**공개SW프로젝트 8조 진행상황 #1**

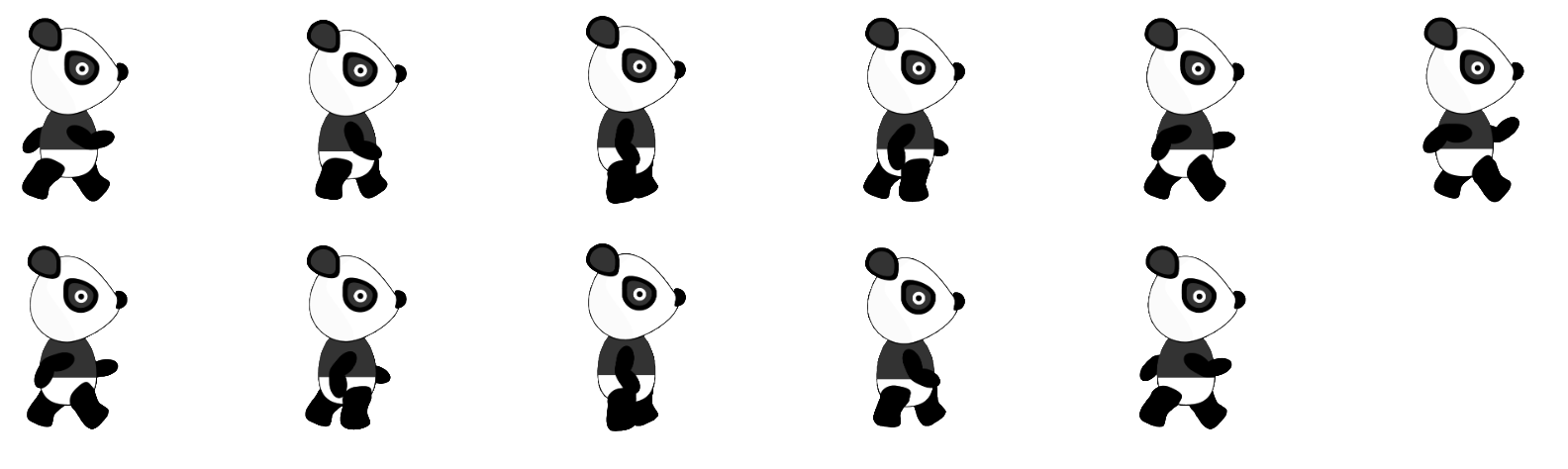
[목차]

[게임 디자인]

|  |  |
| --- | --- |
| 장르 | 2D 디펜스게임 |
| 적 | 배고픈 판다들, 설탕을 훔치려 함 |
| 플레이어 | 컵케이크 타워를 이용해 판다들을 물리쳐야 함. 컵케이크 타워에서는 여러가지 발사체를 쏴서 판다를 공격함. |
| 게임 룰 | 여러 종류의 컵케이크 타워가 있고, 맵의 곳곳에 배치 가능. 레벨마다 판다가 따라가야 하는 패스가 있고, 종착점에는 플레이어가 지켜야 하는 슈거캐슬이 있음. 판다가 설탕을 너무 많이 훔쳐서 슈거미터가 0이 되면 게임오버. 판다를 모두 없애면 승리. 판다를 사냥할 때 마다 플레이어는 일정량의 캔디를 얻음. 플레이어는 캔디를 사용해 새로운 컵케이크 타워를 구매하고 업그레이드하여 더욱 강력해질 수 있음. |

[그래픽 리소스]

