




가상현실 및 실습



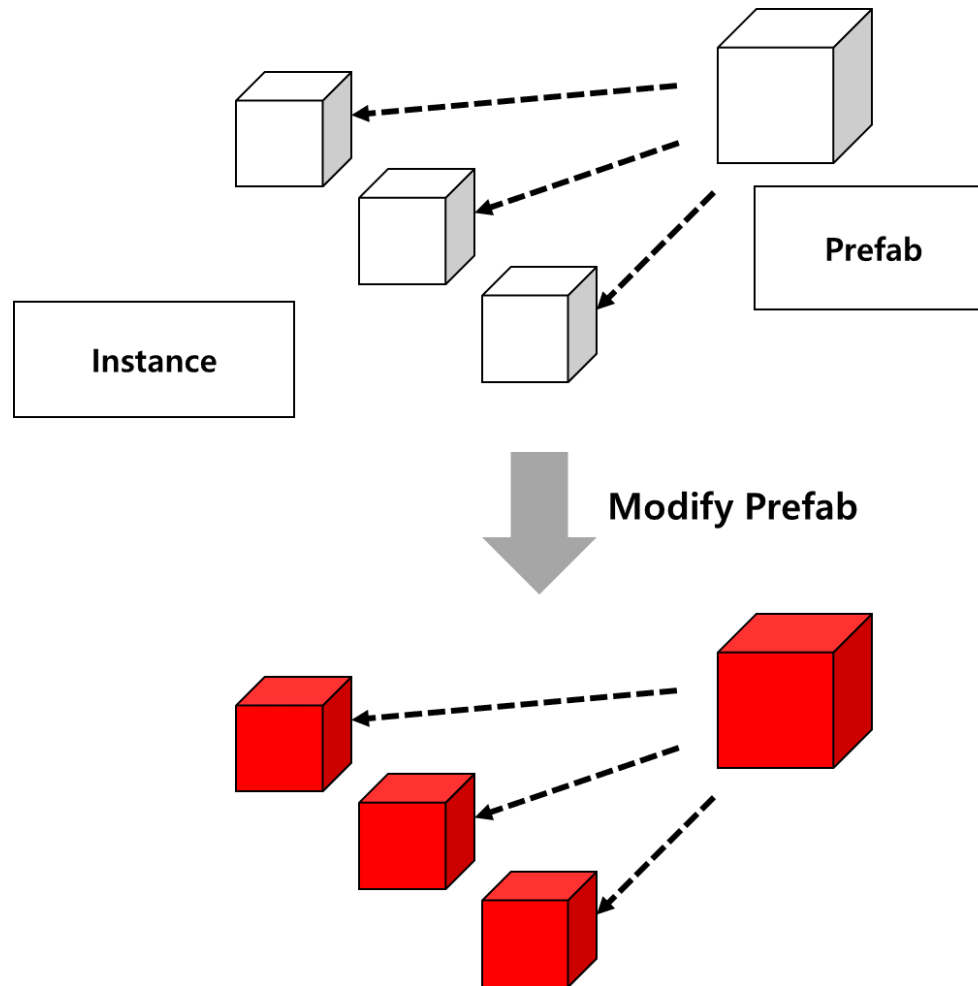
패키지를 통한 프리팹 관리
및 스크립트 개념



패키지를 통한 프리팹 관리

1. 패키지를 통한 프리팹 관리

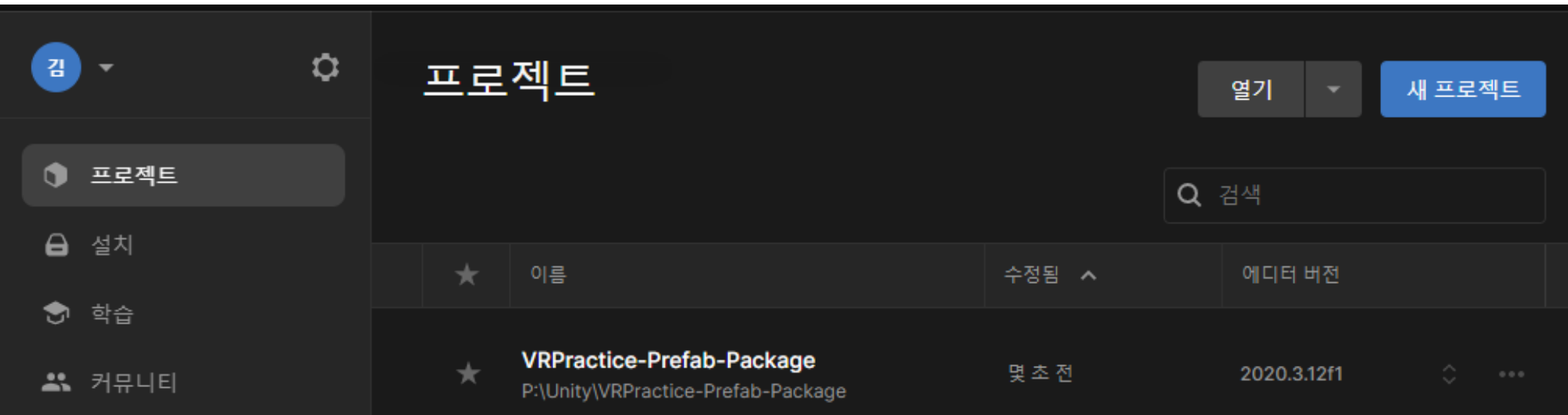
프리팹(Prefab): 재사용 가능한 게임 오브젝트 에셋을 의미



1. 패키지를 통한 프리팹 관리

1) 준비사항

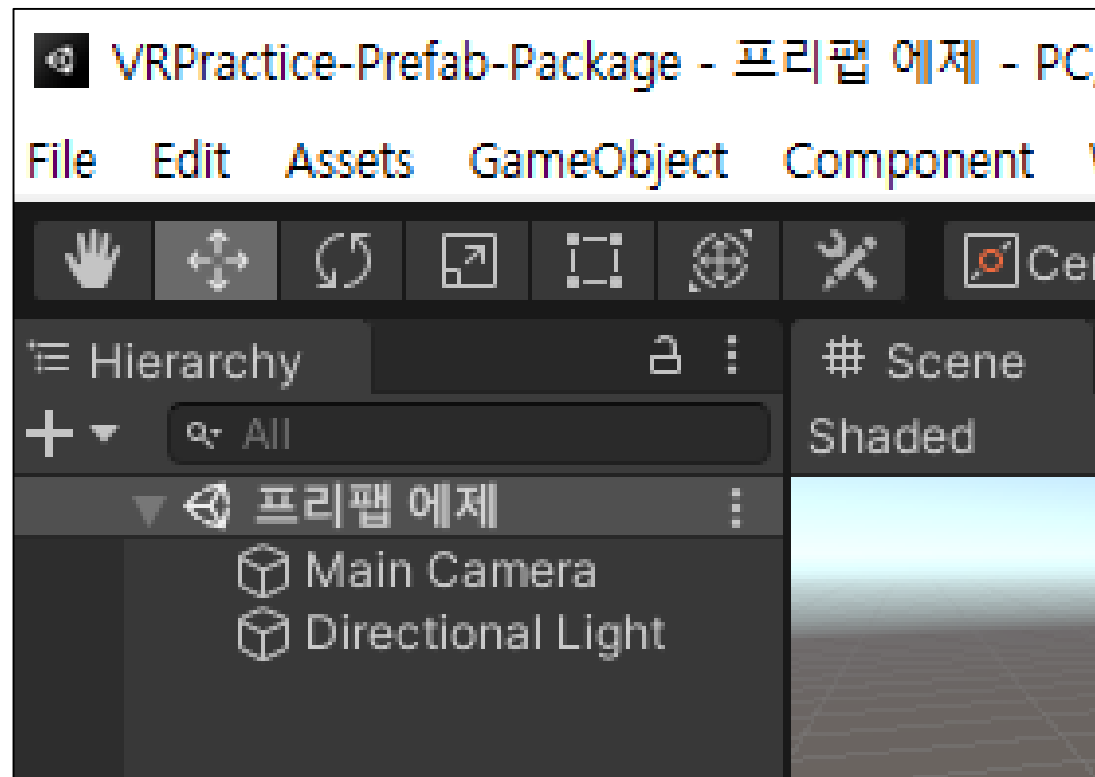
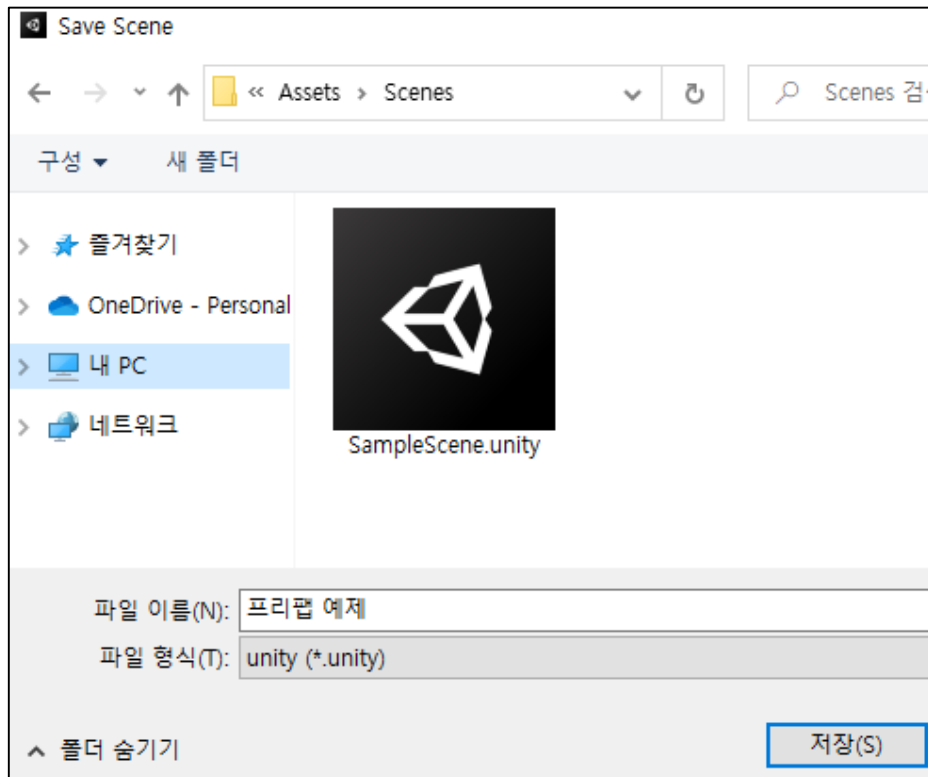
- 새로운 프로젝트 **VRPractice-Prefab-Package**를 생성



