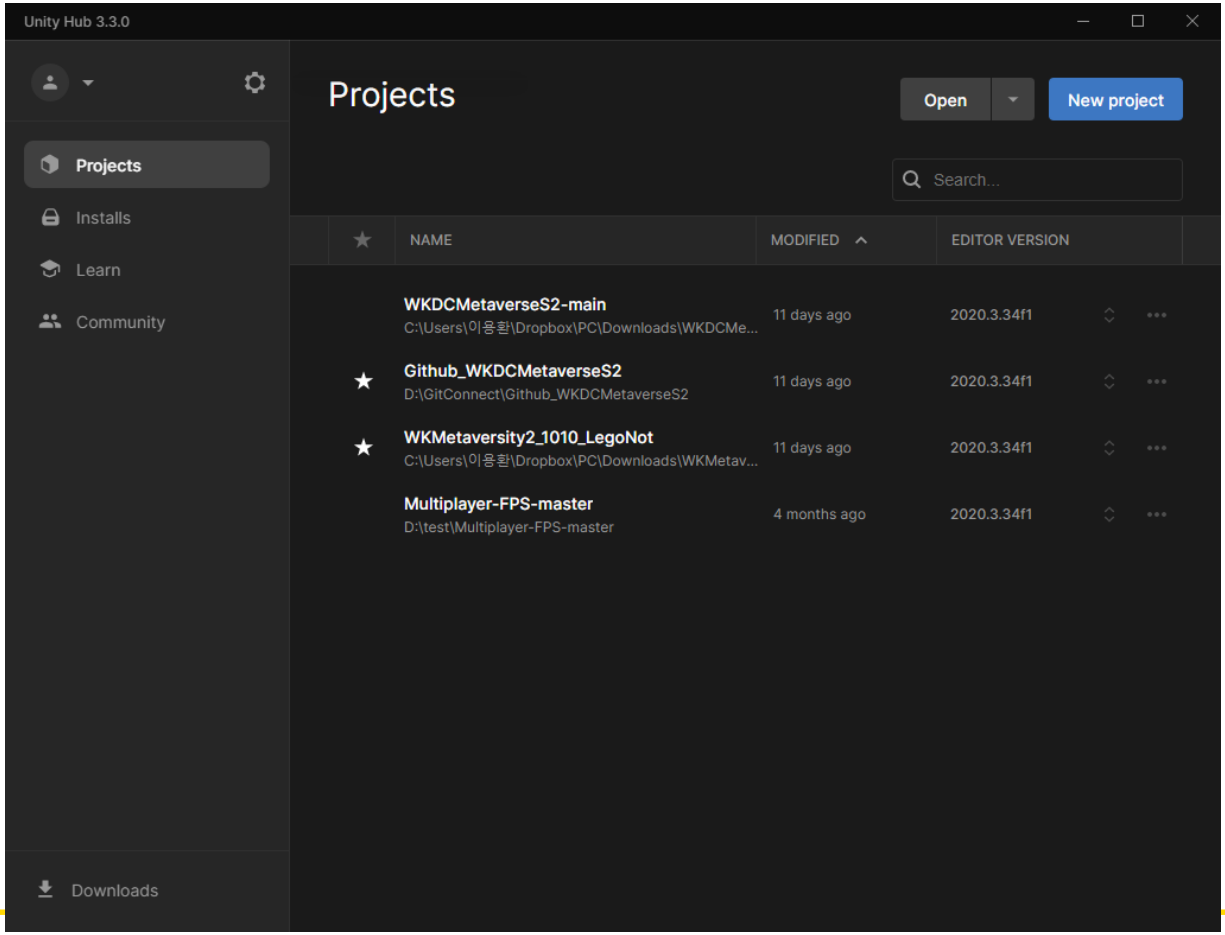
Unity Basic

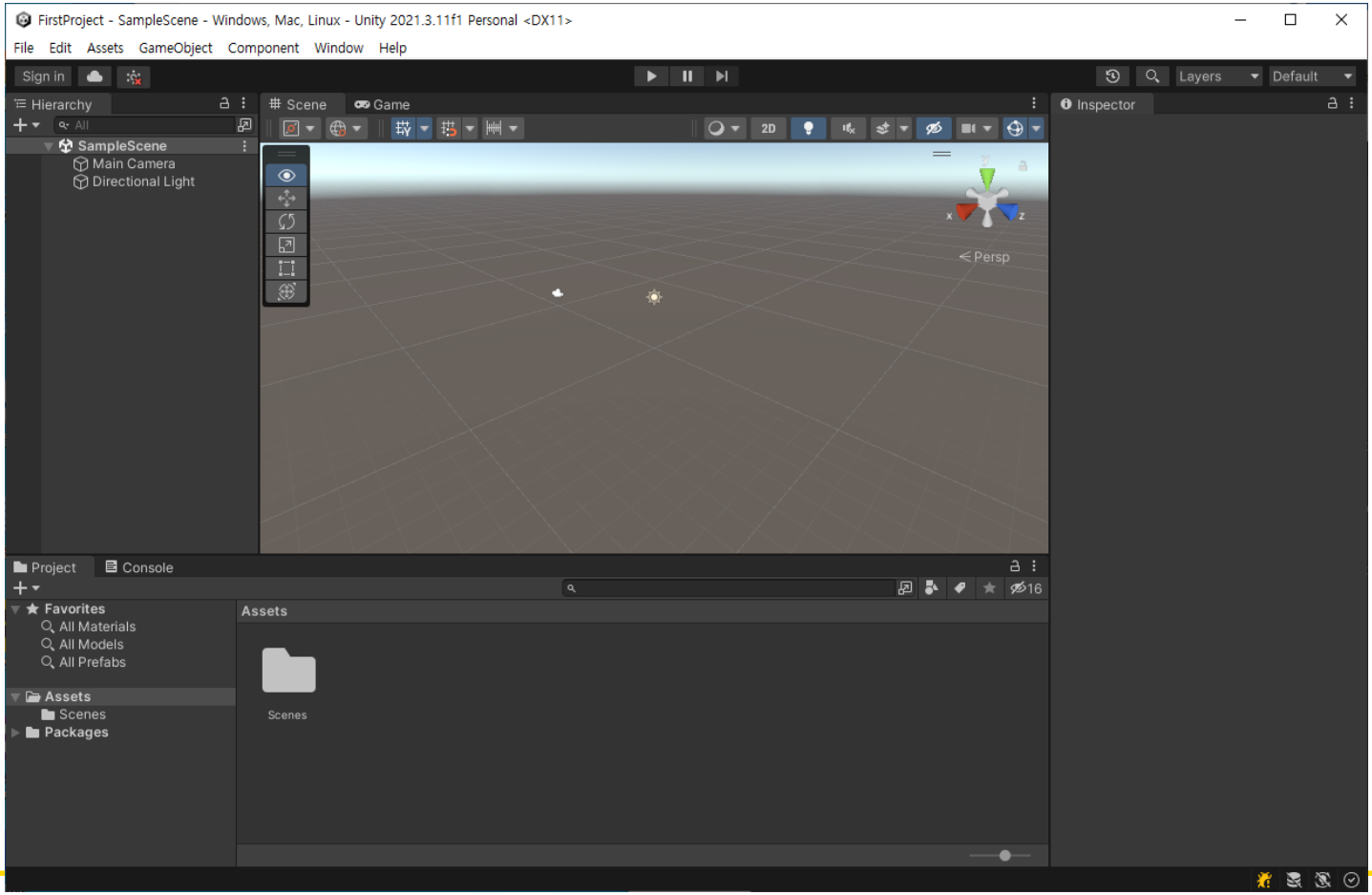
학습내용

1. Unity Hub, New Project 생성
2. Unity Interface
3. 오브젝트 추가 및 그룹핑
4. 오브젝트 매핑 및 물리력
5. 오브젝트 이동 및 충돌
6. 프리팹 활용 및 파티클
7. UI 활용 (캔버스)
8. 씬 전환

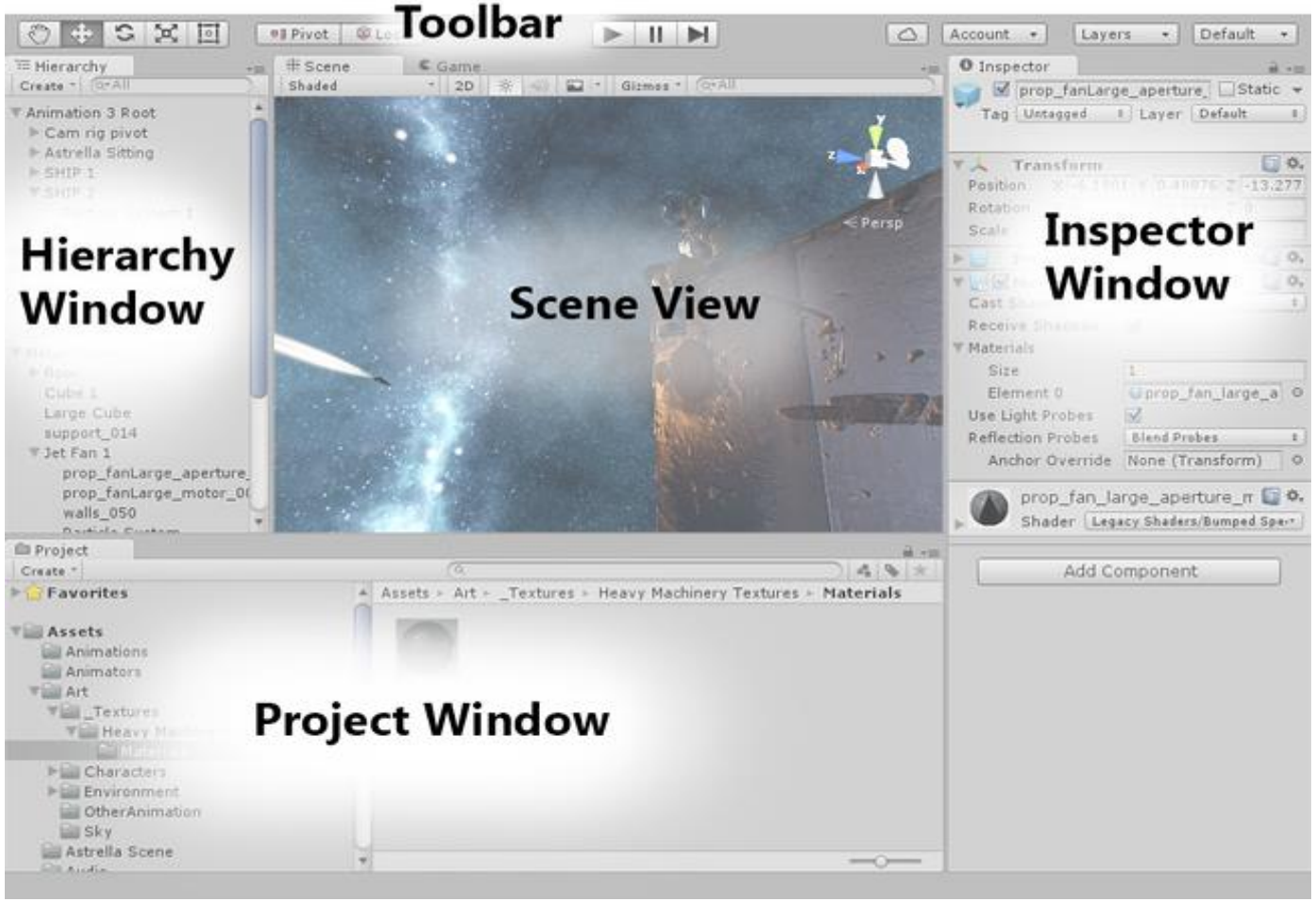
- Unity Hub
  - Unity 설치 및 계정 관리, 사용자 프로젝트, 라이선스, Tutorial 등을 관리하는 프로그램



