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.

GameObject: 캐릭터 등의 item.

* 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.
  + Sleeping Mode > Never Sleep: 원래 rigidbody 컴포넌트는 object가 움직임을 멈추면 계산하지 않는다. 하지만 damage zone등은 움직이지 않아도 계산이 되어야 하기 때문에 Never Sleep으로 설정해주면 된다.
* Box Collider 2D: 물리 시스템에게 물체가 고체라는 것을 알려주기 위한 컴포넌트. Rigidbody는 물리특성을 추가하고 시뮬레이션에 참여시키는 용도, collider은 단순 충돌감지와 충돌체의 형태 정의 용도이다.
  + Edit Collider로 물체의 collider의 크기를 변형할 수 있다.
  + Is Trigger 를 활성화 하면 트리거의 형태로 변하게 된다. 즉 충돌해서 통과하지 못하는 형태가 아닌, 특정 효과를 발동시키는 트리거가 됨.
* Layer: 지정한 object들이 충돌하지 않도록 층을 바꿔주는 기능.
  + launch등의 상황에서 쏘자마자 캐릭터와 충돌하여 projectile이 사라지는 걸 방지해준다.
  + Inspector > Layer dropdown > Add Layer로 해서 특정 layer들을 구분하여 목표 object들을 원하는 layer에 두면 된다.
  + 이제 특정 layer들이 부딪히지 않게 하기 위해선, Edit > Project Settings > physics 2D > Layer Collision Matrix 를 선택하여 설정을 바꿔주면 된다. 원래 모든 layer은 다른 모든 layer과 충돌하도록 설정되어 있지만, 특정 layer이 충돌하지 않도록 체크를 해제해주면 된다.

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. 원하는 object에 부착하려면 드래그 앤 드롭 하면 됨.
* 수정은 컴퓨터에 설치되어 있는 IDE로 진행된다.
* Public 변수로 설정하게 되면 다른 코드 외에도 unity editor의 inspector에서도 수정이 가능하다.
* Health 등 필요한 변수는 새로 만들어야 한다.
* 가끔 특정 코드에 있는 변수를 사용하고 싶은데 public으로 만들기는 오류가 생길 것 같고, 읽기만 하면 되는 경우가 있다. currentHealth가 읽고 싶은 변수라면 public int health {get {return currentHealth;}} 이런 식으로 해당 스크립트에 getter 코드를 만들어주면 된다.
* Functions:
  + Update: 게임이 진행되는 동한 규칙적으로 반복되는 instruction.
  + Debug.Log(variable): 콘솔 창에 변수를 띄우는 함수.
  + Mathf.Clamp(제한 해야 하는 변수, 최소한의 값, 최대한의 값)은 경계 값 사이로 수를 고정시키는 함수
  + Start: 게임이 시작하면 발동되는 함수.
  + OnTriggerEnter2D(Collider2D other) object가 다른 object와 충돌 시 호출되는 함수.
  + PlayerController controller = other.GetComponent<PlayerController>(); 이게 OnTriggerEnter2D 안에 있으면 object와 충돌한 다른 object의 PlayerController Script를 reference해서 controller변수에 저장 한다. 만약 충돌한 object가 이 script를 가지고 있지 않다면 변수엔 아무것도 저장되지 않는다.
  + 그럼 이 controller를 어떻게 쓰느냐? Controller.changeHealth(1)처럼 PlayerController에 저장되어 있는 함수를 호출할 수 있다.
  + Awake는 해당 script가 붙어있는 object 생성과 동시에 호출되는 함수이다. Start는 gameobject가 생성된 바로 다음 프레임에 호출되지만, Awake는 즉시 호출된다. projectile들의 rigidbody2d 등은 여기에 저장하면 좋다.
  + Rigidbody2d.AddForce(direction\*force): direction 은 vector2값이고 force는 float값이다. Rigidbody를 특정 힘과 방향으로 매 프래임마다 움직이게 하는 함수.
  + void OnCollisionEnter2D(Collision2D other) 충돌하는 collision2d 물체를 other에 저장하고, 충돌시에 호출되는 함수.
  + Void Launch(InputAction.CallbackContext context) 여기서 집중해야 할 부분은 인자. Launch는 그냥 만든 함수고, 지정한 InputAction이 일어나면 이 함수를 부르도록 설정 해야하기 때문에 이때 쓰는걸 callback이라 한다. context에는 이 함수를 호출하는 input action에 대한 정보가 담겨져 있다.
  + GameObject projectileObject = Instantiate(projectilePrefab, rigidbody2d.position + Vector2.up \* 0.5f, Quaternion.identity); Instantiate 는 3개의 인자를 받는다. 첫번째는 gameobject reference, 두번째 인자에서 지정해주는 position에 해당 gameobject의 복제품을 만들고, 세번째 인자가 지정해주는 회전을 넣는다. Quaternion은 회전을 표현하는 수학적 operator이다.
  + 위의 코드는 캐릭터의 스크립트에 있는 코드인데, 이제 instantiate를 했으니 발사를 해야한다. 그리고 발사하는 코드는 projectile에 있으므로 Projectile projectile = projectileObject.GetComponent<Projectile>(); projectile.Launch(moveDirection, 300); 의 두문장을 넣어주면 projectile의 launch를 300의 힘으로 호출하면서 완성.
  + 그리고 Launch 상태임을 animator이 알아야 하기 때문에 animator.SetTrigger(“Launch”);를 넣어주면 된다.
  + 그리고 Launch 액션을 inputaction의 형태로 하나 만들어줘야 하는데, move action과 똑같은 방식으로 만들고 launchAction.Enable();을 start에 넣어준 뒤, launchAction.performed += Launch;코드를 넣어준다. 이 코드는 input action의 performed property를 사용하는 방법으로 Launch 함수를 callback하는 역할을 한다. +=는 이벤트, 즉 launchAction이 발생했을 때, callback에 호출할 함수를 넣는 코드이다. 만약 함수를 빼고 싶다면 -= 를 쓰면 된다.
  + 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).
  + Mathf.Approximately(x,y)는 x와 y가 대략 맞는 값인지를 true나 false로 표현. 둘 다 소수기 때문에 사소한 차이를 무시하기 위한 함수다.
  + (여긴 바껴도 됨 moveDirection).Normalize();는 길이를 1로 맞추지만 방향을 유지하는 함수.
* Types:
  + Vector2 type: 변수의 type. 2개의 수 값을 저장할 수 있기 때문에 2D 위치 좌표에 적합하다.
  + 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.

