Unity 정리

2D adventure 튜토리얼은 2022.3 LTS로 진행됨.

Scene: 각 단계 혹은 챕터 별로 하나의 신들을 넣는 것이 시작하기 좋은 practice.

* project에서 Assets > Scenes 파일에, Create > Scene으로 씬 제작.

Sprite: 캐릭터 등의 item 제작 시 필요한 캐릭터의 사진.

* Project 창에서 Art > Sprites > Characters(이건 프로젝트마다 다를 듯) 에서 원하는 사진의 화살표를 눌러 제작된 sprite 선택. 2D 게임에서 사용되는 스프라이트인 경우, Sprite (2D and UI)를 선택 해준다(Scene view toolbar 에서도 2D mode 선택하기. 참고로 mode에 따라 카메라가 바뀌는 등 변화가 있다).
* 그냥 넣었을 시 작게 보일 수 있음. Sprite의 Pixel Per Unit(PPU? Unity 유닛 하나에 들어가는 pixel 수를 나타냄)을 고쳐주면 된다
* Inspector에선 선택한 item의 attribute를 확인 가능.
  + Transform: item의 위치정보.
    - Position: item이 어디에 놓이는지. 0,0으로 두면 화면 한가운데 위치함.
* Component 추가: inspector 창에서 가능한 선택지를 찾아서 골라도 되고, Script 등을 직접 드래그 앤 드롭 해도 된다.
* Sprite 크기 조정: grid를 선택하여 속성들을 보면 Cell Size를 확인 가능.
  + X와 Y값을 확인하면 셀 하나당 넓이와 높이가 각각 몇 Unit인지 알 수 있다.
  + 이를 바탕으로 적용하기 위해선 sprite 선택, Pixels Per Unit 속성을 확인해야 한다. 하나의 Unit당 들어가야 하는 Pixel의 수를 나타냄.
  + 즉 하단의 preview image에서 sprite의 size를 확인하여 속성을 수정하여 크기를 맞춰주면 된다. 64\*64 size인 경우 하나의 unit에 100이 들어갈 수 있는데 스프라이트는 64 뿐이기에 작아보이는 현상이 일어나는 것. PPU값을 64로 수정하거나, Grid의 Cell size를 0.64로 수정해주면 된다.
  + 만약 object의 자체 크기를 조절하는 경우, 보통 원하는 대로 늘리다 보면 사진 자체가 늘어나서 형태가 변하는 경우가 있다.
  + 이게 싫으면 Transform > Scale (0,0,0), Sprite Renderer > Draw Mode > Tiled로 설정
  + 저렇게 설정하면 sprite 설정이 mode에 적합하지 않다는 오류 생성. 이를 고치려면 Tile Mode > Adaptive, project의 sprite 원본 설정 Mesh Type > Full Rect.
* Tile과 다르게 다른 object들은 위치에 따라 무엇이 위에 rendering 되는지를 바꿔줘야 하는 경우가 있음.
  + Project > settings > Renderer2D, General > Transparency Sort Mode > Custom Axis로 설정해준다.
  + Transparency Sort Axis 값을 x=0 y=1 z=0로 설정 해준다. 그럼 이제 y축 상 더 위에 있는 object들이 뒤로 가게 된다.
  + 캐릭터 혹은 item 의 Sprite Sort Point를 pivot으로 맞춰준다. 그러면 pivot 값에 따라서 무엇이 앞에 있는지를 unity가 결정한다.
  + Pivot: sprite의 anchor. Rotation도 pivot을 중심으로 돌고, position도 pivot이 캐릭터의 위치에 온다(0,0으로 설정하고 pivot을 발에 두면 발이 0,0에 간다는 뜻)
    - Inspector 에서 pivot 설정 변경 가능.
    - 혹은 여러 개를 한번에 바꾸고 싶다면 sprite editor에서 가운데의 파란 동그라미를 조절하여 위치를 바꾸거나 settings에서 값을 변화하여 위치를 바꿀 수 있음.
* Rigidbody: unity가 물리 계산을 할 대상에게만 주는 컴포넌트.
  + 만약 캐릭터가 2d라면 rigidbody2d를 사용해야 한다.
  + 중력 작용을 원치 않는다면 Gravity Scale을 0으로 하기
  + 캐릭터가 움직이는 동안 회전하는 것을 원치 않는다면 Constraints > Freeze Rotation > Z.
  + 캐릭터의 움직임을 이것과 싱크로하기 위해선 Rigidbody2D rigidbody2d; 처럼 선언해주고
  + Vector2 position = (Vector2)rigidbody2d.position + move \* 3.0f \* Time.deltaTime; 처럼 rigidbody2d변수의 position값을 이용하여 변환시켜준 뒤
  + rigidbody2d.MovePosition(position); 처럼 rigidbody의 포지션을 다시 변환시켜주면 된다.
* Box Collider 2D: 물리 시스템에게 물체가 고체라는 것을 알려주기 위한 컴포넌트.
  + Edit Collider로 물체의 collider의 크기를 변형할 수 있다.