- New Project
  - 생성된 빈 프로젝트 화면

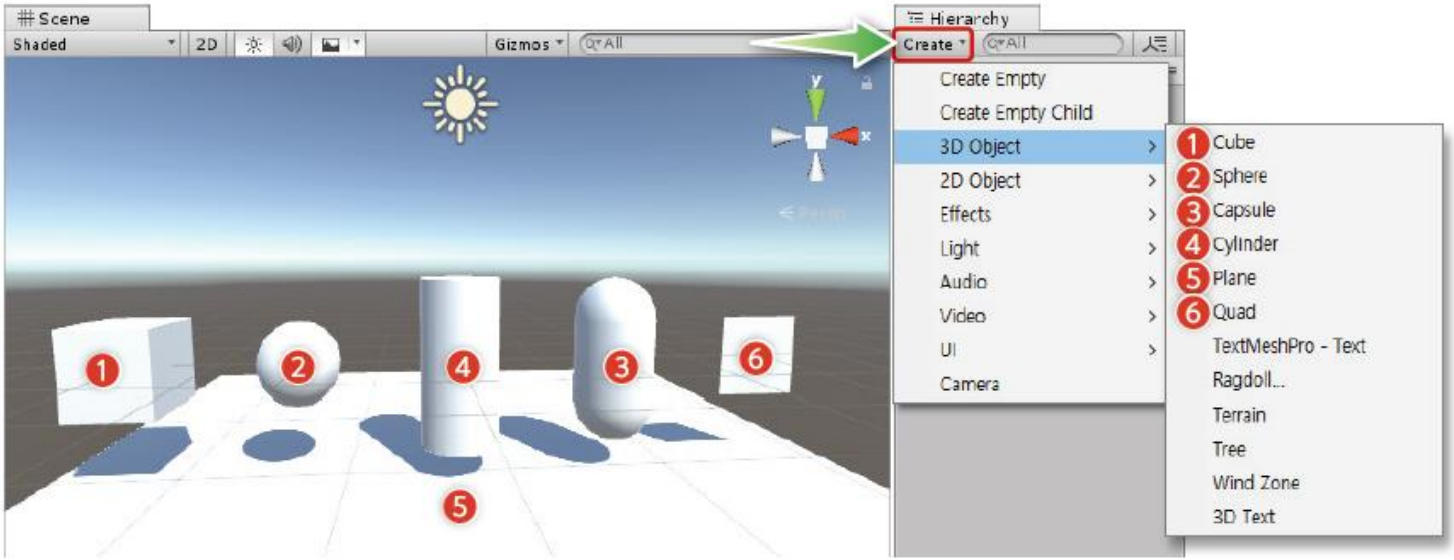


- Unity Interface



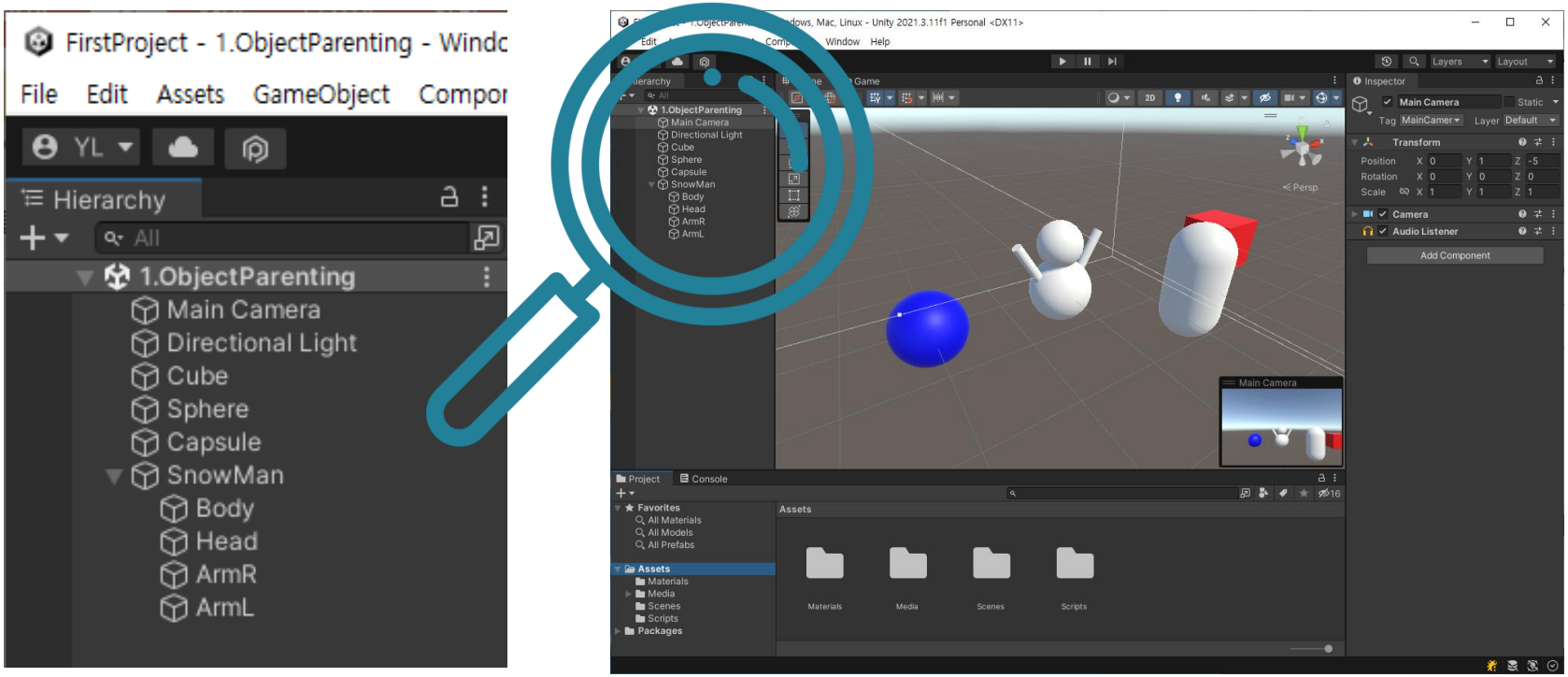
오브젝트 추가 및 그룹핑

- 기본 오브젝트 추가
  - 게임 오브젝트 : 게임에 필요한 모든 요소 (캐릭터, 총, 조명 등)
  - 게임의 일정 부분은 유니티가 제공하는 **기본 오브젝트** (Primitive Object) 를 이용하여 만들 수 있음
  - 하이러리키 [Create – 3D Object] OR 메뉴 [GameObject – 3D Object] 클릭하여 해당 오브젝트를 씬에 추가



  - Cube (정육면체), Sphere (구), Capsule (캡슐), Cylinder (원기둥), Plane (눕혀진 평면), Quad (세워진 평면)

- 그룹핑 (Grouping, Parenting)
  - 여러 개의 오브젝트를 하나로 묶는 작업
  - 오브젝트가 독립된 요소로 분리되어 있으면 이동, 충돌 판정 등의 처리가 번거로워 오브젝트를 하나의 객체로 묶는 것이 편리



씬에 Sphere 와 Cylinder 오브젝트 추가

Transform 속성

Object	이름	Position	Rotation	Scale
Sphere	Body	0, 0, 0	0, 0, 0	1, 1, 1
	Head	0, 0.75, 0	0, 0, 0	0.7, 0.7, 0.7
Cylinder	Arm1	-0.5, 0.5, 0	0, 0, 30	0.15, 0.4, 0.15
	Arm2	0.5, 0.5, 0	0, 0, -30	0.15, 0.4, 0.15

