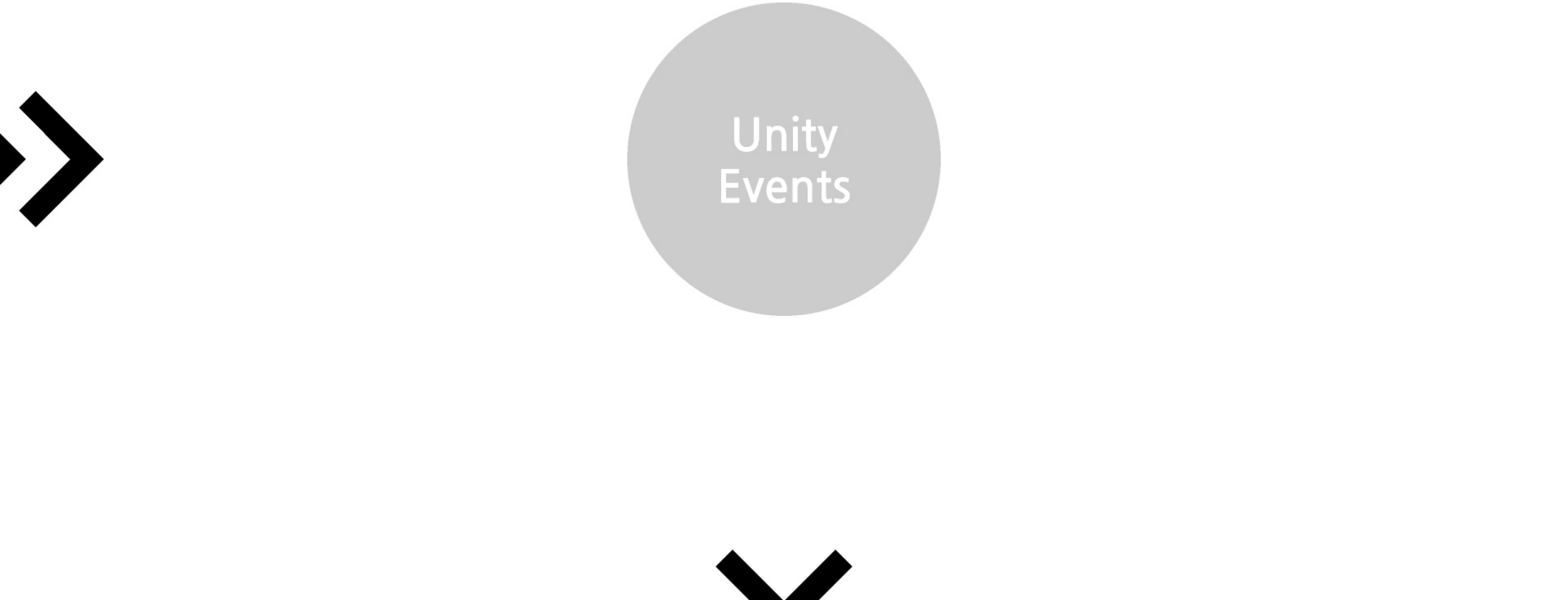
이벤트와 델리게이트

챌린지반 특강 2강

김하연 튜터

Unity 이벤트 시스템



Input Events

- 키보드 입력
  - Input.GetKeyDown(KeyCode key)
  - Input.GetKey(KeyCode key)
  - Input.GetKeyUp(KeyCode key)
- 마우스 입력
  - Input.GetMouseButtonDown(int button)
  - Input.GetMouseButton(int button)
  - Input.GetMouseButtonUp(int button)
- 게임패드/조이스틱 입력
  - Input.GetAxis(string axisName)
- 기타 입력
  - Input.touchCount
  - Input.GetTouch(int index)

```
void Update() {
    // 스페이스바가 눌렸을 때
    if (Input.GetKeyDown(KeyCode.Space)) {}

    // "W" 키가 눌려있는 동안
    if (Input.GetKey(KeyCode.W)) {}

    // "S" 키가 떼어졌을 때
    if (Input.GetKeyUp(KeyCode.S)) {}

    // 마우스 좌클릭이 눌렸을 때
    if (Input.GetMouseButtonDown(0)) {}

    // 마우스 우클릭이 눌려있는 동안
    if (Input.GetMouseButton(1)) {}

    // 마우스 중앙 클릭이 떼어졌을 때
    if (Input.GetMouseButtonUp(2)) {}

    // 게임패드의 수평축과 수직축 입력 값
    float horizontal = Input.GetAxis("Horizontal");
    float vertical = Input.GetAxis("Vertical");

    if (horizontal != 0 || vertical != 0) {}
}
```

