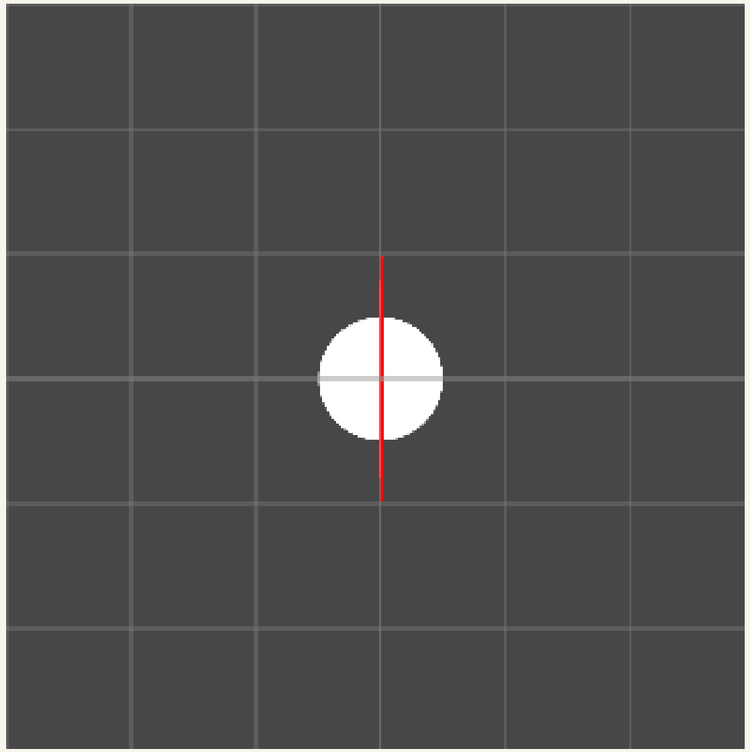
유니티 및 C# 스터디

made by 박준영

목차

오브젝트 이동



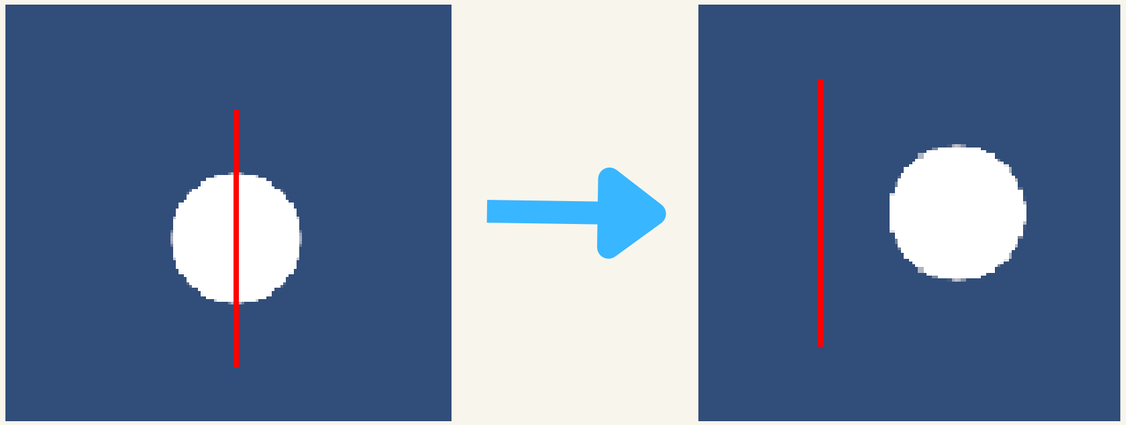
Scene창

- 2D에서는 X,Y 좌표를 사용함
- 오른쪽은 +값 왼쪽은 -값
- 위는 +값 아래는 +값

오브젝트 이동

```
Unity 메시지 | 참조 0개
private void Awake()
{
    // 새로운 위치 = 현재 위치 + (방향 * 속도)
    transform.position = transform.position + new Vector3(1, 0, 0) * 1;

    // transform.position += Vector3.right * 1;
}
```



움직임 스크립트

transform : 내가 소속되어 있는 게임 오브젝트의 Transform 컴포넌트

position : 좌표

- Awake()에서 1회 실행
- 현재 transform.position에 방향 * 속도를 더해 새로운 위치를 설정

- left, right, up, down은 유니티에서 제공하고 있음
- right의 경우 (1, 0 ,0)으로 설정됨

오브젝트 이동

```
Unity 메시지 | 참조 0개
private void Update()
{
    transform.position += Vector3.right * 1 * Time.deltaTime;
}
```

스크립트

- 사양이 좋지 않은 컴퓨터(60회 호출)

- 사양이 좋은 컴퓨터(120회 호출)

- Time.deltaTime이 없으면 컴퓨터 사양에 따라 이동하는 거리가 다름

■Time.deltaTime

- 이전 Update() 종료부터 다음 Update() 시작까지의 시간
- ex) 1분에 Update()가 60번 호출된다면 Time.deltaTime은 1

- 사양이 좋지 않은 컴퓨터가 60초에 Update()를 60번 호출할동안 사양이 좋은 컴퓨터는 60초에 120번 호출
- 이때 각각의 Time.deltaTime 값은 1, 0.5
- Time.deltaTime을 곱하면 이동거리가 같아짐