UI: ui 시스템 도입을 위한 방법은 두가지, 최신의 단순화 버전인 ui toolkit과 gameobject기준의 옛 버전인 the unity UI package가 있다. 후자는 어려운 대신 더 복잡한 기능을 가질 수 있다.

* UI Document: UI toolkit이 사용하는 텍스트 파일.
  + Project > Assets 에 Create > UI Toolkit > UI Document로 만들기.
  + 해당 파일을 열면 UI Builder로 편집 가능.
  + Unity Editor에서 Hierarchy > UI Toolkit > UI Document 컴포넌트 추가. 그리고 해당 컴포넌트에 만든 UI Document를 source로 잡아준다.
  + 캔버스 사이즈를 게임 화면과 같은 크기로 조절하기 위해선 GameUI.uxml 을 선택 > Inspector > Canvas Size > Match Game View. 배경색을 바꾸고 싶다면 Canvas Background에서 color 바꾸기
  + 새로운 UI 요소 만들기. Library에서 VisualElement 컨테이너 가져와서 Viewport에 놓기.
  + VisualElement의 이름 바꾸기, Inlined Styles > Background > Image property를 Sprite로 교체하여 결정.
  + Background > Scale Mode > scale – to - fit으로 설정하면 이미지가 늘어나는 것을 방지 할 수 있음.
  + Flex > Grow parameter를 0으로 설정하면 height를 결정해줄 수 있다.
  + Size 항목에서 입력값으로 사이즈 조절 가능. pixel값, VisualElement의 부모 element의 퍼센티지 중 결정 가능.
  + UIDocument > UI Document component의 PanelSettings > Scale Mode > Scale with Screen Size, Screen Match Mode > Match Width or Height, Reference Resolution을 x:1920 y:1080으로 설정. UI는 1920,1080을 기준으로 스케일링 됨.
  + 만약 특정 요소를 하나의 sprite의 안에 넣을 거라면(ex.증감하는 체력바) 해당 sprite에 드래그 앤 드롭하면 된다. 그러면 child element로 제작됨. 이렇게 하면 parent element의 일정 퍼센티지 크기로 설정하는듯 다양한 이점이 있다.
  + Child element의 size를 100으로 설정하고 type을 %로 잡아주면 parent element를 꽉 채우게 됨.
  + UI 시스템은 특정 element의 child element들은 전부 위에서 아래 순으로 그리기 때문에, 가장 위에 있는 element가 맨 아래에 깔리게 된다.
  + 특정 물건의 위치를 바꾸기 위해선 Position > Absolute로 설정해줘야 함.
* UIDocument Script 관련:
  + UIHandler 등의 이름을 가진 관리용 script를 제작한다. UIDocument game object에 붙힌다.
  + UIDocument uiDocument = GetComponent<UIDocument>(); 이 방식으로 UIDocument game object에 있는 UI Document 컴포넌트를 불러온다.
  + VisualElement healthBar = uiDocument.rootVisualElement.Q<VisualElement>("HealthBar"); UI hierarchy는 rootVisualElement를 통해 접근 가능하다.
  + Q는 query의 줄임으로, Hierarchy에서 특정 visual element를 찾을 수 있다. 이번 케이스에선 오른쪽의 이름을 사용하는 방법.
  + healthBar.style.width = Length.Percent(CurrentHealth \* 100.0f); UI Builder을 이용하여 healthbar의 width를 조절하는 중임. 만약 특정 값을 줘버린다면 pixel size로 인식하기 때문에 Length 함수를 써야한다. Length 함수가 퍼센티지 값으로 전달해주는 역할을 한다.
  + UIHandler는 게임 내에 하나밖에 없는 존재이므로, static으로 선언하여 reference 없이도 접근할 수 있도록 설계하는 것이 좋다. 이렇게 class의 instance가 딱 한 개만 필요할 때, 하나의 class가 UI관련된 모든 것을 다룰 때 사용하기 좋은 디자인 패턴을 singleton 패턴이라 한다.
  + 이를 하기 위해선 public static UIHandler instance { get; private set; } 처럼 static 요소를 만들어준다. 이제 프로젝트의 어디서든 UIHandler.instance 를 치면 컴퓨터가 이해하고 해당 property를 가져온다.
  + 그리고 이 instance property에 UIDocument의 현재 instance를 담아야 하기 때문에 Awake 함수에 instance = this;를 작성한다. 그러면 UIDocument object는 게임 시작과 동시에 생성되기 때문에 Awake는 즉시 시작된다.
  + 이제 PlayerController 등의 다른 스크립트에서 UIHandler.instance.SetHealthValue(currentHealth / (float)maxHealth); 의 방식으로 UIHandler를 호출 할 수 있다. 해당 문장에선 UIHandler의 SetHealthValue 함수를 호출하여 일정 비율의 소수를 전달하여 표시할 체력을 알려주는 과정이다.