Prefab: 만약 게임 오브젝트의 설정을 바꿨고 해당 설정이 추가된 아이템을 저장해놓고 싶다면 prefab으로 저장해놓으면 된다.

* 보통 project 내부에 prefabs 폴더를 만들어 저장
* Item을 project창으로 드래그 드롭해서 저장 할 수 있다.
* Prefab editor로 object에 변화를 주게 되면 해당 prefab을 사용하는 모든 object에 변화가 적용된다.
* Prefab이 존재하는 object를 개별로 수정하게 되면 파란색 바가 옆에 생성되며 해당 파트가 prefab과 다르다는 것을 나타냄. 위의 override 기능을 이용하여 prefab과 다른점을 되돌리거나 prefab에 적용시킬 수 있음.

Script: item들의 활동을 정의하는 스크립트. C#을 사용한다.

* Assets에 Script를 모아두는 폴더 제작. Create > C# Script의 방식으로 제작한다.
* 수정은 컴퓨터에 설치되어 있는 IDE로 진행된다.
* Public 변수로 설정하게 되면 다른 코드 외에도 unity editor의 inspector에서도 수정이 가능하다.
* Functions:
  + Update: 게임이 진행되는 동한 규칙적으로 반복되는 instruction.
  + Debug.Log(variable): 콘솔 창에 변수를 띄우는 함수.
* Types, methods:
  + Vector2 type: 변수의 type. 2개의 수 값을 저장할 수 있기 때문에 2D 위치 좌표에 적합하다.
  + Transform.position: method. Item의 transform 컴포넌트(위의 inspector 확인할 것)에 저장되어 있는 X,Y 값을 가져옴(아마 Vector3 형식의 변수에 저장하게 되면 z도 저장할 듯).
  + 0.1f: 소수를 사용하기 위해선 뒤에 f를 붙여줘야 함(floating point)

Tilemap: 특정 sprite를 가리키는 tile들을 관리하는 컴포넌트.

* Hierarchy 에서 우클릭으로 2D Object > Tilemap > Rectangular 를 선택하면 Grid 와 Tilemap이 생성된다.
* Grid는 tile을 고르게 놓기 위한 gameobject. Scene에 자동 생성.
* Tilemap 은 Grid의 자식 gameobject로, 세팅하는 타일들을 렌더링 하기 위한 도구.
* 보통 tile들은 Assets > Art > Tiles 에 저장한다.
* 새로운 tile을 만들기 위해선 Create > 2D > Tiles > Rule Tile으로 만든다.
* 해당 tile의 sprite를 수정하기 위해선 Inspector 상에서 Select Sprite를 이용하여 지정하면 된다.
* Draw Order: 시각 요소들이 렌더링 되는 순서. 이 순서가 제대로 맞춰져 있지 않으면 캐릭터 위에 tile이 그려지는 등의 일이 일어날 수 있음.
  + Tilemap을 선택, Tilemap Renderer 컴포넌트의 Order in Layer 속성을 수정.
  + 낮을수록 더 먼저 그려진다. 즉 플레이어의 0 보다 낮아야 함.
* Tileset: tile의 palatte를 만드는데 사용할 수 있는 sprite가 여러 개 들어있는 이미지 파일. Sprite sheet라고도 부르며, 연관성이 있는 sprite를 모아둠으로써 Unity가 자원을 아끼면서 rendering이 가능하도록 하고 모둠 전체에 변화를 주기 용이하게 된다.
  + 자르기 전까지는 하나의 sprite 형식이다.
  + 자르려면 sprite mode를 multiple로 설정하고, 크기를 맞춰주고 apply를 해준다.
  + Sprite editor > slice > type 을 Grid By Cell Count 설정. Column 과 Row를 원하는 수로 설정. 그 다음 Slice를 선택하면 하나의 sprite가 여러 개로 잘린다.
  + 그 다음 tile palette에 넣고 일반 tile처럼 색칠용으로 쓰면 된다.
  + 9 – slicing: sprite를 늘릴 때, 경계선 부분은 늘어나지 않도록 하는 방법.
    - Mesh Type > Full Rect 설정.
    - Sprite Editor에서 경계선 설정.
    - 원하는 방법에 따라 Draw Mode를 Sliced 혹은 Tiled로 설정. Sliced는 가운데의 image가 알아서 늘어남. Tiled는 늘어나지 않고 여러 개가 생성.
    - Tiled 상태에서 tile mode를 바꿔서 또 늘어나는 방법을 좀 바꿀 수 있다.
* Collision: 특정 tile들에 collider를 추가하여 통과할 수 없는 등의 효과 넣기.
  + Tilemap GameObject 선택 > add component > Tilemap Collider 2D 추가.
  + Tile의 sprite가 모여 있는 폴더로 가서, collider를 없앨 tile들을 선택한 뒤, collider type을 none으로 설정하면 나머지는 collider이 존재해서 통과할 수 없다.
  + 모든 tile들이 따로 collider를 가지게 되면 계산이 복잡해 게임이 무거워진다. 이를 막기 위해
  + Composite Collider 2D를 tilemap gameobject에 추가하기. (자동으로 rigidbody 2d도 추가됨)
  + Tilemap collder 2d > Used by composite 설정, rigidbody 2d > body type > static 으로 설정. 이렇게 해주면 tilemap gameobject가 움직이지 않는 요소라는 것을 unity가 알고 계산을 최적화 할 수 있다.

