player && player.isLive)
}

} // class GameMgr
```

탄환의 충돌 확인

▶ Bullet 충돌 처리

- Collider의 [Is Trigger]를 **true**로 할 경우 **OnTriggerXXX**를 이용하여 **충돌 확인이 가능하다**
- Capsule의 Tag를 **Player**로 변경, Cylinder(x4)의 Tag를 **Respawn**으로 변경



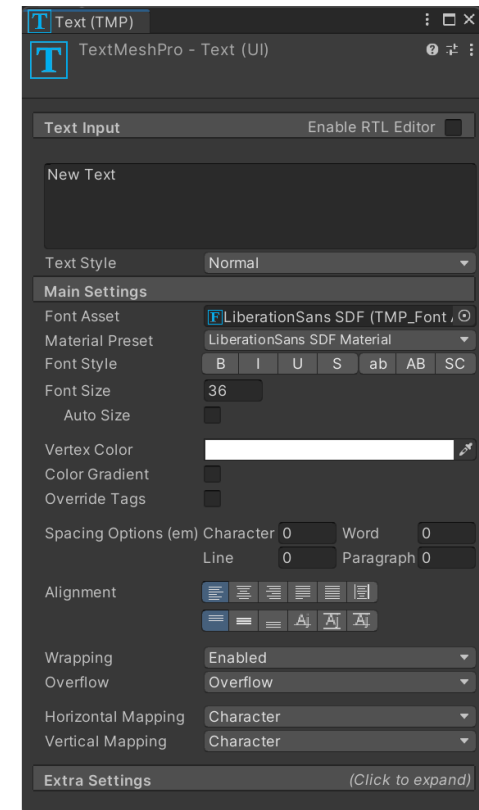
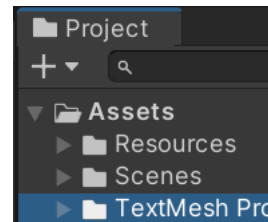
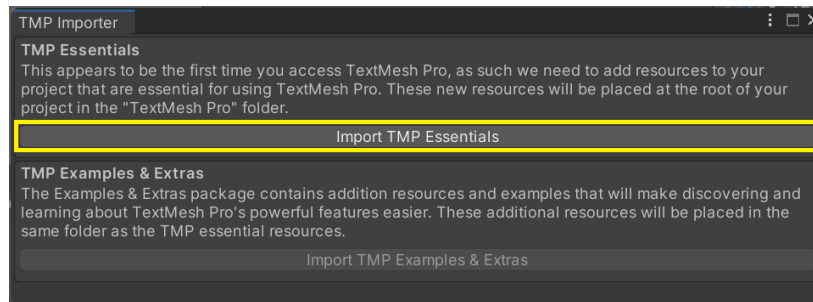
```
public class Bullet : MonoBehaviour
{
    ...
    private void OnTriggerEnter(Collider other)
    {
        // 충돌 물체를 tag를 이용하여 확인.
        string tag = other.tag;
        if (tag.Equals("Player"))
        {
            // XXX?.Invoke() : null 확인 연산자, Delegate에 값이 있다면(null이 아니라면) Method를 실행.
            OnCollisionPlayerEvent?.Invoke();
        }
        else if (tag.Equals("Respawn") || other.name.Equals(name)/*같은 탄환끼리 충돌 예외 처리*/) return;