* NPC 대화창 UI 만들기:
  + UI 제작:
    - 새로운 UI Document를 제작하고 이름을 붙힌다.
    - 대화창의 배경이 될 VisualElement를 넣고 Background라 이름 붙힌다.
    - Image property에서 원하는 sprite를 지정한다.
    - Height와 Width를 100%으로 설정하여 전체 캔버스를 덮게 한다.
    - 이제 대화 내용이 될 글을 넣을 차례. Label 요소를 찾아 Background의 child 요소로 넣는다
    - Height, Width는 100%로 설정
    - Margin & Padding 에서 맨 위의 Margin에 원하는 값을 넣으면 모든 방향으로 적용된다.
    - Text section에서 폰트, 사이즈, 색깔 등을 지정한다. 만약 text가 자동 줄넘김이 되도록 원한다면 Wrap도 수정한다.
    - 맨 위의 attribute에서 text에 NPC가 했으면 좋겠는 말들을 넣는다.
  + GameUI에 해당 대화창을 넣기:
    - GameUI Document를 열고, Library > project > NPCDialogue document를 드래그 드롭한다. 이러면 해당 UIDocument를 인스턴스화하는 효과가 있다.
    - 이제 position을 absolute로 바꾸고, 원하는 대로 position 값을 바꿔주면 크기도 변환이 된다.
  + 해당 UI를 raycasting 이후에 잠깐 동안만 실행되도록 설정하기:
    - UIHandler script를 연다.
    - Script의 상단에 필요한 variable들을 추가한다. 예시에선 public float displayTime = 4.0f;, private VisualElement m\_NonPlayerDialogue;, private float m\_TimerDisplay; 이었다. m\_NonPlayerDialogue는 visualElement에 대한 reference를 담는 변수이다.
    - Start에서 m\_NonPlayerDialogue를 초기화해준다. m\_NonPlayerDialogue = uiDocument.rootVisualElement.Q<VisualElement>("NPCDialogue");, m\_NonPlayerDialogue.style.display = DisplayStyle.None;, m\_TimerDisplay = -1.0f; 이다. 두번째 줄은 해당 VisualElement를 시작할 때 숨겨준다. 세번째 줄은 display 타이머를 -1로 설정하여 UI가 등장하지 않는 경우의 default 값으로 사용하기 좋은 값이다.
    - 이제 update 함수에서 m\_TimerDisplay 값이 0 이상이면 time.deltaTime 값을 빼주고, 그 안에 0이하가 되면 displaystyle.None으로 전환하도록 하는 코드를 넣는다
    - 아래에는 displaydialogue라는 새로운 함수를 만들고, 호출되었을 때 m\_NonPlayerDialogue.style.display = DisplayStyle.Flex; 로 바꿔주고 m\_TimerDisplay 값도 displayTime으로 설정해준다.
  + NPC 스크립트 편집:
    - NonPlayerCharacter 스크립트를 제작
    - Start와 Update를 삭제한다. NPC gameobject를 인증하여 raycast가 해당 gameObject를 확인할 수 있도록 하는 것이 목적이기 때문.
    - NPC에게 해당 스크립트를 assign
    - PlayerController 스크립트에서 raycasting의 결과 문장에 NonPlayerCharacter character = hit.collider.GetComponent<NonPlayerCharacter>(); 를 넣어준다. 이러면 character에 NonPlayerCharacter type의 instance인 npc의 정보가 들어가게 된다.
    - 이후 character이 null 이 아니라면, 즉 raycast가 정상적으로 이루어졌다면 UIHandler.instance.DisplayDialogue();로 uihandler의 함수를 호출해준다.
  + NPC dialogue를 설정할 수 있는 public으로 만들기
    - UIHandler 스크립트에 private Label 타입의 변수를 생성해준다. 이후 해당 변수에 uiDocument의 NPCDialogue visualelement의 NPCDialog Label을 쿼리로 저장해준다. 이건 uiDocument.rootVisualElement.Q<Label>(“NPCDialog”[참고로 여긴 이름이니깐 ui tool에서 저장한 레이블 이름으로 입력하자])로 해도 되고, 아까 만들었던 m\_NonPlayerDialogue에 저장되어 있는 visualelement의 인스턴스에서 m\_NonPlayerDialogue.Q<Label>(“NPCDialog”)로 가져와도 된다.
    - 이후 DisplayDialogue 함수에서 string값의 인자를 하나 더 받도록 하고, m\_NonPlayerDialogueLabel.text를 해당 인자로 설정해주도록 만든다.
    - 그리고 NonPlayableCharacter 스크립트엔 public string 변수인 npcdialog를 추가해준다. 이제 Unity 에서도 쉽게 NPC별로 편집이 가능해졌다.
    - PlayerController에선 character.NPCText를 변수에 저장하여 접속하고, 그걸 UIHandler.instance.DisplayDialogue의 인자로 전달해주면 완성이다.

