유니티 에서 사용되는 게임 요소의 구성

오브젝트와 컴포넌트

오브젝트 : <Secne 뷰><Hierarchy 뷰>에서 선택, 컴포넌트를 담는 그릇

컴포넌트 : 오브젝트에 부착할 스크립트

유니티의 물리엔진

**콜라이더(Collinder)**

물체 간의 충돌을 계산하는 영역, 실제 물리성질이 적용되는 구역(형태)

**리지드바디(Rigid Body)**

물리적인 성질을 적용하기 위한 컴포넌트

**-트리거 인 콜라이더(Trigger in Collinder)**

트리거는 콜라이더 내부의 개념으로, 콜라이더와 다르게 물리적 충돌처리는 하지 않지만 충돌한 경우 충돌했다는 값을 받아오는 역할을 함. (충돌을 해도 뚫고 지나가면 충돌을 해도 무슨 소용인가 그리고 또 왜 있나 ? 왜 어렵게 저런 걸  만들어 두었나 짜증이 나시는 분도 있을텐데 유니티는 처음부터 게임을 만들기 위해 만들어진 엔진입니다 .트리거는 FPS에서 쓰입니다 .

총알에 트리거를 적용하면 좋죠 원래 총알이 사람 몸에 맞으면 어찌 되죠? 뚫고 나가죠?

물론 한 명만 죽일 수 있게 프로그래밍 된 게임도 있지만 요새는 뒤에 있는 사람도 뚫고 나간 총알이 맞추는 게임도 있습니다. 총알 같은 게 사람 몸 맞고 튕겨 나오면 이상하겠죠?  무슨 핸콕도 아니고? 그래서 뚫고 나가되 총알을 맞았다는 것만 알려주는 역할을 하게 됩니다.

**[출처]** [[8강] unity3d 여러가지 충돌 인식 방법 Trigger, Collision, Raycast](http://blog.naver.com/squall_town/30116949074)|**작성자** [squall\_town](http://blog.naver.com/squall_town))