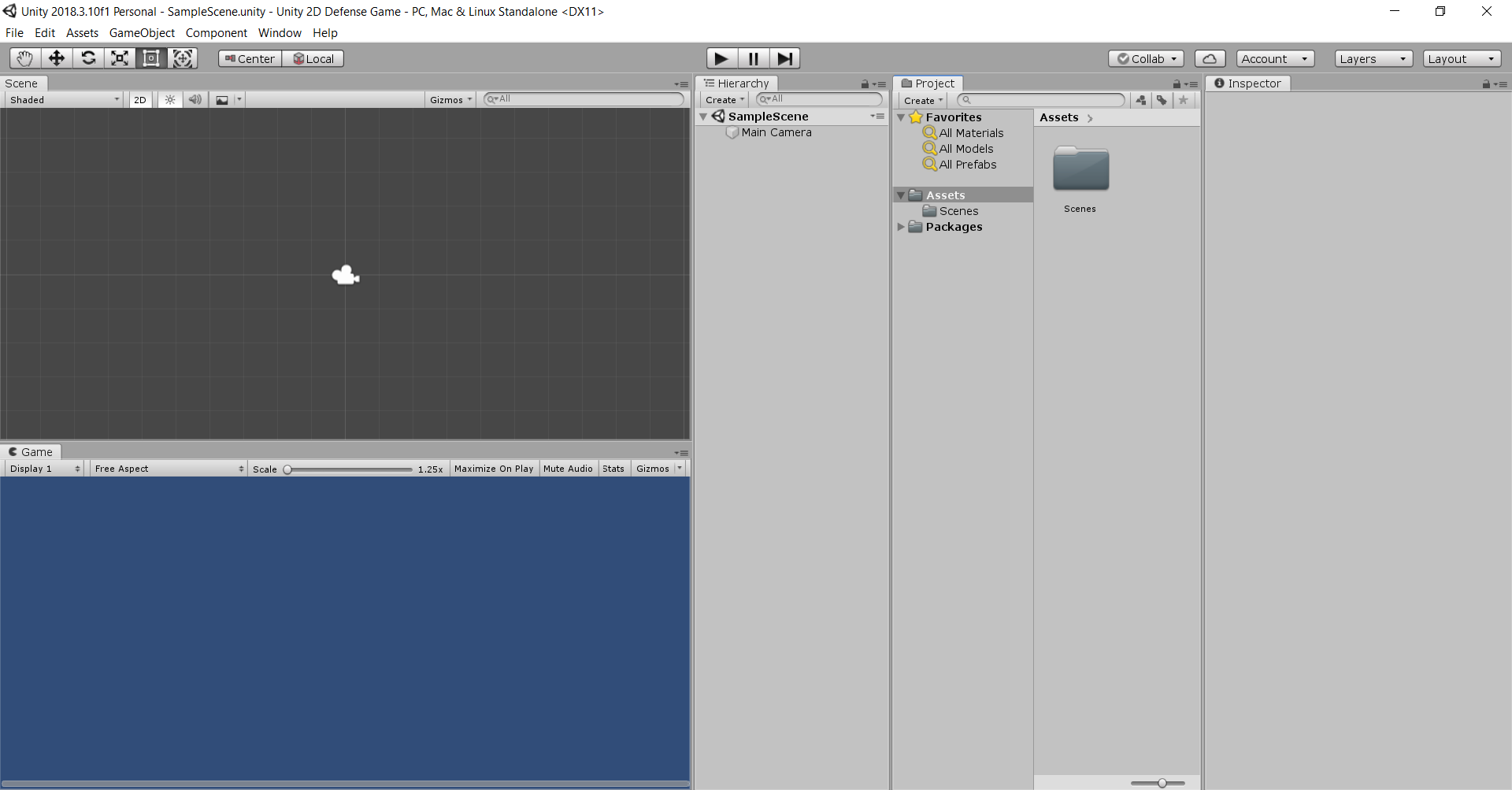
참고문헌의 출판사 홈페이지 (<http://acornpub.co.kr/book/start-unity5-2d-game>)에서 그래픽 리소스를 다운받음. 개인용 및 상업용 프로젝트에 사용 가능한 리소스임. 추후 그래픽 변경 가능.