Animation: Animator 컴포넌트는 game object의 애니메이션을 실행하기 위해 사용된다. 요약하자면 animation data는 animation clip이라는 특별한 asset에 저장되고, 해당 animation clip들을 Animator 컴포넌트에 있는 Animator Controller asset으로 컨트롤하여 애니메이션을 제작한다.

* Animator:
  + 원하는 Game object의 Inspector > Add Component > Animator
  + Animations 폴더 안에 Create > Animator Controller에 이름을 넣고 새로 제작.
  + 이후 game object의 animator component에 controller 요소를 제작한 animator controller로 지정.
  + Animator은 두개의 핵심 요소로 이루어짐: 왼편의 The Layers and Parameters pane 는 layer, 여러 개의 3D 애니메이션을 추가하는데 쓰이고, Parameter은 animation controller에 Script가 정보를 전달하는데 쓰인다. Layout area는 애니메이션과 transition의 시각적인 representation을 나타낸다.
  + Entry state는 애니메이션이 시작하면 가장 처음 실행될 state이다. 현재 방향에 따라 다른 애니메이션을 실행해야 하는데, parameter에 따라서 여러 개의 애니메이션 중 선택하게 해주는 도구가 Blend Tree이다.
  + 먼저 연결되어 있는 animation들을 삭제하고, Create State > From New Blend Tree로 Blend Tree를 생성한다.
  + 선택한 뒤, 2d 게임을 만드는 예시에서는 Blend Type을 2D Simple Directional로 잡아준다.
  + Parameter은 이미 자동으로 만들어진 Blend 값이 있는데, 새로운 걸 만들려면 Parameters > Add > Float(이건 값에 따라 다름). Blend 값도 이름을 바꿔서 쓰던가 하면 된다.
  + 이제 Inspector에서 Parameters dropdown을 만들어준 Move Y(만든 값에 따라 다름)으로 설정해준다.
  + Motion list에서 Add > Add MotionField로 각 애니메이션당 한 개씩 만들어준다.(이번 예시에선 네개 방향에 따라 달라지므로 네개를 만듦)
  + Motion picker로 각 animation들을 각 motion field에 지정해준다.
  + 그리고 motion Field의 property들의 값을 지정해준다.(예시에선 Left는 (-0.5, 0), Up은(0,0.5)등으로 설정 함.)
  + 어떻게 작동하는지 보고 싶다면 Inspector의 가장 아래 있는 Blend Tree preview 창에서 play를 누른 뒤 빨간 동그라미를 움직여보면 된다.
  + 여러 Blend Tree가 연결되어 있는 형태를 State Machine이라 한다. Player를 위한 state machine을 만든다면 Idle, Moving, Hit, Launch의 형태로 각각 방향에 따라 다른 애니메이션을 실행해야 하는 상태의 네가지이다.
  + 각 state사이에는 transition을 나타내는 화살표가 있는데, 사이에 화살표가 없으면 하나의 애니메이션에서 다른 애니메이션으로 전환이 불가능하다.
  + Transition 화살표를 선택하면 3D 애니메이션에서 자주 쓰이는 설정들을 건드릴 수 있다(애니메이션들이 합쳐지는 경우가 많기 때문). 2D 상황에서 쓰이는 설정들만 보자면
    - HasExitTime: 조건이 바뀌는 순간 바로 다른 애니메이션을 실행할지, 아니면 해당 애니메이션이 종료된 이후 다음 애니메이션을 실행할지를 결정하는 요소.
