3D콘텐츠 이론 및 활용

9주(2). 충돌처리 및 오브젝트 함수

- 오브젝트 생성 함수 및 소멸 함수
- 컴포넌트의 스크립트 처리
- 랜덤 함수
- 타임 클래스
- 충돌처리

학습개요

학습목표

- 스크립트로 오브젝트의 (무한)생성과 소멸처리를 할 수 있다.
- 컴포넌트 속성값을 스크립트로 처리할 수 있다.
- 필수 함수(랜덤, 타임, 충돌)를 이해하고 게임에 적용해 볼 수 있다.

학습내용

- 오브젝트 생성 함수 및 소멸 함수
- 컴포넌트의 스크립트 처리
- 랜덤 함수
- 타임 클래스
- 충돌처리

Unity 스크립팅

Random.Range()

- 사용자가 설정한 두 수의 범위내에서 무작위로 수를 리턴
- 예)
  10 ~ 20 미만의 정수 값을 리턴
  10 ~ 20 미만의 실수 값을 리턴

```
public class random : MonoBehaviour {
    int randomInt;
    float randomFloat;

    void Update () {
        randomInt = Random.Range (10, 20);
        randomFloat = Random.Range (10.0f, 20.0f);
        print (randomInt);
        print (randomFloat);
    }
}
```

Unity 스크립팅

Time.deltaTime - 타임클래스

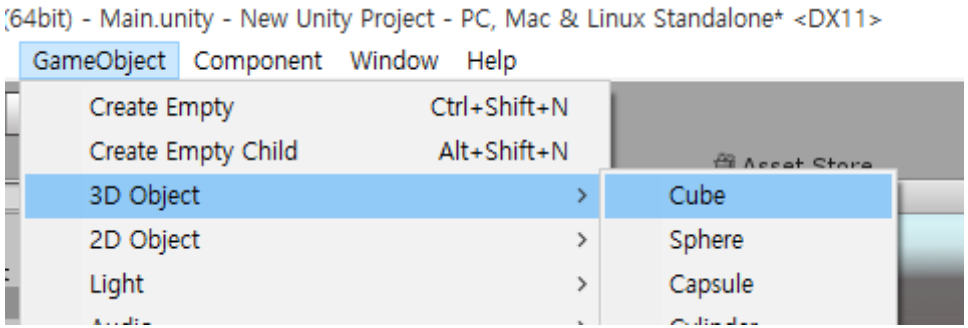
- 마지막 프레임을 계산하는데 걸린 시간, float 형으로 지정
- 시간을 활용하여 이벤트에 활용하는 경우에 사용
- 예)
  - 시간이 되면 정해진 함수나 명령을 처리(게임오버, 이동, 폭발, 소멸 등)
  - 일정시간 후에 총알이 발사되는 스크립트 작성
  - 게임의 남은 시간을 카운트

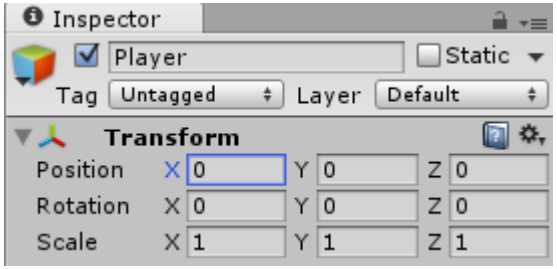
```
float setTime = 3.0f;
float Timer=0.0f;

void Update () {
    if (Timer >= setTime) {
        Timer = 0.0f;
        print ("시간이 초과되었습니다");
    } else {
        Timer += Time.deltaTime;
    }
}
```

1. GameObject 생성

1) 큐브 생성 및 위치 확인

- Cube 게임 오브젝트 생성
- 생성된 cube 계층 뷰에서 확인
- Cube 이름을 Player 로 변경
- Player 의 위치 값 (0,0,0) 확인