Tile palatte: tile을 정리하고 tilemap의 grid에 적용하기 편하도록 도움이 되는 도구.

* Window > 2D > Tile Palatte를 선택하면 사용할 수 있다.
* 새로 만들기 위해선 No valid Palatte, Create New Tile Palatte> Create New Palatte를 하면 됨.
* 타일들이 보관되어 있는 폴더에서 타일을 드래그앤드롭으로 가져오면 된다.
* 팔레트에서 원하는 타일을 선택하여 grid에 놓는 행위를 painting이라 한다.
* Tile Palatte에서 원하는 것을 선택한 뒤, Brush 툴을 이용하여 칠하면 됨.
* 지우기 위해선 Eraser을 선택하거나 D를 누르면서 선택하면 된다.

Input System: 코드에 using UnityEngine.InputSystem; 을 넣어줘야 한다.

* 완전히 하나의 버튼에 넣는 경우: Keyboard.current.leftArrowKey.isPressed 등
* Public InputAction varname;의 형식으로 설정하면 editor에서 input을 설정 가능.
* Action Properties(기어 아이콘) > Action Type - Button으로 설정.
* Add(+ 아이콘) > Add Binding 선택, 새로 생긴 binding 클릭시 선택 가능.
* Path > Listen 으로 누르는 버튼을 사용 가능.
* 모든 action은 기본적으로 비활성화 상태이기 때문에 start() 함수의 안에 varname.Enable()의 문장을 넣어 활성화 해줘야 한다.
* 이제 varname.isPressed()의 형식으로 input을 사용할 수 있다.
* Binding에 Add Up/Down/Left/Right Composite를 추가하면 composite binding을 추가하여 대각선 움직임도 가능하게 한다. Sub-binding에 필요한 버튼들을 할당해주면 된다.
* Varname.ReadValue<Vector2>();의 형식으로 varname의 현재 값을 가져올 수 있다. 이걸 Vector2 move = varname.Read…로 작성하면 변수에 저장이 가능하고, 이걸 Vector2 position = (Vector2)transform.position + move \* 0.1f의 형식으로 활용 가능하다. 캐릭터에 위치를 적용하기 위해선 transform.position = position처럼 사용하면 된다.

Frame rate: 기본적으로 unity는 60 프레임으로 돌아간다.

* 기본 프레임 설정을 바꾸고 싶은 경우 QualitySettings.vSyncCount = 0; Application.targetFrameRate = 10; 처럼 설정하면 fps는 10이 된다.
* 기본적으로 Update 함수는 각 frame마다 호출된다. 때문에 update에 캐릭터의 속도를 넣는 등의 선택을 하면 frame 수에 따라 캐릭터의 속도가 변하게 된다.
* 계산식에 Time.deltaTime을 곱하는 식으로 포함하면 frame과는 독립적으로 움직일 수 있다.
* 하지만 rigidbody를 추가하게 되면 캐릭터가 떨게 된다. Unity가 Rigidbody 항목을 계산하는 방법 때문. 이를 해결하기 위해선 rigidbody의 포지션과 object의 포지션을 같게 해주는게 좋은데, 물리 계산은 안정적이기 위해선 일정한 interval이 필요하다. 즉 update로는 안됨.
* 이를 해결하는게 FixedUpdate로, 물리 시스템과 같은 주파수를 가진다. 그래서 이 함수안에 넣게 되면 물리 시스템과 같은 interval로 신호를 보낼 수 있음.

Game Object