    - Conditions list: transition의 발생을 조절한다. 두가지 방법으로 발생하는데 아무런 조건이 설정되지 않았을 때, 현재 애니메이션의 실행이 끝난 뒤에 transition이 일어난다. 이는 특정 이벤트에 연결되어 있는 애니메이션들(Hit처럼)에게 좋은 방법이다. 혹은 조건이 설정되었을 때, 정의된 조건이 맞춰지면 transition이 일어난다(speed가 0.1이하일 때 moving에서 idle로 바뀌는 것처럼).
  + Trigger parameters: parameter의 종류에는 여러가지가 있는데, Hit과 Launch등의 행동은 trigger parameter로, 참 거짓 값을 받는 인자이다.
* Script로부터 값 전달:
  + Animator animator로 스크립트의 맨 위에 변수를 생성해준다.
  + animator = GetComponent<Animator>();로 Start 함수 안에 Animator 컴포넌트를 변수에 저장해준다.
  + animator.SetFloat(“Move X” ,0); animator.SetFloat(“Move Y”, direction) 으로 Update안의 가로 방향으로 움직이는 함수에 넣어주면 1이거나 -1인 direction 값이 Move Y에 전달되고 Move X엔 0이 전달되어 맞는 애니메이션이 재생된다.
  + 필요한 경우 새로운 변수를 만들어야 하는데, Player character은 enemy와 다르게 가만히 있는 경우가 있기 때문에 Vector2 moveDirection = new Vector2(1,0); 처럼 새로운 변수를 만들어 줘야 한다. 캐릭터가 가만히 있을 때는 Move X, Move Y가 둘다 0이 되는데, 그러면 state machine에게 줄 방향이 없기 때문에 이걸 전달해 주는 것. New는 클래스의 객체를 생성할 때 사용하는 키워드다. 안쓰면 오류남.
  + 이제 코드를 만들면, move.x, 와 move.y가 0이 아닐때만 moveDirection.Set(move.x, move.y)로 설정해주면 된다. 그러면 만약 둘다 0이라면 마지막으로 전달된 방향을 기억하고 있기 때문에, 오른쪽으로 움직이다가 멈춘다면 오른쪽 idle애니메이션이 실행될 것이다.
  + animator.SetFloat(“Move X”, moveDirection.x);로 설정하는데, animator.SetFloat(“Speed”, move.magnitude)처럼 속도는 magnitude 를 써야한다.
  + 맞은 경우엔 트리거를 발동 해야 하기 때문에 changehealth function에 animator.SetTrigger(“Hit”);을 두면 된다.
* Animation Clip:
  + 이제 animation clip을 만들어야 함. Window > Animation > Animation으로 새 창을 만들기. 왼측엔 animating이 되는 propertie들이 담겨있고, 오른쪽은 keyframe을 보여주는 timeline이다.
  + 대상 game object를 선택한 뒤, Create로 새로운 클립을 만들면 된다. 이것도 특정 파일에 모아두기.
  + Project > Art > animation에 쓰일 sprite가 모여있는 곳으로 가서 대상 sprite들 모두 선택하여 드래그 드롭.
  + 보통 처음 놓게 되면 빠르거나 속도가 맞지 않을텐데, 이를 고치기 위해선 More > Show sample rate를 눌러 바꿔주면 된다. 기본 설정은 60으로, 각 sprite가 1초의 1/60만큼 재생된다는 뜻이다.
  + 만약 sprite의 방향을 바꾸려면, Animation 창에서 add property > sprite renderer > sprite renderer.Flip X 등을 추가해주면 된다. 그리고 첫 frame 부터 마지막 frame 까지 Sprite Renderer.Flip X 가 enable 되어있는지 확인할 것.
  + 복잡한 경우엔 사용할 수 없지만, 간단한 애니메이션이 반복되는 캐릭터 생성시 해당 애니메이션 클립에 사용될 sprite들을 hierarchy로 가져오는 것으로 애니메이션이 포함된 gameobject를 생성할 수 있다.