1. 패키지를 통한 프리팹 관리

1) 준비사항

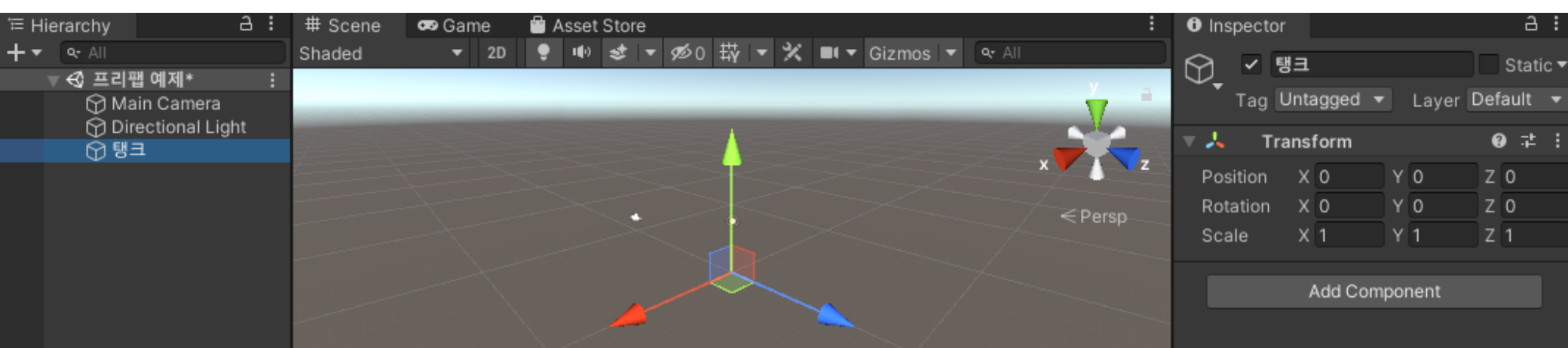
- 새로운 메뉴 > File > New Scene을 선택하여,
새로운 씬 **프리팹 예제**를 생성



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

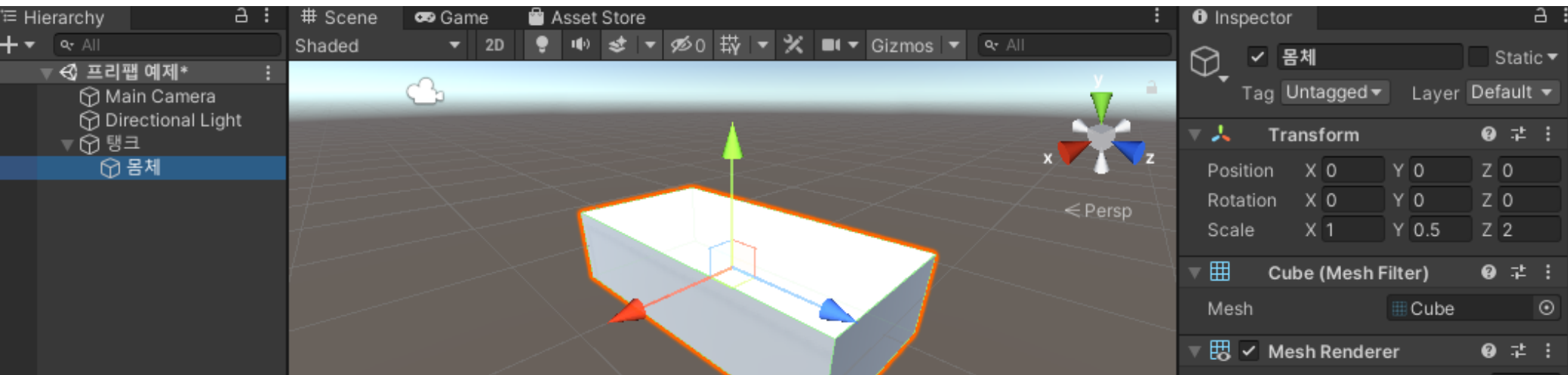
- 빈 게임 오브젝트를 생성하고 이름을 **탱크**로 설정
- **탱크** 게임 오브젝트의 **Transform** 컴포넌트 속성을 다음과 같이 설정
 - 위치: 0, 0, 0
 - 회전: 0, 0, 0
 - 스케일: 1, 1, 1



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

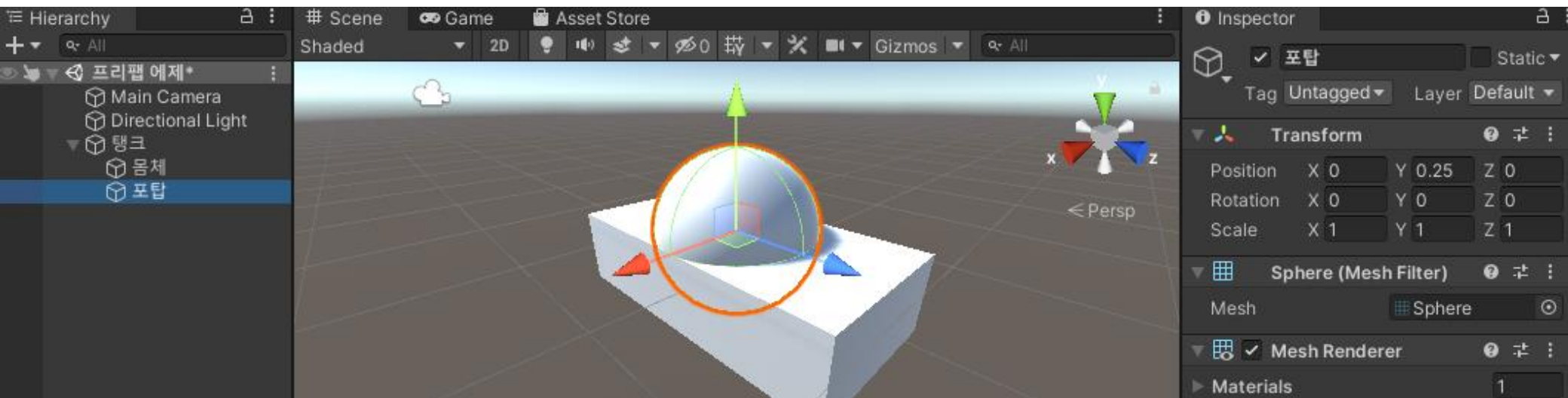
- 프리미티브 게임 오브젝트인 Cube 게임 오브젝트를 생성
- 이름을 **몸체**로 변경 후 다음과 같이 설정
 - 부모 게임 오브젝트: 탱크
 - Transform - 위치: 0, 0, 0
 - Transform - 회전: 0, 0, 0
 - Transform - 스케일: 1, 0.5, 2



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

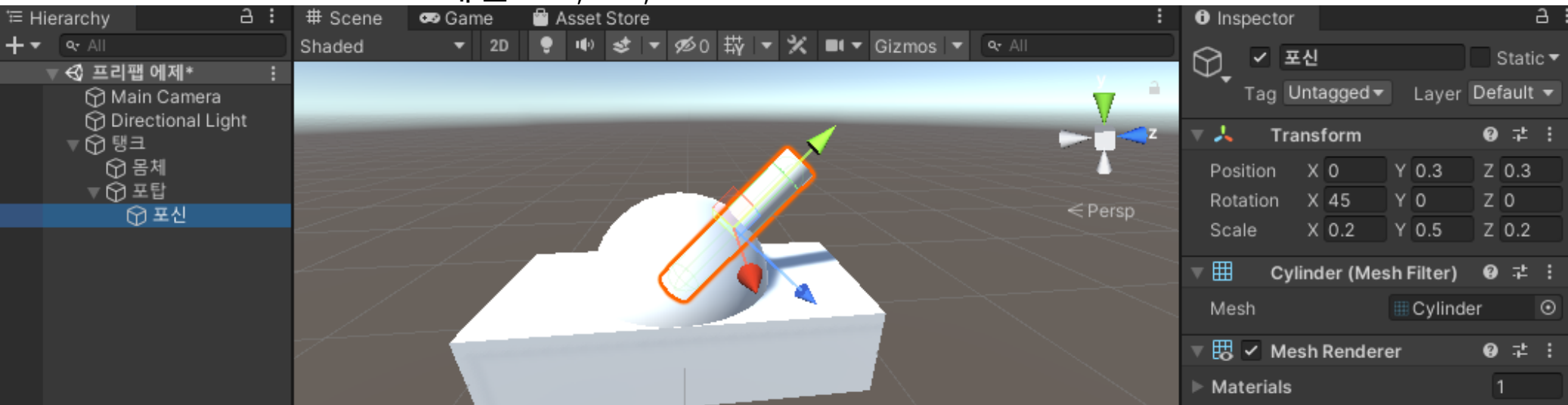
- 프리미티브 게임 오브젝트인 Sphere 게임 오브젝트를 생성
- 이름을 **포탑**으로 변경 후 다음과 같이 설정
 - 부모 게임 오브젝트: 탱크
 - Transform - 위치: 0, 0.25, 0
 - Transform - 회전: 0, 0, 0
 - Transform - 스케일: 1, 1, 1