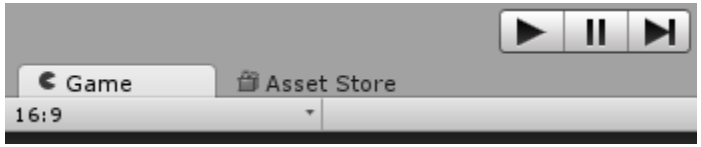




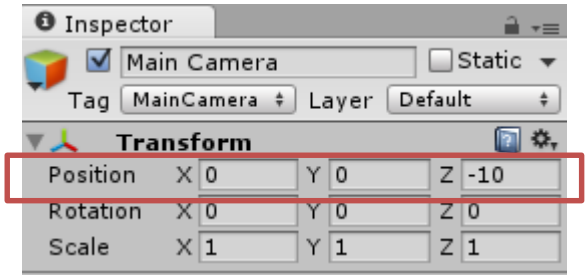
1. GameObject 생성

2) 화면 구성 조정

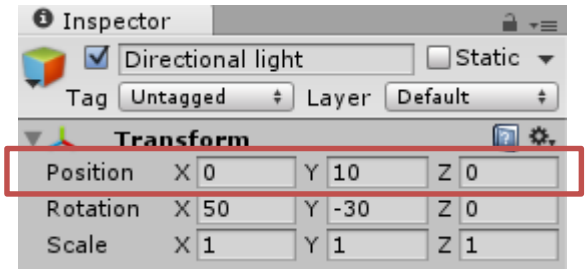
- Play 버튼으로 실행 및 중단
- 화면 비율 (16:9)로 조정



- Main Camera의 위치 조절



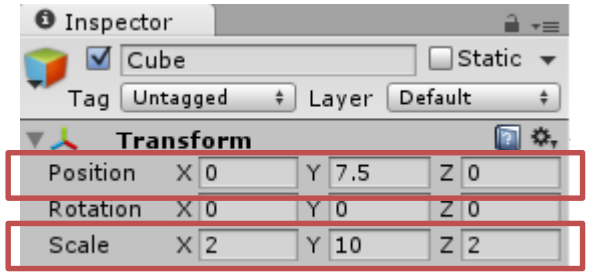
- Directional light 광원 추가 및 위치조정



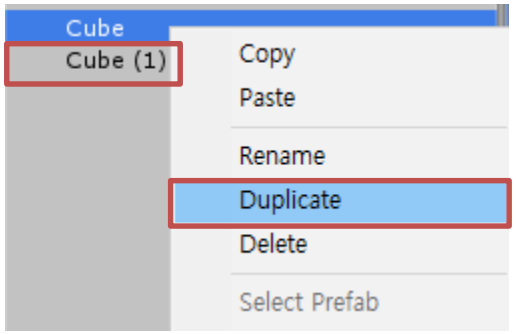
3. 게임 오브젝트 관리

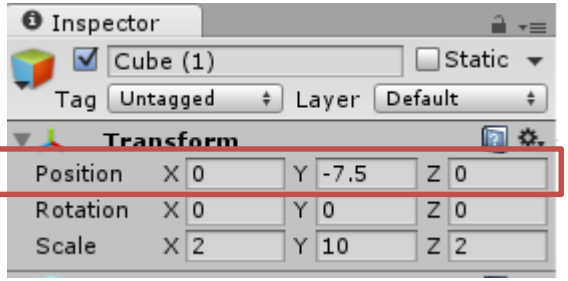
1) 벽 만들기

- 새로운 큐브 생성 및 위치, 스케일값 변경



- 큐브 복제
  - 마우스 우클릭 후 Duplicate
  - 복제된 Cube (1) 확인
  - 위치 값 수정



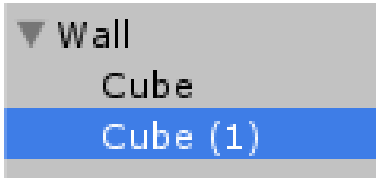


