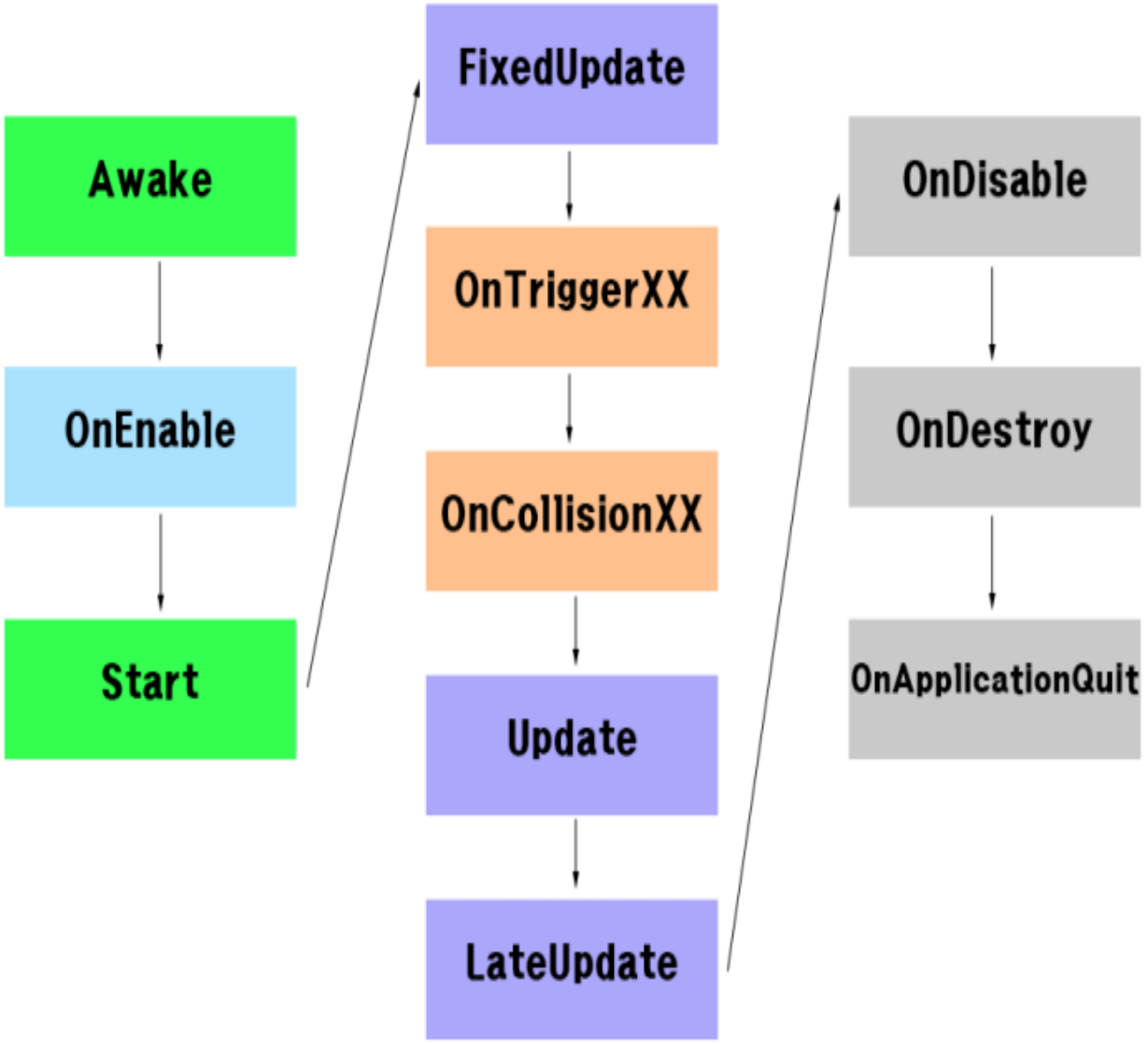
# 유니티
# 이벤트 메소드

학과 : 소프트웨어학과

학번 : 2019975070

이름 : 한재훈



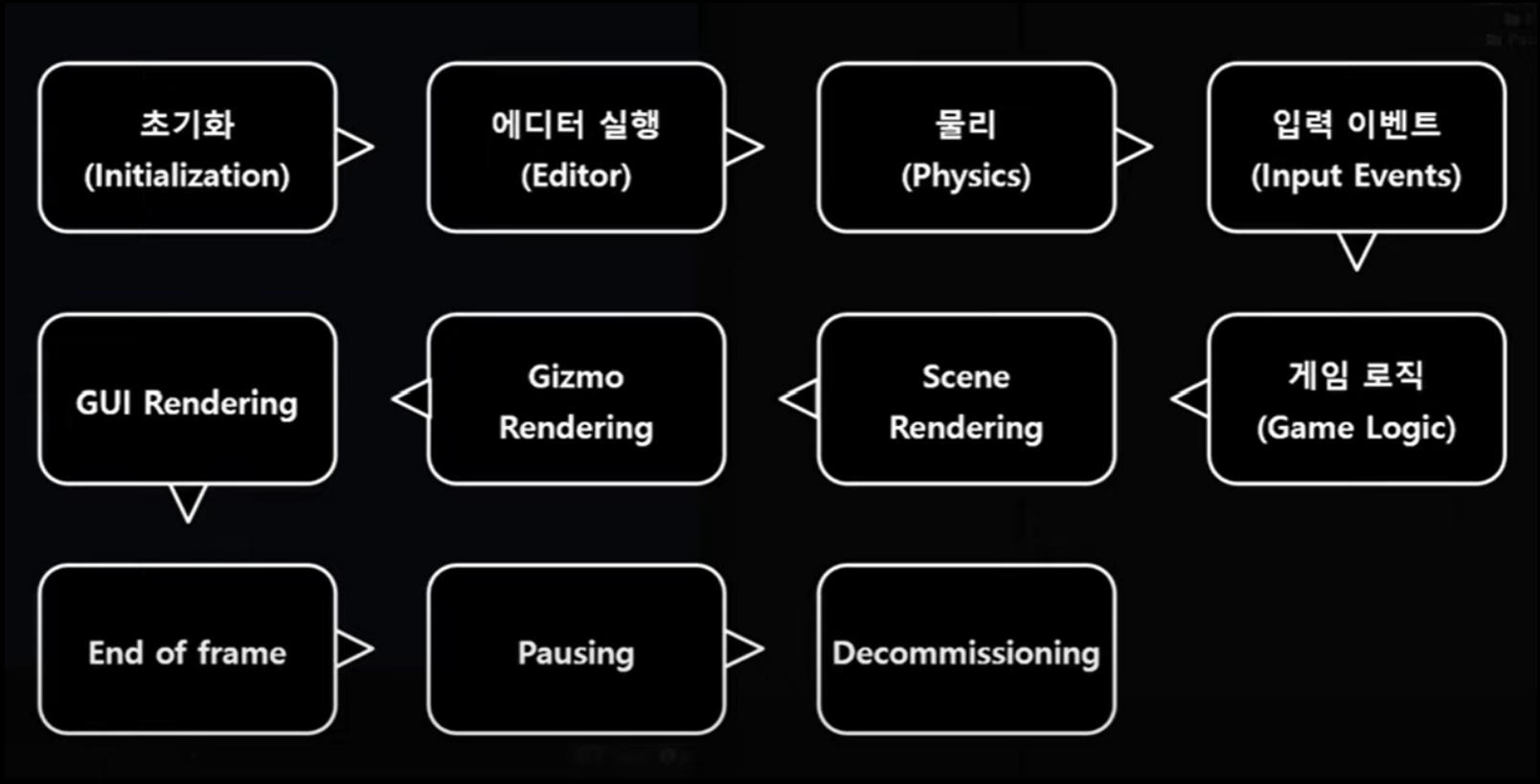
# 목차

- 유니티 이벤트 함수란?
- 이벤트 함수의 실행 순서
- 상황에 따른 이벤트 함수 분류
- 상속과 이벤트 함수
- 주요 이벤트 함수들
  + Awake(), Start(), Update(), FixedUpdate(), LateUpdate(), OnTriggerEnter(Colider other)등

# 유니티 이벤트 함수란?

- **Unity(유니티)**는 게임 개발을 위한 인기 있는 게임 엔진 및 개발 환경으로, 게임 오브젝트의 동작 및 상호작용을 제어하기 위해 사용되는 여러 이벤트 함수를 제공합니다. 이러한 이벤트 함수는 스크립트를 사용하여 게임 오브젝트의 동작을 제어하고 상호작용하는 데 유용합니다.
- **Unity**의 이벤트 함수는 특정한 상황 또는 이벤트가 발생할 때 자동으로 호출되는 함수입니다. 이러한 함수들은 **MonoBehaviour 클래스**를 상속받은 스크립트 컴포넌트에 구현됩니다. 이벤트 함수를 정의하면 **Unity**가 특정 시점에 그 함수를 호출하여 게임 오브젝트의 동작을 제어할 수 있습니다.

# 이벤트 함수 실행 순서


초기화
(Initialization)
에디터 실행
(Editor)
물리
(Physics)
입력 이벤트
(Input Events)
게임 로직
(Game Logic)
Scene
Rendering
Gizmo
Rendering
GUI Rendering
End of frame
Pausing
Decommissioning

# 이벤트 함수 실행 순서

- https://docs.unity3d.com/kr/2021.3/Manual/ExecutionOrder.html
  유니티 문서의 스크립트 라이프사이클 플로우차트를 통해 정확한 이벤트 함수 실행 순서를 확인할 수 있습니다.

- 주요 함수로 쉽게 표현하자면
  **Awake -> Start -> Fixed Update -> OnTrigger -> Update -> LateUpdate-> OnDisable**
  으로 진행 된다고 생각하시면 됩니다.