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

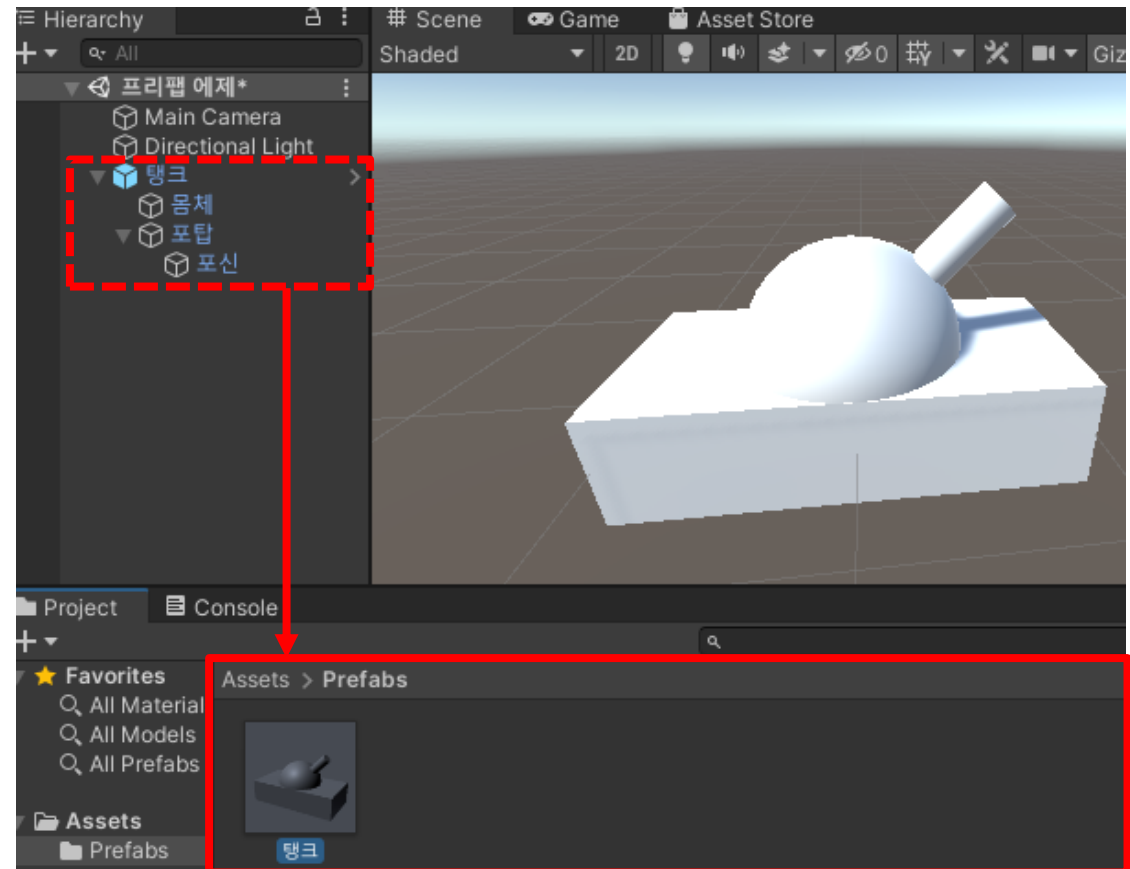
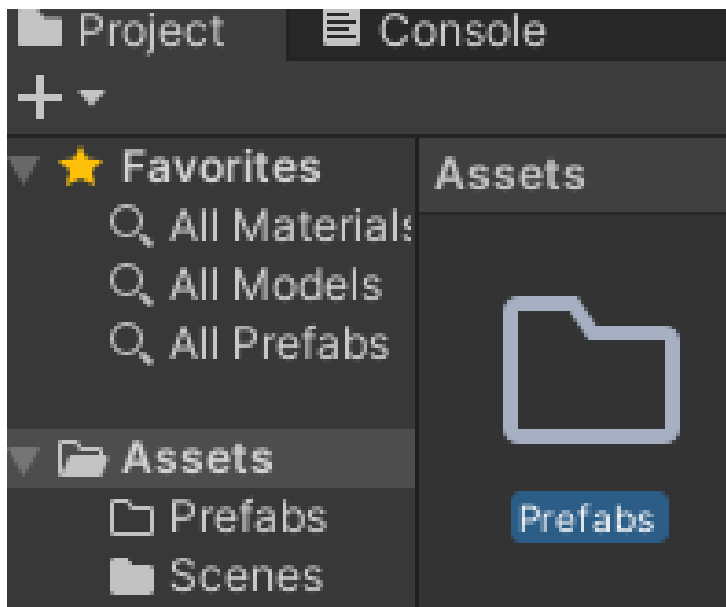
- 프리미티브 게임 오브젝트인 Cylinder 게임 오브젝트를 생성
- 이름을 포신으로 변경 후 다음과 같이 설정
 - 부모 게임 오브젝트: 포탑
 - Transform - 위치: 0, 0.3, 0.3
 - Transform - 회전: 45, 0, 0
 - Transform - 스케일: 0.2, 0.5, 0.2



1. 패키지를 통한 프리팹 관리

3) 프리팹 생성 방법

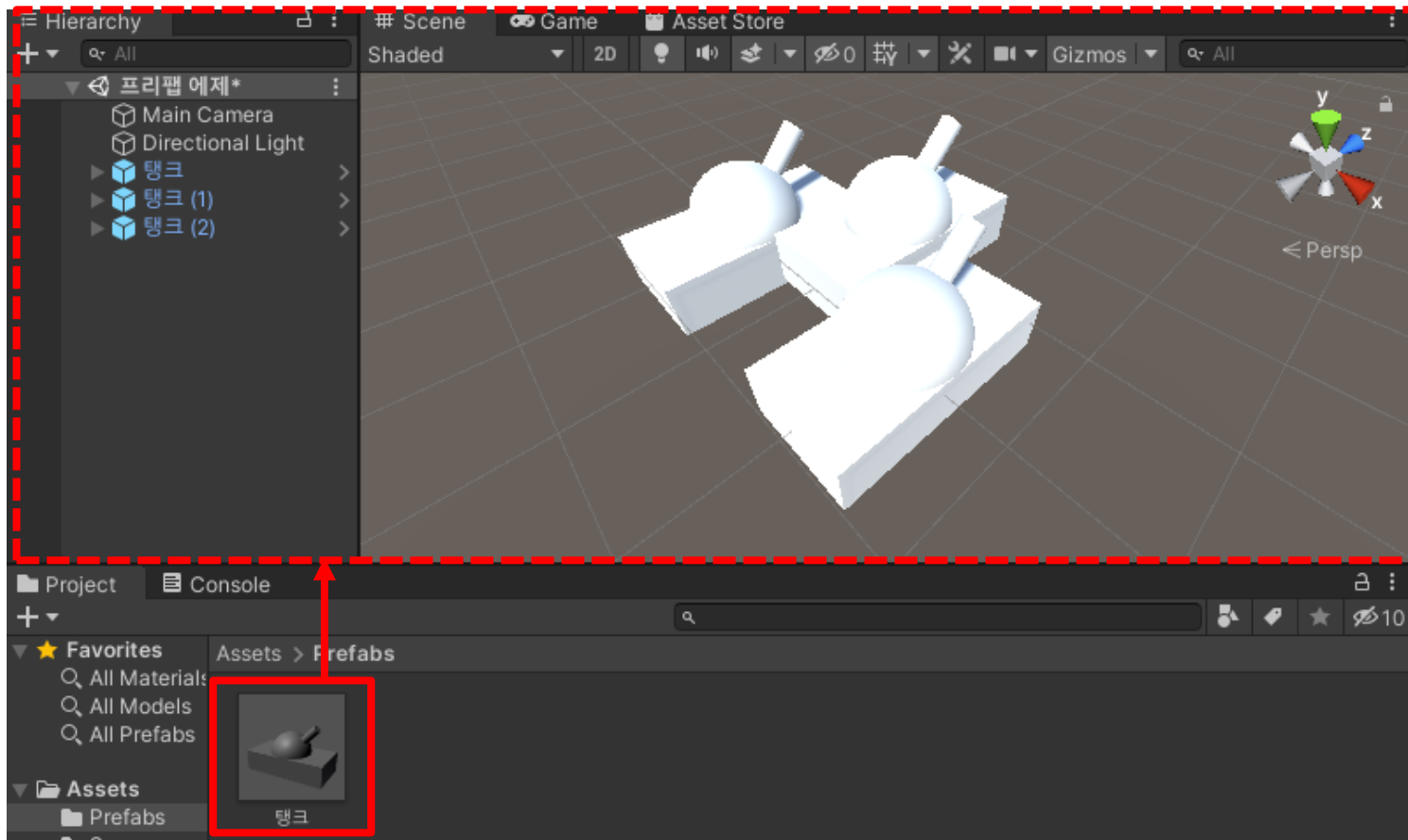
- 새 폴더 **Prefabs** 생성
- **탱크** 게임 오브젝트를
프로젝트 창으로 드래그 앤 드롭



1. 패키지를 통한 프리팹 관리

4) 프리팹 사용 방법

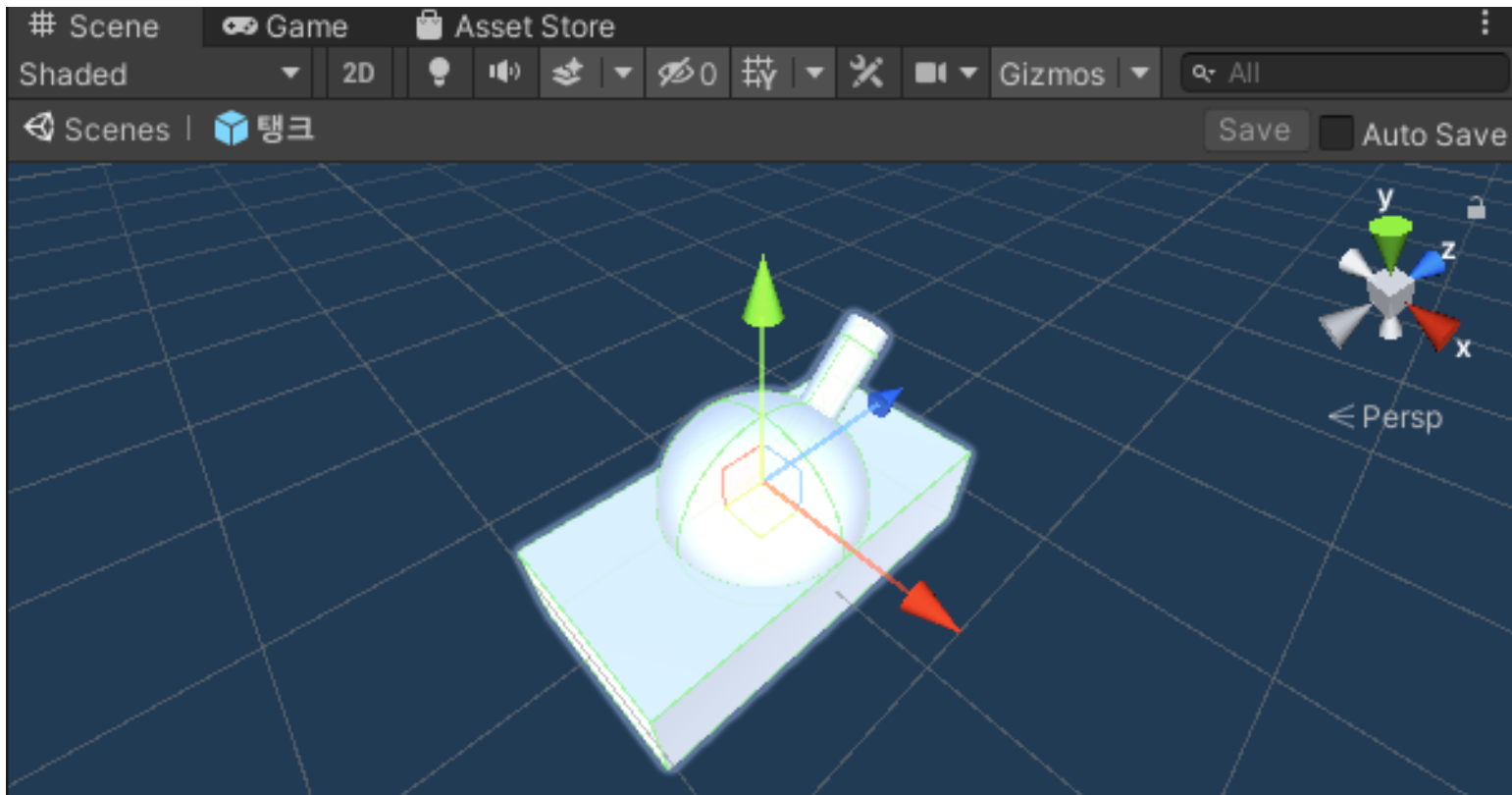
- 프로젝트 창의 탱크 프리팹을 씬 뷰 또는 계층 창으로 드래그 앤 드롭