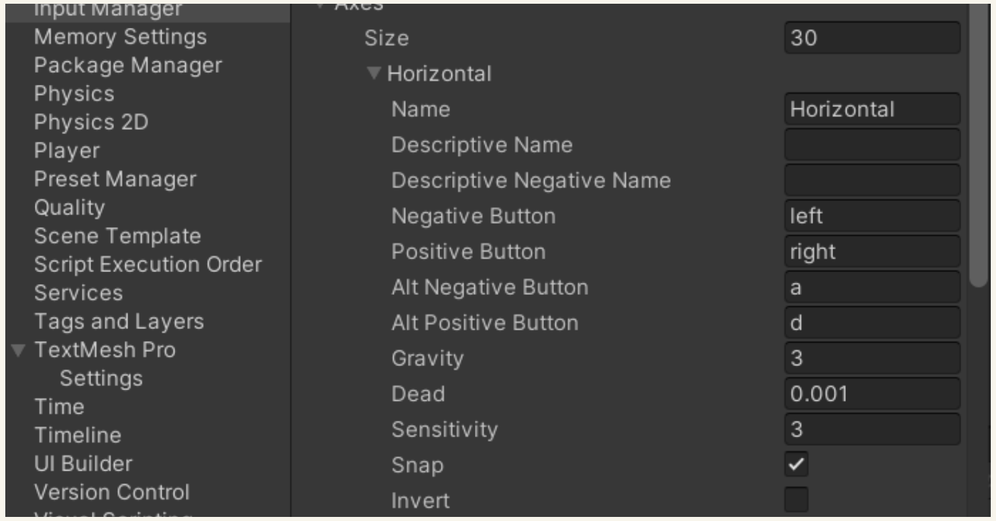
오브젝트 이동

```
private float moveSpeed = 5.0f;                    // 이동 속도
private Vector3 moveDirection = Vector3.zero;      // 이동 방향

Unity 메시지 | 참조 0개
private void Update()
{
    float x = Input.GetAxisRaw("Horizontal");      // 좌우 이동
    float y = Input.GetAxisRaw("Vertical");        // 위아래 이동

    // 이동방향 설정
    moveDirection = new Vector3(x, y, 0);

    //새로운 위치 = 현재 위치 + 방향 * 속도
    transform.position += moveDirection * moveSpeed * Time.deltaTime;
}
```



스크립트와 InputManager

■Input.GetAxisRaw(“string”)

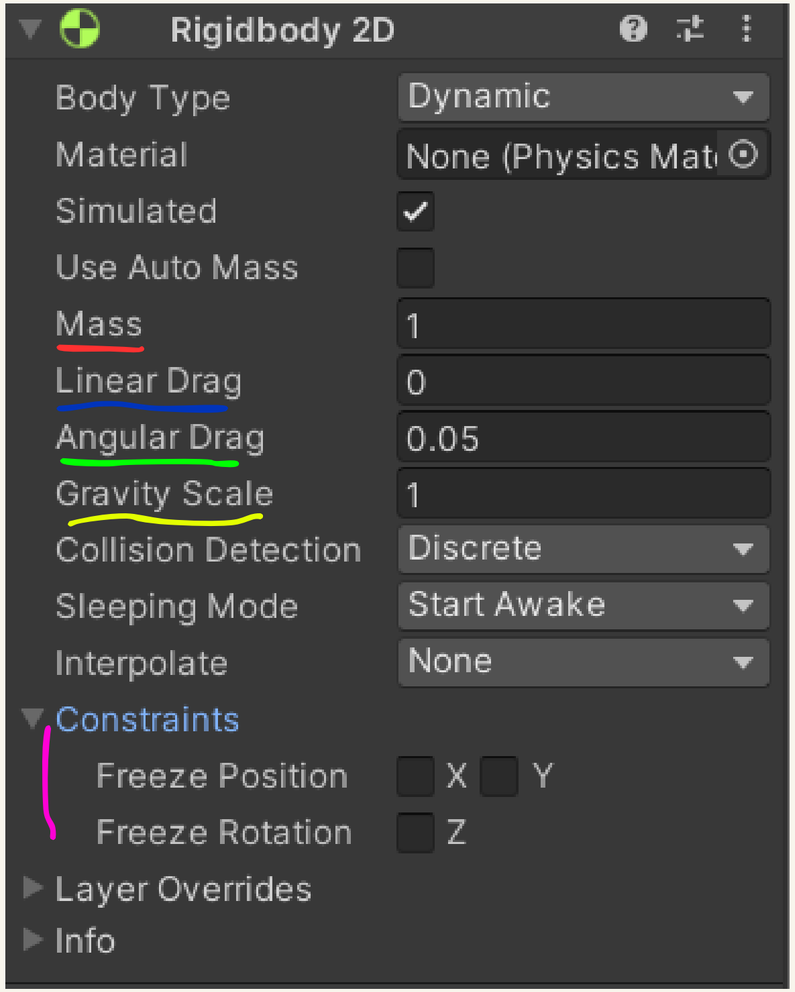
- 수평, 수직 버튼 입력을 받으면
  -1, 0, 1의 값을 반환함

■Input.GetAxis(“string)