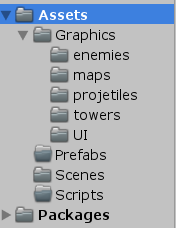
▲컵케이크 타워, 게임 맵, 악당 판다 이미지 소스

.[프로젝트 생성]

<Unity 2D Defense Game>이라는 이름의 프로젝트 생성. 유니티 버전 2018.3.10f1 Personal Edition 사용.



▲ 프로젝트 생성 직후의 화면



▲ Assets 폴더의 하위 폴더 내용. Graphics폴더의 각 하위 폴더에는 다운로드받은 그래픽 리소스들을 종류별로 분리하여 저장하였다.

프로젝트 asset(게임 제작에 사용되는 리소스들)들의 관리를 위해 폴더를 만들어 정리함. (폴더는 필요에 따라 추가 가능)

|  |  |
| --- | --- |
| 폴더 이름 | 내용 |
| Graphics | 게임 내 사용할 그래픽 소스들을 저장. 그래픽의 종류에 따라 다시 하위폴더들로 나누어 저장 |
| Prefabs | Game object의 템플릿이 되는 prefab들을 저장 |
| Scenes | 게임 내의 각 scene들을 저장. 게임 레벨마다 다른 scene사용 |
| Scripts | 게임 내에서 사용되는 C# 스크립트를 저장 |

[Sprite]

Sprite(스프라이트)는 유니티 2D게임의 기본요소. 이미지(image)로써 생각할 수 있으나 조금 다른 개념. 예를 들어 하나의 이미지 파일에는 여러 개의 sprite가 존재할 수 있음.

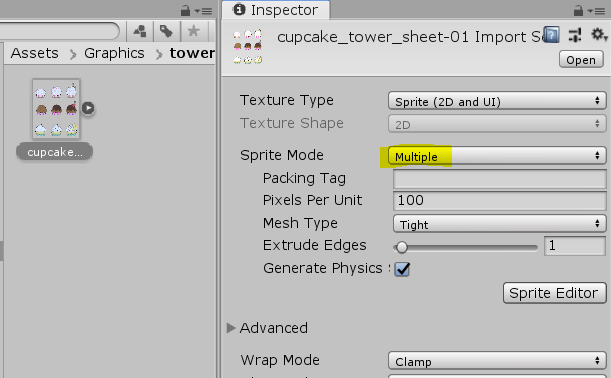
여러 개의 sprite를 가지고 있는 하나의 이미지 파일을 sprite sheet라고 함. Sprite sheet를 사용하는 이유는 게임 성능상의 이유와 그래픽 리소스의 체계적인 관리를 위해서임.



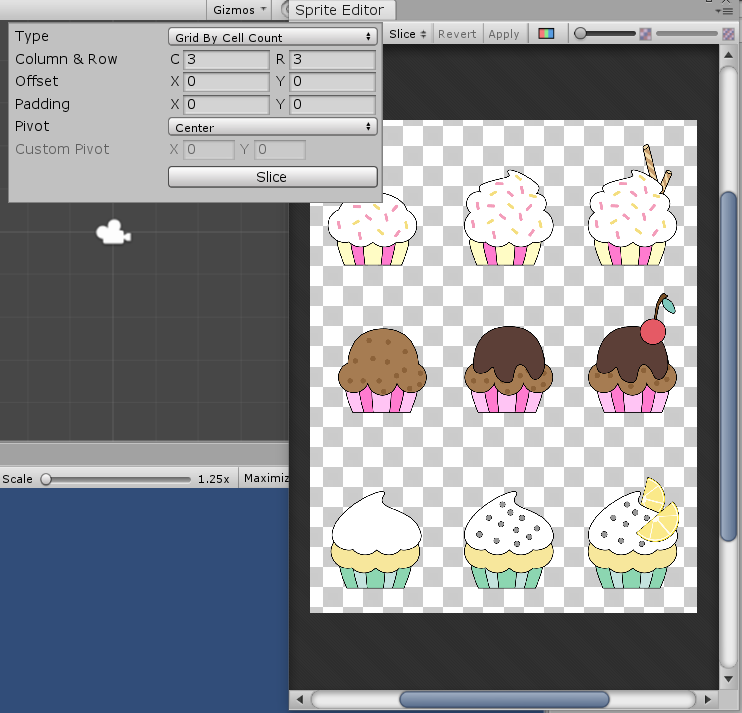
▲sprite sheet의 예제. 하나의 이미지파일에 여러 개의 sprite가 있다.