1. 패키지를 통한 프리팹 관리

5) 프리팹 변경 방법

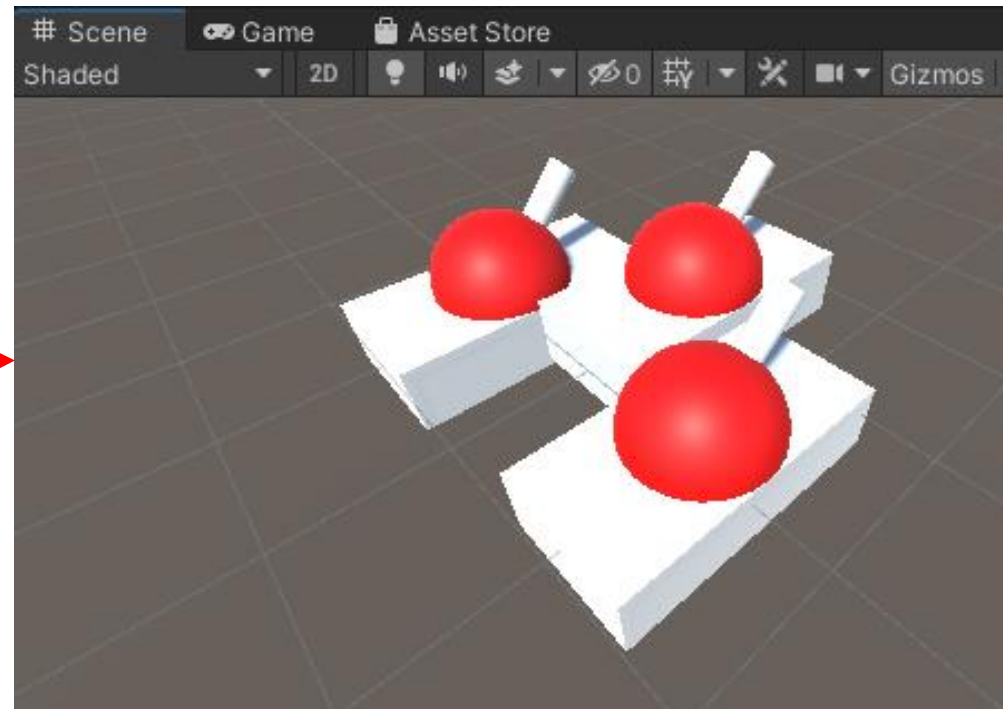
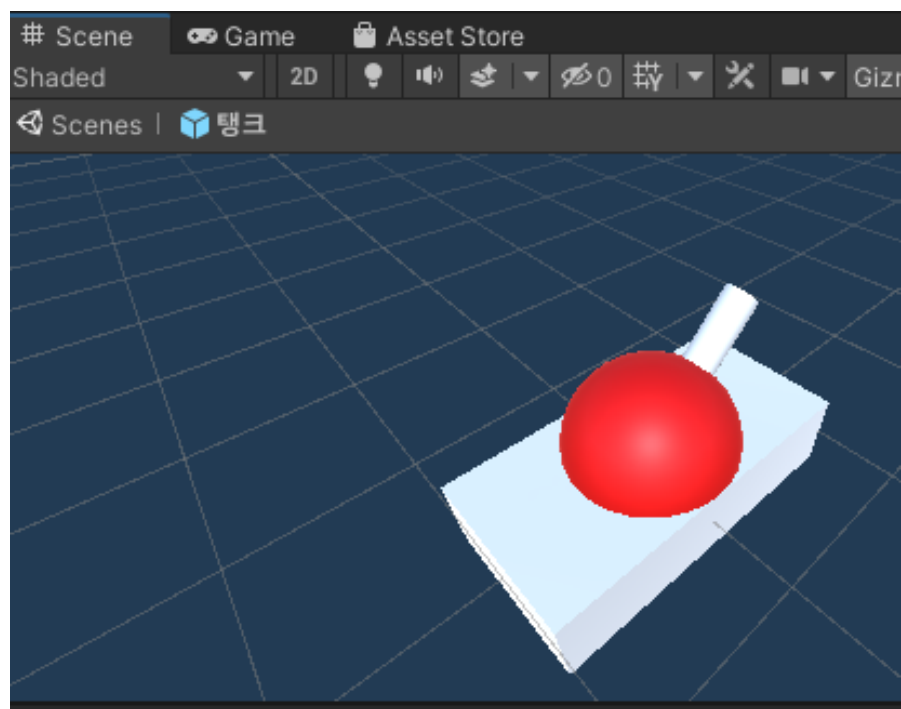
- 프로젝트 창의 **탱크** 프리팹을 열면, 프리팹 수정 화면이 활성화됨



1. 패키지를 통한 프리팹 관리

5) 프리팹 변경 방법

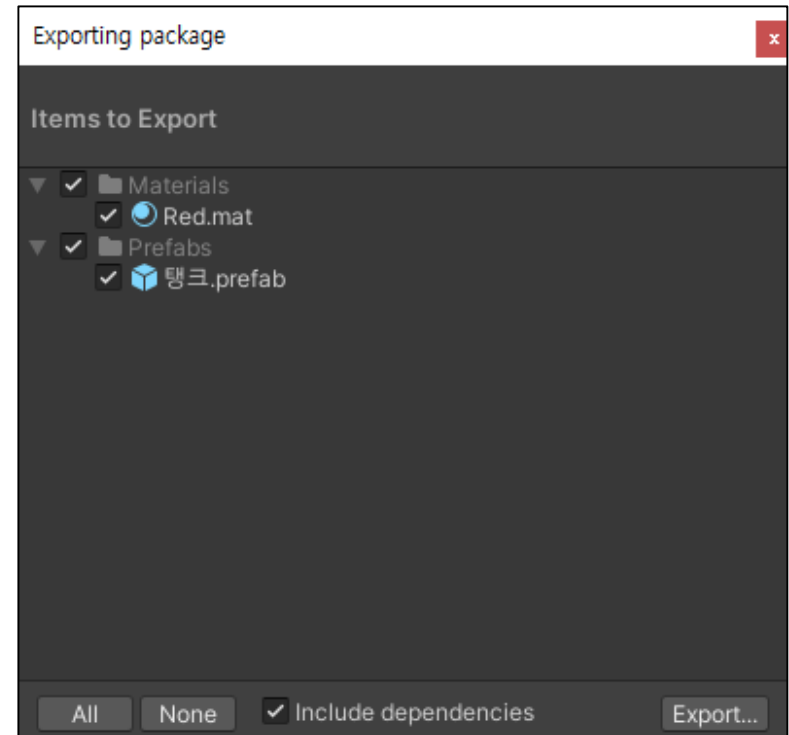
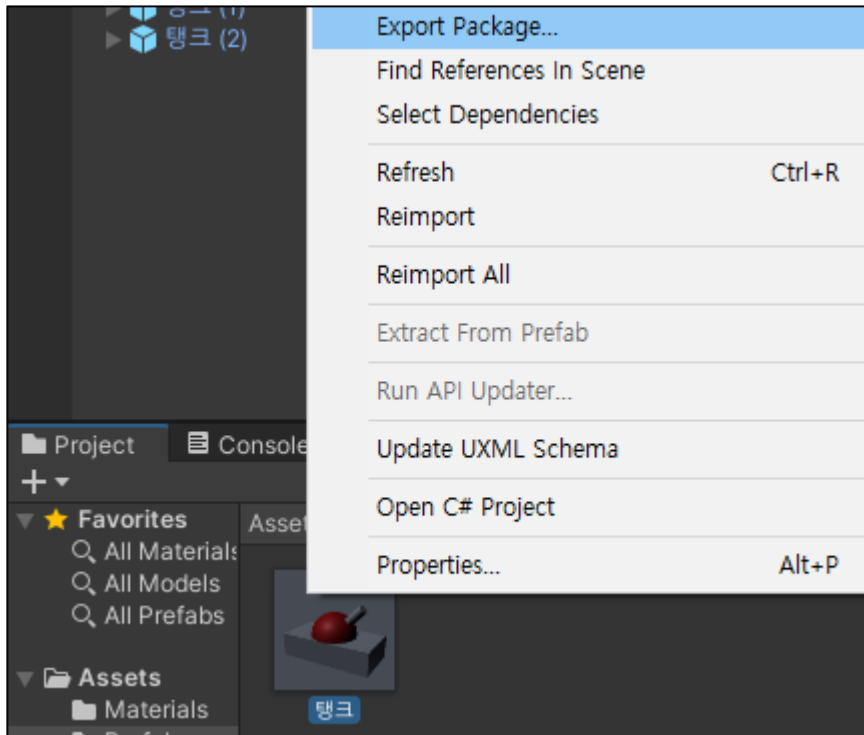
- 프리팹 수정 화면에서 프리팹을 수정하면, 인스턴스들에게도 수정 사항이 적용됨