- 수평, 수직 버튼 입력을 받으면
  -1, 0, 1의 값으로 서서히 증가시킴
- 부드러운 이동을 표현할 때 사용

- Edit->Project Settings
  ->Input Manager에서 확인 가능

오브젝트 물리와 충돌



Rigidbody

■Rigidbody2D

- 2차원 공간에서 오브젝트의 물리와 중력을 담당하는 컴포넌트

-Mass : 오브젝트의 질량

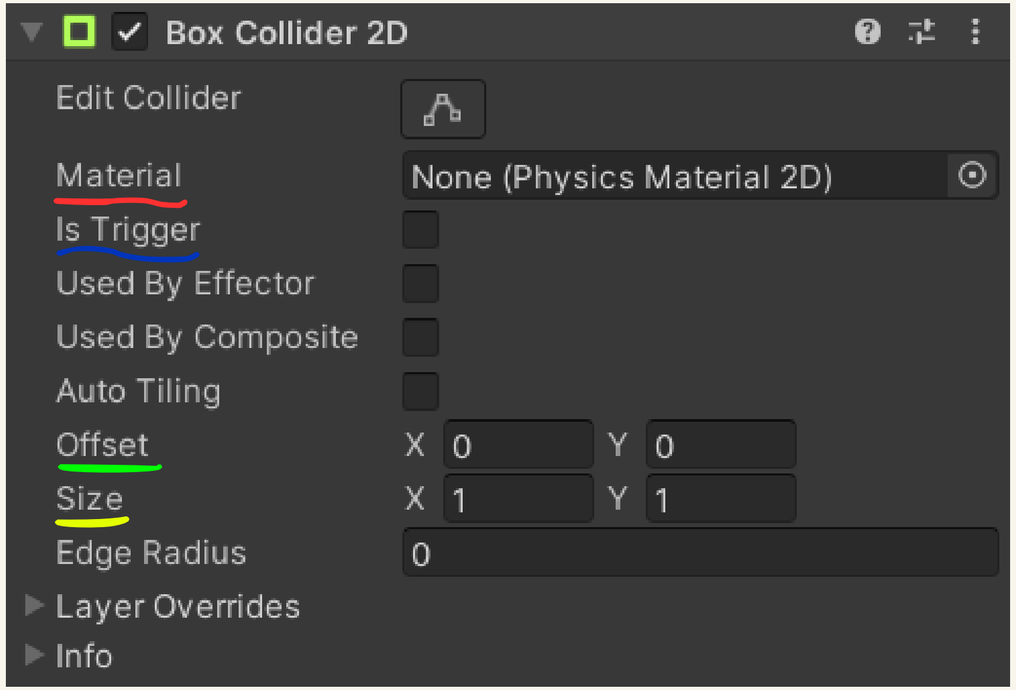
-Linear Drag : 위치 움직임에 대한 마찰력

-Angular Drag : 회전 움직임에 대한 마찰력

-Gravity Scale : 오브젝트 중력 계수(-9.81 * Gravity Scale)

-Constraints : 체크된 축은 외부로부터 받은 물리력에 의해 이동, 회전하지 않음

오브젝트 물리와 충돌



Collider

■Collider2D

- 2차원 공간에서 오브젝트의 충돌 범위를 나타내는 컴포넌트

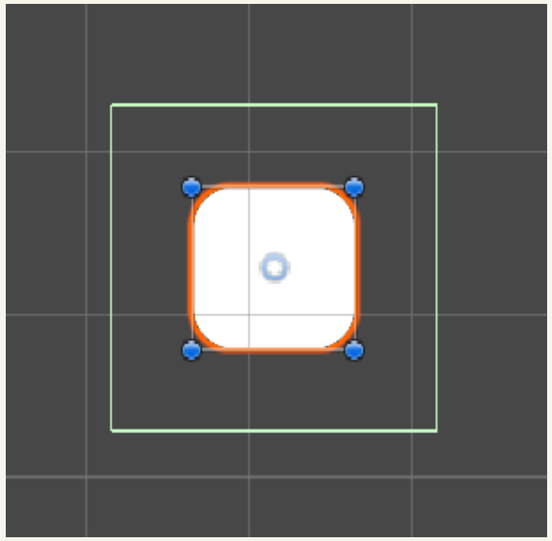
-Material : 해당 오브젝트의 마찰력 등을 설정할 수 있는 물리 메테리얼 등록 가능