– 그룹핑 방법

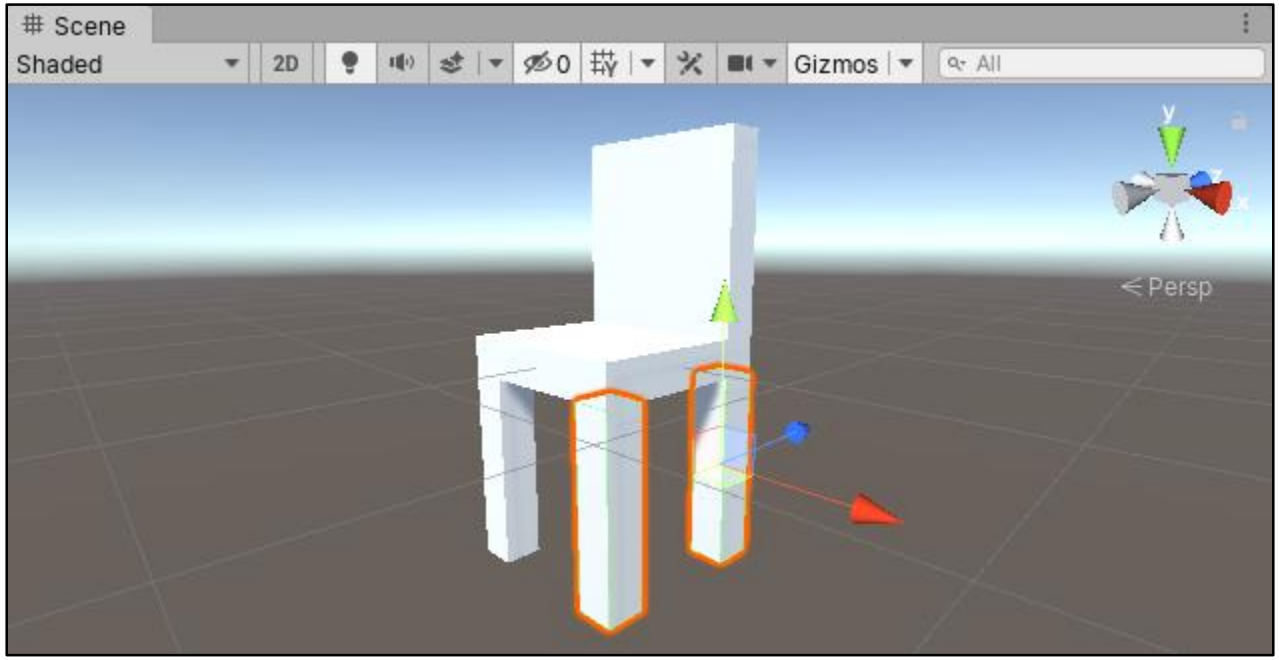
① 주체가 되는 오브젝트를 정하고, 나머지를 (해당 오브젝트의) 하위 오브젝트로 설정

② 빈 오브젝트를 새로 만들고, 오브젝트 전체를 빈 오브젝트의 하위 오브젝트로 설정

- 그룹핑 해제 방법 : 해당 오브젝트를 하이러리키의 빈 곳으로 드래그

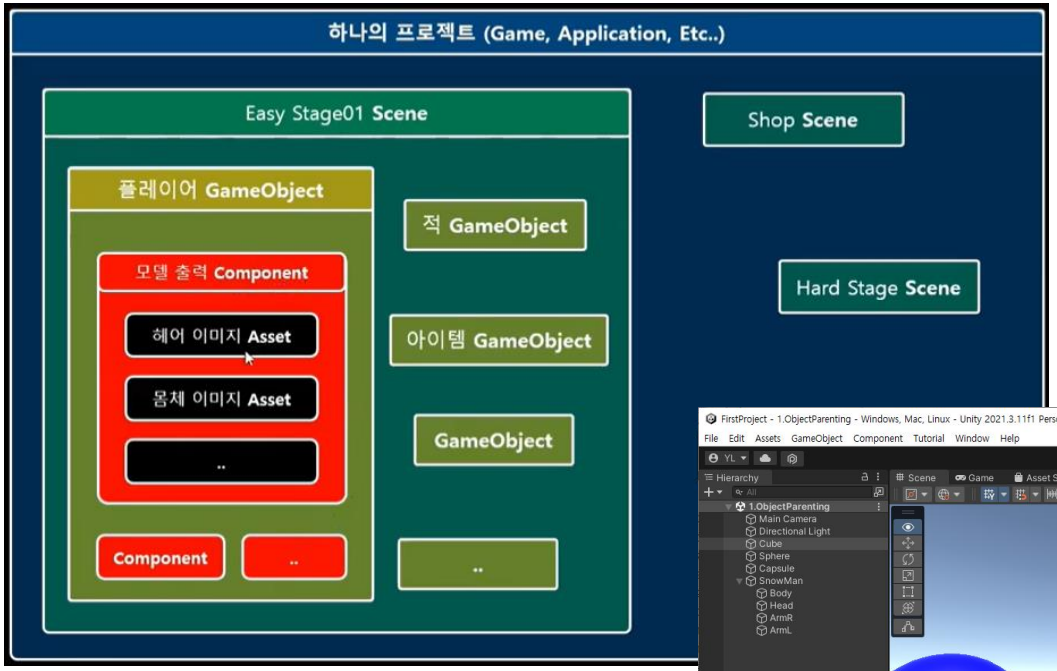
Practice

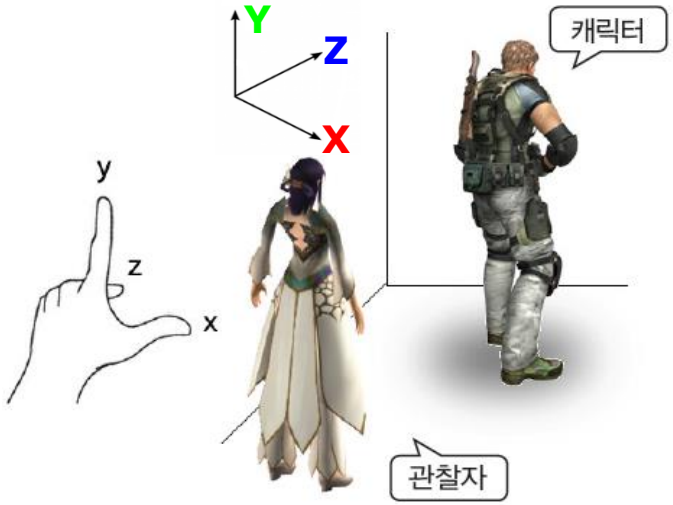
- 아래의 그림을 보고, 기본 오브젝트를 활용하여 게임 오브젝트를 만들어 봅니다