# 상황에 따른 이벤트 함수 분류 -1-

- **씬이 처음 시작될 때 (First Scene Load)**
  + Awake , OnEnable
- 에디터에서 (Editor)
  + Reset, OnValidate
- **첫번째 프레임 업데이트 전에**
  + Start
- **프레임 사이**
  + OnApplicationPause

# 상황에 따른 이벤트 함수 분류 -2-

- **업데이트 순서**
  + FixedUpdate -> Update -> LateUpdate
- **애니메이션 업데이트 루프**
  + OnStateMachineEnter, OnStateMachineExit, Fire Animation Events, StateMachineBehavior(OnStateEnter/OnStateUpdate/OnStateExit)등등
- **유용한 프로파일 마커**
  + State Machine Update, ProcessGraph, ProcessAnimation등
- **렌더링**
  + OnPreCull,, OnBecameVisible/OnBecameInvisible등

# 상황에 따른 이벤트 함수 분류 -2-

- **코루틴**
  + Yield, yeild WaitForSeconds, yield WaitForFixedUpdate등
- **오브젝트를 파괴할 때**
  + OnDestroy
- **종료할 때**
  + OnApplicationQuit, OnDisable등등

- 자새한 내용은 Unity 문서에 상세히 정리되어있으므로 한번 씩 읽어보시면 좋겠습니다.

# 스크립트(클래스) 구성

- **MonoBehaviour Class** : 유니티에서 제공하는 클래스로 스크립트에 작성된 클래스를 게임오브젝트의 컴포넌트로 적용하고, 컴포넌트화 되었을 때 사용할 수 있는 여러 기능을 담고 있는 클래스
- 스크립트(클래스)는 게임 오브젝트의 컴포넌트로 활용됩니다.
- 스크립트에는 객체의 데이터를 저장할 변수를 저장하고, 유니티 지원 함수와 사용자 정의 함수를 통해 기능을 추가할 수 있습니다. 변수를 저장할때 public으로 저장한다면 unity 에디터 내에서 값을 임의로 개발자가 수정 가능합니다.

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트 | 참조 0개
public class PlayerMovement : MonoBehaviour
{
    public float speed = 10.0f;

    Unity 메시지 | 참조 0개
    void Awake()
    {

    }

    Unity 메시지 | 참조 0개
    void Start()
    {

    }

    Unity 메시지 | 참조 0개
    void FixedUpdate()
    {

    }

    Unity 메시지 | 참조 0개
    void OnCollisionEnter(Collision collision)
    {

    }

    참조 0개
    void UpdateMove()
    {

    }
}
```

# 유니티의 메시지와 브로드캐스팅 시스템

- 독립적인 형태의 컴포넌트 내부에 있는 메소드를 실행시키기 위해 사용하는 방법

# 상속과 이벤트 함수

- 만약 부모 클래스의 Start함수가 있고 자식클래스에도 Start함수가 있으며 , 오브젝트엔 자식 클래스 스크립트가 연결되어있는 상황에서 상속을 받을 때는 이벤트 함수인 Start가 아닌 Init 가상함수를 따로 만들어서 이를 자식이 오버라이딩한 Init 에서 `base.Init()`으로 호출하게 하고 이 Init 을 자식의 Start()안에서 실행시키는 식으로 코딩하면 될 것 같다.

  오브젝트엔 자식 클래스 스크립트가 붙어있는 상황에서 만약 부모에만 Start()가 있으면 이땐 상속처리가 되어 자식 클래스는 부모 클래스의 Start()를 물려 받아 실행한다.