-Is Trigger : 위치 움직임에 대한 마찰력

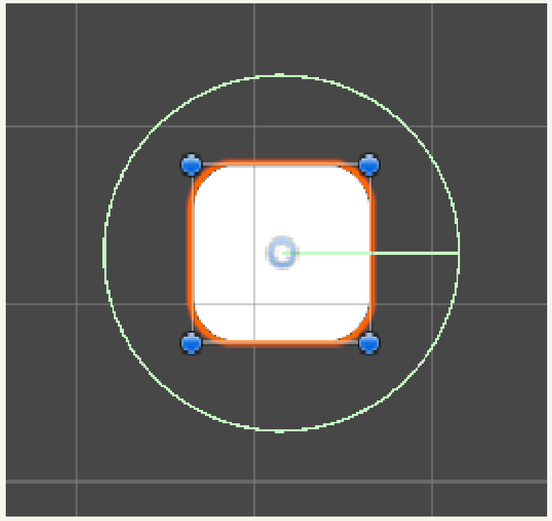
-Offset : 충돌 범위 중심점 위치

-Size : 충돌 범위 크기

오브젝트 물리와 충돌



Box Collider 2D



Circle Collider 2D

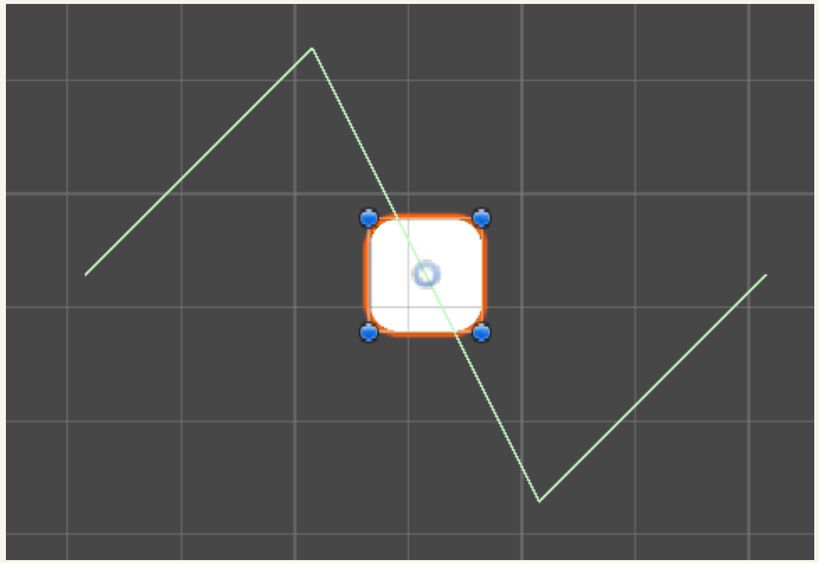
Box Collider 2D

- 사각형 범위의 충돌 범위
- Offset : 충돌 범위 중심점
- Size : 충돌 범위 크기

Circle Collider 2D

- 원 범위의 충돌 범위, 연산 속도가 가장 빠름
- Offset : 충돌 범위 중심점
- Radius : 충돌 범위 반지름 크기

오브젝트 물리와 충돌



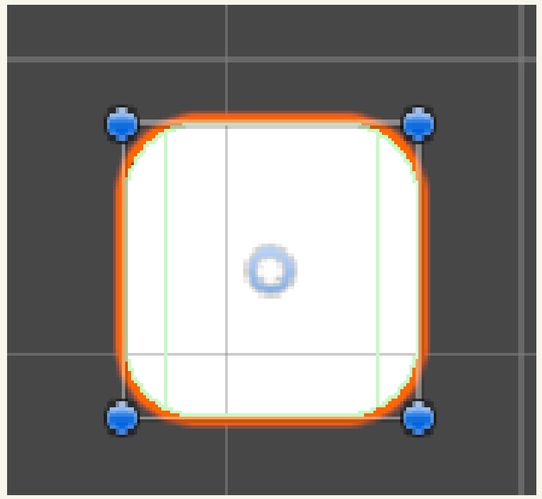
Edge Collider 2D

점 개수와 위치

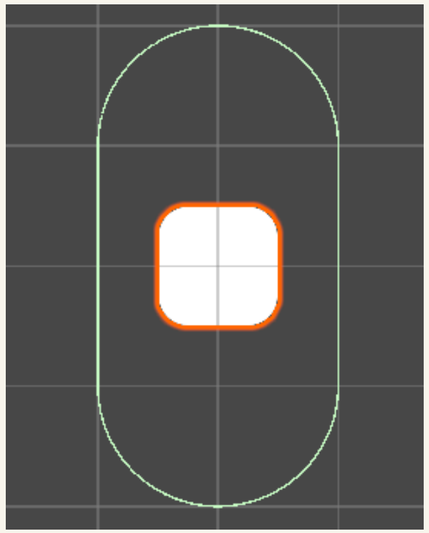
Edge Collider 2D

- 점의 개수, 점의 위치를 설정할 수 있어서 다양한 곡선 형태로 충돌 범위 표현 가능
- 주로 2D 게임의 바닥 충돌에 사용
- Offset : 충돌 범위 중심점
- Edge Radius : 충돌 선의 두께
- Points : 선을 이루는 점의 개수와 각 점의 위치

오브젝트 물리와 충돌



Polygon Collider 2D



Capsule Collider 2D

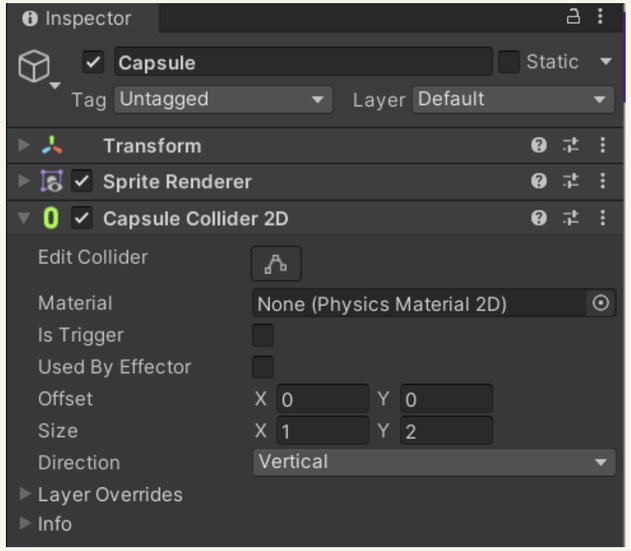
Polygon Collider 2D

- 텍스처의 모양과 비슷한 형태로 충돌 범위 생성
- Points로 수정 가능
- Offset : 충돌 범위 중심점
- Points : 선을 이루는 점의 개수와 각 점의 위치

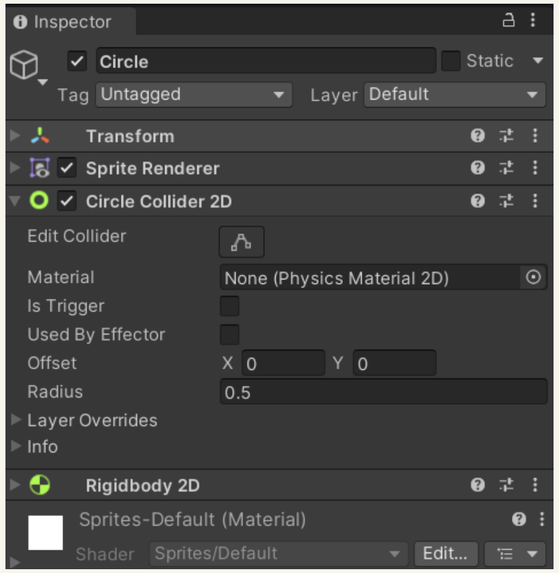
Capsule Collider 2D

- 캡슐 모양의 충돌 범위 생성
- 주로 사람 형태의 캐릭터에 사용됨
- Offset : 충돌 범위 중심점
- Size : 충돌 범위 크기
- Direction : 둥근 캡슐이 표현되는 방향
- Vertical : 위/아래, Horizontal : 좌/우

오브젝트 물리와 충돌



Cpasule 정보



Circle 정보

서로 다른 두 오브젝트가 충돌하기 위한 필수 조건

- 두 오브젝트 모두 Collider2D 컴포넌트를 가지고 있어야 함
- 둘 중 하나 이상의 오브젝트가 물리 처리를 담당하는 Rigidbody2D 컴포넌트를 가지고 있어야 함

오브젝트 물리와 충돌

```
private float moveSpeed = 5.0f;                    // 이동 속도
private Rigidbody2D rigid2D;

Unity 메시지 | 참조 0개
private void Awake()
{
    rigid2D = GetComponent<Rigidbody2D>();
}
Unity 메시지 | 참조 0개
private void Update()
{
    float x = Input.GetAxisRaw("Horizontal");      // 좌우 이동
    float y = Input.GetAxisRaw("Vertical");        // 위아래 이동

    rigid2D.velocity = new Vector3(x, y, 0) * moveSpeed;
}
```

스크립트

게임 오브젝트의 컴포넌트에 접근하는 방법

- GetComponent<컴포넌트 이름>();