1. 패키지를 통한 프리팹 관리

6) 패키지를 통한 프리팹 추출

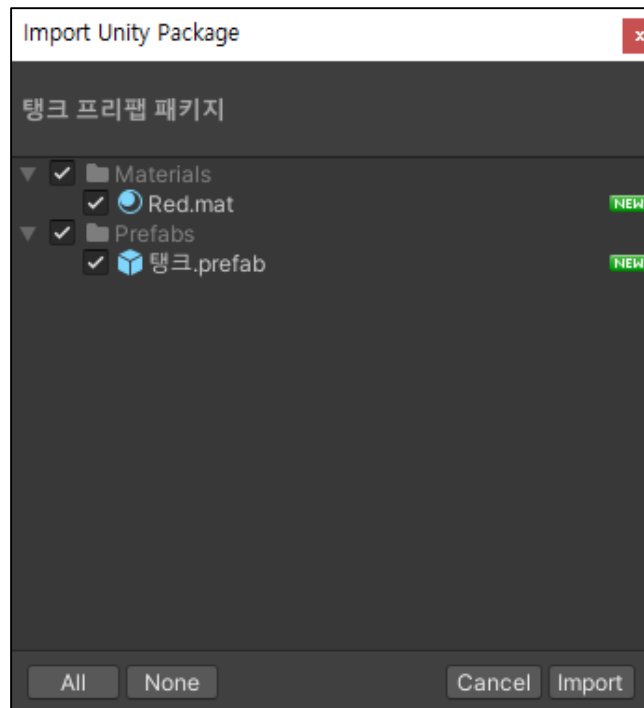
- 프로젝트 창에서 추출할 프리팹을 선택 > 마우스 우 클릭 > Export Package... 선택
- 패키지 추출 화면이 나타나면 Export 버튼 선택하여 프리팹을 패키지로 저장 가능*



1. 패키지를 통한 프리팹 관리

7) 패키지를 통한 프리팹 불러오기

- 추출된 패키지는 다른 프로젝트에서 다음의 과정을 통해 불러올 수 있음
 - ① 파일 관리자 프로그램 > 패키지 파일을 드래그 > 프로젝트 창에 드롭
 - ② 메뉴 > Assets > Import Package > Custom Package... 선택
 - ③ 프로젝트 창 > Import Package > Custom Package... 선택





스크립트 개념

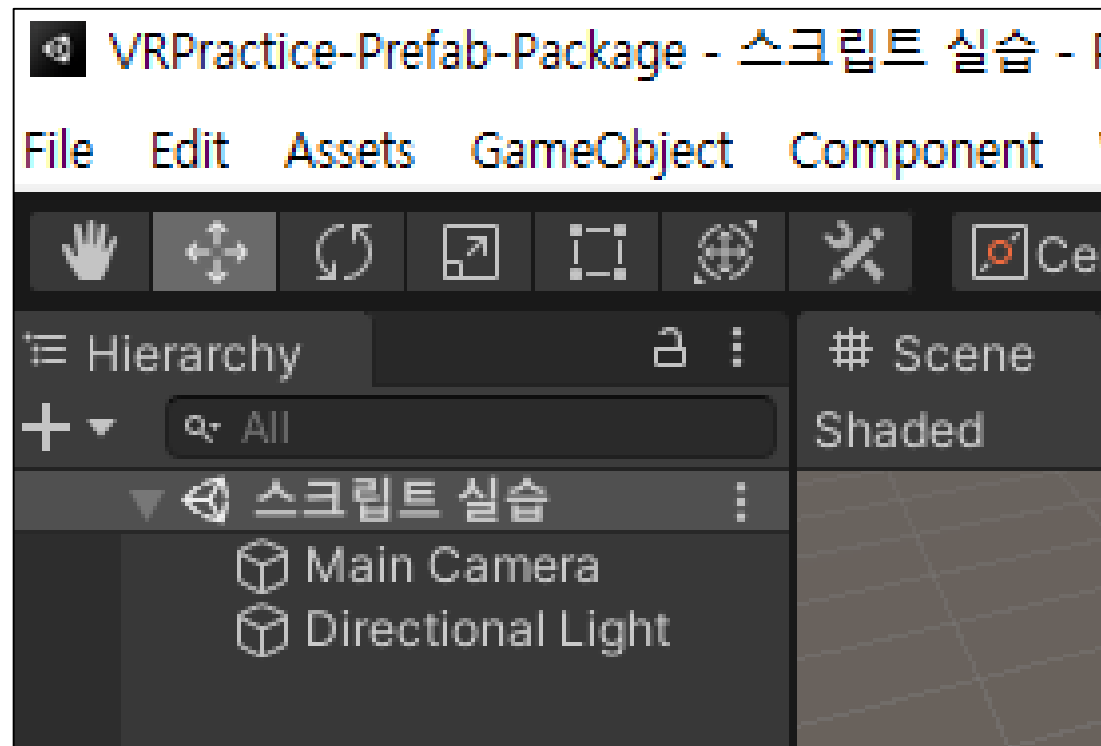
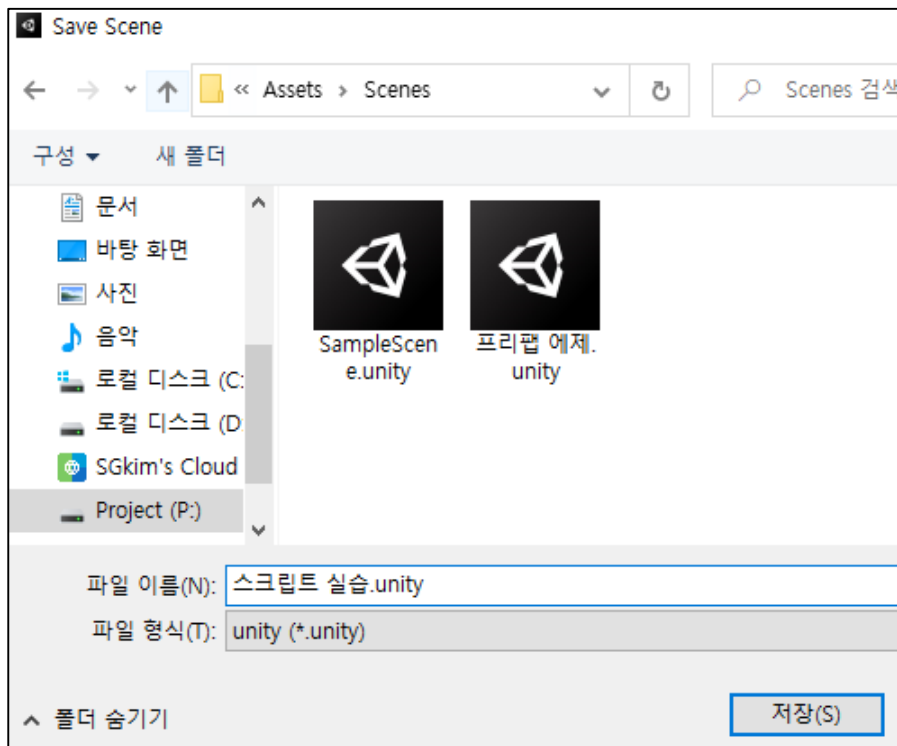
2. 스크립트 개념

- Unity에서 스크립트를 생성하면 기본적으로 **Monobehaviour**이라는 클래스를 상속받은 클래스 형태로 스크립트가 생성됨
- Monobehaviour은 다양한 특징을 가지는데 이 중 가장 큰 특징은 바로 생명 주기(life cycle)에 따라서 특정 메서드를 호출하는 것임

2. 스크립트 개념

1) 준비사항

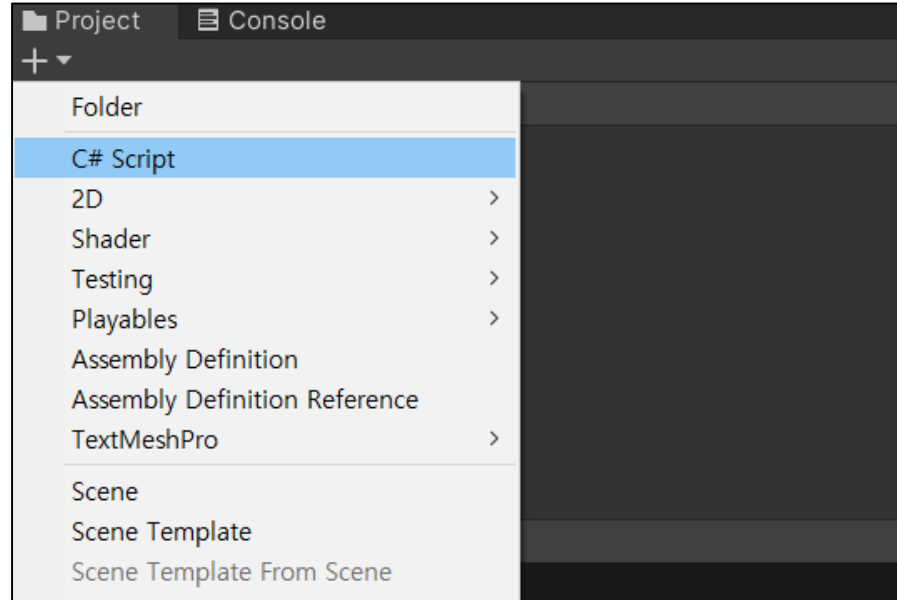
- 새로운 메뉴 > File > New Scene을 선택하여,
새로운 씬 스크립트 실습을 생성