[Sprite Asset 준비하기]

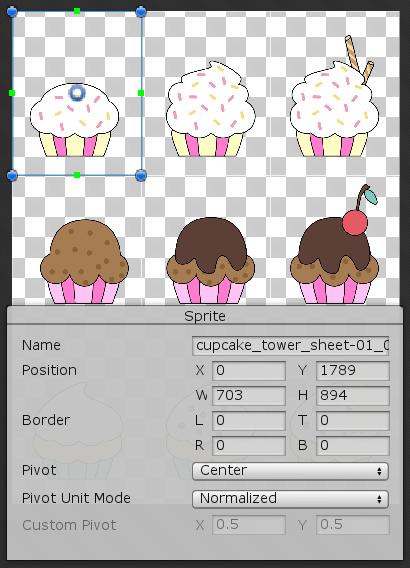
앞서 다운로드 받은 이미지 파일들을 sprite editor를 이용하여 sprite들로 만든다. 이렇게 만들어진 sprite들은 게임 내의 game object를 만드는 데에 사용된다.



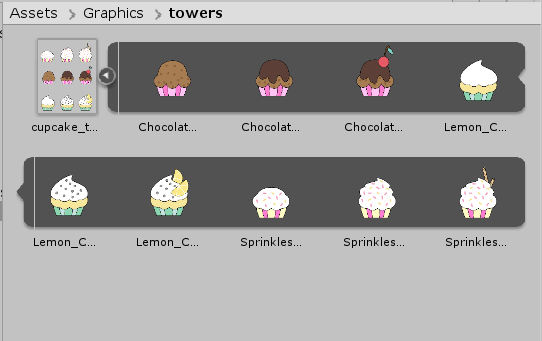
컵케이크 타워 sprite sheet의 sprite mode를 multiple로 설정한다. 이 설정의 의미는 이미지 파일에 여러 개(multiple)의 sprite가 존재하고 있다는 뜻이다. 설정을 바꾼 후 밑의 “Sprite Editor”버튼을 누른다.



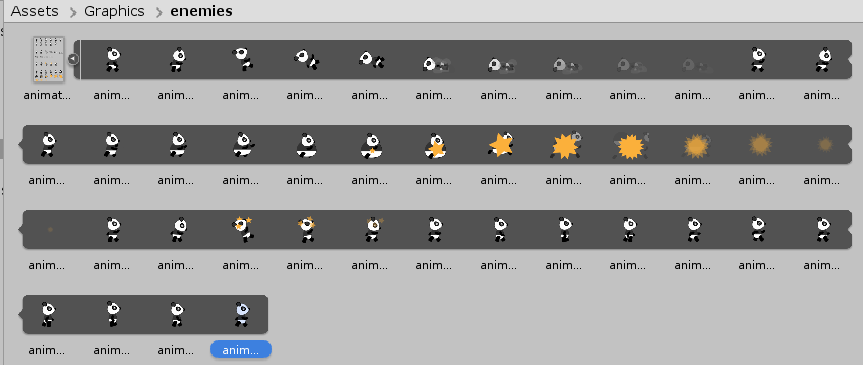
slice의 type을 Grid By Cell Count로 바꾸고, row와 column을 각각 3으로 지정한다. 이렇게 하면 이미지 전체를 3X3으로 slice하여 9개의 sprite를 생성할 수 있다.