- 컴포넌트와 동일한 타입의 변수 생성
- 컴포넌트 정보를 얻어와서 변수에 저장
- 컴포넌트 정보가 저장된 변수를 사용

현재 방법과 같이 클래스 변수를 생성하고 컴포넌트 정보를 한번 저장하면 현재 클래스 내부 어디서든 rigid2D 변수를 이용해 Rigidbody2D 컴포넌트 정보를 바꾸거나 얻을 수 있음

오브젝트 물리와 충돌

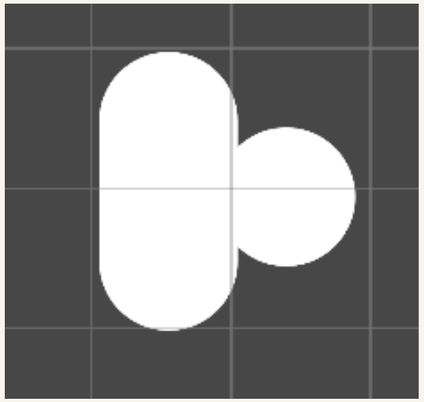
```
private float moveSpeed = 5.0f;                    // 이동 속도
private Vector3 moveDirection = Vector3.zero;      // 이동 방향

Unity 메시지 | 참조 0개
private void Update()
{
    float x = Input.GetAxisRaw("Horizontal");      // 좌우 이동
    float y = Input.GetAxisRaw("Vertical");        // 위아래 이동

    // 이동방향 설정
    moveDirection = new Vector3(x, y, 0);

    //새로운 위치 = 현재 위치 + 방향 * 속도
    transform.position += moveDirection * moveSpeed * Time.deltaTime;
}
```

transform 스크립트



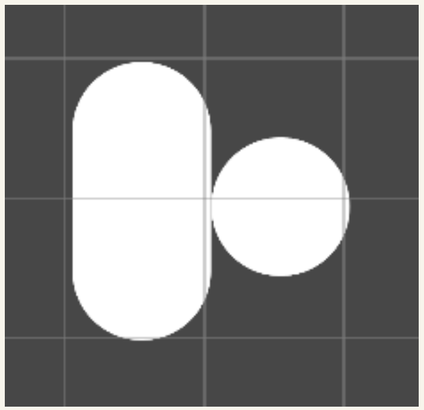
transform.position을 사용하게 되면
장애물에 막히지 않고 뚫을려고 함

```
private float moveSpeed = 5.0f;                    // 이동 속도
private Rigidbody2D rigid2D;

Unity 메시지 | 참조 0개
private void Awake()
{
    rigid2D = GetComponent<Rigidbody2D>();
}
Unity 메시지 | 참조 0개
private void Update()
{
    float x = Input.GetAxisRaw("Horizontal");      // 좌우 이동
    float y = Input.GetAxisRaw("Vertical");        // 위아래 이동

    rigid2D.velocity = new Vector3(x, y, 0) * moveSpeed;
}
```

Rigidbody2D 스크립트



Rigidbody2D를 사용하면 장애물에
막혀 그대로 멈출 수 있음

오브젝트 물리와 충돌

```
[SerializeField]
private Color color;
private SpriteRenderer spriteRenderer;

Unity 메시지 | 참조 0개
private void Awake()
{
    spriteRenderer = GetComponent<SpriteRenderer>();
}

Unity 메시지 | 참조 0개
private void OnCollisionEnter2D(Collision2D collision)
{
    spriteRenderer.color = color;
}

Unity 메시지 | 참조 0개
private void OnCollisionStay2D(Collision2D collision)
{
    Debug.Log("충돌중");
}

Unity 메시지 | 참조 0개
private void OnCollisionExit2D(Collision2D collision)
{
    spriteRenderer.color = Color.white;
}
```

충돌 이벤트 스크립트

[SerializeField]

- 해당 변수의 바로 윗줄에 작성
- 인스펙터창에서 변수의 옵션을 조절할 수 있게 해준다.

OnCollisionEnter2D() : 두 오브젝트가 충돌하는 순간 1회 호출

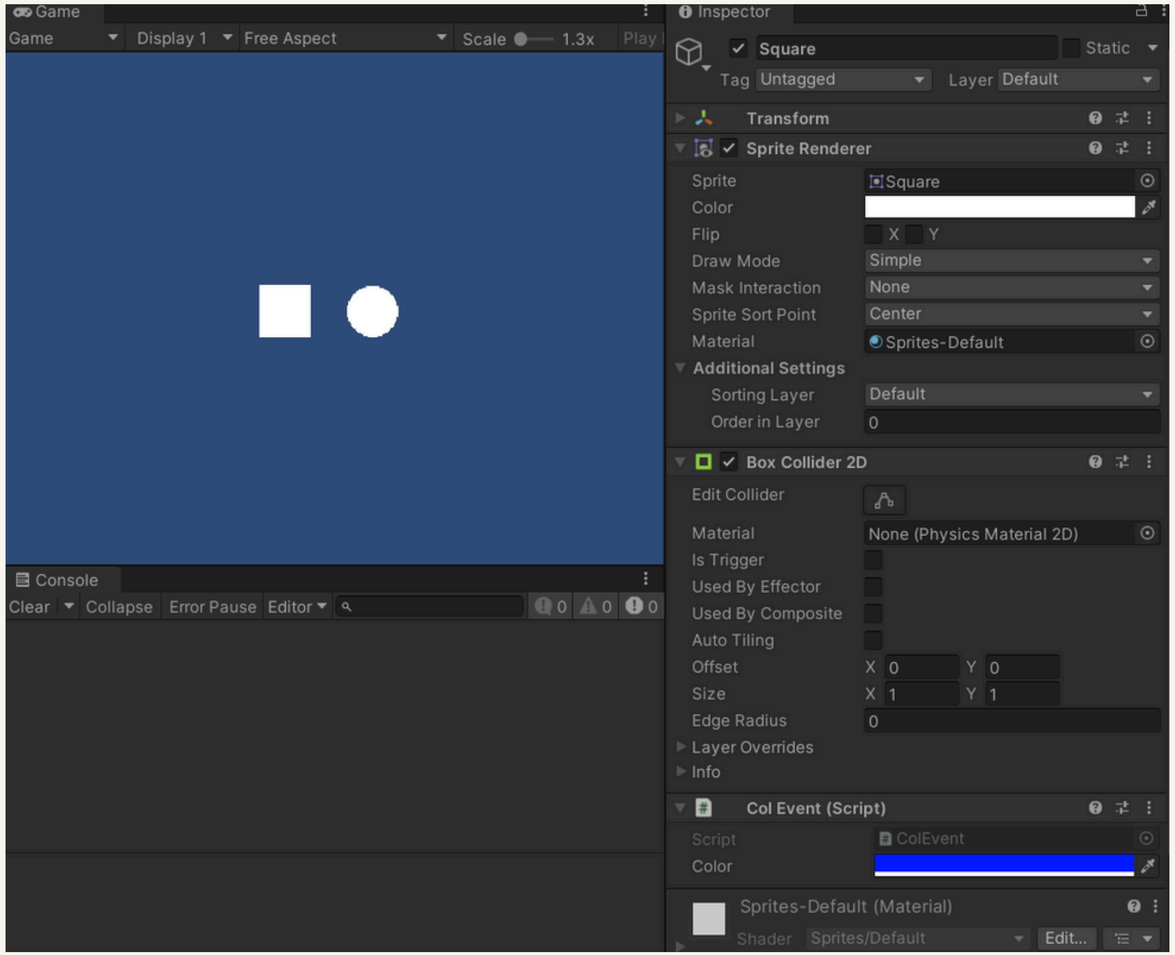
OnCollisionStay2D() : 충돌 직후 맞닿아 있는 동안 매 프레임 호출