2. 스크립트 개념

2) 스크립트 생성 방법

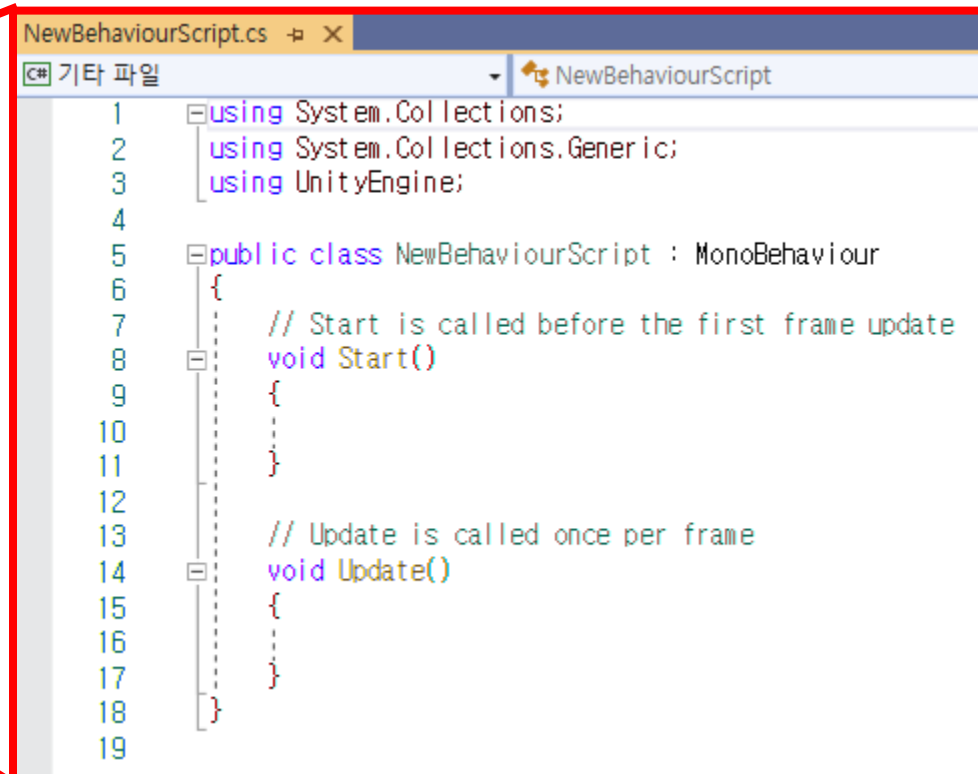
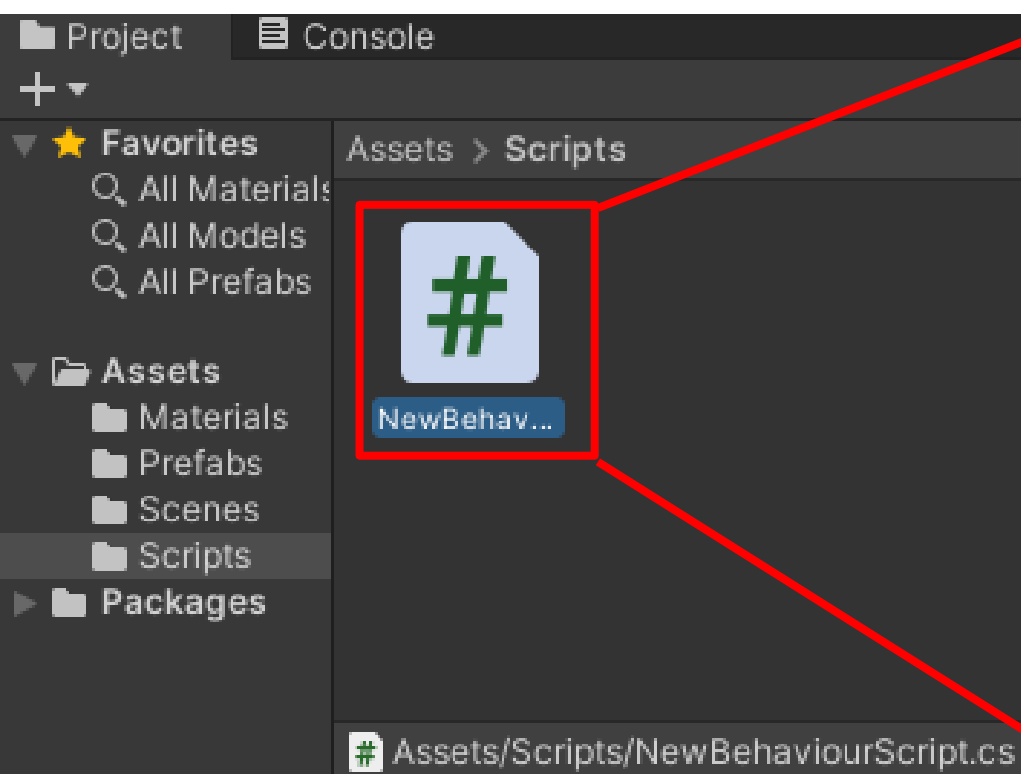
- 새 폴더 **Scripts** 생성
- Scripts 폴더 내에 새로운 스크립트를 생성
 - ① 메뉴 > Assets > Create > C# Script
 - ② 프로젝트 창 > 생성 메뉴 > C# Script
 - ③ 프로젝트 창 > 마우스 우 클릭 > Create > C# Script



2. 스크립트 개념

2) 스크립트 생성 방법

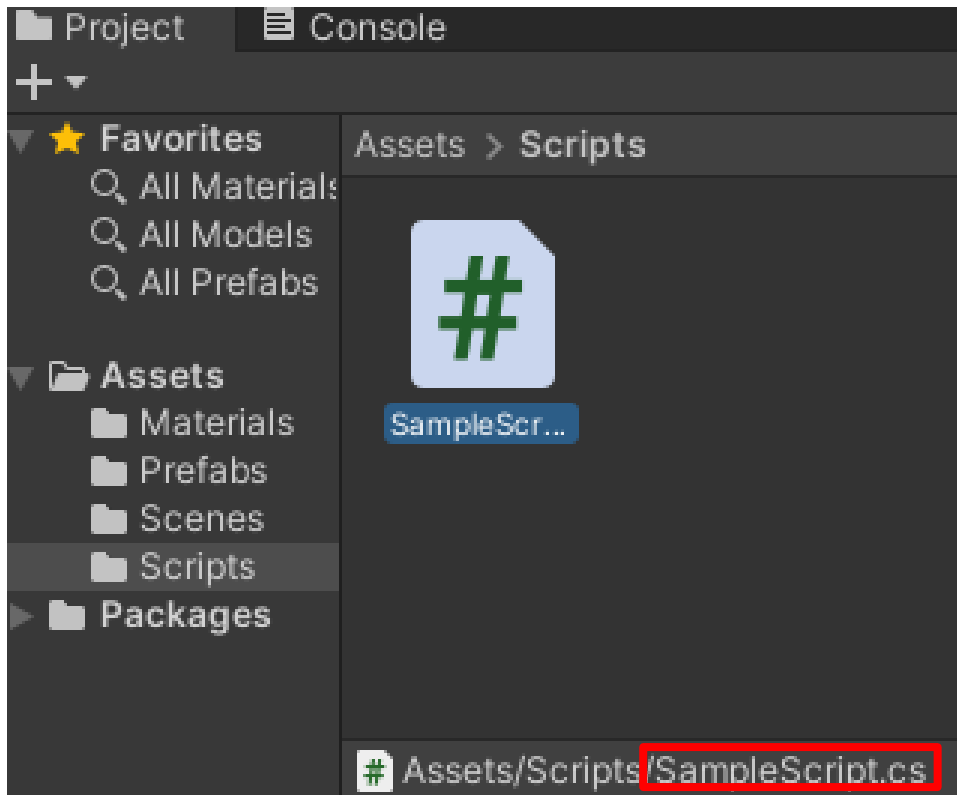
- 생성된 스크립트를 열면 코드 편집기가 실행됨



2. 스크립트 개념

3) 스크립트 적용 방법

- 스크립트를 적용하기 위해선 반드시 1) 스크립트 파일 이름과
2) MonoBehaviour를 상속받은 클래스의 이름이 동일해야 함



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

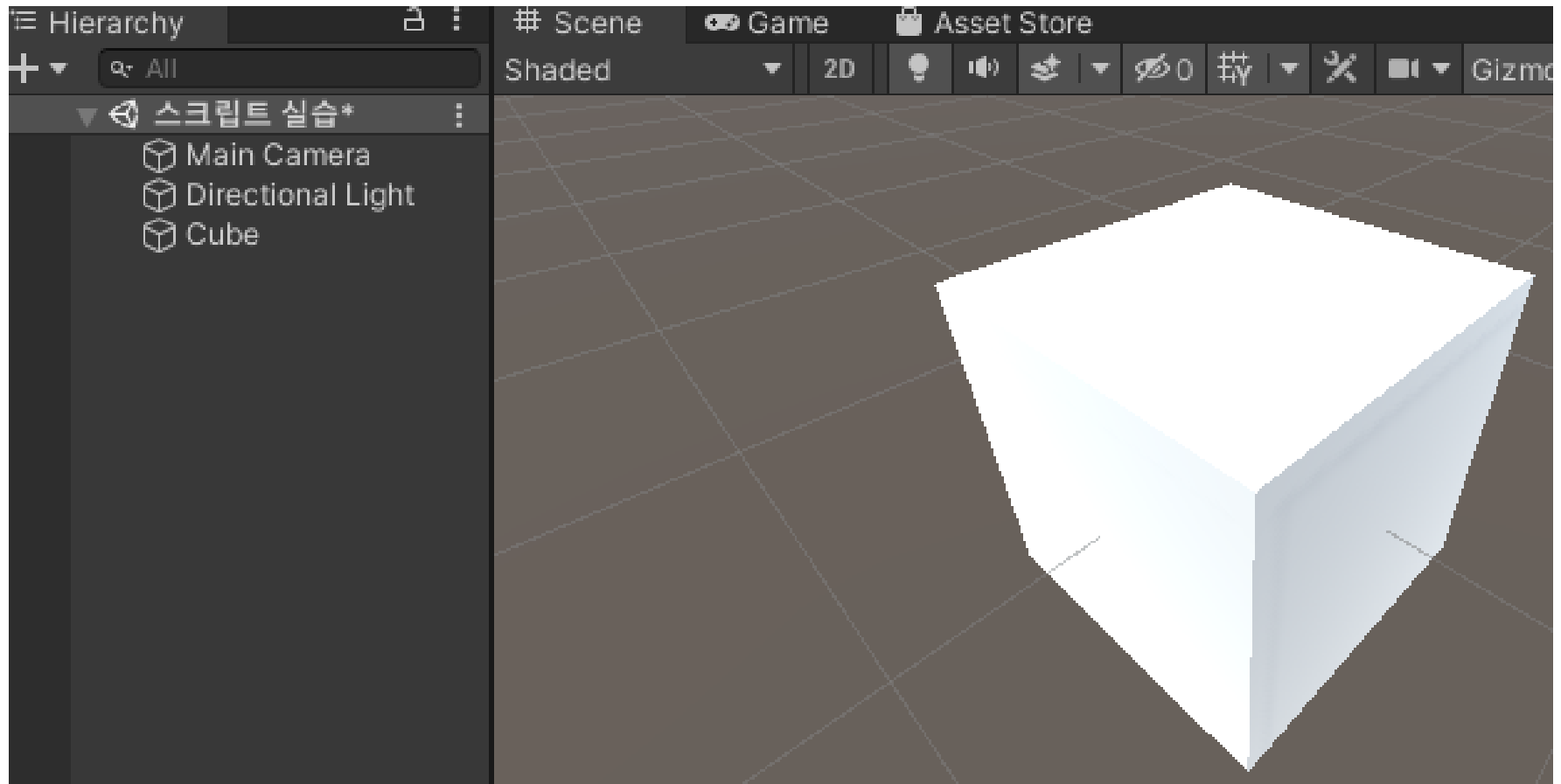
public class SampleScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        ...
    }