9개로 slice된 각각의 sprite들의 이름을 지정하고 저장 해 준다.



Graphics 폴더를 확인하면 원래 하나의 sprite sheet에서 9개의 sprite들이 생성되어 저장되어있는 것을 확인할 수 있다.



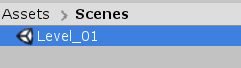


▲판다와 발사체 sprite sheet를 slice하여 각각의 개별 sprite로 만들었다

같은 방법으로 적 판다와 발사체(projectiles) sprite sheet도 slice해 준다. 참고로 Sprite sheet의 빈 공간은 sprite로 저장되지 않으니 Grid By Cell Count를 이용해 모두 동일한 사이즈로 slice해주면 된다.

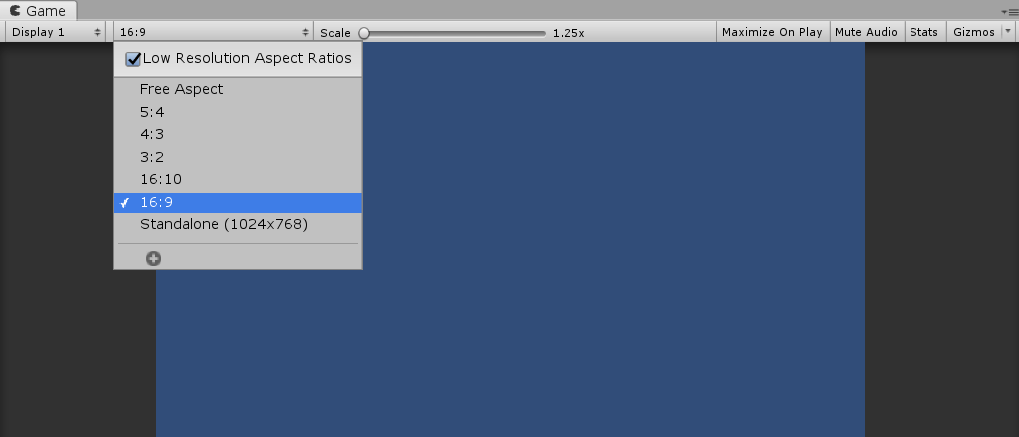
[씬(Scene)설정]

게임은 여러 레벨 스테이지로 구성될 수 있다. 이 때 하나의 레벨은 유니티에서 하나의 scene으로 만들어 관리한다. 현재 작업중인 scene은 게임의 첫 번째 레벨이 될 것이므로, level\_01이라는 이름으로 저장한다.



[화면 비율(aspect ratio) 설정]

보편적인 게임 화면 비율인 16:9로 게임 전체의 비율 세팅을 변경한다.



Game 탭에서 16:9비율을 선택한다.

[Z-buffering]

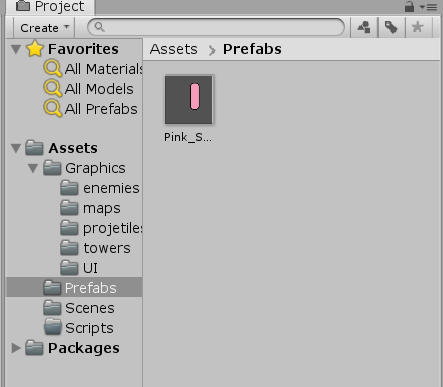
z-버퍼링이란 유니티에서 2D요소들의 렌더링 순서를 설정할 때 쓰인다. z값이 낮을수록 우선순위가 낮게 렌더링된다. 예를 들어, 맵 이미지는 캐릭터 이미지 뒤에 있어야 하므로 맵 이미지의 z-버퍼링 값을 더 높게 준다.(먼저 렌더링되어야 하므로) 우리 프로젝트의 main camera의 z값은 -10이다. 이것보다 낮은 값은 카메라의 뒤에 렌더링되므로 볼 수 없다.