3. 게임 오브젝트 관리

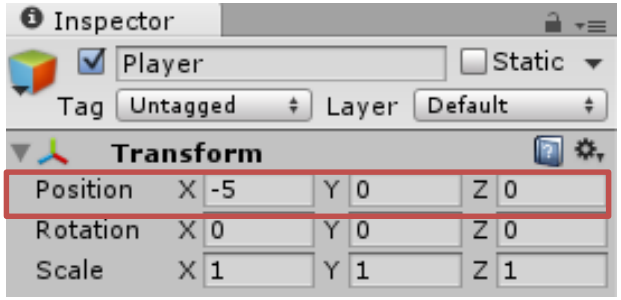
3) 큐브 계층 관리

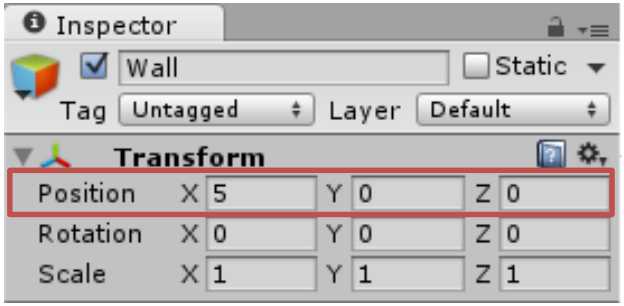
- 여러 개의 관련 오브젝트를 계층형태로 구성하고 관리

- 빈 오브젝트를 생성하고 이름을 Wall 로 변경
- 2개 Cube 객체를 Wall 로 드래그하여 계층화



4) Player, Wall 위치 재조정

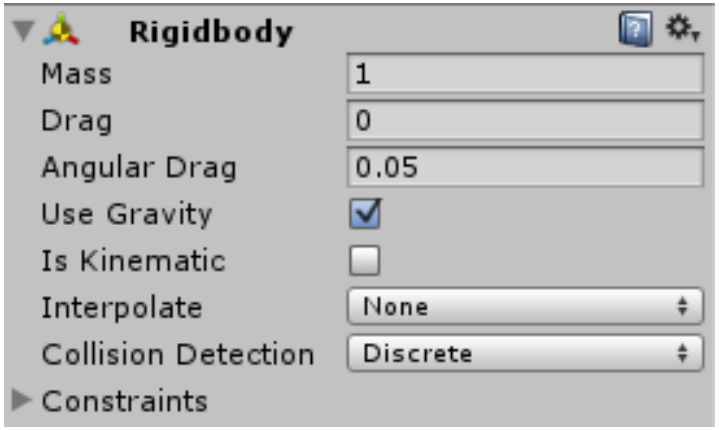




4. 플레이어에 중력 추가

1) Rigidbody 컴포넌트 추가

- Component > physics > Rigidbody 또는 객체의 Add component 버튼



5. 씬 저장

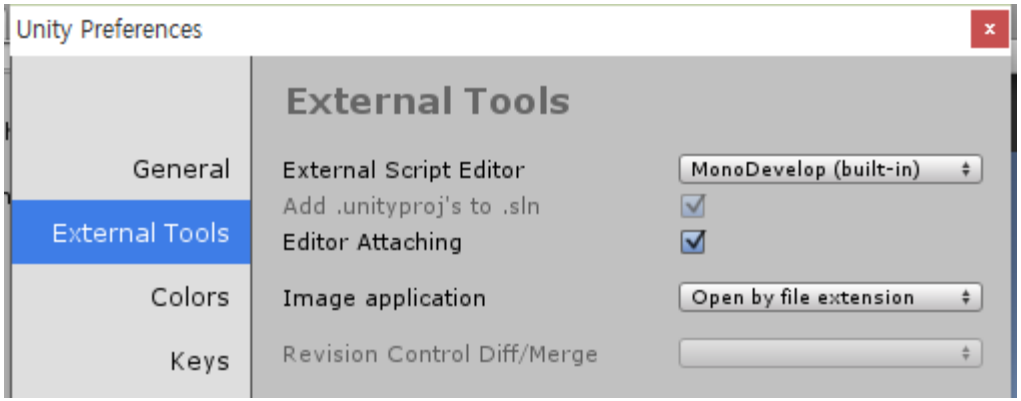
- File > Save Scene을 선택하고 Assets 폴드에 Main 이름으로 저장
- 파일은 Main.unity 형식으로 저장됨

  - 씬에는 모든 객체를 포함하고 있지 않으며 객체들의 배치 및 구조 정보만 가지고 있음.