        gameObject.SetActive(false);
        // 사용이 끝난 오브젝트는 반드시 Object Pool에 돌려 주어야 한다.
        GameMgr.Instance.PushObjectPool(this);
    }
}
```


타이머

▶ Text Mesh Pro(TMP)

- 프로젝트에서 처음 TextMeshPro UI를 추가하면 TMP Importer가 활성화 된다
- TMP Importer가 활성화 되면 반드시 추가 리소스를 Import 해주어야 한다
- Text Input : 출력할 내용, 서식 문자(링크) 사용 가능
- 공식 매뉴얼 : [링크](#)



타이머

▶ Timer 만들기

- UI => Text Mesh Pro

◆ Rect Transform

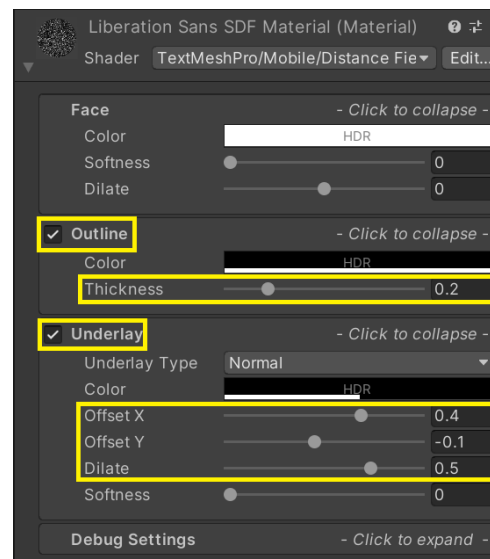
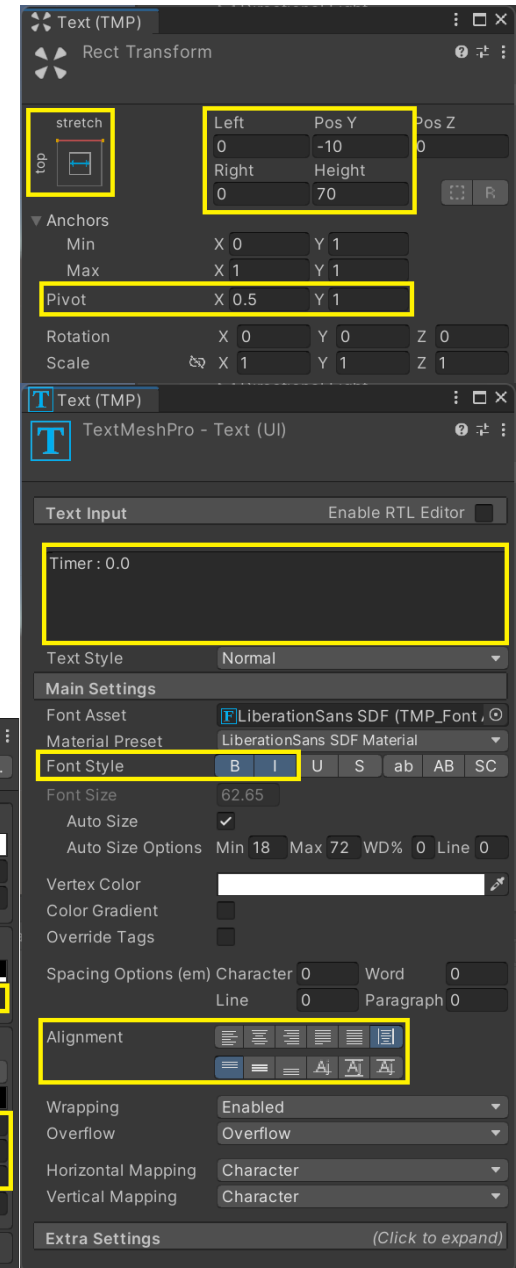
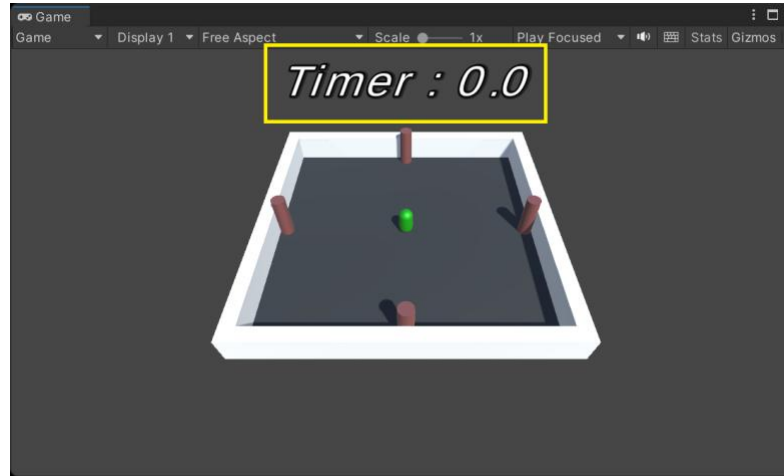