카메라 위치 조정

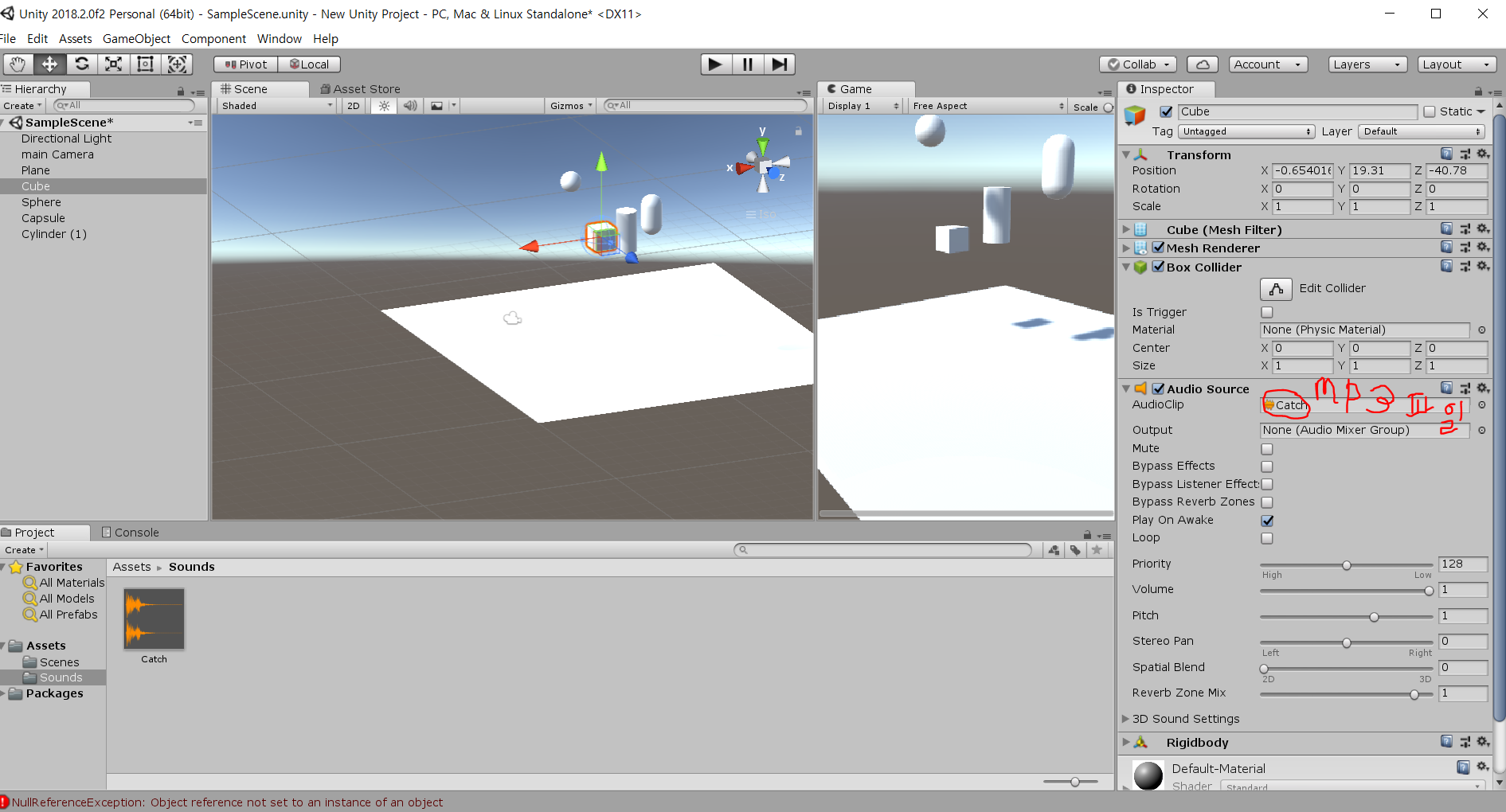
스크린화면에서 원하는 앵글을 잡은 후 <Ctrl+Shift+F>를 누른다.

사운드

리스너(Listener)

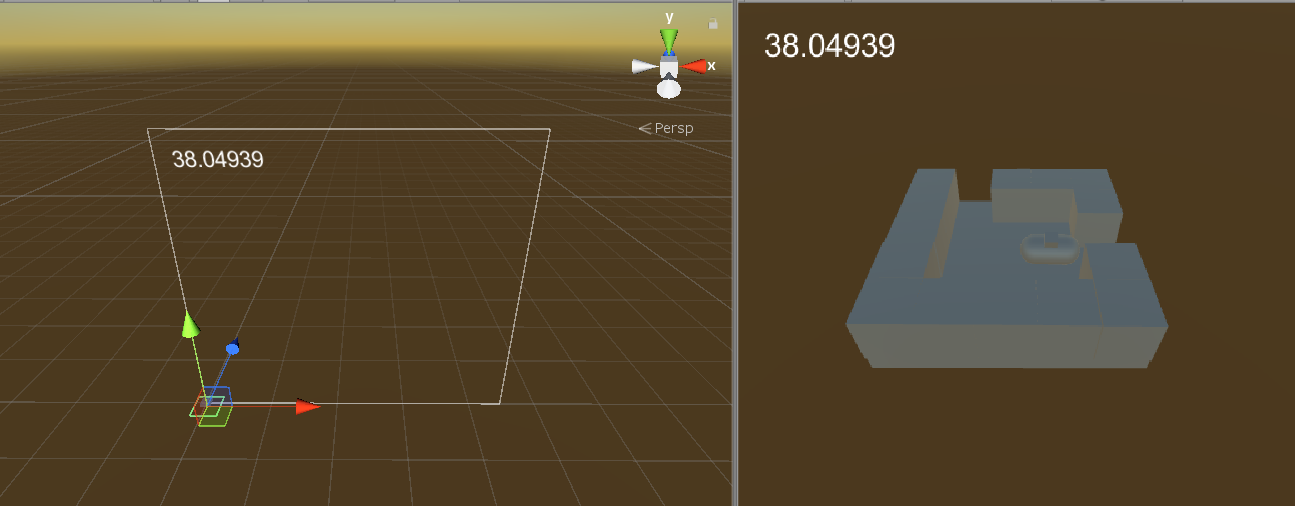
리스너는 Camera오브젝트의 Inspecter에 Audio Listener로 찾을 수 있는데, 이는 사운드를 듣는 역할을 하며 한Secene에 한 개만 존재해야 한다.