```
// 부모 클래스
public override void Init()
{

}

// 자식 클래스
private void Start()
{
    Init();
}

public override void Init()
{
    base.Init();  // 부모의 Init() 실행

    // ...
      }
```

# 주요 이벤트 함수 (초기화)

- **Awake()** : 씬이 시작될 때 1회 호출
- **Start()** : 첫 번째 Update() 호출 직전 1회 호출출
- **OnEnable()** : 컴포넌트가 활성화 될 때마다 1회 호출

- 호출 순서 : Awake() -> OnEnable() -> Start()

# Awake()

- 데이터를 **초기화하는 목적**으로 만들어진 이벤트 함수
- 현재 씬이 실행된 직후 **1회 호출**된다.

- Awake() 메소드는 게임 오브젝트가 활성화 되어 있을 때 호출된다.
  + 컴포넌트(Awake() 메소드가 포함되어 있는)가 비활성화 되어 있어도 호출된다.
  + 게임 오브젝트가 비활성화 되어 있을 경우 게임 오브젝트가 활성화 되었을 때 호출된다.

# Start()

- Awake() 메소드와 마찬가지로 **초기화를 목적**으로 만들어진 이벤트 메소드
- 첫번째 프레임 업데이트가 실행되기 직전에 **1회 호출** 된다.
  + 호출 순서 : Awake() -> Start()

- Start() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

- 코루틴 형태로 호출이 가능하다.
  + Private IEnumerator Start() {..}

# *Awake()와 Start()의 차이점!

- 둘 다 오브젝트가 생성될 때 (스크립트가 최초로 실행될 때) 최초 1회 실행되는것은 같다.
- <차이점>
  + Awake()
    - 코루틴 실행이 안된다. **Start()보다 먼저 실행된다.**
    - **스크립트(컴포넌트)가 비활성화인 상태에서도 실행된다.** 꺼져있어도 실행 됨. 오브젝트가 활성화되있기만 하면 된다.
    - 오브젝트는 SetActive(true) 해야 하고 & 스크립트 this.enabled = false; 인 상태에선 호출된다는 얘기!
    - 오브젝트 자체가 비활이면 Awake()도 실행 안된다.
  + Start()
    - 코루틴 실행이 가능하다.
- **즉, Awake는 오브젝트가 활성화되자마자 실행되고, 뒤이어 OnEnable과 Start는 스크립트(컴포넌트)가 활성화 되야 실행된다는 얘기다.**

# OnEnable()

- 컴포넌트가 비활성화 되었다가 **다시 활성화 될 때 마다 1회 호출**된다.
  + 최초 호출 순서 : Awake() -> OnEnable() -> Start()

- OnEnable() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

# 주요 이벤트 함수 (업데이트)

- **Update()** : 씬이 시작된 후 매 프레임 호출
- **LateUpdate()** : 현재 씬의 모든 Update() 메소드 실행 직후 호출
- **FixedUpdate()** : 1초에 정해진 횟수만큼 호출 (default : 1초 50회)

# Update()

- 현재 씬이 실행된 후 **매 프레임마다 호출**되는 이벤트 함수
  + 프레임 속도는 환경마다 다르기 때문에 물리 처리를 update() 함수에서 해주면 환경에 따라 물리 처리 오차가 발생할 수 있으므로 비추천
- Update() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

# FixedUpdate()

- 프레임의 영향을 받지 않고 정해진 횟수만큼 호출(고정적이고 동일한 시간)
  + 이 같은 경우로 환경에 상관없이 물리처리를 오차 없이 실행시킬 수 있다.
- 기본 값은 0.02로 1초에 50회 호출
  + Edit -> Project Settings - Time의 "Fixed Timestep"에서 호출 주기 설정 가능

- FixedUpdate() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

# LateUpdate()

- 현재 프레임에서 모든 게임 오브젝트의 Update()가 호출된 후 호출

- LateUpdate() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

- **Tip. LateUpdate()의 경우 플레이어 캐릭터, 카메라와 같이 서로 다른 오브젝트가 존재할 때, 플레이어 캐릭터를 쫓아다니는 카메라를 구현할 때 활용할 수 있다.**

  **플레이어 캐릭터가 Update()를 이용해 움직이고 난 후 카메라는 LateUpdate()에서 플레이어어의 위치를 바탕으로 이동을 할 수 있다.**

# *Update()와 FixedUpdate()의 차이점!

- 차이점
  + FixedUpdate
    - 프레임마다 호출되지 않는다. 독립적인 타이머가 존재하여 정해진, 고정적인 시간 간격으로 호출된다.