Raycasting: 직선의 선을 발사하여 collider과 부딪히는지 확인하는 액션. 시작점, 방향, 길이가 있다.

* 이를 이용하여 NPC에게 말을 거는 액션을 위한 함수를 만들 수 있다. void FindFriend(InputAction.CallbackContext context) 로 하면 InputAction을 callback으로 받는 이전의 코드와 같다.
* 안의 문장은 RaycastHit2D hit = Physics2D.Raycast(rigidbody2d.position + Vector2.up \* 0.2f, moveDirection, 1.5f, LayerMask.GetMask("NPC"));의 코드이다. 해석하자면 RaycastHit2D 변수가 raycast의 결과를 저장하는 용도의 변수이고, Physics2D.Raycast는 여러가지 버전이 있지만 여기서 사용하는 것은 4개의 인자를 받는 함수이다.
  + 첫 인자는 ray의 시작 포인트로, 예시에선 playerCharacter의 position + 살짝 위로 하여 캐릭터의 중심 정도에서 시작된다.
  + 두번째 인자는 방향으로, 예시에선 현재 캐릭터가 향한 방향을 전달한다.
  + 세번째 인자는 ray의 최대 사거리로, 여기선 소리지르는 것처럼 먼 곳에서 하지 못하도록 짧은 1.5로 설정해두었다.
  + 네번째 인자는 layer mask를 정의한다. 여기서 정의해준 layer외에는 무시하여 ray가 거치지 않는다. 예시에선 NPC에게만 말을 걸 생각이기 때문에 mask에 NPC만 넣어두었다.
* 해당 문장 이후 if(hit.collider != null)을 넣는다. NPC의 collider이 raycast와 실제로 부딪히는지 확인하는 문으로, 정상적으로 hit이 되었다면 RaycastHit2D형 변수 hit에 NPC의 collider이 저장되었을 것이다.