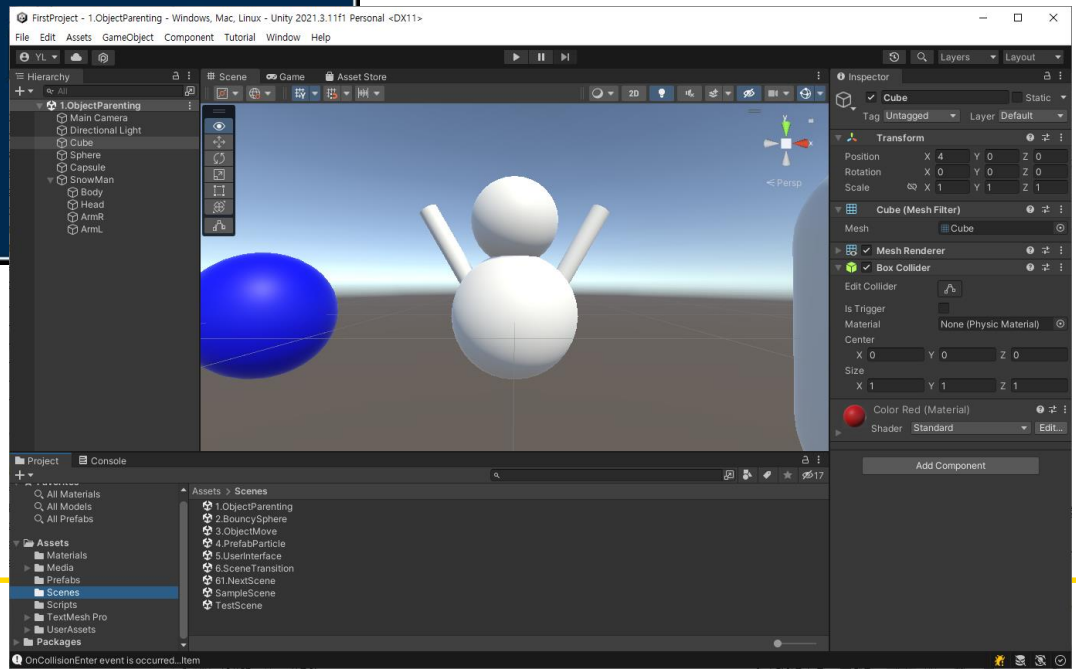


참조

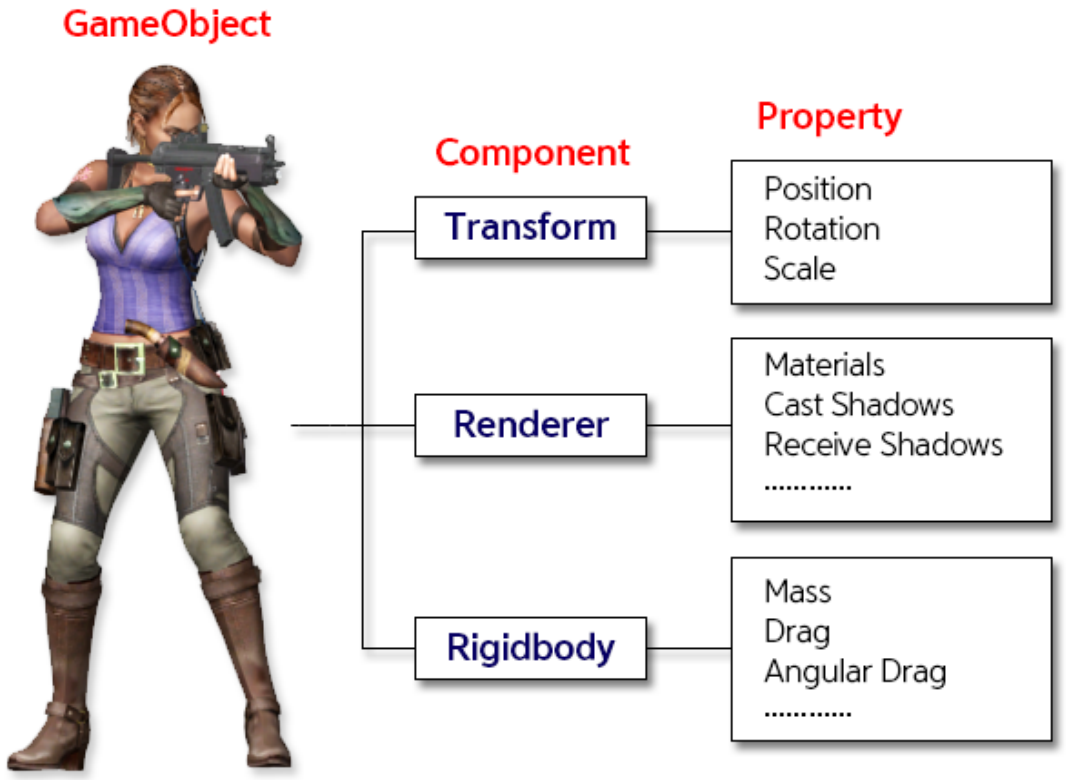
- 프로젝트, 씬, 게임 오브젝트 와 컴포넌트



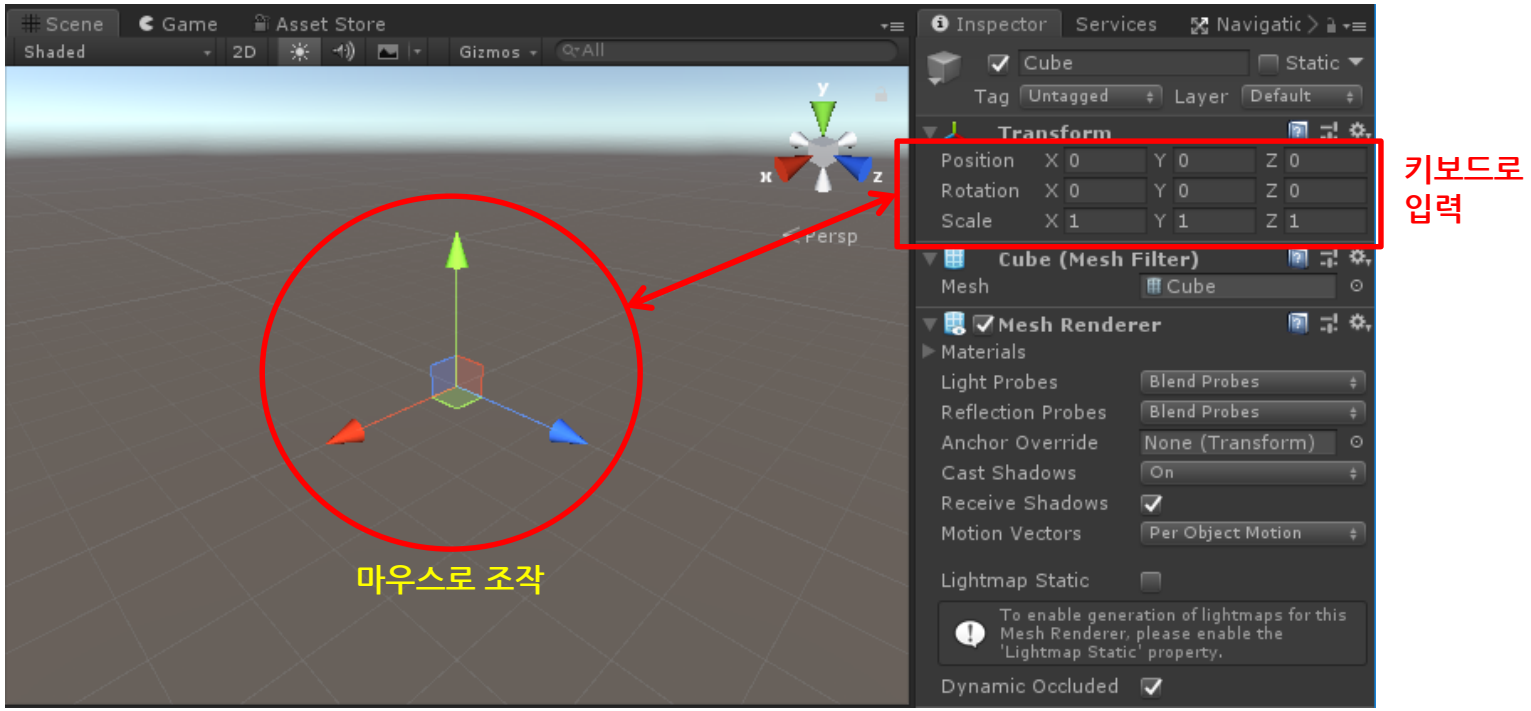




- 컴포넌트와 속성
  - **속성**(Property, 프로퍼티) : 오브젝트의 위치, 방향, 질량, 색상 등의 개별적인 값
  - **컴포넌트**(Component) : 서로 관련된 속성을 묶음 (인스펙터에서 볼드체로 표시됨)

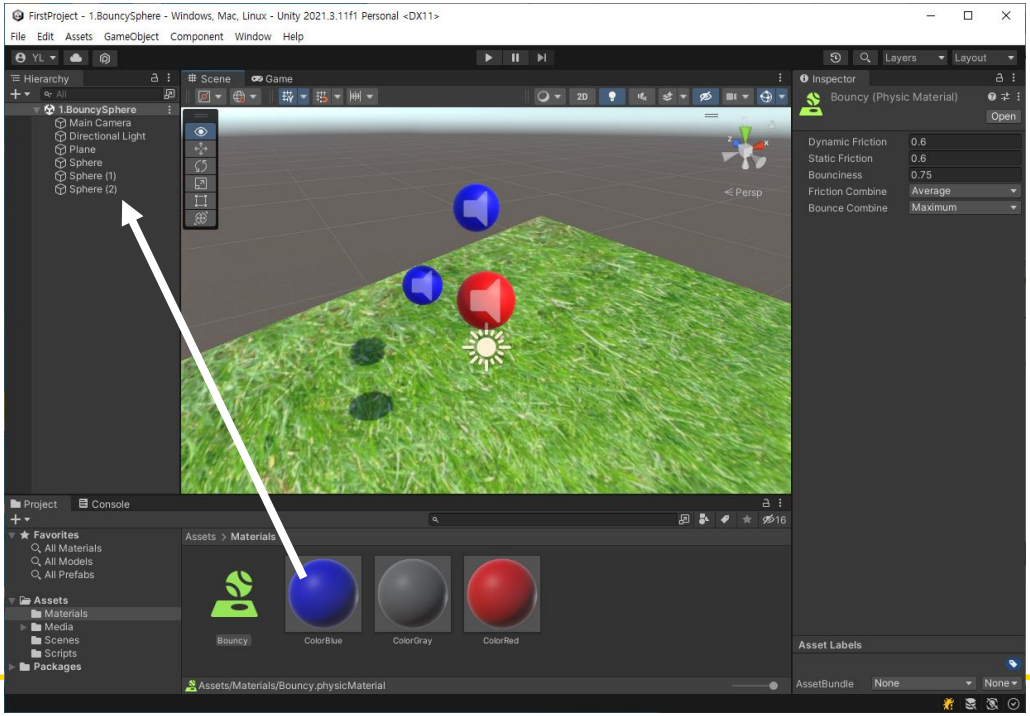


- 컴포넌트 속성 변경
  - 인스펙터에서 컴포넌트의 속성을 변경하면, 씬 뷰에 즉각적으로 반영됨
  - 씬 뷰에서 오브젝트에 대한 이동/회전/스케일 등을 조작하면, 곧바로 인스펙터의 속성에 결과가 반영됨

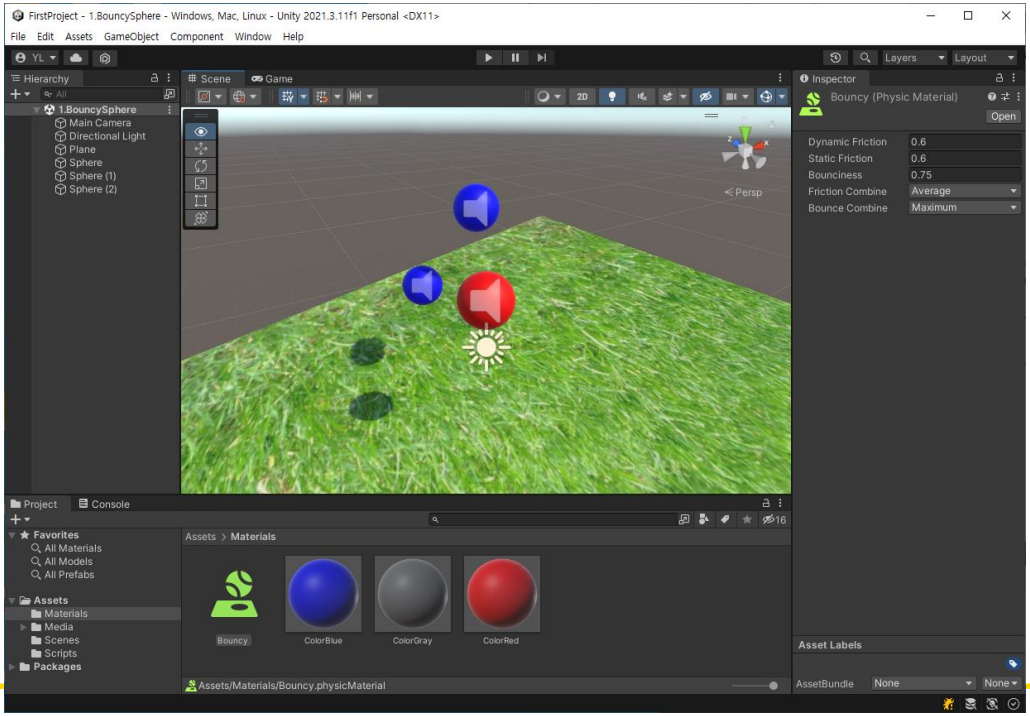


오브젝트 매핑 및 물리력 적용

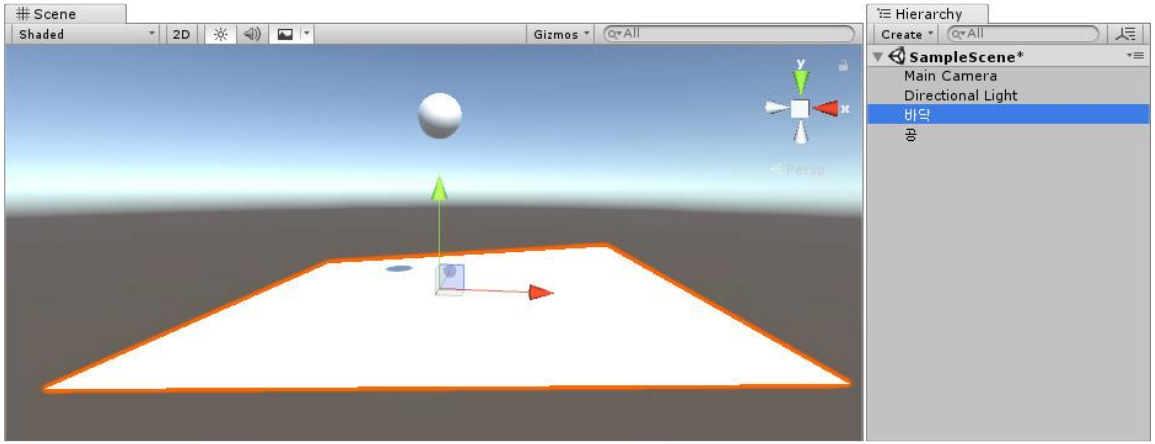
- 매핑
  - 오브젝트의 표면을 컬러나 이미지로 씌우는 작성
  - 매핑하기 위해 매터리얼(Material)이 필요
  - 매터리얼을 오브젝트로 Drag and Drop 하면 매핑이 이뤄짐
  - 매핑된 매터리얼이 삭제되면, 오브젝트의 색상이 음영 없는 **자주색**으로 바뀜