Unity Engine Events

- 생명주기 이벤트
  - Awake, Start, Update, FixedUpdate, LateUpdate
  - OnEnable/OnDisable, OnDestory
- 렌더링 이벤트
  - OnRenderObject, OnPreCull, OnPostRender

- 이벤트의 목적과 타이밍을 잘 이해하고 사용할 것
- 과도한 Update 사용은 피할 것
- 물리 계산은 FixedUpdate에서 사용. 고정된 시간 간격으로 호출되기 때문!

```
0 references
public class LifecycleExample : MonoBehaviour {
    0 references
    void Awake() {
        Debug.Log("오브젝트가 생성될 때 호출됩니다.");
    }

    0 references
    void Start() {
        Debug.Log("첫 번째 Update 호출 전에 실행됩니다.");
    }

    0 references
    void Update() {
        Debug.Log("매 프레임마다 호출됩니다.");
    }

    0 references
    void FixedUpdate() {
        Debug.Log("고정된 시간 간격으로 호출됩니다 (주로 물리 계산에 사용).");
    }

    0 references
    void LateUpdate() {
        Debug.Log("모든 Update 메소드가 호출된 후에 실행됩니다.");
    }

    0 references
    void OnEnable() {
        Debug.Log("오브젝트가 활성화될 때 호출됩니다.");
    }

    0 references
    void OnDisable() {
        Debug.Log("오브젝트가 비활성화될 때 호출됩니다.");
    }

    0 references
    void OnDestroy() {
        Debug.Log("오브젝트가 파괴될 때 호출됩니다.");
    }
}
```

Physics Evnets

- **충돌 이벤트**
  - OnCollisionEnter: 다른 물체와 충돌할 때 처음 호출
  - OnCollisionExit: 충돌한 물체로부터 떨어질 때 호출
  - OnCollisionStay: 다른 물체와 충돌하고 있는 동안 지속적으로 호출
- **트리거 이벤트**
  - OnTriggerEnter: Collider가 다른 Collider와 겹칠 때
  - OnTriggerExit: Collider가 다른 Collider로부터 떨어질 때 호출함
  - OnTriggerStay: 두 Collider가 겹쳐져 있는 동안 지속적으로 호출함

- **많은 수의 트리거와 충돌 이벤트는 게임의 성능을 저하시킴**
- **물리 이벤트는 물리 엔진의 업데이트 주기에 따라 호출되므로 Update() 메소드와는 다른 타이밍으로 실행될 수 있음.**

Animation Events

- **애니메이션 상태 변화 이벤트**
  - OnStateEnter: 애니메이션의 시작
  - OnStateUpdate: 애니메이션 상태가 진행되는 동안 매 프레임에 호출
  - OnStateExit: 애니메이션 상태가 종료될 때 호출

- OnStateUpdate는 사용을 주의할 것. 매 프레임 호출되므로 성능과 직접적 영향.
- 애니메이션 상태 변화 이벤트는 애니메이션 클립과 연결되어 있어야 함.

UI Events

- **버튼 클릭, 드래그, 드롭 이벤트**
  - OnClick, OnDrag, OnDrop
- **UI 상호작용 이벤트**
  - IPointerClickHandler, IPointerEnterHandler, IPointerExitHandler