Dynamic Camera: 플레이어를 따라다니는 카메라.

* Cinemachine package: Unity에서 제공하는 코딩 없이 자동으로 카메라 움직임을 설정할 수 있도록 해주는 패키지.
  + Cinemachine package를 package manager을 이용하여 설치.
  + Hierarchy > Cinemachine > 2D Camera 생성. 그러면 Virtual Camera Game Object 생성 됨.
  + Lens Ortho Size 5로 설정. 이러면 원래의 카메라 사이즈와 같아짐.
  + Unity 같은 3d 어플리케이션엔 카메라에 두가지 모드가 있다. Perspective mode, 마치 실제로 보는 듯한 원근감을 주어 카메라에서 멀어지는 모든 선들이 하나의 점으로 향하게 되는 방법과, Orthographic, 카메라에서 나오는 모든 선들이 평행함을 유지하는 방법. 2D에선 Orthographic을 사용하고, 2D template는 자동으로 Main Camera를 orthographic으로 설정해준다.
  + Lens Ortho Size를 5로 설정하면 카메라의 높이의 반에 다섯개의 unit이 들어가는 설정이다. 폭의 크기는 resolution에 따라 변하지만, vertical height은 늘 고정되게 설정하기 때문.
  + PlayerCharacter을 CinemachineVirtualCamera 컴포넌트에 드래그 앤 드롭 한다. 그러면 카메라가 플레이어를 따라다님.
  + CinemachineVirtualCamera > Extension > Add Extension, Cinemachine Confinder 2D 추가.
  + 제한이 될 boundary값이 필요하기 때문에, empty gameObject 생성 > CameraConfiner라고 이름짓기. Box Collider 2D 추가, 원하는 boundary만큼 설정.(만약 Polygon Collider 2D를 사용했다면 끝부분을 더 부드럽게 할 수 있는데 Points > Paths > Element 0 > X,y값을 .5로 설정)
  + 이제 Virtual Camera로 돌아와서, CameraConfiner에 새로 만든 GameObject assign.
  + 이러면 PlayerCharacter과 collide 해서 바로 밖으로 밀려나기 때문에, Layer를 다르게 해서 충돌하지 않게 하면 된다.

Audio: 게임에서 발생하는 모든 사운드를 diegetic sound라 하고, 경험과는 별개로 그냥 존재하는 사운드트랙 등을 Non-diegetic 사운드라 한다. 이걸 위해 사용되는 제작 프로그램들은

* [**Wwise Spatial Audio**](https://www.audiokinetic.com/products/wwise/): A plugin for more advanced spatial audio effects, used by both indies and AAA studios alike.
* [**Audacity**](https://www.audacityteam.org/): An open source audio application that lets you record and edit audio effects. It’s particularly useful for making small edits to existing sounds.
* [**Beepbox**](https://www.beepbox.co/): A free online tool for creating instrumental melodies.

등이 있다.