6. 스크립트로 오브젝트 조작하기

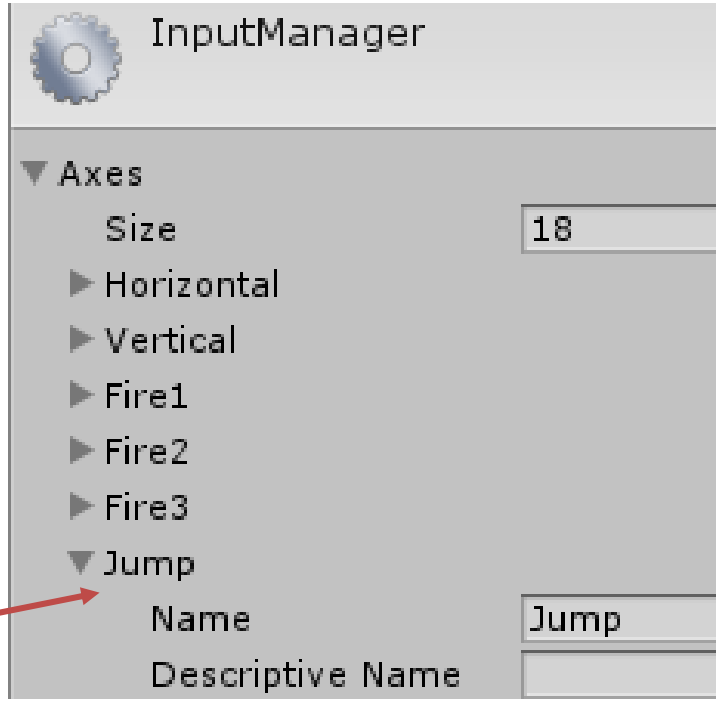
1) 스크립트 생성

- Assets > create > c# script 선택 또는 에셋창에서 우클릭
  - 생성된 파일 이름은 Player 수정한다. (나중에 이름을 변경을 하면 문제가 발생할 수 있다)
  - 파일을 더블 클릭하면 개발 툴이 실행됨. (비주얼스튜디오 또는 모노디벨롭)



6. 스크립트로 오브젝트 조작하기

2) 점프키를 입력할 때 처리할 스크립트 작성



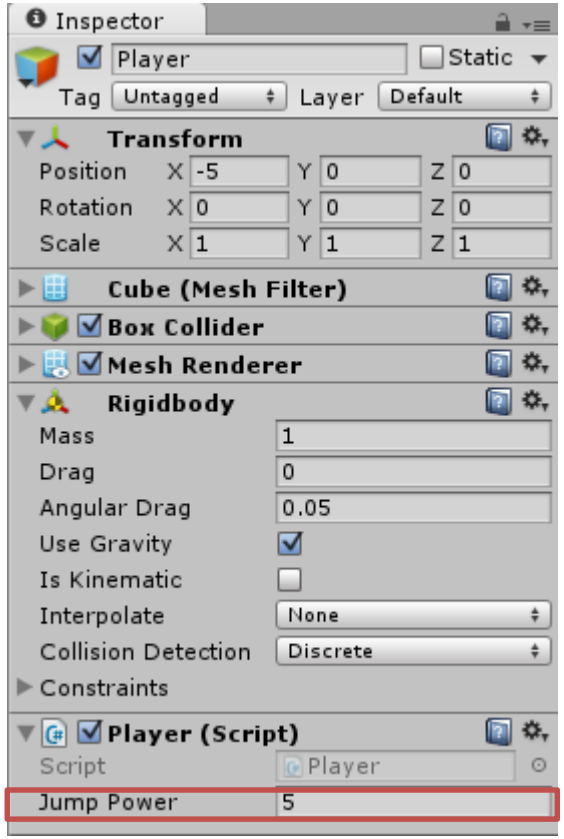

```
using UnityEngine;
using System.Collections;

public class Player : MonoBehaviour {



    // Update is called once per frame
    void Update () {
        if (Input.GetButtonDown ("Jump")) {
            GetComponent<Rigidbody> ().velocity = new Vector3(0, 5, 0);

        } // 컴포넌트의 속성 값을 스크립트에서 변경
    }
}
```