OnCollisionEnter2D() : 두 오브젝트가 떨어져서 충돌이 종료되는 순간 1회 호출

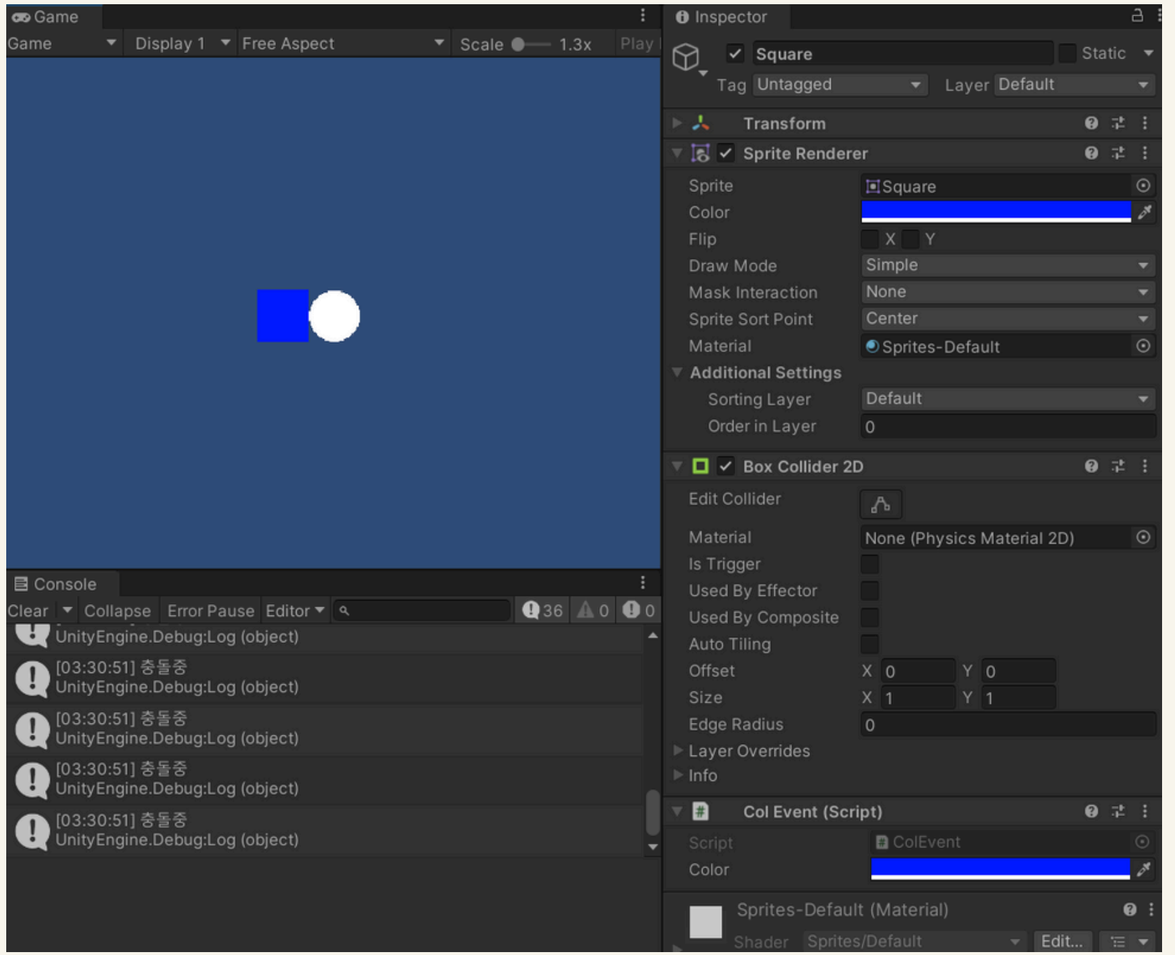
매개변수 collsion

- 현재 컴포넌트를 가지고 있는 오브젝트에 부딪힌 오브젝트 정보

오브젝트 물리와 충돌



충돌 전



충돌 후

*인스펙터창Color 변수의 기본 값은 rgba = 0000이므로 투명도값도 변경해줘야 함

오브젝트 물리와 충돌

```
[SerializeField]
private GameObject moveObject;
[SerializeField]
private Vector3 moveDirection;
private float moveSpeed;

Unity 메시지 | 참조 0개
private void Awake()
{
    moveSpeed = 5.0f;
}
Unity 메시지 | 참조 0개
private void OnTriggerEnter2D(Collider2D collision)
{
    moveObject.GetComponent<SpriteRenderer>().color = Color.black;
}
Unity 메시지 | 참조 0개
private void OnTriggerStay2D(Collider2D collision)
{
    moveObject.transform.position += moveDirection * moveSpeed * Time.deltaTime;
}
Unity 메시지 | 참조 0개
private void OnTriggerExit2D(Collider2D collision)
{
    moveObject.transform.position = new Vector3 (0, 6, 0);
}
```