- 물리력 적용 - Rigidbody (리지드바디)
  - 물리엔진을 컴포넌트로 만든 것
  - 중력, 마찰, 충돌의 판정 등에 관여
  - 물체가 충돌 반응을 일으키려면, 두 물체 중에 적어도 어느 하나는 리지드바디가 있어야 하며, 충돌 이벤트는 리지드바디가 있는 오브젝트에만 발생



씬에 Plane와 Sphere 오브젝트 추가



- 오브젝트 속성

Object	Name	Position	Rotation	Scale
Plane	바닥	0,0,0	0,0,5	1,1,1
Sphere	공	0,4,0	0,0,0	1,1,1

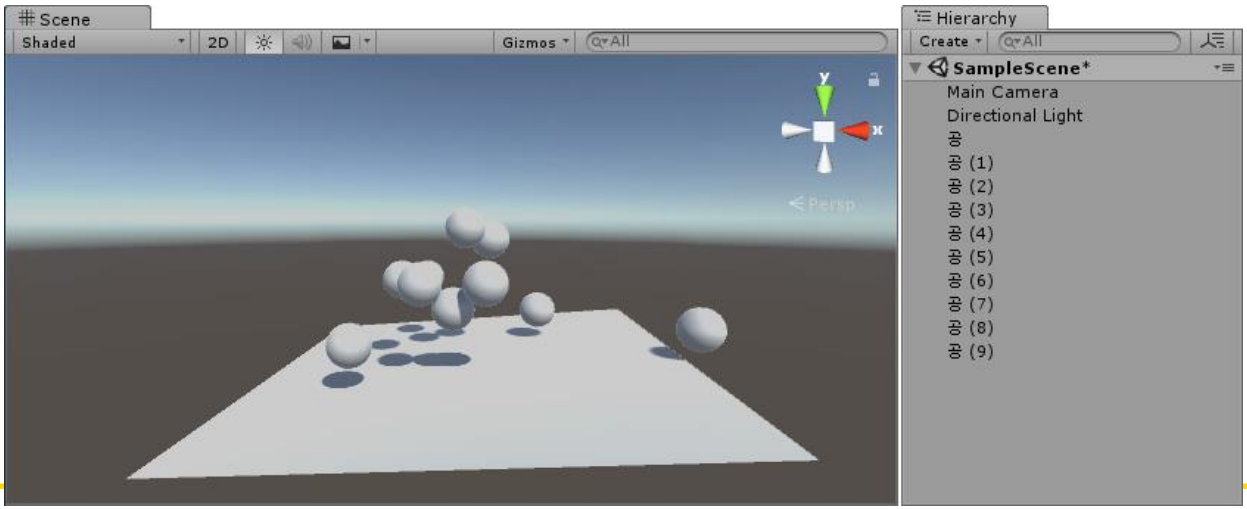
– 게임을 실행하면, 바닥과 공에 모두 물리적 영향을 받지 않는 무중력 상태이므로 게임 뷰에서는 **아무런 변화가 없음**

1. Sphere에 리지드바드 추가
2. Sphere 와 Plane 오브젝트에 텍스처를 입힘 (매핑)
3. 오브젝트 반사(반발력 부여) – Physics Material을 생성하고 매핑
4. (반발력에 의해) Sphere가 띄어 오르게 되며, 이때 사운드를 추가하여 효과음을 부여

   오디오 클립을 오브젝트에 드래그하여 AudioSource를 추가

   충돌 판정 스크립트를 작성 (다음 슬라이드 참조)

   스크립트를 Sphere에 드래그
5. Sphere를 복제(^D)하여 여러 개를 배치하고 게임을 실행

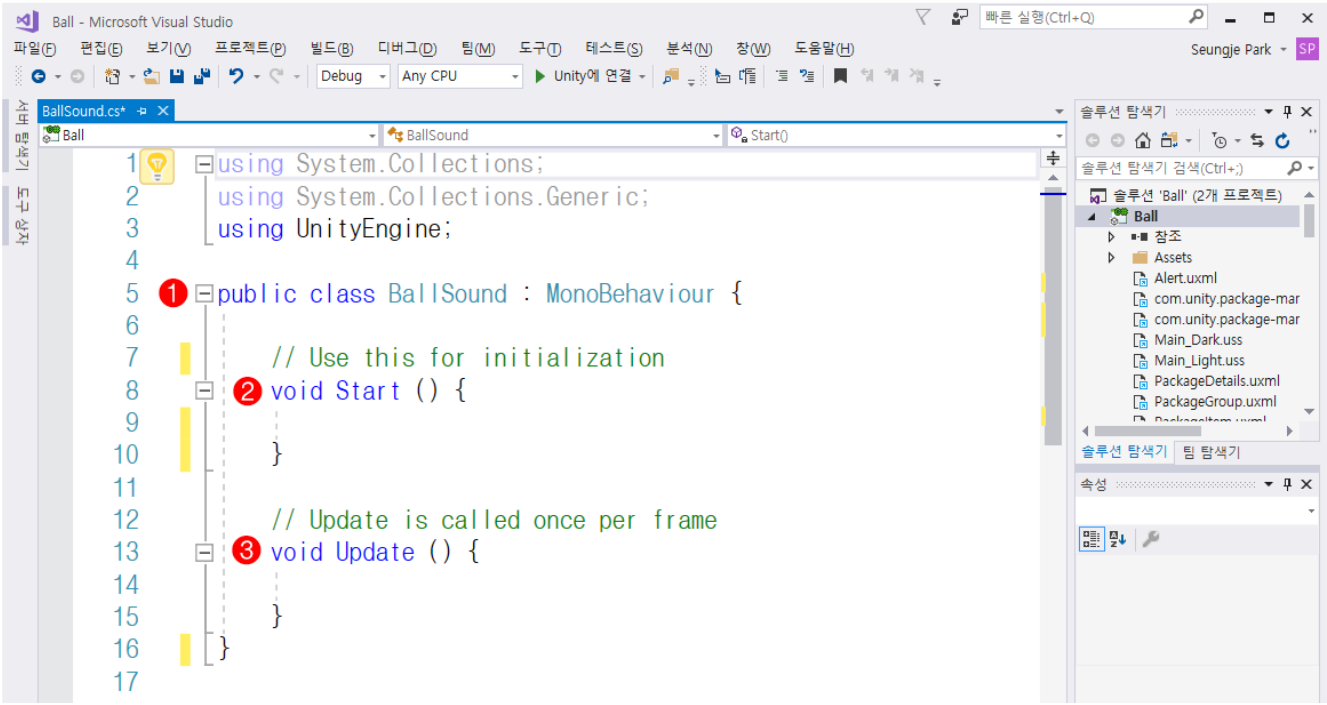


★ 게임 실행 중에 변경된 속성은 (게임 실행이 끝나면) 모두 무시됨


```
public class SphereBouncySound : MonoBehaviour
{
    // AudioSource를 저장할 변수 선언
    AudioSource ballAudio;

    // Start is called before the first frame update
    void Start() {
        // 시작할 때 컴포넌트 열기
        ballAudio = GetComponent<AudioSource>();  // 씬이 로딩되면 AudioSource를 변수로 읽어 들임
    }

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

    }

    // 충돌 처리 함수
    void OnCollisionEnter(Collision collision) {  // 오브젝트 충돌이 발생하면 호출되는 이벤트 함수
                                                  // 충돌의 상대편 정보가 매개변수 collision으로 전달됨
        ballAudio.Play();   // AudioSource에 할당된 오디오 클립을 재생
        Debug.Log("공과 충돌한 오브젝트 = " + collision.GameObject.name);
    }
}
```

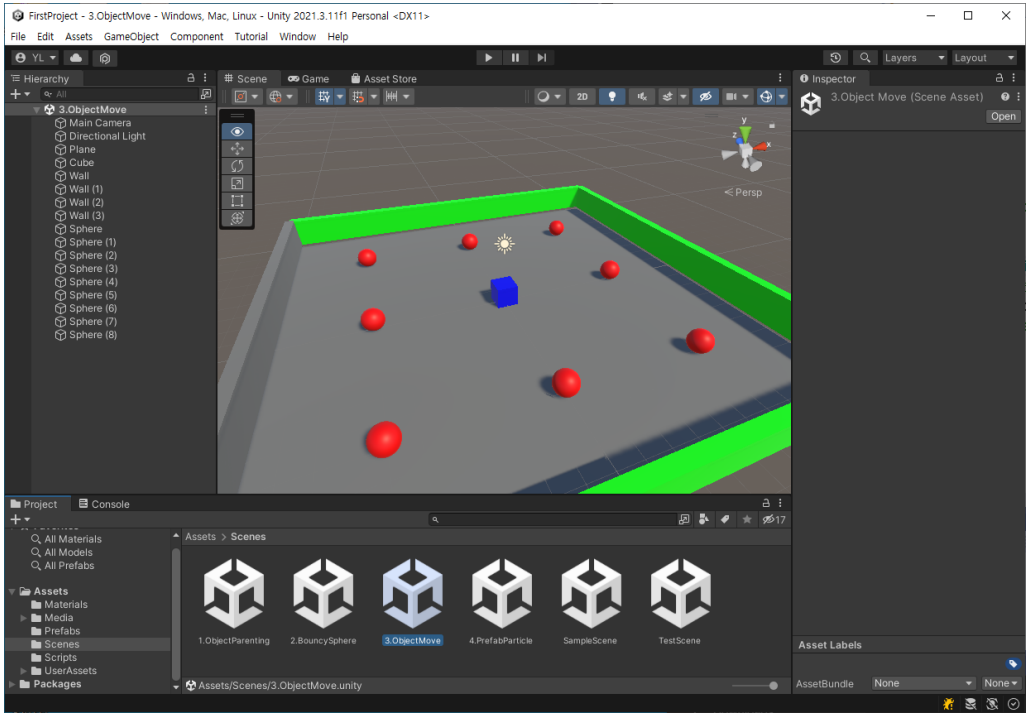
참조

- 비쥬얼 스튜디오에 생성된 클래스
  - public class // class 선언부. Class명과 스크립트의 파일명이 동일해야 함
  - void start() // 스크립트가 실행될 때 한번 호출되는 콜백 함수
  - void update() // 게임의 프레임마다 자동으로 호출되는 콜백 함수