- 리지드바디의 Y축 중력 값을 public 변수로 선언하고 사용하면 편리하다

6. 스크립트로 오브젝트 조작하기

3) 스크립트 Save 및 Player에게 적용하기

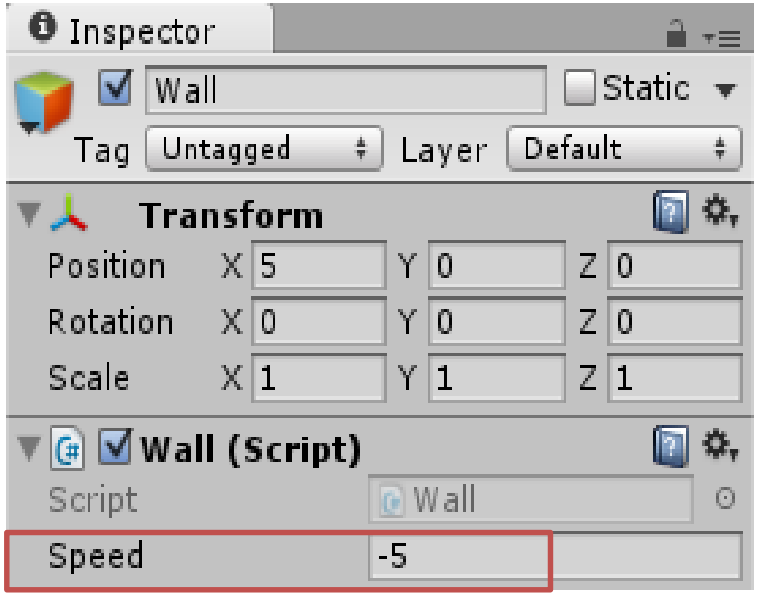
- File > save
- 저장된 파일을 드래그 하여 Player 객체에 놓으면 연결됨
- Inspector에서 Jump Power값을 5로 변경
- 게임 실행 후 스페이스 키를 눌러 확인



6. 스크립트로 오브젝트 조작하기

4) 벽을 왼쪽으로 이동 시키는 스크립트

- Wall.cs 스크립트를 새로 생성
- 저장 후 Wall 객체에 스크립트 연결
- Speed 값을 -5로 수정 후 게임실행




```
using UnityEngine;
using System.Collections;

public class Wall : MonoBehaviour {
    public float speed;

    void Update () {
        this.transform.Translate (speed * Time.deltaTime, 0, 0);
    }
}
```

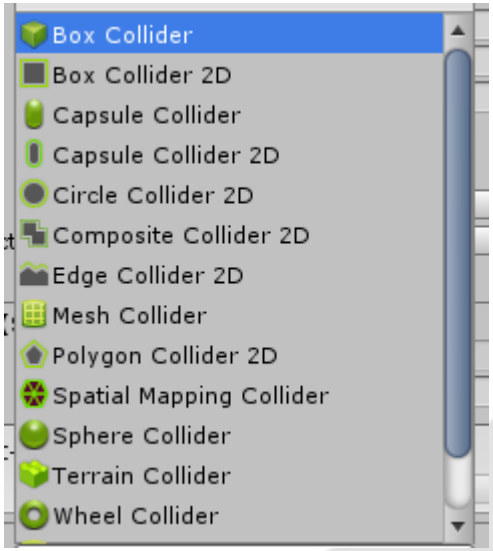
6. 스크립트로 오브젝트 조작하기

5) 충돌처리 스크립트

- 플레이어와 벽이 충돌하면 게임오버 처리
- Player.cs 파일의 끝 부분에 충돌처리 코드를 추가하고 실행

```
using UnityEngine.SceneManagement; // 추가해야 함

// 충돌이 일어나면 자동 실행되는 함수
    void OnCollisionEnter (Collision other)
    {
        SceneManager.LoadScene("Main");
    }
```

- Collider는 객체와 가장 유사한 형태의 콜라이더를 사용해야 효과(처리시간, 자원활용성)가 좋음
- 충돌이 되었는지 감지하기 위해서는 Rigidbody가 추가로 사용되어야 함.



6. 스크립트로 오브젝트 조작하기

Collision 함수 3가지

- Collision은 물리적인 연산을 하며 충돌을 감지
- Rigidbody의 [is Kinematic] 속성이 꺼져 있어야 작동됨
- 두 객체가 접촉 했을 때 서로 튕겨지는 효과가 일어 남
- 함수의 파라메터로 Collision 객체가 들어오며 “any”을 이용해 충돌한 GameObject에 대한 처리를 할 수 있음 (예, tag 처리)

```
- void OnCollisionEnter(Collision any) { }
- void OnCollisionStay(Collision any) { }
- void OnCollisionExit(Collision any) { }
```

- Enter - 충돌이 시작되는 순간 호출
- Stay - 충돌이 되고있을 때 매 프레임마다 호출
- Exit - 충돌이 끝날 때 호출

참고: [is Kinematic] '외부의 힘이 영향을 미치지 않는 오브젝트' 라고 체크하면 이 옵션을 가진 오브젝트에 대해서는 물리엔진이 해당 오브젝트에게 가해지는 힘의 크기와 방향 등을 계산하지 않게 되어 시스템의 부하를 줄일 수 있음