    // Update is called once per frame
    void Update()
    {
        ...
    }
}
```

2. 스크립트 개념

3) 스크립트 적용 방법

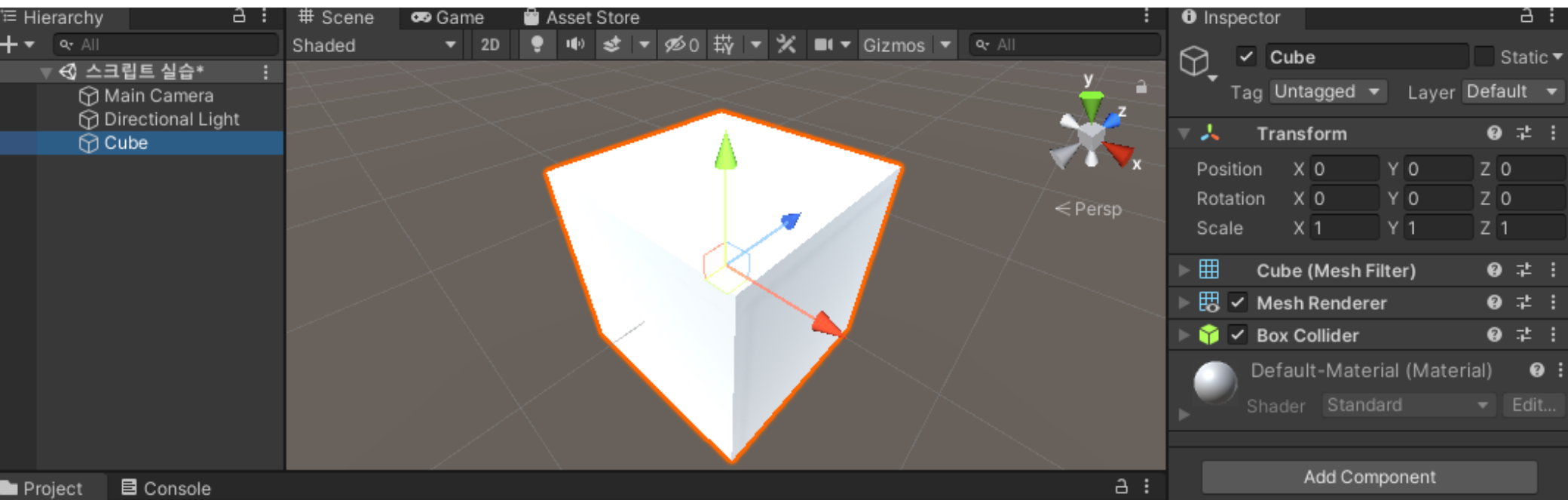
- 스크립트를 적용할 프리미티브 게임 오브젝트인 Cube를 생성



2. 스크립트 개념

3) 스크립트 적용 방법

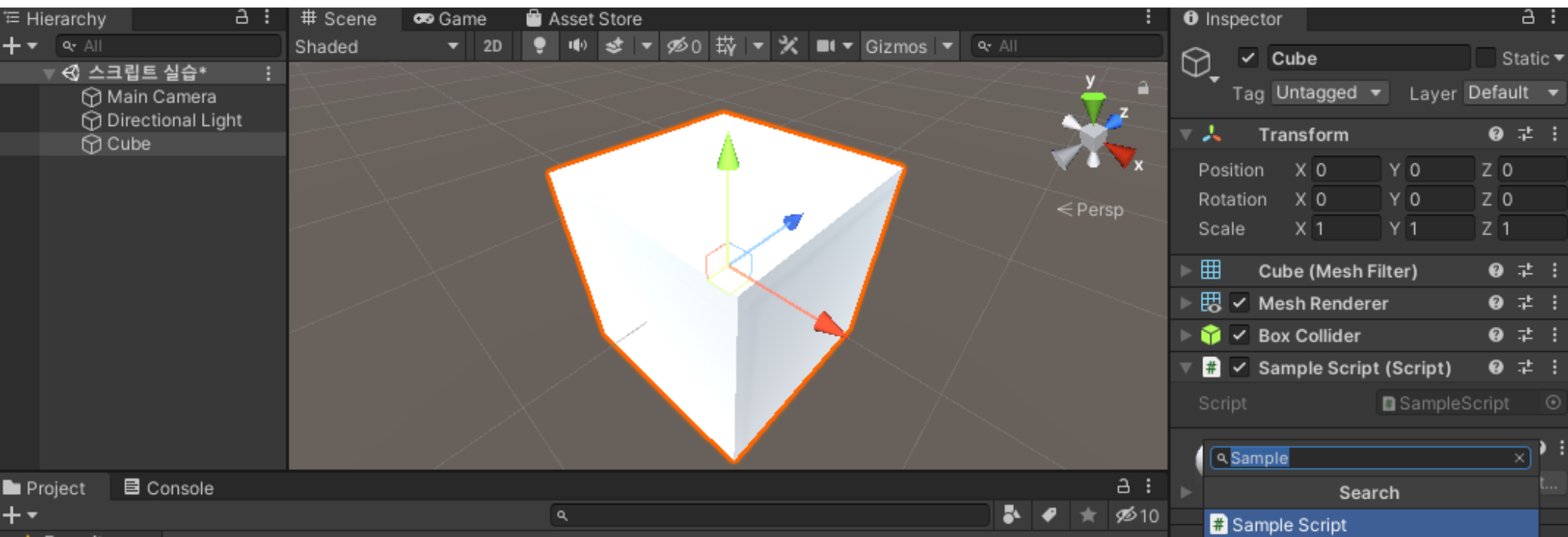
- 인스펙터 창으로 이동 후 Add Component 버튼 선택



2. 스크립트 개념

3) 스크립트 적용 방법

- 적용할 스크립트의 이름을 검색 후 선택



2. 스크립트 개념

4) Debug.Log(문자열)

- Debug.Log(문자열)는 콘솔 창에 문자열을 출력하는 메서드임
- 다음과 같이 Cube에 적용된 스크립트의 Start 메서드를 수정

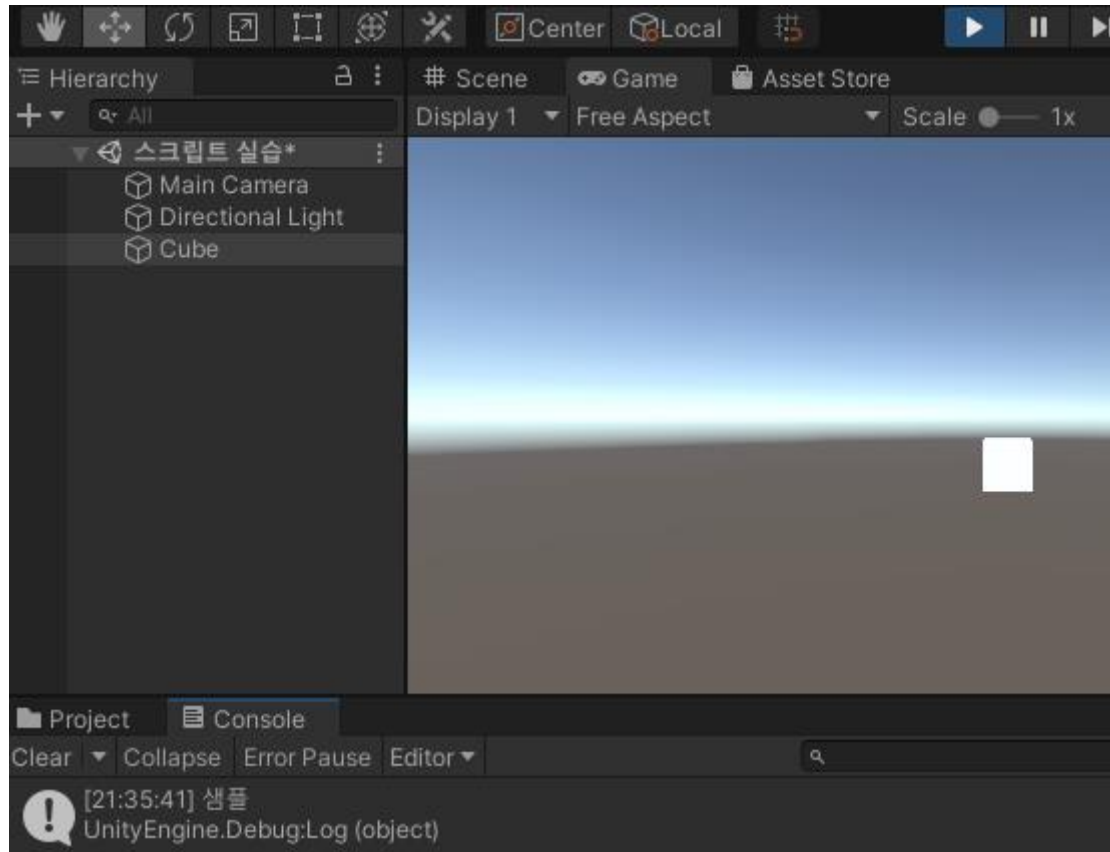
```
public class SampleScript : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("샘플");
    }