우리 프로젝트에서 사용할 게임 오브젝트들은 종류에 따라 z값을 다음과 같이 설정하기로 한다.

|  |  |  |
| --- | --- | --- |
| 요소 | Z값 | 이유 |
| 맵 | 0 | 맵은 모든 요소들 중 가장 뒤에 있으므로 가장 먼저 렌더링 되어야 한다. |
| 적들 | -1 | 적들은 타워 뒤를 지나가는 것처럼 보여야 하며, 발사체도 적들을 때릴 때 보여야 하므로 적들은 타워와 발사체보다는 먼저 렌더링 되어야 한다. |
| 발사체 | -2 | 발사체는 타워보다 뒤에 놓인다. |
| 타워 | -3 | 타워는 플레이어의 가장 중요한 유닛이므로 다른 게임 요소들에 비해 가장 위에 보인다. |
| 맵 오버레이 | -9 | 맵 오버레이는 맵의 이미지 요소들 중 포어그라운드, 즉 가장 가까지 보여야 할 요소들을 가리킨다. 카메라에서 가장 가까운 요소이므로 -9값을 주었다. |
| 메인카메라 | -10 | 카메라의 z값은 디폴트값 그대로이다. |

[Prefab(프리팹)]

Prefab이란 이미 필요한 컴포넌트들로 조립되어 씬에 배치하기만 하면 되는 오브젝트이다. 씬에 배치하였을 때 비로소 Game Object가 된다. 객체지향 프로그래밍에서 클래스와 인스턴스와의 관계를 생각하면 쉽다. Prefab을 만드는 방법은 Hierarchy 메뉴에 있는 game object를 project 패널의 폴더 중 하나에 드래그 해 주면 된다.



▲ Prefab을 만들고, 씬 안에 prefab의 인스턴스들을 배치해 본 모습.

[Script(스크립트)]

유니티의 게임 오브젝트들에 C# 언어로 된 스크립트를 컴포넌트로서 추가할 수 있다. 스크립트를 만들기 위해서는 project패널에서 우클릭->Create->C# Script를 선택한다.

**스크립트 파일의 이름은 그 스크립트의 메인 클래스의 이름과 같아야 한다.(중요)**

|  |  |
| --- | --- |
| 스크립트 변수 종류 | 설명 |
| private | 스크립트 내에서만 사용 가능. |
| public | 모든 스크립트에서 액세스 가능. inspector창에서도 열람 및 설정이 가능. getter와 setter 함수를 만들어 관리하면 좋다. |

스크립트 생성 시 생성되는 클래스는 Monobehavior클래스에서 파생된 두 개의 메인 함수를 가진다. 다음은 그 함수들에 대한 설명이다.

|  |  |
| --- | --- |
| 메소드 이름 | 설명 |
| Start() | 게임이 시작되었을 때 처음이자 마지막으로 한 번만 호출된다. 변수초기화 등의 작업을 수행하는 용도로 쓰인다. |
| Update() | 게임의 매 frame마다 실행된다. 유니티로 만든 게임은 fps(frames per second)를 기준으로 동작하기 때문에 컴퓨터간 실질적인 연산 속도의 차이로 인해 게임의 동작이 달리지지 않음을 보장한다.  인풋 감지, 물리적 동작 등 리얼타임으로 연산되어야 하는 코드를 써 넣으면 된다. 그러나 Update함수는 정말 자주 호출되므로, 필요 이상으로 많은 코드를 넣으면 게임 성능이 저하되기 때문에(렉걸림 현상 등) 주의해야 한다. |

[발사체 스크립팅]

모든 발사체에 공통적으로 적용시킬 스크립트를 작성한다. 모든 초기 단계의 스크립트는 추후에 성능 개선을 위해 수정될 수 있다.