6. 스크립트로 오브젝트 조작하기

충돌처리와 트리거 처리의 차이

Collision처리 예

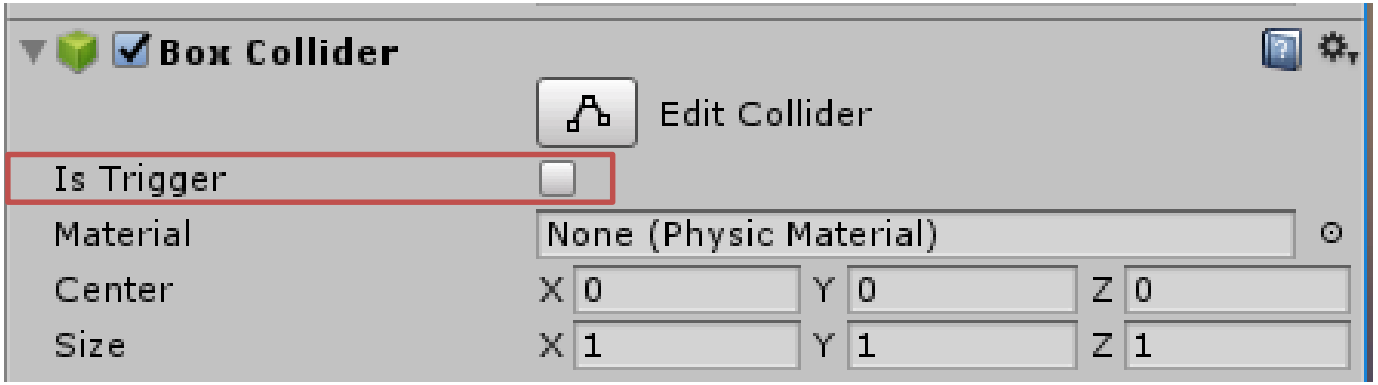
Trigger 처리 예

6. 스크립트로 오브젝트 조작하기

Trigger 함수의 종류

- GameObject간의 물리적 연산을 하지 않고 충돌을 감지
- 두 객체가 접촉 했을때 서로 튕겨 나가지않고 그냥 통과하게 됨.
- Trigger를 쓰기 위해서는 해당 Collider의 [Is Trigger] 항목을 체크해야 함

- void OnTriggerEnter(Collider any) { }
- void OnTriggerStay(Collider any) { }
- void OnTriggerExit(Collider any) { }

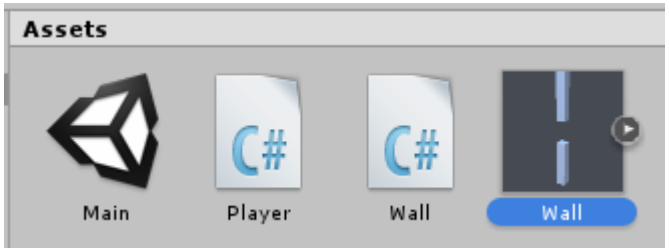


7. 게임 오브젝트 연속 생성

프리팹 기능과 스크립트를 이용하여 자동으로 벽을 생성하고 삭제

1) 벽 프리팹 만들기

- Prefab : 오브젝트를 파일로 저장하고 필요할 때 마다 불러와서 재 사용 가능하도록 저장한 게임오브젝트
- Wall 을 드래그 하여 에셋 폴드에 놓으면 프리팹이 생성됨



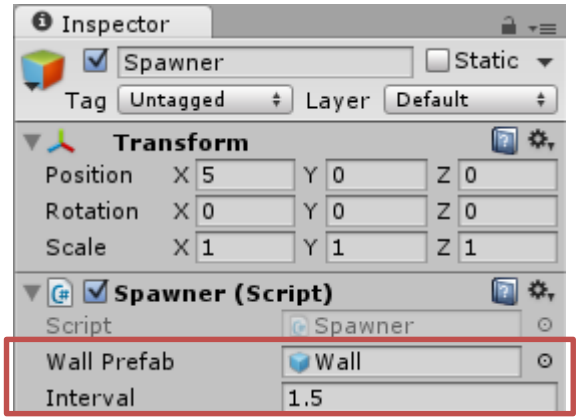
2) 스포너 만들기 (Spawner)