소스(Source)

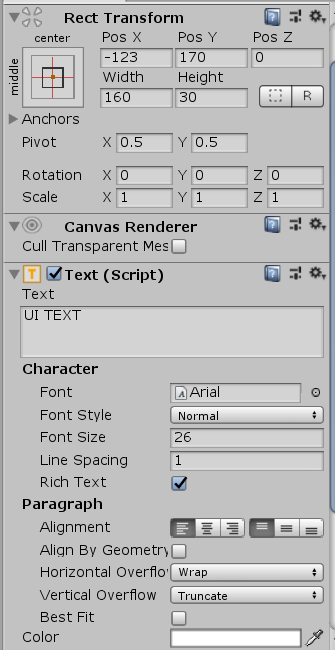
AudioSource는 오브젝트에 부착할 수 있는 컴포넌트로 소리를 발생시킨다.  자세한 내용 p72

**UI**

<http://blog.naver.com/PostView.nhn?blogId=gold_metal&logNo=220523485312&parentCategoryNo=&categoryNo=40&viewDate=&isShowPopularPosts=false&from=postView> 이 블로그 참고

**1-UI Text 추가하기 (아래는 완성본)**

GameObject>UI>Text를 이용해 간단히 추가할 수 있다. Canvas에 다양한 GUI요소(Text,Button)을 추가해 화면에 나타낼 수 있다.

옆은 Text의 인스펙터 창이다.

Rect Transform은 unity Game화면상의Text배치를

이는 택스트의 크기, 색, 깊이 등을 설정하는 것이다.

인스펙터창을 통해 GUIText의 원하는 위치를 잡고

다음은 스크립트로 넘어간다.

나는 게임 전체에 적용될 Manager스크립트에 작성했는데 블로그처럼 Text에 직접 스크립트를 부착하는것도 좋다. 개인적으로 블로그의 방법이 더 좋아보인다

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI; **//이것을 꼭 추가해야 스크립트에서 UI를 다룰 수 있다**

public class Manager : MonoBehaviour {

public int Count;//자동 초기화

public float \_time;

bool End;

public Text Text\_time; public변수로 해야한다.

private void OnTriggerEnter(Collider other)

{

if (other.tag == "Box") //테그를 이용하면 이렇게 스크립트 안에서 쉽게 호출가능함

{

Count = Count + 1;

}

if (Count >= 1)

{

End = true;

}

}

void Start () {

Text\_time = GetComponent<Text>();

}

void Update () {

if (End == false)

{

\_time += Time.deltaTime;

}

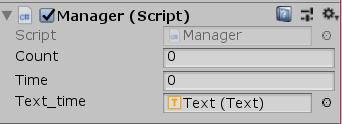
Text\_time.text = \_time.ToString();

//\_time을 string타입 변수로 변환한 것은 GUI Text는 string타입의 변수만 출력할 수 있기 때문이다.

}

}

그 후 위 스크립트의 경우는 Text스크립트에 직접 추가한 것이 아니므로 연결을 해줘야 한다.

public변수로 지정하면 옆과같이 에디터창에서 편집 할 수 있게 되는데

Text 변수옆 작은 동그라미를 눌러

지정하면UI Text에서 시간이 나타나게 된다.  
뿐만 아니라 버튼이나 다양한 GUI도 추가할 수 있다. To be continue…….