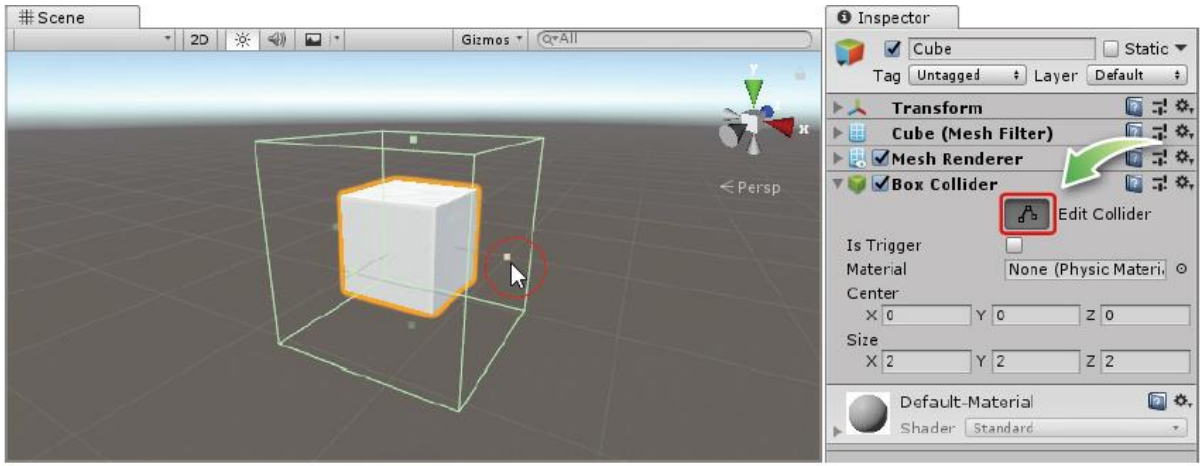
오브젝트 이동 및 충돌

- 게임오브젝트 이동
  - transform의 Position에 직접 접근하여 (값을) 설정해주는 방법
  - transform의 Translate() 함수를 이용하는 방법
  - RigidBody를 설정한 후, velocity(속도) 값을 설정해주는 방법 (물리 시뮬레이션에서 사용)



- 충돌 - Collider (콜라이더)
  - 충돌 : 2개의 오브젝트가 서로 접촉하는 상태
  - 콜라이더 : 오브젝트의 충돌을 판정하기 위한 영역
  - 두 물체의 콜라이더가 서로 접촉하면, 충돌 이벤트가 발생
  - 충돌에 반응하려는 오브젝트는 반드시 콜라이더가 있어야 함

  (콜라이더가 없는 오브젝트는 충돌이 발생하지 않으므로, 물체가 이동할 때 다른 오브젝트를 뚫고 지나감)
  - 메뉴 [Component – Physics - ...] 에서 추가

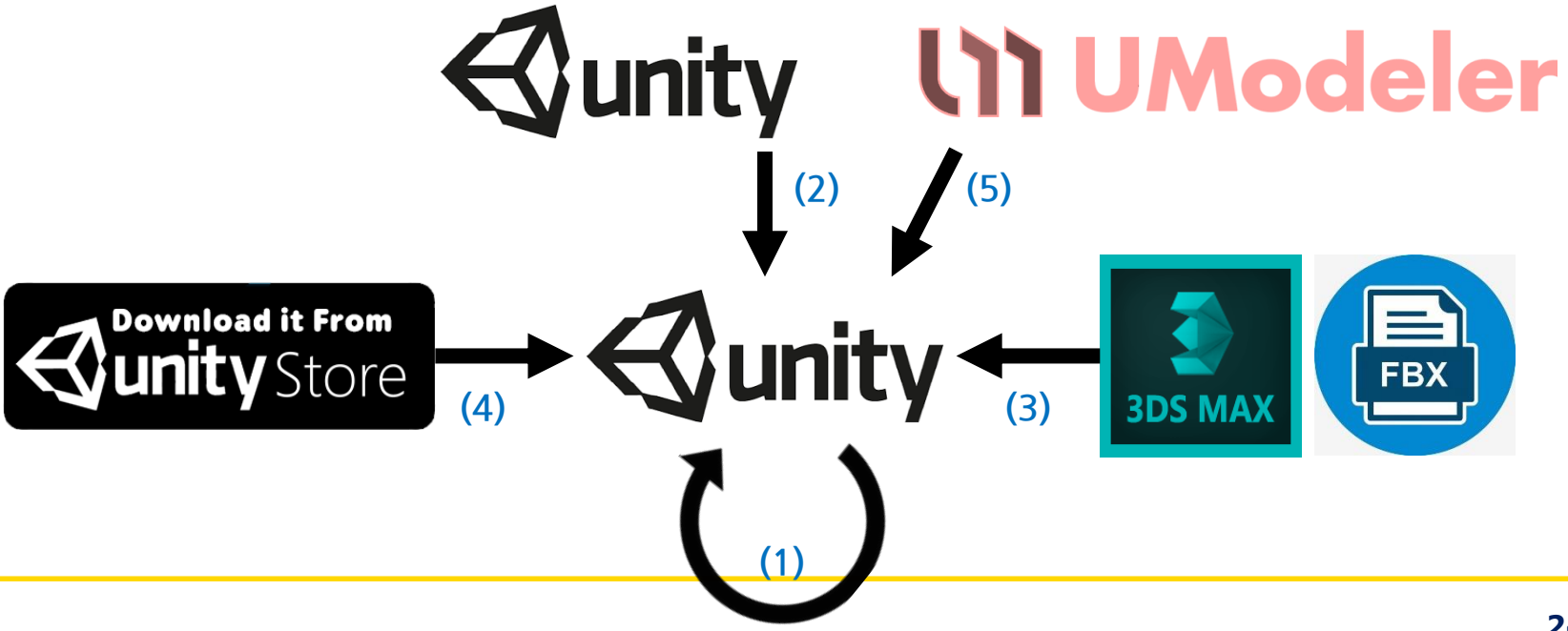


1. 이동 스크립트를 작성하여 Cube 오브젝트에 드래그
2. 게임 실행 모드에서 키보드의 화살표 키를 입력으로 오브젝트 이동 여부를 확인
3. Wall 과 Sphere 오브젝트를 다수 개 생성하여 배치
4. Cube 오브젝트에 리지드바드를 추가하고 이동하면서 Wall 과 Sphere 오브젝트와 충돌 실험
   (실행 모드에서 Wall/Sphere 오브젝트 Collider 속성의 IsTrigger 를 체크하면서 충돌과 관통을 실험)
5. 충돌 스크립트를 작성하여 Cube 오브젝트에 드래그 (또는 이동스크립트에 추가)
   (충돌을 인지하면, 충돌 오브젝트를 제거 – 예로, 아이템을 획득하면 획득 아이템은 화면에서 제거)
6. 충돌 오브젝트에 Tag 를 부여하여 (특정) 대상만을 구분하여 제거함

```
public class ObjectMove : MonoBehaviour
{
    float moveSpeed = 10f;  // 이동속도

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        // 현재 프레임에서 이동할 거리
        float amount = moveSpeed * Time.deltaTime;

        // 오브젝트의 전방으로 이동
        // (1) Position 이동
        //transform.position += new Vector3(0, 0, 1);
        // (2) Translate() 함수 이동
        // 현재 위치에서 지정한 방향과 거리만큼 오브젝트를 이동시키는 함수
        //transform.Translate(Vector3.forward * moveAmount);    // 상대좌표

        // 키보드 입력
        if (Input.GetButtonDown("Jump")) { Debug.Log("Jump, Space bar is pressed.."); }

        // 키보드로부터 오브젝트 이동
        // 전후(Vertical) 좌우(Horizontal) 이동키를 받음
        float vert = Input.GetAxis("Vertical");
        float horz = Input.GetAxis("Horizontal");
        // 입력 값 만큼 오브젝트 이동
        transform.Translate(new Vector3(horz, 0, vert) * moveAmount);
    }
}
```

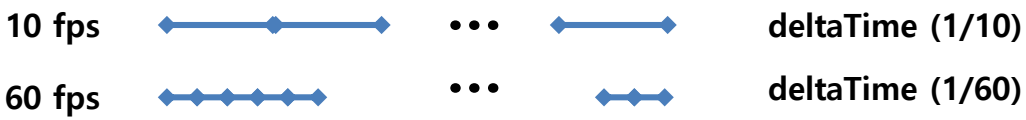

```
public class ObjectMoveAndCollide : MonoBehaviour
{
    // 이 부분에 ObjectMove 스크립트와 동일하게 (여기서는 공간부족으로 생략함)

    // 충돌 처리
    private void OnTriggerEnter(Collider other) {
        Debug.Log("OnTriggerEnter event is occurred.." + other.gameObject.tag);
    }
    private void OnCollisionEnter(Collision collision) {
        Debug.Log("OnCollisionEnter event is occurred..." + collision.gameObject.tag);

        // 충돌이 발생한 오브젝트가 아이템이면
        if (collision.gameObject.tag == "Item")
        {
            // (1) 오브젝트 제거
            //Destroy(collision.gameObject);
            // (2) 오브젝트를 화면에서 안보이게
            collision.gameObject.SetActive(false);
            // (3) 오브젝트에서는 충돌 판정만 하고,
            // 충돌에 따른 세부처리는 충돌 대상 오브젝트에서 처리하는 것이 효율적임.
            // (충돌 대상 오브젝트는 collision.gameObject)
            collision.gameObject.SendMessage("DestroySelf", transform.position);
        }
    }

}
```