- 빈 오브젝트를 만들어 이름을 Spawner으로 변경
- 게임에서 스포너는 캐릭터나 아이템이 나온다/내보낸다의 의미의 용어

7. 게임 오브젝트 연속 생성

3) 스포너 스크립트 작성

- Spawner.cs 작성 후 Spawner 객체에 연결
- Inspector 설정




```
    public GameObject wallPrefab;
    public float interval=3.0f;
    float Timer=0.0f;

    void Update () {
        if (Timer >= interval) {
            Timer = 0.0f;
     Instantiate (wallPrefab, transform.position, transform.ro
tation); // wallPrefab 을 위치와 회전을 적용하여 클론
        } else {
            Timer += Time.deltaTime;
        }

    }
```

7. 게임 오브젝트 생성 삭제

4) 게임 오브젝트의 생성 위치를 랜덤처리

- Spawner.cs 수정

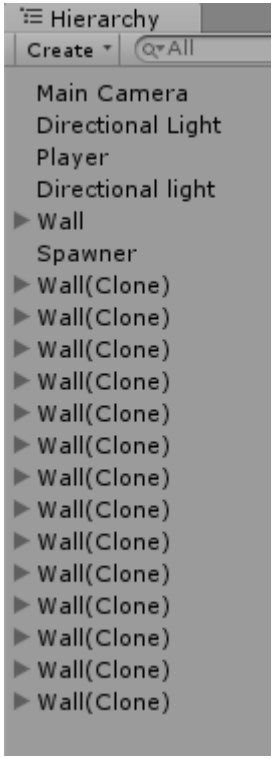
```
    public GameObject wallPrefab;
    public float interval = 3.0f;
    float Timer = 0.0f;
public float range = 3.0f;
    void Update ()
    {
        if (Timer >= interval) {
            Timer = 0.0f;
transform.position = new Vector3 (transform.position.x,
Random.Range (-range, range), transform.position.z);

            Instantiate (wallPrefab, transform.position, transform.rota
tion);
        } else {
            Timer += Time.deltaTime;
        }
    }
```

8. 게임 오브젝트 삭제

1) 불 필요한 게임 오브젝트 삭제

- 벽이 계속해서 생성되면 컴퓨터 자원을 소모하게 되므로 일정 시간이 지나면 게임 오브젝트를 제거하도록 해야 함.
  - Wall.cs 스크립트의 Start() 함수내에 소멸자 추가

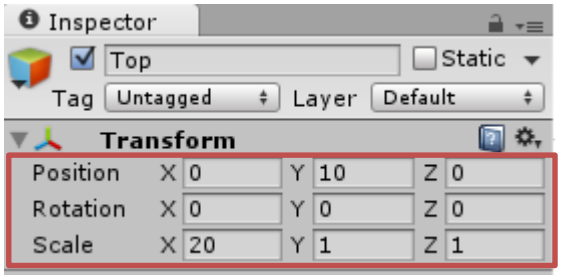
```
void Start () {
    Destroy (gameObject, 10f);
     // 생성 후 10초 후 소멸되도록 처리.
}
```

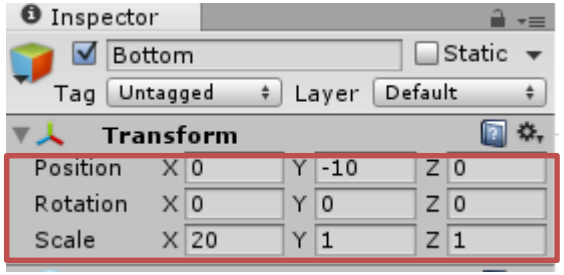


9. 게임 완성하고 공개하기

1) 화면에 상, 하단에 벽 추가

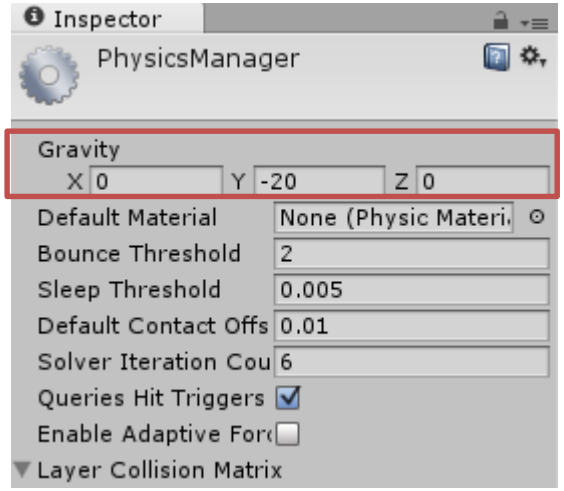
- 새로운 cube 2개를 생성하고 이름은 Top, Bottom 으로 지정
- 각각의 큐브 설정 값 변경