- Anchor : (top, stretch)
- Pivot : (0.5, 1)
- Pos Y : -10
- Left, Right : 0
- Height : 70

◆ Text(TMP)

- Font Style : Bold, Italic
- Alignment : Geometry Center
- Auto Size : true

◆ Material

- Outline(true)
- Thickness : 0.2
- Underlay(true)
- Offset X : 0.4
- Offset Y : -0.1
- Dilate : 0.5



타이머

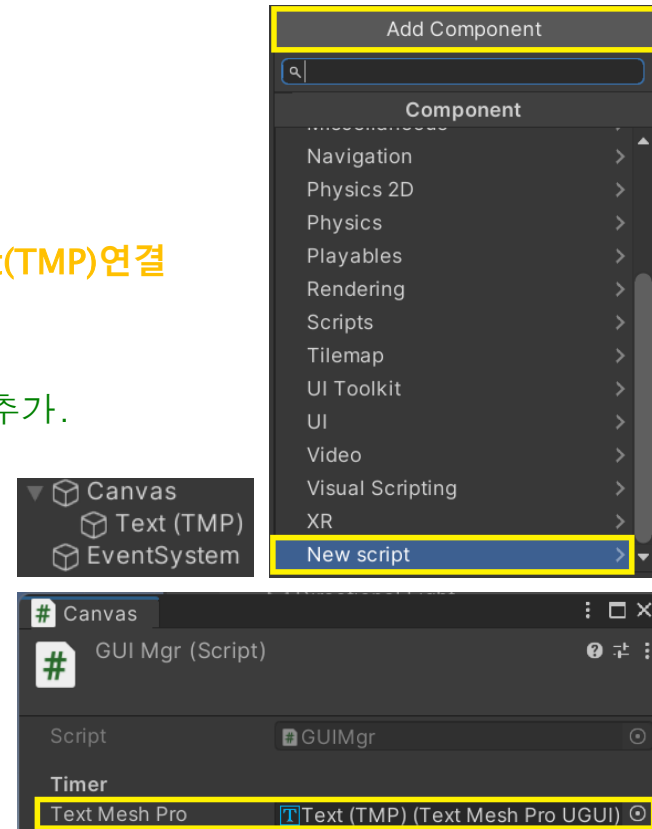
▶ GUIMgr 스크립트

- Canvas에 **GUIMgr** 스크립트 생성 및 추가
- Inspector에서 **GUIMgr**의 TextMeshPro 변수에 **Text(TMP)** 연결
- 시간을 계산해서 Text(TMP)를 이용하여 씬에 출력

using TMPro; // TextMeshProUGUI를 사용하기 위해서 추가.