    // Update is called once per frame
    void Update()
    {
    }
}
```

2. 스크립트 개념

4) Debug.Log(문자열)

- 툴 바의 플레이 버튼을 선택하여, 문자열이 출력되는 것을 확인



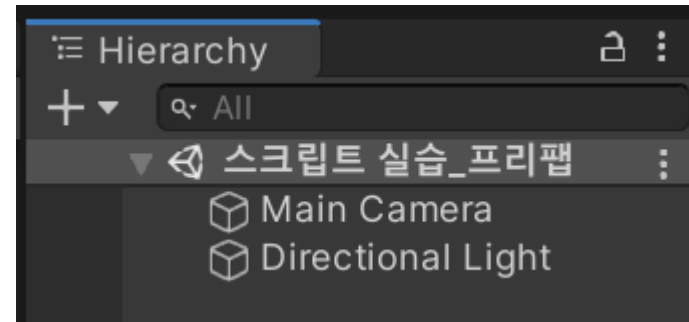
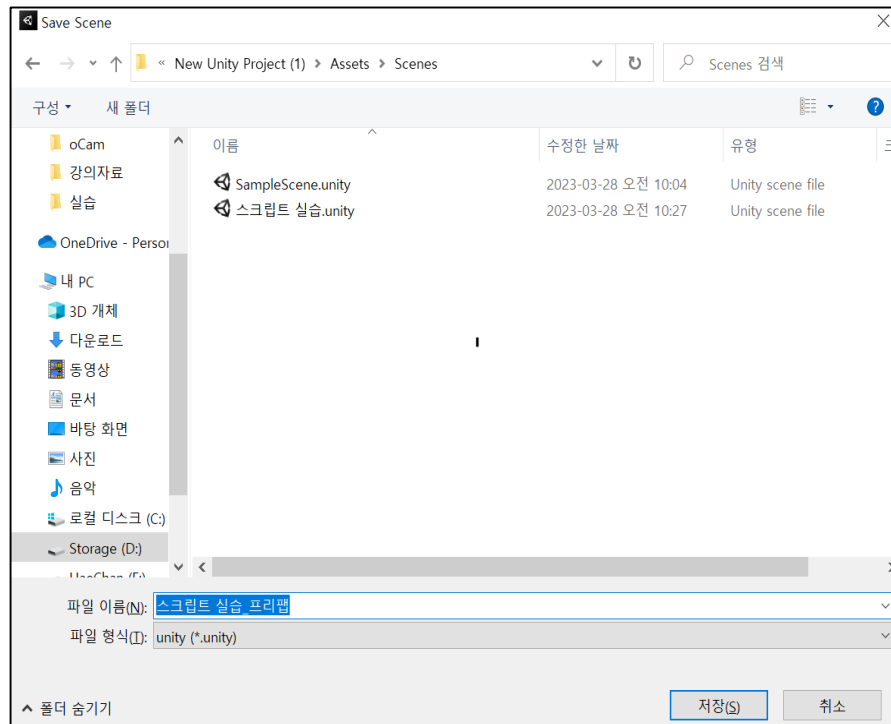


스크립트 실습

3. 스크립트 개념

1) 준비사항

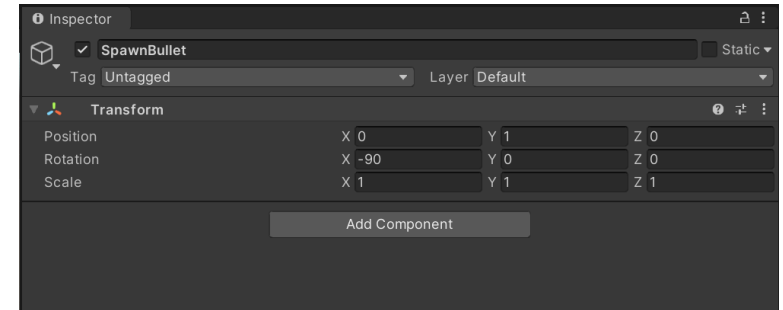
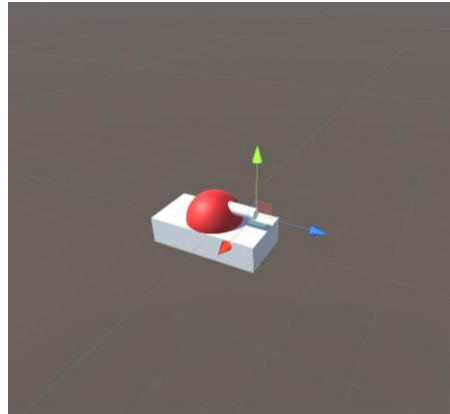
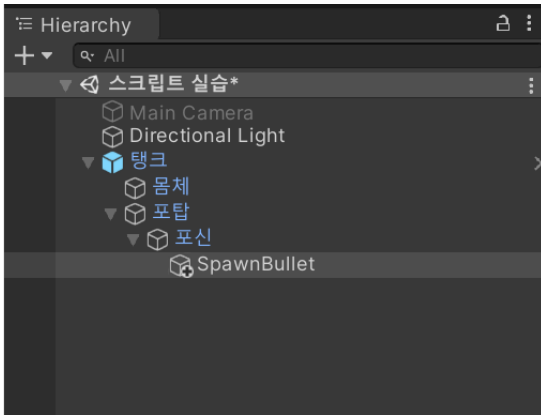
- 새로운 메뉴 > File > New Scene을 선택하여,
새로운 씬 스크립트 실습을 생성



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

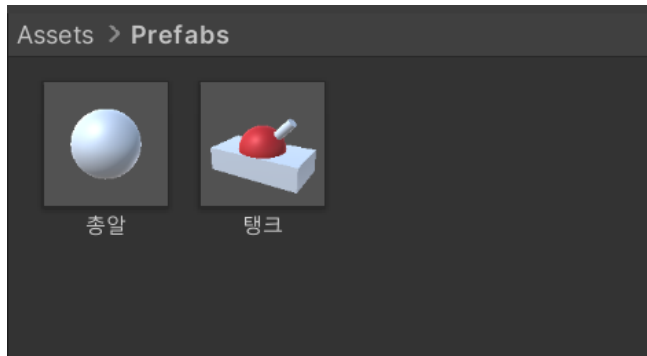
- 빈 게임 오브젝트를 생성
- 이름을 **SpawnBullet** 으로 변경 후 다음과 같이 설정
 - 부모 게임 오브젝트: 포신
 - Transform – 위치: 0, 1, 0
 - Transform - 회전: -90, 0, 0
 - Transform - 스케일: 1, 1, 1



1. 패키지를 통한 프리팹 관리

2) 오브젝트 배치

- 프리미티브 게임 오브젝트인 Sphere 게임 오브젝트를 생성
- 이름을 **총알**로 변경 후 다음과 같이 설정
 - Transform - 위치: 0, 0, 0
 - Transform - 회전: 0, 0, 0
 - Transform - 스케일: 0.2, 0.2, 0.2
- 프리팹 폴더 내 프리팹으로 저장



2. 스크립트 개념

4) Fire 스크립트

- 모든 게임 오브젝트 및 컴포넌트는 스크립트에서 제어 가능함
- Public은 외부에서 사용할 수 있도록 함
- Instantiate(오브젝트, 위치, 회전) : 오브젝트를 지정한 위치 및 회전에 따라 생성함
- Input.GetKeyDown : 키코드 버튼이 눌렸는지 확인함

```
public class Fire : MonoBehaviour
{
    public GameObject bulletPrefab;
    public Transform spawnBullet;

    // Update is called once per frame
    [Unity 메시지 참조 0개]
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Space))
        {
            var fire = Instantiate(bulletPrefab, spawnBullet.position, Quaternion.identity);
        }
    }
}
```

2. 스크립트 개념

4) Bullet 스크립트

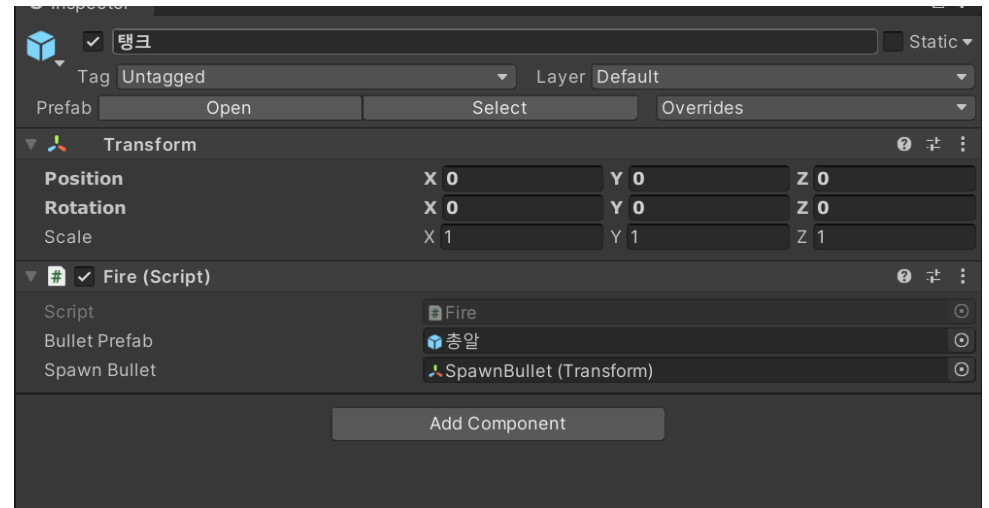
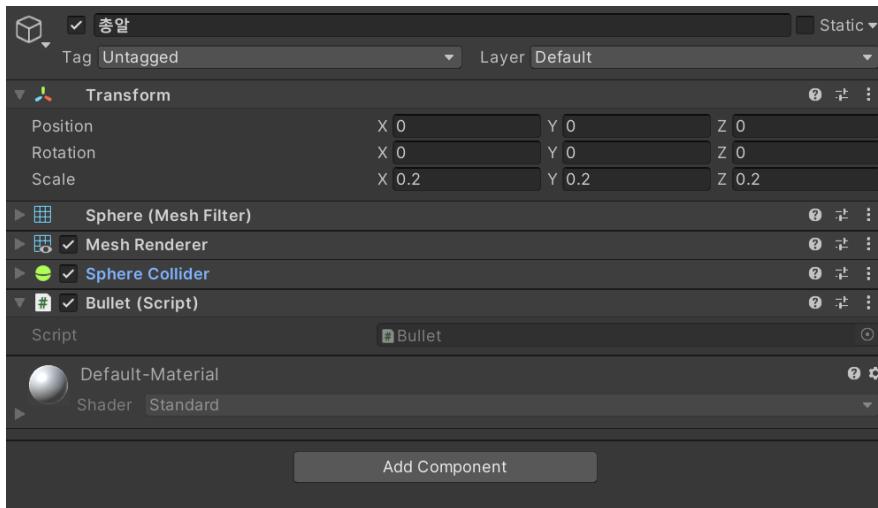
- Translate(Vector3) : Vector3의 방향으로 이동

```
☞ Unity 스크립트(자산 참조 1개) | 참조 0개
public class Bullet : MonoBehaviour
{
    // Update is called once per frame
    ☞ Unity 메시지 | 참조 0개
    void Update()
    {
        transform.Translate(Vector3.forward);
    }
}
```


2. 스크립트 개념

4) 스크립트 추가

- Bullet Script를 총알 오브젝트에 추가
- Fire Script를 탱크 오브젝트에 추가



2. 스크립트 개념

4) 메인카메라

- Transform – 위치: 0, 1, -0.5
- Transform - 회전: 0, 0, 0
- Transform - 스케일: 1, 1, 1