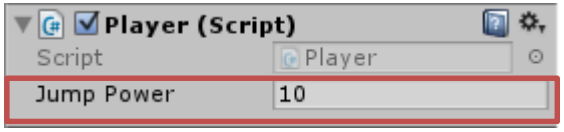




2) 게임 중력 변경하기

- 중력을 조절해서 플레이 해 보자
  - Edit > Project settings > physics에서 중력값을 -20으로 더 낮추고,
  - Player 의 Jump Power는 10으로 올려서 난이도 조절

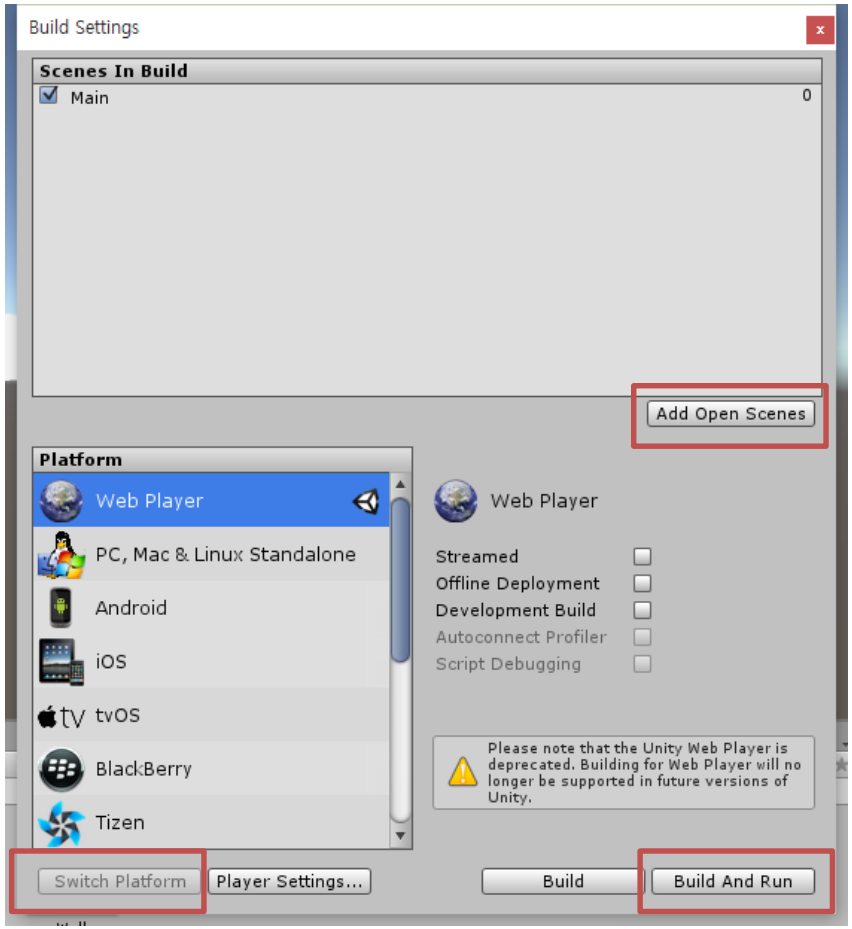




8. 게임 완성하고 공개하기

3) 웹용으로 빌드하고 실행

- 빌드 할 씬을 추가 함
- 플랫폼은 Web Player로 지정 후 Switch Platform으로 확인



학습정리

1. Prefab을 포함한 모든 객체와 컴포넌트의 기능은 스크립트를 통해 처리할 수 있다.

2. Collision 처리는 물리적인 충돌처리가 가능하고 Trigger는 감지의 역할을 수행한다.

3. 게임 중 생성된 게임오브젝트가 무한 생성되면 컴퓨터 자원을 소모하게 되므로 화면을 벗어났거나 일정 시간이 지나면 게임 오브젝트를 소멸되도록 처리해 주어야 한다.