이벤트와 델리게이트

챌린지반 특강 2강

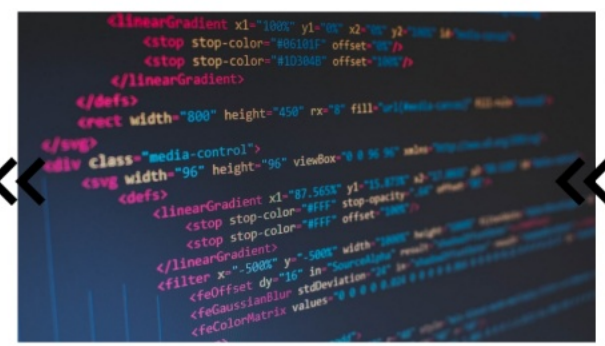
김하연 튜터

Events

- 장점
  - **캡슐화**: 이벤트는 클래스 외부에서 직접 발생시킬 수 없어서, 캡슐화를 통해 안전한 코드 관리가 가능함.
  - **의사소통 강화**: 이벤트는 컴포넌트 간의 소통을 명확하게 하므로, 가독성과 유지보수성을 향상시킴.
- 단점
  - **이해도 요구**: 이벤트의 개념과 사용 방법을 이해하는 데 시간이 소요될 수 있음
  - **오버헤드**: 이벤트 시스템이 복잡해질 수록 성능에 미치는 영향도 커짐

Delegates

- 장점
  - **유연성**: 다양한 메소드를 동일한 델리게이트 변수에 할당할 수 있음
  - **코드 재사용 및 분리**: 코드의 재사용성을 높이고, 컴포넌트 간의 결합도를 낮춤
- 단점
  - **복잡성**: 델리게이트의 사용이 과도하면 코드의 복잡성이 증가할 수 있음
  - **메모리 누수 위험**: 잘못 관리되면 메모리 누수를 일으킴



Unity 이벤트 시스템

Unity Events

Customize Events

C# 델리게이트와 이벤트
개발자가 정의한 특정조건이나 행동에 반응하는 사용자 정의 이벤트

- **델리게이트**
  - 델리게이트는 C#에서 메소드를 참조하는 타입
  - 메소드의 참조를 변수에 저장하고, 다른 메소드로 전달하거나 메소드 호출을 동적으로 결정할 수 있음
- **이벤트**
  - 이벤트는 델리게이트를 기반으로 함
  - 특정 상황이 발생했을 때 이벤트를 구독하는 모든 메소드를 호출하는데 사용됨 (멀티캐스트)

Customize Events

C# 델리게이트와 이벤트
개발자가 정의한 특정조건이나 행동에 반응하는 사용자 정의 이벤트

- **델리게이트**
  - 델리게이트는 C#에서 메소드를 참조하는 타입
  - 메소드의 참조를 변수에 저장하고, 다른 메소드로 전달하거나 메소드 호출을 동적으로 결정할 수 있음
- **이벤트**
  - 이벤트는 델리게이트를 기반으로 함
  - 특정 상황이 발생했을 때 이벤트를 구독하는 모든 메소드를 호출하는데 사용됨 (멀티캐스트)

Delegates

- **장점**
  - **유연성**: 다양한 메소드를 동일한 델리게이트 변수에 할당할 수 있음
  - **코드 재사용 및 분리**: 코드의 재사용성을 높이고, 컴포넌트 간의 결합도를 낮춤
- **단점**
  - **복잡성**: 델리게이트의 사용이 과도하면 코드의 복잡성이 증가할 수 있음
  - **메모리 누수 위험**: 잘못 관리되면 메모리 누수를 일으킴

```
1 reference
public delegate void MyDelegate(int value);
MyDelegate myDelegate = MyFunction;

void MyFunction(int number) {
    // 실행할 코드
}

myDelegate(5);
```

Events

- **장점**
  - **캡슐화**: 이벤트는 클래스 외부에서 직접 발생시킬 수 없어서, 캡슐화를 통해 안전한 코드 관리가 가능함.
  - **의사소통 강화**: 이벤트는 컴포넌트 간의 소통을 명확하게 하므로, 가독성과 유지보수성을 향상시킴.
- **단점**
  - **이해도 요구**: 이벤트의 개념과 사용 방법을 이해하는 데 시간이 소요될 수 있음
  - **오버헤드**: 이벤트 시스템이 복잡해질 수록 성능에 미치는 영향도 커짐

```
0 references
public event MyDelegate MyEvent;

// 구독
someObject.MyEvent += MyEventHandler;

// 발행
if (someCondition) {
    MyEvent?.Invoke(10);
}
```