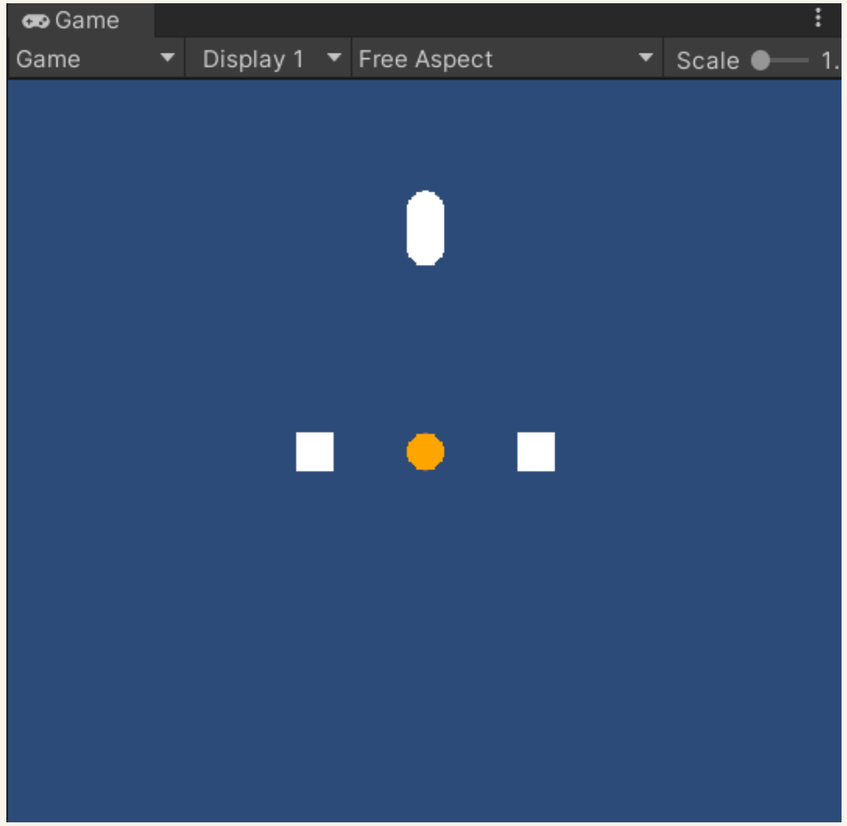
트리거 이벤트 스크립트

OnTriggerEnter2D() : 두 오브젝트가 충돌하는 순간 1회 호출

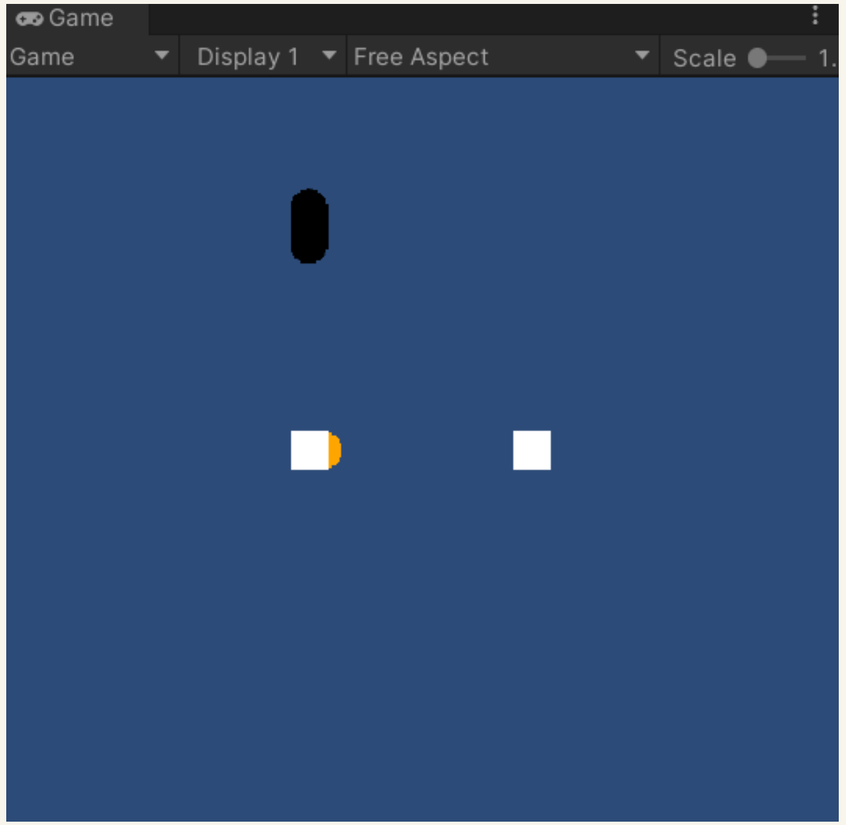
OnTriggerStay2D() : 충돌 직후 맞닿아 있는 동안 매 프레임 호출

