Using UnityEditor

[MenuItem(“../..”) : 유니티 메뉴바 편집 기능

EditorWindow : 에디터창(hierarchy, inspector 같은)을 만들 때 상속 받는 Class

Public static EditorWindow GetWindow(Type t, bool utility = false, string title = null, bool focus = true) : 해당하는 창이 있으면 그 창을 보여주고 없으면 새로 만들어서 보여줌

EditorGUILayout : 에디터 창에 넣을 수 있는 여러가지 레이아웃을 미리 만들어 놓은 Class

Public static EditorGUILayout.TextField : 글을 쓸 수 있는 공간을 만드는 함수, 이름, 정보를 받을 변수, GUI스타일, 옵션을 선택적으로 사용할 수 있다.

EditorApplication : 어플리케이션에 관한 내용을 총괄하는 클래스

EditorApplication.isPlaying : 어플리케이션이 동작하고 있는지(Scene이 작동중인지, 유니티 내에서 play중인지)를 bool로 반환 동작하면 true

EditorUtility.DisplayDialogComplx : 제목, 내용, 수락, 거절, 취소의 의미를 가진 문자열을 사용한다. 흔히 보는 ~~하시겠습니까? 할 때 만들어지는 창이다.

EditorSceneManage.SaveScene : scene을 저장할 때 사용한다.

EditorSceneManage.OpenScene : 원하는 Scene을 원하는 모드로 가져오는 함수

AssetDatabase : 여러 에셋에 접근하여 관리하는 클래스

AssetDatabase.FindAssets : AssetDatabase를 통해서 에셋을 찾는 함수

AssetDatabase.CopyAsset : A에 있는 에셋을 복사하여 B에 붙여넣는 함수

AssetDatabase.Refresh : 변경된 에셋이 있을 경우 그 에셋을 AssetDatabase가 인식하기 위해 모든 에셋을 다시한번 읽어들이는 함수

EditorBuildSettingsScene : BuildSettings창의 정보를 가진 클래스이다.

EditorBuildSettings : BuildSettings창을 수정할 수 있는 기능을 가진 클래스이다.

Using UnityEngine

[RequireComponent(typeof(ComponentName))] : ComponentName에 해당하는 컴포넌트를 게임오브젝트에 자동으로 추가해준다. 실행됐을 때만 추가하는 것이 아닌 이 명령문?을 가진 컴포넌트를 게임객체에 추가하는 순간 컴포넌트들이 같이 추가된다.

Mathf : 수학 함수를 모아 놓은 구조체

Mathf.Approximately : 두 값을 비교하여 충분히 유사할 경우 true를 반환한다. (어느정도 유사해야 하는 것 인지는 잘 모르겠음, 내 생각으로는 9.99999… 와 10이 유사한 정도의 유사성일 것이라 생각함)

Vector2 : 2차원 Vector값을 가지고 있는 구조체

Vector2.sqrMagnitude : 벡터의 제곱에 해당하는 값을 가진 property, 이게 왜 함수가 아닌지 잘 모르겠음, 자주 사용해서 성능적으로 이득을 보기 위해서 이렇게 되어있나?

MonoBehaviour : Unity 스크립트(Component)가 파생되는 기본 클래스이다.

MonoBehaviour.Reset() : 컴포넌트를 게임객체에 처음 추가시켰을 때 또는 컴포넌트를 처음의 상태로 바꾸고 싶을 때 사용 하는 함수이다.

[contextMenu] : 상황에 맞는 메뉴에 명령을 추가 할 수 있다는데 어떤 것인지 모르겠다.

[SelectionBase] : 선택하는 상황에서 우선적으로 선택하라는 방향표 같은 역할을 하는 것 같다.

[AddComponentMenu] : 사용자가 만든 스크립트를 유니티에 등록시킨다. Inspector 창의 addComponent에 추가된 것을 확인할 수 있다

[Range(min, max)] : 이후에 만들 변수의 최소와 최대치를 정해주는 방법

[ExecuteInEditMode] : 해당하는 스크립트가 게임 실행중이 아니더라도 주요 함수를 실행시키는 방법, 화면에 필터링을 걸 때, 실행시키지 않으면 확인할 수 없는 것을 보완해줌

Using ShaderLab

Shader “name” : 쉐이더의 이름

SubShader : 실제 쉐이더 구현부분, 여러 개의 SubShader가 있을 수 있으며 각각 성능에 따라 사용하는 등의 사용처가 나뉠 떄 사용된다.

Pass : 오브젝트의 렌더링의 단위를 의미, SubShader 안의 Pass의 수만큼 오브젝트가 렌더링 된다.

Material : 물체 표면의 재질을 정의 하는 방식

* Diffuse(r, g, b, a) : 재질의 Diffuse 색상을 r, g, b로 지정
* Ambient(r, g, b, a) : 위와 같음

Lighting On : 해당 쉐이더에서 고정 기능 파이프라인의 표준 라이팅을 적용한다는 의미

Properties : 쉐이더 외부로부터 개발자에 의해 설정될 수 있는 파라메터 정의

* \_MainTex ("Texture", 2D) = "white" {} :Texture를 파라메터화 시킨 것
* \_MyColor(“Main Color”, COLOR) = (0, 0, 1, 1) : 색상을 파라메터화 시킨 것

SetTexture[\_MainTex] : []안의 변수(텍스처)를 텍스쳐로 사용하는 것

* 그냥 이렇게 써주면 값이 이상하기 때문에(명암이 없다던가..) {Combine texture \* primary} 같은 것을 사용한다.
* 맨뒤에 double을 써주면 결과값을 2배해준다.

D

Using Animator

Readonly int m\_HashForwardSpeed = Animator.StringToHash(“ForwardSpeed”) : 애니메이터에 있는 ForwardSpeed 파라메터를 m\_HashForwardSpeed에 저장하는 함수