변수와 그 종류

Int, float, bool, string, GameObject, Transform, Vector2, Vector3, enum

변수의 사용권한을 설정하면 스크립트창이 아닌 유니티 에디터화면에서도 수정을 할 수 있고 다른 스크립트와 접근이 가능해진다. 사용권한을 선언하는 방식은 변수선언시 앞에 public을 쓰면 된다. 예)public int A;

함수와 그 종류

함수에는 시점함수, 일반함수, 기본제공함수가 있다.

***기본제공함수***는 c언어의 printf(“%d”,1); 과 같은 printf로 선언할 필요가 없다.

유니티에서 제공하는 기본제공함수는 매우 많다.

유니티 라이브러리 사이트에서 확인해야한다

***시점함수***는 선언만 해놓으면 특정 시점에 자동으로 실행된다.

유니티에서 제공하는 시점함수로는

Start() 스크립트가 시작할 때, 한번 자동실행

Update() 스크립트가 활성화되어있는 동안, 매프레임마다 반복실행

OnEnable() 이 스크립트가 부착된 오브젝트가 활성화되는 순간마다 한 번씩 실행

OnDisable() 이 스크립트가 부착된 오브젝트가 비활성화되는 순간마다 한 번씩 실행

OnTriggerEnter() Trigger속성의 중돌체가 특정 오브젝트와 부딪힌 순간, 한 번씩 실행

OnTriggerStay() Trigger속성의 중돌체가 특정 오브젝트와 겹쳐있는 동안, 매프래임마다 반복실행

OnTriggerExit() Trigger속성의 중돌체가 자신으로부터 특정 오브젝트가 충돌영역을 벗어나는 순간 한 번씩 실행

등 많이 있으나 여기서 줄인다. 홈페이지 들어가 확인할 것

***일반함수***는 직접 선언 후 start나 다른 시점함수에 의해 함수가 호출되면 실행되는 함수로 우리가 알고있는 함수의 사용법과 비슷할 것이다.

**실습1)Basic\_Move**

**1 - 화살표키로 오브젝트 이동시키기**

함수설명*(이동함수)*

*transform.position = new Vector3(float, float, float);* - 절대좌표를 기준으로 위치시킴(순간이동)

*transform.Translate(new Vector3(float, float, float));* - 상대좌표를 기준으로 위치시킴(매 프레임마다 값만큼 이동)

출처: <http://tenlie10.tistory.com/71>[게임 개발자 블로그]

*Transform.Translate(Vector3이동할 방향과 속도, ~~Space relativeTo = Space.Self~~)*라는 함수는 IrelativeTo가 생략되거나 Space.Self로 설정되면 이동은 변형의 로컬 축에 상대적으로 적용되고, relativeTo = Space.World 인 경우 움직임이 월드 좌표계를 기준으로 적용된다. 이 함수는 변환 방향과 거리로 변형을 이동한다

변수설명

Time.deltaTime 이는 전프레임과 현프레임 간의 시간 간격을 의미하는 변수이다. 성능이 좋은 기기는 간격이 좁고, 성능이 나쁜 기기는 간격이 넓을 것이다. Updata와 같이 매프래임마다 실행되는 함수는 기기의 성능에 따라 다르게 실행되는 격인데, 이를 예방하기위해 주로 쓰인다.