    - 프레임과 관계없이 규칙적으로 호출되므로 물리적인 연산을 할 때 이 곳에서 하는게 좋다.
    - 프레임은 시스템 환경을 따라가므로 컴퓨터 환경이 좋지 않으면 느리고 불규칙적으로 변할 수 있기 때문에 Rigidbody 같은 어떤 물리 효과가 적용된 움직임 처리를 *Update* 안에 구현하는건 좋지 않다.
    - `TimeSCale`에 의존하기 때문에 `Time.timeScale = 0;`이 될 때 실행되지 않는다.
    - `Time.fixedDeltaTime`마다 실행된다. 이는 `0.02`초로 고정되어 있다.
  + Update
    - 프레임마다 호출된다.
    - `TimeSCale`에 의존하지 않기 때문에 `Time.timeScale = 0;`이 될 때도 Update 함수 자체는 실행이 된다.
    - 다만 이 안에서 `deltaTime`을 사용하여 움직임을 제어한게 있었다면 멈춘다!

# 주요 이벤트 함수 (트리거)

- **OnTriggerEnter(Colider other)** : Trigger인 Colider와 충도할 때 자동으로 실행된다.
- **OnCollisionEnter(Collision other)** : Trigger가 체크되지 않은 일반 Colider를 가진 오브젝트와 충돌할 경우 자동으로 실행된다.

# OnTriggerEnter(Colider other)

- Is Trigger가 체크된 Colider와 충돌하는 경우 발생되는 메시지이다.
- 충돌하는 두 오브젝트 중 하나만 Trigger라도 두 오브젝트 모두 이 함수가 실행됨
  + 즉 물리적 충돌은 일어나지 않고 닿기 만 하더라도, 뚫고 지나가지만 그래도 이벤트 발생은 시켜야 하는 경우.
- 오브젝트끼리 충돌하면 유니티에서 OnTriggerEnter 메세지를 ***충돌한 오브젝트들*** 에게 브로드캐스팅 한다.
- 유니티는 충돌한 상대 오브젝트의 정보를`Collider`타입의 `other`가 참조하도록 넘겨준다.
- 이 스크립트가 붙은 오브젝트(나 자신)가 다른 오브젝트와 충돌시 `OnTriggerEnter` 이벤트가 발생하기 위한 조건. 아래 조건을 전부 만족해야 이 이벤트가 발생할 수 있다.
  + 내가 혹은 상대방 둘 중 하나는 꼭 Rigidbody 컴포넌트를 가지고 있어야 한다.
    - `IsKinematic` 체크 여부는 상관 없다.
  + 나 그리고 상대방 둘 다 모두 Collider 컴포넌트를 가지고 있어야 한다.
    - 단, 둘 중 하나라도 `IsTrigger`는 반드시 켜져 있어야 함

# OnTriggerEnter, OnTriggerExit, OnTriggerStay의 차이

- OnTrigger**Enter** : `Enter`는 충돌하는 순간
- OnTrigger**Exit** : `Exit`는 떼어지는 순간. 더 이상 붙어 있지 않는 순간.
- OnTrigger**Stay** : `Stay`는 충돌 중인, 붙어 있는 동안.

# OnCollisionEnter(Collision other)

- **Collider와 Collision의 차이**
  + Collision은 충돌한 상대 오브젝트에 대한 많은 정보를 담고 있다. 나랑 부딪친 오브젝트의 Transform, Collider, GameObject, Rigidbody, 상대 속도 등등이 Collision에 담겨서 들어 온다. 물리적인 정보가 더 많이 들어 있다.
  + Collider는 Collision보다는 담고 있는 정보가 적다. 물리적인 정보는 담고 있지 않아서.
- 이 스크립트가 붙은 오브젝트(나 자신)가 다른 오브젝트와 충돌시 OnCollisionEnter 이벤트가 발생하기 위한 조건. 아래 조건을 전부 만족해야 이 이벤트가 발생할 수 있다.
  + 내가 혹은 상대방 둘 중 하나는 꼭 Rigidbody 컴포넌트를 가지고 있어야 한다.
    - IsKinematic은 꺼져 있어야 함
    - 즉, 둘 중 하나는 꼭 충돌로 인한 물리적인 힘에 영향을 받을 수 있는 상태여야 함. 그래서 OnCollisionEnter는 뭔가 물리적인 힘에 의한 충돌 느낌
  + 나 그리고 상대방 둘 다 모두 Collider 컴포넌트를 가지고 있어야 한다.
    - IsTrigger는 꺼져 있어야 함
- FPS 게임 같은데서 총알이 사람에게 맞으면 총알 입장에선 사람 오브젝트 정보가 Collision으로 들어오게 되므로 그 사람의 체력을 깎거나 하는 처리를 할 수 있다.
    -

# 주요 이벤트 함수 (해체)

- **OnDestroy()** : 게임 오브젝트가 파괴될 때 1회 호출
- **OnApplicationQuit()** : 게임이 종료될 때 1회 호출
- **OnDisable()** : 컴포넌트가 비활성화 될 때마다 1회 호출

# OnDestory()

- **게임오브젝트가 파괴될 때 1회 호출**된다.
- 씬이 변경되거나 게임이 종료될 때도 오브젝트가 파괴되기 때문에 호출된다.

- OnDestroy() 메소드는 게임 오브젝트가 활성화 되어 있을 때 호출된다.