- 게임 오브젝트 생성 방법
  1. Unity 에서 제공하는 3D 오브젝트를 활용하여 생성
  2. 다른 프로젝트에서 임포트
  3. 외부 3D 모델을 임포트
  4. Asset Store 에서 다운로드
  5. UModeler (3D 모델링을 위한 Unity 플러그인) 에서 제작

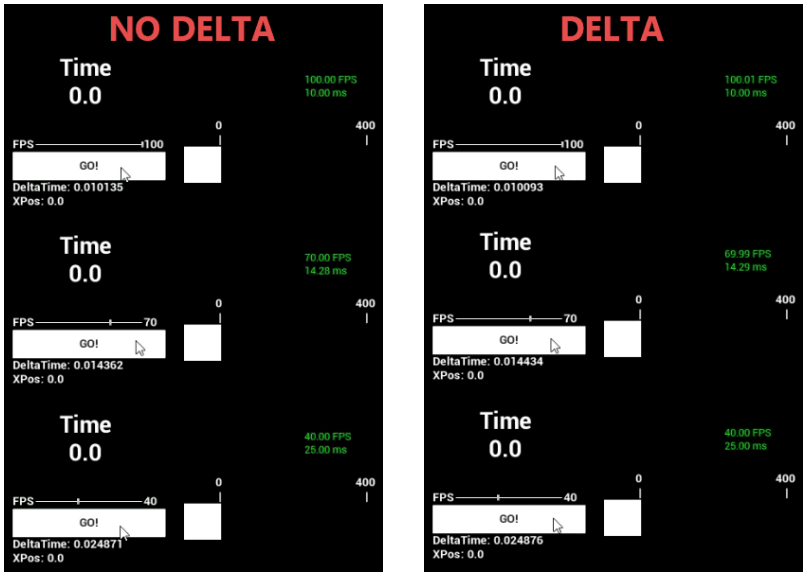


– 프레임 : FPS 와 Delta Time

- 게임은 일련의 절차를 무한정 수행하는 루프의 연속
- deltaTime 을 곱하므로, 동일한 유닛의 이동을 지원하도록



- 늦은 FPS 에서는 큰 deltaTime 을 곱해주고, 빠른 FPS 에서는 작은 deltaTime을 곱해준다
- 이를 통해 (FPS가 빠르건 느리건 상관없이) 동일한 이동 거리를 갈 수 있도록 조정



- **키보드 입력**

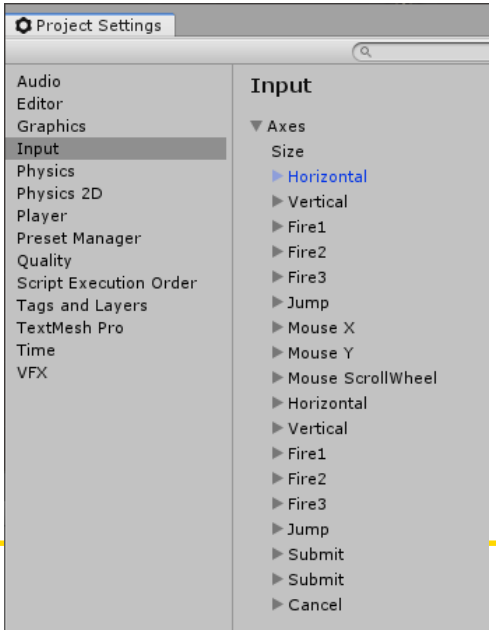
  (1) 미리 설정한 키 조합(버튼) 사용, (2) 특정 키 입력

  - 미리 설정한 키 조합(버튼) 사용
    - [Edit – Project Setting – Input] 에서 미리 설정된 키 조합 사용 (수정가능)
    - GetButton(), GetButtonDown(), GetButtonUp()
    - 버튼으로 설정된 이름으로 동작

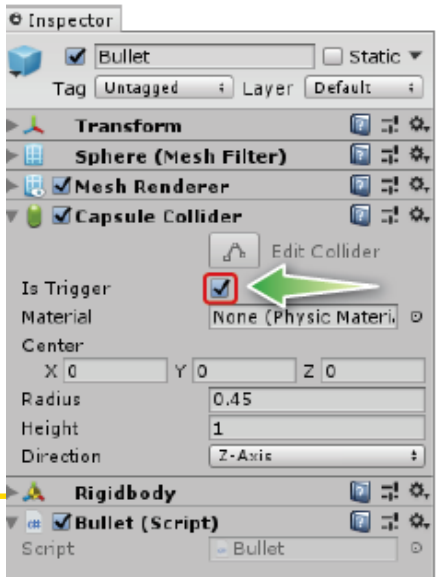
    예) if (Input.GetButton("Fire1")) {...}

  - 특정 키 입력
    - GetKey(), GetKeyDown(), GetKeyUp()
    - 키보드의 키-코드를 인식하여 동작

    예) if (Input.GetKeyDown(RightArrow)) {...}



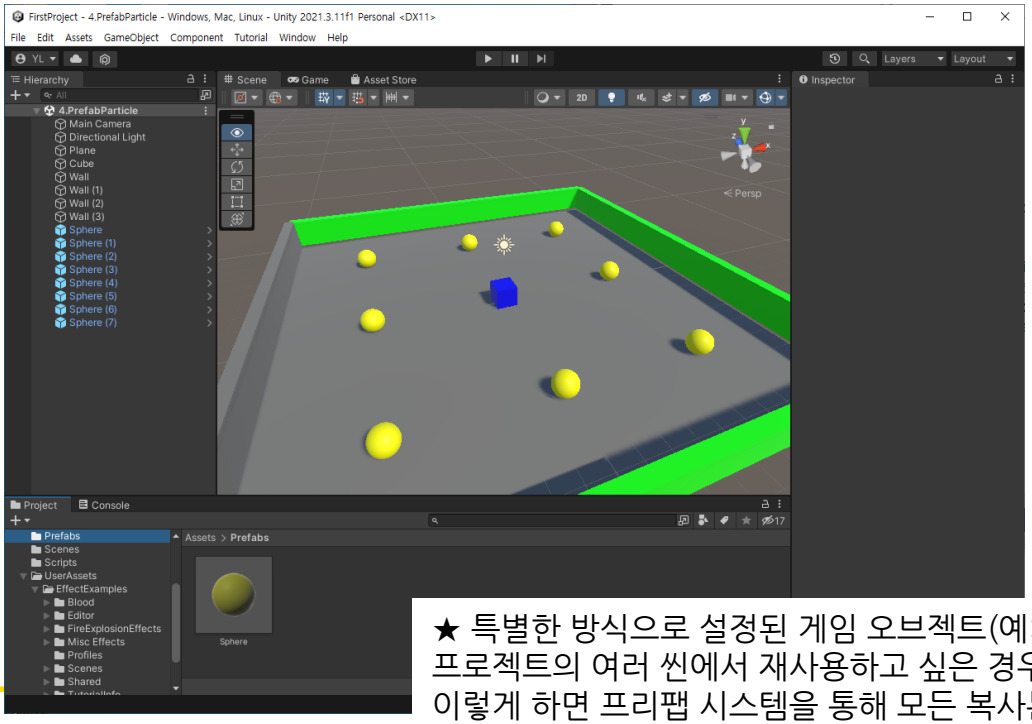
- **충돌** 판정 및 처리
  - 충돌이 일어나기 위해서는, 두 GameObject 가 모두 Collider 를 가지고 있어야 하며, 둘 중 하나는 Rigidbody 를 가지고 있어야 함 (움직이는 오브젝트가 Rigidbody를 갖는 게 일반적)
  - 충돌이 발생할 때, 충격을 가하는 오브젝트가 반사되느냐 관통하는냐에 따라 발생하는 이벤트가 달라지며, 반사 및 관통 여부는 Collider의 Is Trigger 옵션으로 설정
    - Is Trigger ON : 오브젝트 관통, Trigger 이벤트 발생
    - Is Trigger OFF : 오브젝트에서 반사, Collision 이벤트 발생
  - 적어도 하나가 Trigger On 이면 Trigger 이벤트가, 둘 다 Off 면 Collision 이벤트가 발생
  - 충돌 이벤트의 매개변수의 type
    - OnTrigger … (Collider other) Trigger 이벤트
    - OnCollision … (Collision collision) Collision 이벤트



프리팹 활용 및 파티클 적용

- 프리팹 : 재사용 가능한 애셋 (씬에 새로운 프리팹 인스턴스를 만들기 위한 템플릿)
  - 하이러리키의 오브젝트를 프로젝트 브라우저로 드래그하여 프리팹을 만듬
- 파티클 : 매우 작은 이미지나 매쉬를 시뮬레이션하고 랜더링하여 시각효과를 생성
  - 불, 연기, 액체 등과 같은 동적 오브젝트를 구현할 때 유용 (디자인 영역)

(1) 메뉴 [.. – Effects - Particle System] 클릭하여 추가, (2) Particle Asset 을 임포트



★ 특별한 방식으로 설정된 게임 오브젝트(예: NPC, 장면의 소품 또는 일부)를 씬의 여러 장소 또는 프로젝트의 여러 씬에서 재사용하고 싶은 경우 해당 게임 오브젝트를 프리팹으로 변환해야 함. 이렇게 하면 프리팹 시스템을 통해 모든 복사본을 자동으로 동기화할 수 있기 때문에 게임 오브젝트를 단순히 복사해서 붙여 넣는 것보다 더 효율적임