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BasicMove : MonoBehaviour {

public int Speed;//public으로 사용권한을 지정함으로써 에디터창에서 Speed조절 가능

void Start () {

}

void Update () {

if (Input.GetKey(KeyCode.LeftArrow))

{

transform.Translate(Vector3.left \* Speed \* Time.deltaTime);

}

if (Input.GetKey(KeyCode.RightArrow))

{

transform.Translate(Vector3.right \* Speed \* Time.deltaTime);

}

if (Input.GetKey(KeyCode.UpArrow))

{

transform.Translate(Vector3.forward \* Speed \* Time.deltaTime);

}

if (Input.GetKey(KeyCode.DownArrow))

{

transform.Translate(Vector3.back \* Speed \* Time.deltaTime);

}

}

}

**2 – 회전을 이용해 오브젝트가 전방을 주시하게 만들기p141**

함수설명*(회전함수)*

Input.GetAxisRaw((string axisName)

조이스틱이나 키보드로 입력받은 값을 -1~1(이산적인 입력이라면 -1, 0, 1)로 반환

Quaternion.LookRotation(Vector3 forward, Vector3 upwards = Vector3.up)

지정된 전방 및 위쪽 방향으로 회전을 한다.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class BasicMove : MonoBehaviour {

public float Speed;//public으로 사용권한을 지정함으로써 에디터창에서 Speed조절 가능

Vector3 lookDirection;

void Start () {

}

void Update () {

if (Input.GetKey(KeyCode.LeftArrow)||

Input.GetKey(KeyCode.RightArrow)||

Input.GetKey(KeyCode.UpArrow)||

Input.GetKey(KeyCode.DownArrow))

{

float a = Input.GetAxisRaw("Vertical");

float b = Input.GetAxisRaw("Horizontal");

lookDirection = a \* Vector3.forward + b \* Vector3.right;**//씬기한 문제풀이 기법**

transform.rotation = Quaternion.LookRotation(lookDirection);

//방향 회전하고(입력방향으로 전방회전)

transform.Translate(Vector3.forward \* Speed \* Time.deltaTime); //전방으로 방향이동

}

}

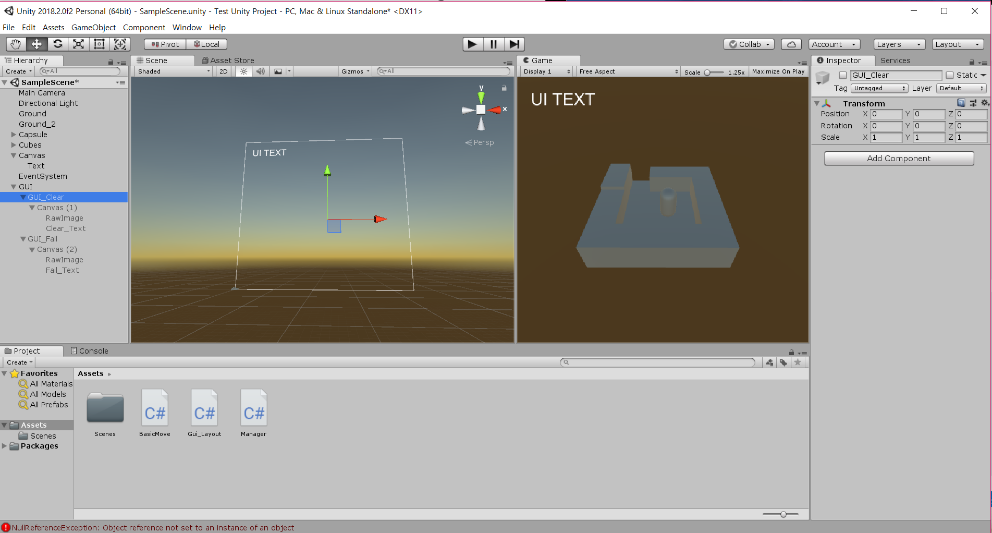
}

Tip 오브젝트 배치시 키보드v키를 누르면 모서리와 모서리를 편하게 맞출 수 있다

Tag(태그)는 하나 이상의 GameObjects에 연결하는 단어입니다. 예를 들어, “Player”와 “Enemy” 태그를 플레이어 제어 가능한 캐릭터와 비 플레이어 제어 가능한 캐릭터로 각각 정의하거나, “Collectable” 태그를 플레이어가Scene에서 회수 가능한 아이템으로 정의하거나 하며 이어갑니다. *태그의 목적은 명확하게 GameObject를 스크립팅에서 확인하는 것입니다.* 스크립트 코드를 작성할 때 찾으려는 태그에 속하는 게임 오브젝트를 찾을 수 있습니다. 이것은 GameObject.FindWithTag() 함수를 사용하여 실현할 수 있습니다.