  + 컴포넌트가 비활성화 되어 있어도 호출된다.
  + 게임 오브젝트가 비활성화 되어 있을 경우 게임 오브젝트가 활성화 되었을 때 호출된다.

# OnApplicationQuit()

- **게임이 종료될 때 1회 호출**된다
  + Unity Editor에서는 플레이 모드를 중지할 때 호출된다.

- OnApplicationQuit() 메소드는 게임 오브젝트트가 활성화 되어 있을 때 호출된다.
  + 컴포넌트가 비활성화 되어 있어도 호출된다.
  + 게임 오브젝트가 비활성화 되어 있을 경우 게임 오브젝트가 활성화 되었을 때 호출된다.

# OnDisable()

- **컴포넌트가 비활성화 될 때 마다 1회 호출**된다.

- OnDisable() 메소드는 게임 오브젝트, 컴포넌트가 활성화 되었을 때만 호출된다.

# 주요 이벤트 함수 (기즈모)

- OnDrawGizmos()
  + Unity Editor의 Scene VIew에만 출력되는 선, 도형
  + 게임 제작 시 광선, 충돌 범위와 같이 눈에 보이지 않는 것을 확인할 때 사용하는 이벤트 함수

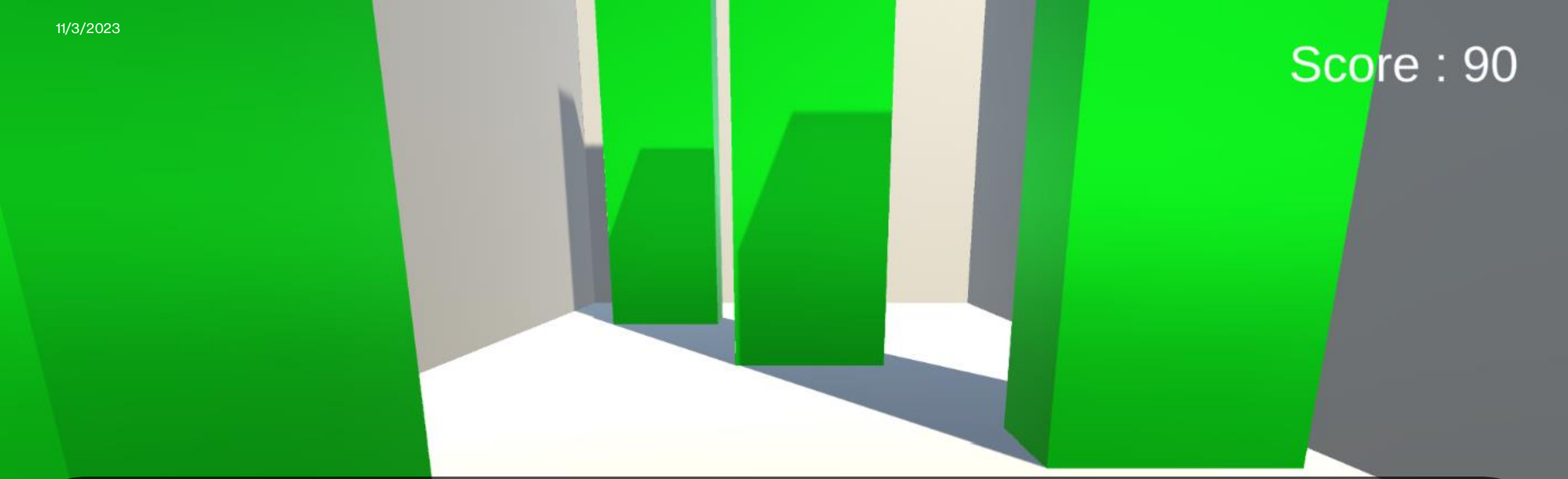
- OnDrawGizmos() 메소드는 게임 오브젝트가 활성화 되어 있을 때 호출된다.
  + 컴포넌트가 비활성화 되어 있어도 호출된다.
  + 게임 오브젝트가 비활성화 되어 있을 경우 게임 오브젝트가 활성화 되었을 때 호출된다.

# 사용자 지정 이벤트

- 해당 이벤트에 원하는 함수가 들어 있는 스크립트들을 드래그 해 와서 발동시킬 함수들을 선택하면 해당 이벤트를 발동시켰을 때 등록한 함수들도 다 같이 실행된다.
  - ***using UnityEngine.Events;***
  - **myEvent라는 이름의 사용자 지정 이벤트 변수를 선언한다.**
  - **이제 유니티에서 myEvent 슬롯이 열릴텐데 여기에 원하는 스크립트를 드래그 앤 드롭해준다.**
  - **원하는 함수들을 선택한다.**
  - **invoke() 해주면 등록한 함수들이 모두 실행된다.**

# Invoke()

- MonoBehaviour 에서 지원하는 함수로, 함수 혹은 이벤트를 실행시킨다.
  + 이벤트이름.Invoke() : 이벤트 발동
  + Invoke(string) : 이름을 문자열로 넣으면 그 이름의 함수를 실행시킨다.
  + Invoke(string, floaT) : 매개 변수로 시간도 넣을 수 있다. *Invoke("Restart", 5f)* 해주면 5초 뒤에 Restart() 함수를 실행시켜라라는 의미다.




# Unity 이벤트 함수를 활용하여 만든 예제

- 사용 이벤트 함수
- Start() , Update()

# Player 컴포넌트

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

Unity 스크립트(자산 참조 1개)|참조 0개
public class Player : MonoBehaviour
{
    private Rigidbody rigid;
    private float moveForce = 7.0f;
    private float x_Axis;
    private float z_Axis;

    // Start is called before the first frame update
    Unity 메시지|참조 0개
    void Start()
    {
        rigid = GetComponent<Rigidbody>();
    }

    // Update is called once per frame
    Unity 메시지|참조 0개
    void Update()
    {
        x_Axis = Input.GetAxis("Horizontal");
        z_Axis = Input.GetAxis("Vertical");

        Vector3 velocity = new Vector3(x_Axis, 0, z_Axis);
        velocity *= moveForce;
        rigid.velocity = velocity;
    }
```

```
// OnCollisionEnter 함수는 트리거와 충돌하는 순간 호출됩니다.
Unity 메시지|참조 0개
void OnCollisionEnter(Collision collision)
{
    if (collision.gameObject.CompareTag("Enemy"))
    {
        // "Enemy" 태그를 가진 오브젝트와 충돌한 경우 게임 종료
        GameOver();
    }
}

참조 1개
void GameOver()
{
    // 게임 오버 시 원하는 동작 수행
    // 여기서는 간단하게 현재 씬을 재로드하는 것으로 대체합니다.
    int currentSceneIndex = SceneManager.GetActiveScene().buildIndex;
    SceneManager.LoadScene(currentSceneIndex);
}
```