1. Sphere 오브젝트를 프리팹으로 만듬
2. (씬에 존재하는) Sphere 오브젝트는 제거하고 모두 프리팹으로 대체함

   ★ 프리팹을 사용하는 장점

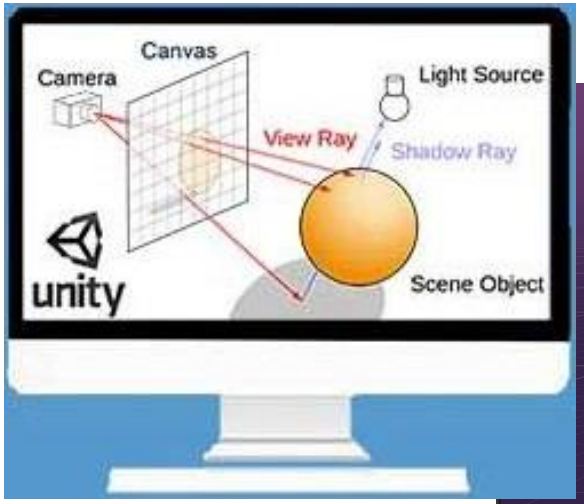
   프리팹을 수정하는 경우 vs. 씬에 있는 오브젝트를 수정하는 경우
3. 충돌 시, SendMessage()를 이용하여 충돌 처리 과정을 충돌 오브젝트의 스크립트로 분리
4. 충돌 오브젝트(아이템)을 제거하면서 파티클 효과를 추가

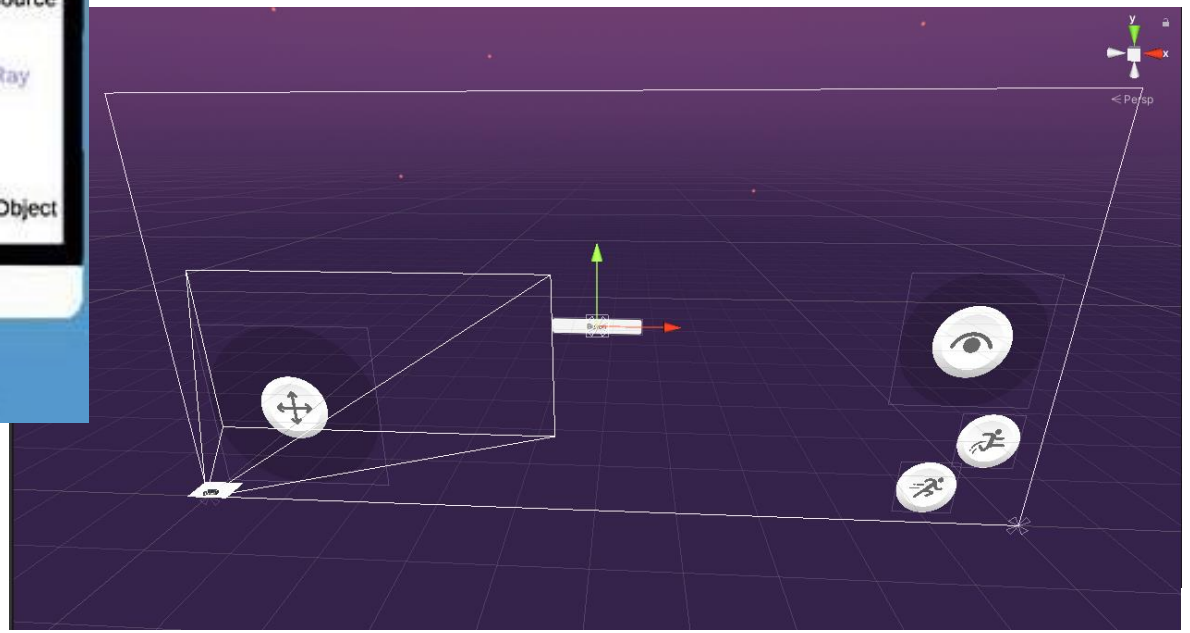
   파티클은 AssetStore 에서 무료 애셋을 검색하여 사용 (Unity Particle Pack 5.x)
5. 파티클이 계속 유지되고 있을 경우, 파티클 오브젝트 제거


```
public class ObjectDestroyWithParticle : MonoBehaviour
{
    // 오브젝트 제거 파티클
    public Transform explosion;

    // Start is called before the first frame update
    void Start() {
        //Debug.Log("ObjectDestroyWithParticle");
    }

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

    // Object Destroy
    void DestroySelf(Vector3 pos)
    {
        // 오브젝트 제거 시점에서 파티클 실행
        //Instantiate(explosion, pos, Quaternion.identity);
        // 파티클 오브젝트 생성 후, 삭제되지 않는다면 이렇게
        Transform parti = Instantiate(explosion, pos, Quaternion.identity);
        // Delay(지연)을 줘서 오브젝트 제거(destroy)
        Destroy(parti.gameObject, 2);

        //Destroy(gameObject);
        // 오브젝트 제거 대신에, 오브젝트가 화면에 안보이게
        gameObject.SetActive(false);

        // 아이템 획득 수(score)를 올려줌
        GameManager.AddResource();
    }
}
```

UI 활용 - 캔버스 다루기

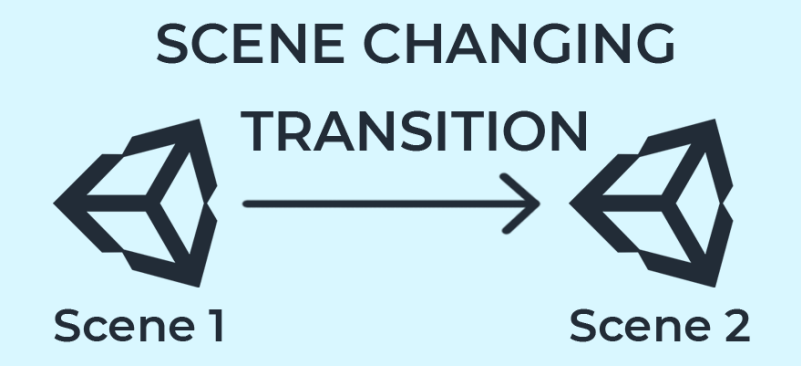
- 캔버스 (Canvas)
  - 캔버스는 모든 UI 오브젝트를 배치하기 위한 영역 (버튼, 텍스트 등의 모든 UI 요소는 Canvas 안에 위치해야 함 = UI 오브젝트는 반드시 캔버스의 자식이어야 함)
  - 메뉴 [GameObject – UI – Canvas]를 클릭하여 캔버스를 추가

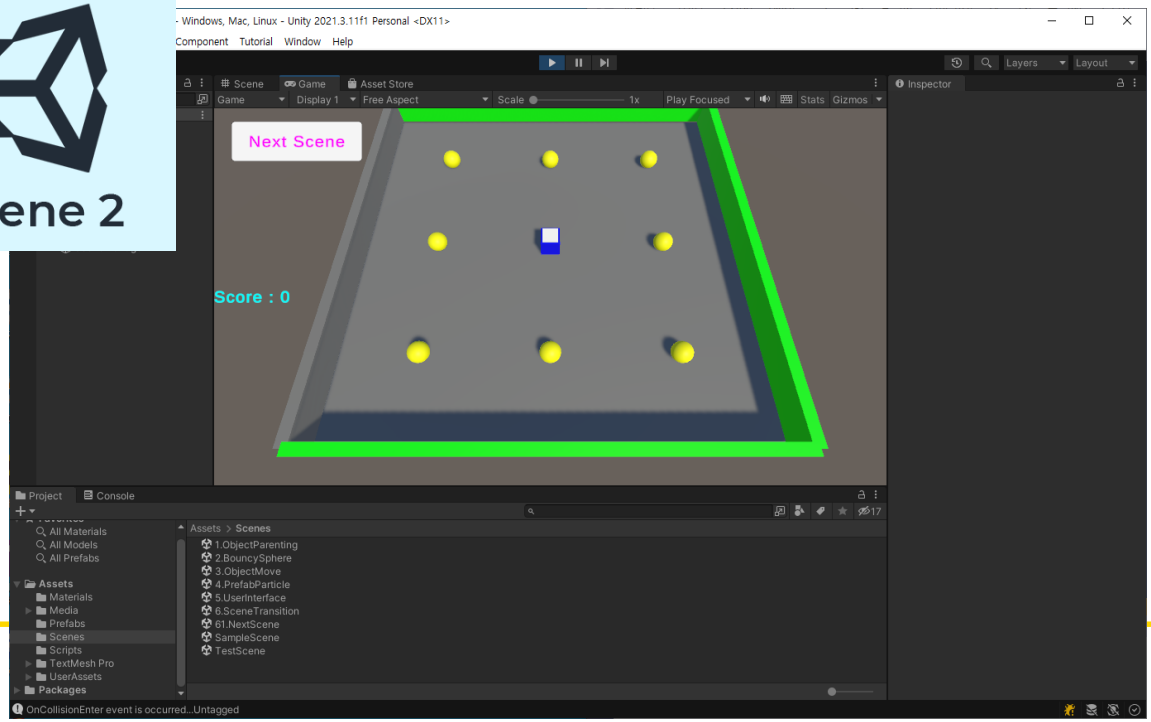




Scene Transition

- Scene 전환
  - SceneManager 를 위해, using UnityEngine.SceneManagement 필요
  - 전환하는 씬은 Building Setting 에 씬으로 등록되어 있어야 함
  - 씬 전환 시에, (필요한) 데이터 전달이 가능 (DontDestroyOnLoad() 사용)





1. Text UI 오브젝트를 씬에 추가
2. GameManager 오브젝트를 생성하여 (게임 내에서) 전체를 다루는 스크립트를 작성
3. 특정 오브젝트와 충돌이 발생할 때 (아이템을 획득할 때) 점수를 부여하여 캔버스의 Text 에 표시
4. Button UI 오브젝트를 씬에 추가
5. 버튼을 클릭하여 다른 씬으로 전환

```
using UnityEngine.UI;
using TMPro;

public class GameManager : MonoBehaviour
{
    //public Text score;
    // unity version이 올라감에 따라, UI text를 쓰기 보다는 TextMeshPro를 사용하는게 나을 듯
    public TextMeshProUGUI resourceText;

    //private int resource;
    [HideInInspector]
    public static int resource;

    // Start is called before the first frame update
    void Start() {
        // public으로 선언해서 (따로) GetComponent<>()를 할 필요 없음
        //resourceText = GetComponent<TextMeshProUGUI>();
        resource = 0;
    }

    // Update is called once per frame
    void Update() {
        resourceText.text = "Score : " + resource.ToString();
    }

    public int GetResource() {
        return resource;
    }

    public static void AddResource(int addCount) {
        resource += addCount;
    }

    public static void AddResource() {
        resource++;
    }
}
```

```
using UnityEngine.SceneManagement;

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

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

    public void NextScene() {
        SceneManager.LoadScene("61.NextScene");
    }
}
```