OnTriggerEnter2D() : 두 오브젝트가 떨어져서 충돌이 종료되는 순간 1회 호출

오브젝트 물리와 충돌



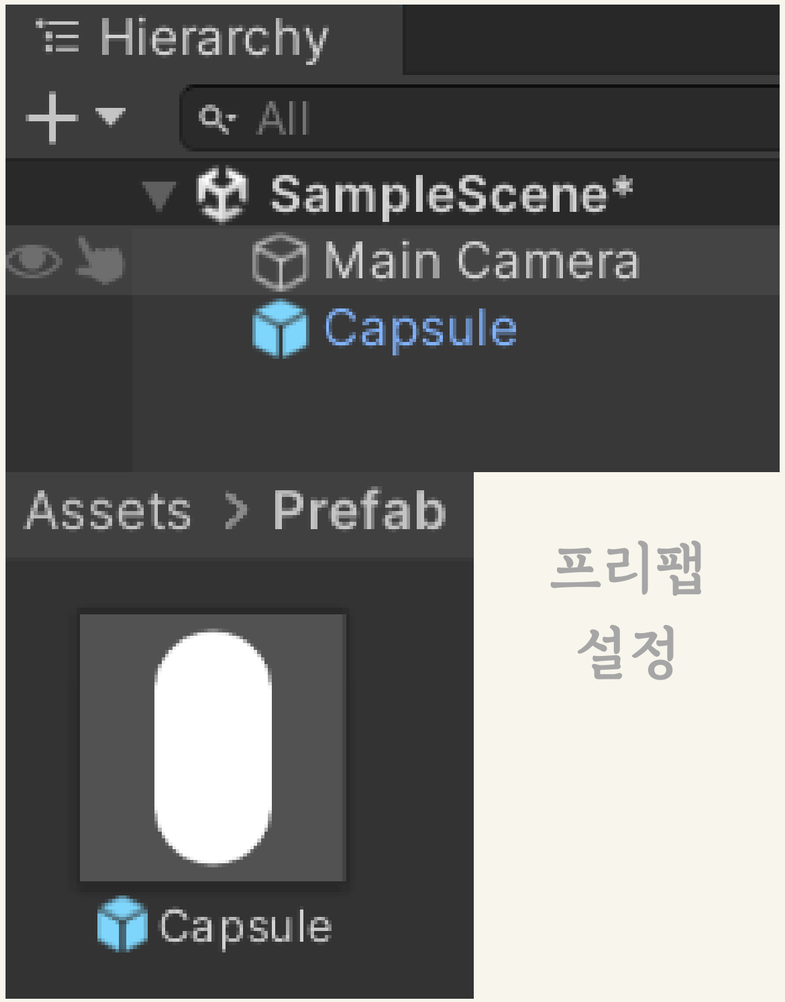
충돌 전



충돌 후

*IsTriger 체크가 되어있어야 함

오브젝트 생성



프리팹 설정

■Prefab

- 게임에 존재하는 게임오브젝트를 Project View에 파일 상태로 저장해 둔 것

프리팹 생성 방법

1. 원하는 오브젝트를 만든다.
2. Hierarchy View에 오브젝트를 Porject View로 드래그&드롭한다.
1. Hierarchy View에 있는 오브젝트를 삭제한다.

오브젝트 생성

```
[SerializeField]
private GameObject CapsulePrefab;

Unity 메시지 | 참조 0개
private void Awake()
{
    Instantiate(CapsulePrefab);
}
```

스크립트

■Instantiate()

- 게임오브젝트(프리팹)를 복제해서 생성
- 복제되는 오브젝트의 모든 컴포넌트 정보가 동일

```
[SerializeField]
private GameObject CapsulePrefab;

Unity 메시지 | 참조 0개
private void Awake()
{
    Instantiate(CapsulePrefab ,new Vector3(3, 3, 0), Quaternion.identity);
}
```

스크립트

- 두번째 매개변수로 위치를 position으로 설정할 수 있으며 세번째 매개변수로 회전을 rotation으로 설정 가능

오브젝트 생성

오일러 (Euler)

3차원의 3개 각도를 표현하기 위해 사용하는 3x3 크기의 행렬
회전 순서에 따라 결과가 달라지기 때문에 회전 순서에 주의해야 함
(유니티에서는 따로 계산할 일이 없기 때문에 걱정할 필요 없음)

장점 : 우리가 알고 있는 0~360의 각도를 표현할 수 있다
단점 : 지속적으로 회전을 하는 연산을 할 때 쿼터니온 보다
연산속도가 느리고, 짐벌락 현상이 발생할 수 있다

※ 짐벌락 : 세 개의 축이 서로 종속적인 관계를 가지고 있기 때문에 발생하는 문제로
회전 연산 도중 축이 하나 사라져 3차원의 오브젝트가 일그러지는 현상

쿼터니온 (Quaternion)

사원수로 3개의 벡터 요소와 하나의 스칼라 요소로 구성 (4개의 -1~1 사이의 값)

장점 : 연산속도가 빠르고, 짐벌락 현상이 발생하지 않는다
단점 : 우리가 알고 있는 0~360의 각도가 아니기 때문에 특정 각도를 표현하기 힘들다

유니티(Unity)

- transform.rotation : 게임오브젝트의 쿼터니온 회전 정보
- transform.localScale : 게임오브젝트의 오일러 회전 정보

※ Inspector View에 보이는 Transform의 rotation은 개발자의 편의를 위해 오일러로 표현!

```
Quaternion q = Quaternion.Euler(0, 0, 0);
```

오일러 회전 정보를 입력해서 쿼터니온 회전 값으로 변경

```
transform.Rotate(new Vector3(1, 0, 0));
```

"x축으로 빙글빙글 돌아라"와 같이 지속적인 회전 함수

회전

오브젝트 생성

```
[SerializeField]
private GameObject CapsulePrefab;

Unity 메시지 | 참조 0개
private void Awake()
{
    Quaternion rotation = Quaternion.Euler(0, 0, 45);

    GameObject clone = Instantiate(CapsulePrefab, Vector3.zero, rotation);

    clone.name = "Capsule001";

    clone.GetComponent<SpriteRenderer>().color = Color.black;

    clone.transform.position = new Vector3(2, 1, 0);

    clone.transform.localScale = new Vector3(3, 2, 1);
}
```

프리팹 스크립트

- GameObject clone에 생성된 복제 오브젝트의 정보를 저장
- clone과 생성된 캡슐 오브젝트는 동일
- clone의 값을 변경하여 이름, 색, 위치 등을 바꿀 수 있음

Thanks