# Player 컴포넌트 설명

- `private Rigidbody rigid;`은 Rigidbody 컴포넌트에 대한 참조를 저장합니다. Rigidbody는 게임 오브젝트의 물리 엔진 동작을 제어하는 데 사용됩니다.
- `private float moveForce = 7.0f;`는 플레이어의 이동 속도를 설정합니다.
- `private float x_Axis;`와 `private float z_Axis;`는 사용자 입력에 따라 플레이어의 움직임을 저장할 변수들입니다.
- `Start()` 함수는 스크립트가 시작될 때 호출되며, 여기서 **`rigid`** 변수에 Rigidbody 컴포넌트를 할당합니다.
- `Update()` 함수는 매 프레임마다 호출됩니다. 여기서 사용자 입력을 감지하고, 입력에 따라 플레이어를 이동시키는 역할을 합니다. **`Input.GetAxis`**를 사용하여 사용자의 화살표 키나 WASD 입력에 따른 이동 방향을 감지하고, 이동 방향에 **`moveForce`**를 곱해 이동 속도를 조절하며, **`rigid.velocity`**를 통해 플레이어를 움직입니다.
- `OnCollisionEnter(Collision collision)` 함수는 플레이어와 다른 오브젝트의 충돌을 감지합니다. 여기서 **`collision.gameObject.CompareTag("Enemy")`**를 사용하여 충돌한 오브젝트의 태그가 "Enemy"인 경우 게임을 종료하도록 설정됩니다.
- `GameOver()` 함수는 게임을 종료하는 동작을 수행합니다. 이 경우, 현재 씬의 인덱스를 가져와서 **`SceneManager.LoadScene(currentSceneIndex)`**를 호출하여 현재 씬을 다시 로드하여 게임을 재시작하게 됩니다. 이렇게 간단한 게임 종료 및 재시작 로직을 구현하는 예시입니다.

# Obstacles 컴포넌트

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트|참조 0개
public class Obstacle : MonoBehaviour
{
    public GameObject cubePrefab; // 큐브 프리팹
    public float spawnInterval = 5.0f; // 생성 간격
    public float cubeSpeed = 5.0f; // 큐브 이동 속도

    private float timeSinceLastSpawn = 0.0f;

    // Update is called once per frame
    Unity 메시지|참조 0개
    void Update()
    {
        timeSinceLastSpawn += Time.deltaTime;
        if (timeSinceLastSpawn >= spawnInterval)
        {
            SpawnCube();
            timeSinceLastSpawn = 0.0f;
        }
    }

    참조 1개
    void SpawnCube()
    {
        // 큐브를 (0, 15, 30) 위치에서 x 좌표가 -7에서 7 사이의 랜덤한 값으로 생성
        float randomX = Random.Range(-7f, 7f);
        Vector3 spawnPosition = new Vector3(randomX, 15f, 30f);

        GameObject newCube = Instantiate(cubePrefab, spawnPosition, Quaternion.ider

        // 큐브에 Rigidbody 컴포넌트를 가져와서 설정
        Rigidbody cubeRigidbody = newCube.GetComponent<Rigidbody>();
        if (cubeRigidbody != null)
        {
            Vector3 direction = Vector3.forward; // 이동 방향 설정
            cubeRigidbody.velocity = direction * cubeSpeed;
        }

        // 큐브를 삭제할 스크립트를 추가
        CubeMover mover = newCube.AddComponent<CubeMover>();
        mover.SetSpeed(5.0f); // Z 좌표 감소 속도 설정
    }
```

# Obstacles 컴포넌트 설명

- `public GameObject cubePrefab;`는 큐브 오브젝트의 프리팹을 연결하기 위한 변수입니다. 큐브의 디자인과 속성은 이 프리팹을 기반으로 생성됩니다.
- `public float spawnInterval = 5.0f;`는 큐브 생성 간격을 설정합니다. 큐브가 얼마나 자주 생성될지를 결정하는 값으로, 5.0초로 설정되어 있습니다.
- `public float cubeSpeed = 5.0f;`는 큐브의 이동 속도를 설정합니다. 큐브가 얼마나 빠르게 이동할지를 결정하는 값으로, 5.0으로 설정되어 있습니다.
- `timeSinceLastSpawn` 변수는 마지막 큐브 생성 이후의 경과 시간을 추적하는 데 사용됩니다.
- `Update()` 함수는 매 프레임마다 호출되며, `timeSinceLastSpawn` 변수를 업데이트하고, 설정된 `spawnInterval` 시간이 경과하면 `SpawnCube()` 함수를 호출하여 큐브를 생성합니다.
- `SpawnCube()` 함수는 큐브를 생성하는 역할을 합니다. 큐브의 생성 위치는 (0, 15, 30)에서 시작하며, x 좌표는 -7에서 7 사이의 랜덤한 값으로 설정됩니다.
- 생성된 큐브에는 Rigidbody 컴포넌트를 가져와서 큐브를 이동시킬 속도와 방향을 설정합니다. 현재 방향은 Z 축 방향으로 설정되어 큐브가 화면 앞쪽으로 움직이게 됩니다.
- 마지막으로 큐브를 삭제할 스크립트 `CubeMover`를 추가하고, Z 좌표를 감소시키는 속도를 설정합니다. 이것은 큐브가 일정 시간마다 Z 좌표를 변경하여 화면에서 벗어나면 삭제되도록 만드는 역할을 합니다.

# CubeMove 컴포넌트

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity 스크립트|참조 2개
public class CubeMover : MonoBehaviour
{
    private float speed = 1.0f;

    참조 1개
    public void SetSpeed(float newSpeed)
    {
        speed = newSpeed;
    }

    // Update is called once per frame
    Unity 메시지|참조 0개
    void Update()
    {
        // 매 프레임마다 Z 좌표를 감소시킴
        Vector3 position = transform.position;
        position.z -= speed * Time.deltaTime;
        transform.position = position;

        // Z 좌표가 -12 이하로 내려가면 오브젝트를 삭제
        if (position.z < -12f)
        {
            Destroy(gameObject);
        }
    }
```