**3-Fail과 Clear창 띄우기**

함수설명

GameObject.SetActive(true or false)

인스펙터창 상단을 보면 선택된 오브젝트의 활성화를 관리 할 수 있는 태그박스가 있다.

setactive라는 함수를 이용해 스크립트에서도 게임오브젝트의 활성화를 조절 할 수 있다.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class Manager : MonoBehaviour {

public int Count;//자동 초기화

public float \_time;

bool End;

public Text Text\_time;

public GameObject ClearGUI;

public GameObject FailGUI;

private void OnTriggerEnter(Collider other)

{

if (other.tag == "Box")//이상하게 씀

{

Count = Count + 1;

if(Count >=16)

{

End = true;

ClearGUI.SetActive(true);

//박스가 16개 모두 gameoverZone을 통과했을 때 End=True가되고 ClearGUI는 활성화 된다.

}

}

else if (other.tag == "Player")

{

End = true;

FailGUI.SetActive(true);

//플레이어가 gameoverZone을 통과했을 때 End=True가되고 Fail GUI는 활성화 된다.

}

}

void Start () {

Text\_time = GetComponent<Text>();

}

void Update () {

if (End == false)

{

\_time += Time.deltaTime;

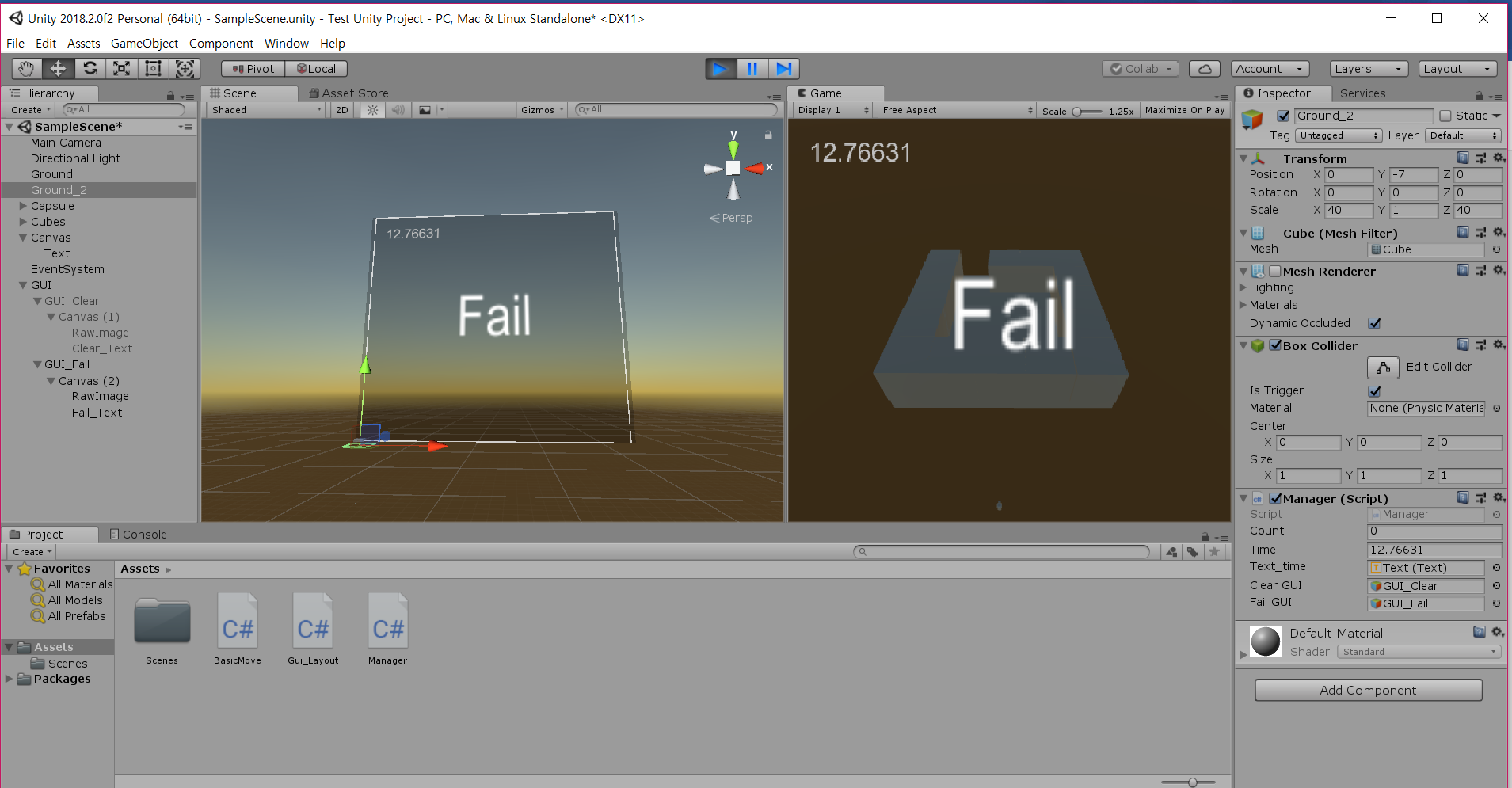
}

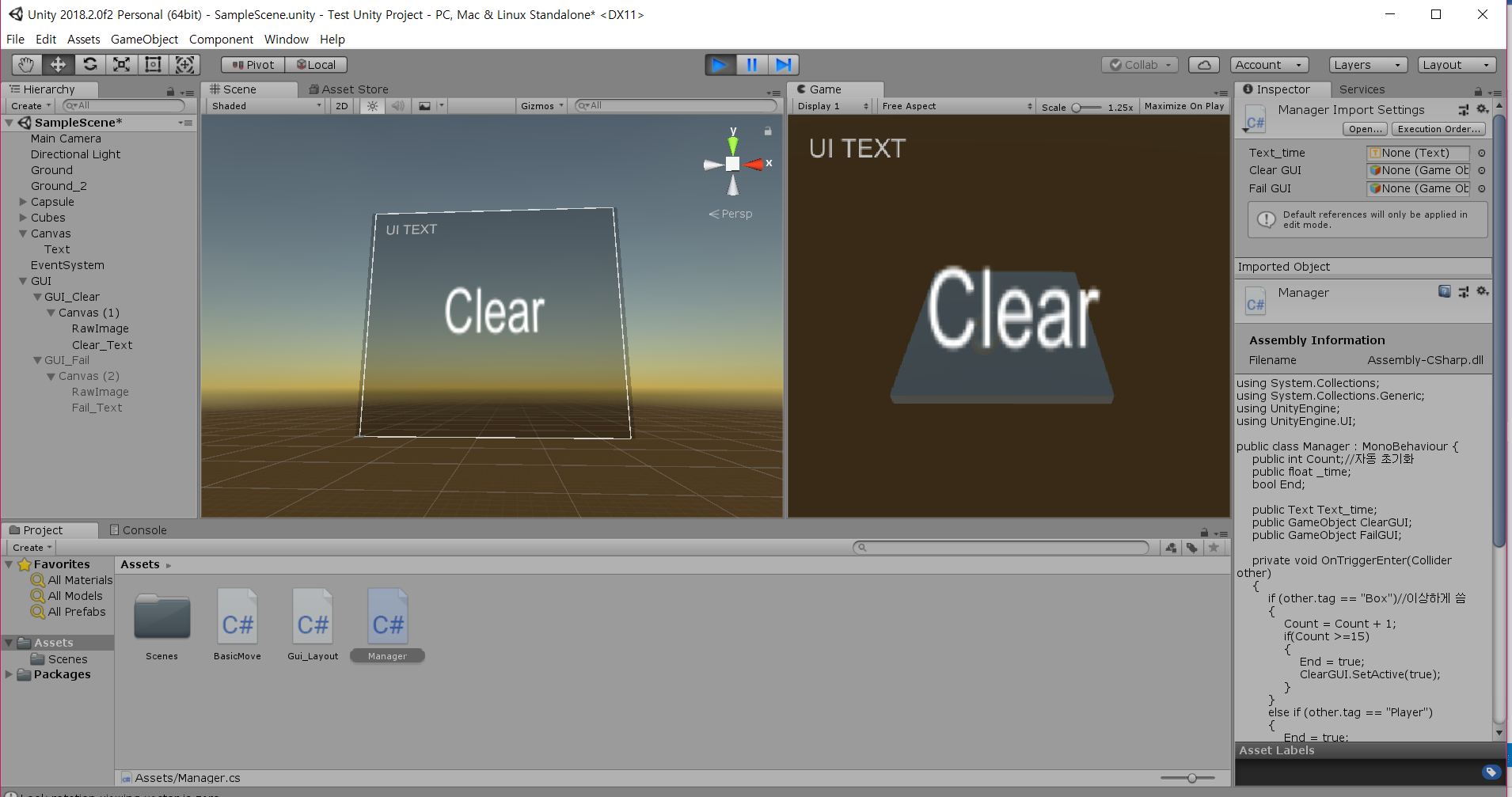
Text\_time.text = \_time.ToString();