```
public class GUIMgr : MonoBehaviour
{
    [Header("Timer")]
    [SerializeField] TextMeshProUGUI textMeshPro;

    public float Timer
    {
        set
        {
            if (textMeshPro) textMeshPro.text = string.Format("Timer : {0:N2}", value);
        }
    }
}
```



타이머

```
public class GameMgr : MonoBehaviour
{
    ...
    GUIMgr UIMgr;
    float time = .0f;

    public void SetGUIManager(GUIMgr mgr) { UIMgr = mgr; }
    ...
    void Initialize() {
        time = .0f;
        ...
    }
    ...
    void Update() {
        if (player && player.isLive) {
            ...
            if (UIMgr) {
                time += Time.deltaTime;
                UIMgr.Timer = time;
            }
        }
    }
} // class GameMgr
```

```
public class GUIMgr : MonoBehaviour
{
    ...
    private void Start()
    {
        Timer = .0f;
        GameMgr.Instance.SetGUIManager(this);
    }
}
```



게임오버

▶ GameOver 화면 구성

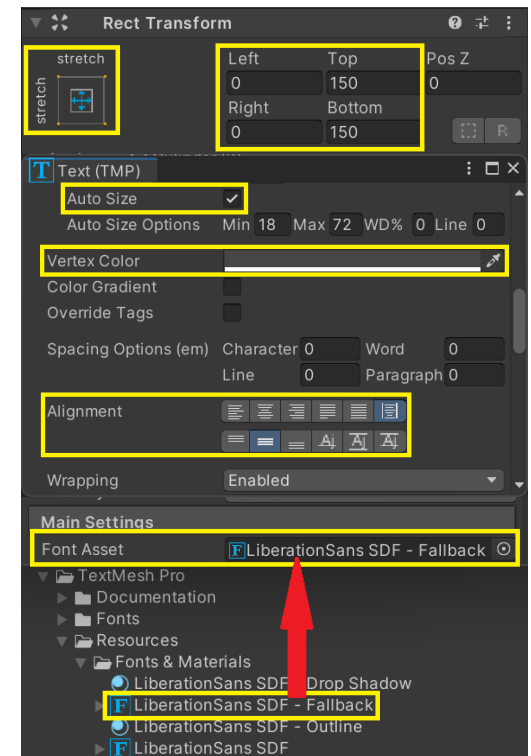
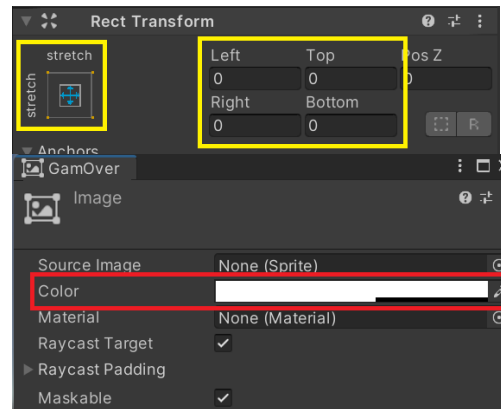
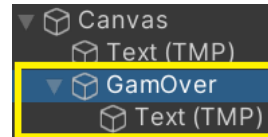
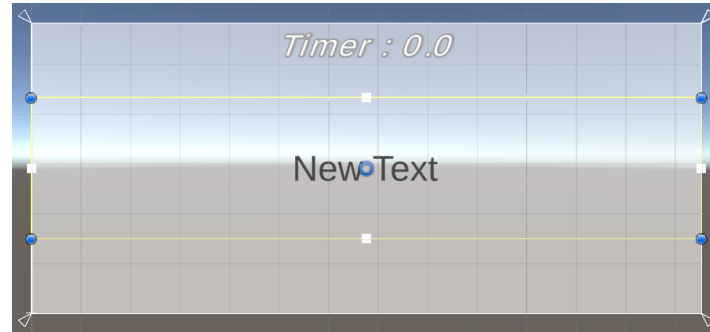
- UI=>Image(GameOver)
- UI=>Text(TextMeshPro)

◆ 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(TMP)
- Auto Size : true
 - Vertex Color : (70, 70, 70, 255)
 - Alignment : 가운데 맞춤
 - Font Asset : Fallback으로 변경



게임오버

▶ GUIMgr 스크립트

- Inspector에서 **GUIMgr**의 gameOverPanel 변수에 **GameOver(GameObject)** 연결
- 최고 생존 시간을 Text(TMP)를 이용하여 씬에 출력

```
public class GUIMgr : MonoBehaviour
{
    ...

    [Header("Game Over")]
    [SerializeField] GameObject gameOverPanel;

    string format = "<b>Press <color=red>Space Bar</color> to Restart</b>\n<i>Best Time : {0:N2}</i>";

    ...

    public float BestTime {
        set {
            if (gameOverPanel) {
                gameOverPanel.SetActive(true);
                TextMeshProUGUI text = gameOverPanel.GetComponentInChildren<TextMeshProUGUI>();
                if (text) {
                    float bestTime = PlayerPrefs.GetFloat("BestTime", .0f);
                    bestTime = Mathf.Max(bestTime, value);

                    text.text = string.Format(format, bestTime);
                    PlayerPrefs.SetFloat("BestTime", bestTime);
                } // if(text)
            } // if(gameOverPanel)
        } // set
    }

    ...

    private void Start()
    {
        ...

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



게임오버

▶ GameMgr 스크립트

- 플레이어가 죽을 경우 UIMgr의 BestTime을 갱신하여 **GameOver GUI가 출력** 되도록 한다

```
public class GameMgr : MonoBehaviour
{
    ...

    void MakeBullet()
    {
        if (bulletPrefab)
        {
            int count = turretsPos.Length;
            for (int i = 0; count > i; i++)
            {
                ...

                // 플레이어가 죽었을 경우 Best Time을 갱신.
                bullet.EventHadleOnCollisionPlayer += () =>
                {
                    if (!player.isLive)
                    {
                        UIMgr.BestTime = time;
                    }
                };

                objectPool.Enqueue(bullet);
            } // for (int i = 0; count > i; i++)
        } // if (bulletPrefab && player)
    }
    ...
}
```

재시작

- Player Input 컴포넌트에서 Event/UI/Restart (CallbackContext)에 GameMgr(GameObject)의 ReStart()연결

using UnityEngine.SceneManagement; // SceneManager 사용을 위해 추가.

```
public class GameMgr : MonoBehaviour
```

```
{  
    private void Awake()  
    {  
        if (null == Instance)  
        {  
            ...  
            SceneManager.sceneLoaded += (scene, mode) => Initialize();  
            return;  
        }  
    }  
    Destroy(gameObject);  
}
```

// sceneLoaded : 씬이 로드될 때 설정한 함수를 호출한다.

// 따라서 Start()에서 Initialize()를 호출하지 않아도 자동으로 호출 된다.

```
/*void Start() { Initialize(); }*/
```

```
public void ReStart(InputAction.CallbackContext context) {  
    if (context.started) SceneManager.LoadScene("SampleScene");  
}
```