# CubeMove 컴포넌트 설명

- `private float speed = 1.0f;`는 오브젝트의 이동 속도를 설정하는 변수입니다. 이 스크립트는 Z 축 방향으로 일정한 속도로 오브젝트를 이동시키는 역할을 합니다.
- `public void SetSpeed(float newSpeed)` 함수는 **speed** 변수를 설정하는 메서드입니다. 이를 통해 외부에서 이동 속도를 변경할 수 있습니다.
- `Update()` 함수는 매 프레임마다 호출됩니다. 이 함수 내에서 오브젝트의 위치를 업데이트하여 부드럽게 이동시킵니다.
- `Vector3 position = transform.position;`는 현재 오브젝트의 위치를 **position** 변수에 복사합니다.
- `position.z -= speed * Time.deltaTime;`는 Z 축 좌표를 **speed** 변수에 지정된 속도로 매 프레임마다 감소시킵니다. 이렇게 함으로써 오브젝트가 Z 축 방향으로 부드럽게 이동합니다.
- `transform.position = position;`은 갱신된 위치를 실제 오브젝트의 위치로 설정합니다.
- `if (position.z < -12f)`는 Z 축 좌표가 -12 이하로 내려가면 오브젝트를 삭제하는 조건을 확인합니다.
- `Destroy(gameObject);`는 조건이 충족될 경우 현재 스크립트가 연결된 게임 오브젝트를 삭제합니다.

# ScoreManger 컴포넌트

```
using System.Collections;
using System.Collections.Generic;
using TMPro;
using UnityEngine;
using UnityEngine.SocialPlatforms.Impl;

Unity 스크립트|참조 0개
public class ScoreManager : MonoBehaviour
{
    public TextMeshProUGUI scoreText;  // UI 텍스트
    private int score = 0;
    private float scoreIncreaseInterval = 1.0f;  // 매 초마다 스코어 증
    private float timeSinceLastIncrease = 0.0f;

    // Update is called once per frame
    Unity 메시지|참조 0개
    void Update()
    {
        // 일정 시간마다 스코어 증가
        timeSinceLastIncrease += Time.deltaTime;
        if (timeSinceLastIncrease >= scoreIncreaseInterval)
        {
            IncreaseScore(10);
            timeSinceLastIncrease = 0.0f;
        }
    }

    참조 1개
    void IncreaseScore(int amount)
    {
        score += amount;
        scoreText.text = "Score : " + score.ToString();
    }
}
```

# ScoreManger 컴포넌트 설명

- `public TextMeshProUGUI scoreText;`는 Unity UI 텍스트 오브젝트를 연결하기 위한 변수입니다. 이 변수를 통해 게임 화면에 현재 점수를 표시합니다.
- `private int score = 0;`는 현재의 점수를 저장하는 변수입니다. 초기 점수는 0으로 설정됩니다.
- `private float scoreIncreaseInterval = 1.0f;`는 스코어를 증가시킬 간격을 설정합니다. 이 경우, 매 초마다 스코어가 증가하도록 설정되어 있습니다.
- `private float timeSinceLastIncrease = 0.0f;`는 마지막 스코어 증가 이후의 경과 시간을 추적하는 변수입니다.
- `Update()` 함수는 매 프레임마다 호출되며, `timeSinceLastIncrease` 변수를 업데이트하고, 설정된 `scoreIncreaseInterval` 시간이 경과하면 `IncreaseScore(10)` 함수를 호출하여 점수를 증가시킵니다. 매 초마다 10씩 스코어가 증가하도록 구현되어 있습니다.
- `IncreaseScore(int amount)` 함수는 스코어를 증가시키는 역할을 합니다. 입력된 `amount`만큼 현재의 점수에 더하고, `scoreText.text`를 통해 UI 텍스트에 현재 점수를 표시합니다.
- 이 스크립트를 게임 오브젝트에 연결하면, 게임 화면에 표시된 UI 텍스트가 매 초마다 10씩 증가하는 형태로 점수가 표시됩니다. 이것은 간단한 스코어 관리 시스템을 구현하는 예시입니다.
-

# 참고자료

- https://docs.unity3d.com/kr/2021.3/Manual/ExecutionOrder.html - Unity 공식 문서(이벤트함수)
- https://ansohxxn.github.io/unitydocs/eventmethod/ - 이벤트 함수 정리