}

}

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*실행화면\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*





**4-레이저 쏘기**

Raycast는 원하는 방향으로 원하는 길이의 레이저를 쏘고 레이저가 닿은 오브젝트의 이름과 그 거리를 읽어오는 기능을 한다.

함수설명(raycast)

Debug.DrawRay(레이저를 발사할 위치, 발사할 거리, 발사할 색)

Debug는 실제 게임화면에서는 출력되지 않지만 에디터창이나 콘솔창, Scene뷰에서 확인이 가능한 함수이다. 즉 위의 함수는 레이(레이저)를 발사할 위치와 거리만큼 Scene뷰에서 보여주는 함수이다.

Physics.Raycast(transform.position, transform.forward, out hit, Range)

Raycast를 발사해 충돌이 일어났는지(true), 아닌지(false)에 대한 결과값을 받아오고 True일 때 중괄호의 실행문을 실행한다. Out hit의 hit은 raycast로부터 정보를 얻기위해 사용되는 구조체를 나타낸다.

rigidbody.AddForceAtPosition(transform.forward \* Power, hit.point)

force와 position이라는 파라미터가 필요하다. Position에 force를 적용하고 그 결과로 오브젝트에 토크와 힘을 적용하는 함수이다. 함수의 파라미터(변수)로 사용된 hit.point는 raycastHit.point로 ray가 충돌한충돌체(hit)의 충돌지점을 나타내며 월드 좌표를 사용한다.

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Shot : MonoBehaviour

{

public float Power = 200f;

public float Range = 10f;

private void Update()

{

Debug.DrawRay(transform.position, transform.forward \* Range, Color.red);

if (Input.GetKeyDown(KeyCode.Space))

{

RaycastHit hit;

if (Physics.Raycast(transform.position, transform.forward, out hit, Range))

{

if (hit.collider.gameObject.tag == "Box")//hit의 정보중 충돌체정보.gameobject.tag

{

Debug.Log(hit.collider.name);

hit.rigidbody.AddForceAtPosition(transform.forward \* Power, hit.point);

}

}

}

}  
}