* Main Camera에겐 이미 Audio Listener 컴포넌트가 있어서 사운드 입력은 자동을 됨. Audio Source 만 이제 있으면 된다.
* Background Audio 만들기:
  + 새로운 GameObject를 생성하고 이름을 짓는다.
  + Audio Source 컴포넌트를 추가한다. AudioClip 프로퍼티에 선택한 BackgroundLoop 애셋을 넣는다. Loop property를 넣어주고, Spatial Blend 값이 2D로 세팅되도록 한다
  + Spatial Blend란 사운드가 spatialized 될지를 정하는데, 이는 listener의 위치에 따라서 다른 스피커에서 소리를 낼지를 정하는 요소이다. 2D로 설정하게 되면 그런거 없이 늘 일정하게 나오는데, 이는 배경음악으로 좋은 선택이다.
* 특정 행위에 맞는 Sound 넣기:
  + 2D게임 같은 단순한 게임에서 이를 하기 가장 좋은 방법은 PlayOneShot이라는 Audio Source function을 사용하는 것.
  + 대상 gameObject에 Audio Source 컴포넌트를 넣는다.
  + PlayerController 스크립트에 새로운 변수 AudioSource audioSource를 넣는다. 컴포넌트도 해당변수에 넣어준다.
  + Public void PlaySound(AudioClip clip)함수를 만들고, audioSource.PlayOneShot(clip);문장을 넣어준다. 이제 특정 audioclip을 한번 실행하는 함수를 만들었기 때문에 기초는 된 셈이다.
  + 이제 해당 행동이 실행 되었을 때, 원하는 audioclip을 전달해주기만 하면 된다. 이를 위해서 healthcollectible 스크립트에 audioclip을 public으로 선언, 그리고 해당 audioclip을 controller.PlaySound(clip)의 형태로 불러주면 된다. 그리고 나서 Unity에서 sound를 assign해주면 끝.
* 늘 발생하는 Sound지만 3d로 넣기:
  + 적의 걸음걸이 Sound를 넣으려 한다. Enemy prefab에 Audio Source를 삽입 > Loop 활성화 > Spatial Blend 3d. 그러면 이제 Enemy에 speaker이 생기고, 파란 동그라미가 돌게 된다. 해당 동그라미가 소리가 가장 크게 들리는 경계선이고, 가장 멀리 뒤뒤 뺐을 때 나오는 동그라미가 가장 작게 들리는 경계선이다. 이 큰 원을 조절해야한다.
  + 3d sound settings > Volume Rolloff가 있는데, Logarithmic Rolloff: 기본으로, 실제 세상에서 소리가 들리는 방식을 따라하는 곡선. Linear: 일정한 선으로 줄어듦. Custom: 걍 맘대로 하는 것.
  + Max Distance를 줄여주면 큰 원이 줄어든다. 이대로 실행해버리면 정상적으로 안되는데, 이 이유는 설정이 3D로 되어있기 때문이다. Scene을 3D로 바꾸면 해결된다.
  + 근데 또 3D로 설정하면 카메라가 위에 있기 때문에 소리의 크고 작음이 정확하지 않다. 이를 해결하기 위해 MainCamera에게 child gameobject를 주어 걔의 높이를 0으로 설정해 듣게 함으로써 rendering도 망치지 않고 게임을 수정하려 한다.
  + MainCamera 우클릭 > create empty로 child gameobject생성 > audio listener 컴포넌트 추가 > position값을 0,0,10으로 설정. 왜 10이냐면 main camera를 기준으로 계산하기 때문에 main camera의 10 앞에 있으면 그게 땅의 높이와 같다. 이제 main camera의 audio listener 컴포넌트를 없애주면 완성.
* 행동에 따라 다른 sound 실행하기:
  + Enemy의 fixed 상태와 walking 상태에 따라 다른 sound를 플레이하려 한다면
  + Audioclip 타입 변수 두개, enemyFixed, enemyNormal을 public으로 선언해주고,
  + Start에서 audioSource에 컴포넌트를 불러올 때, 아래에 audioSource.clip = enemyNormal로 설정하고 audioSource.loop = true;로 설정하여 루프도 넣어준다. 그리고 audioSource.play();를 해주면 완료.그 이후 아래 fixed 상태에서 audioSource.clip을 enemyfixed로 바꾸고 루프도 바꿔주면 된다.
* Sound를 넣을 때 주의할 점: 늘 game sound의 source에게 audio clip변수를 추가해야한다. 즉 플레이어 캐릭터가 맞거나 무언갈 던질 때 나는 소리는 PlayerController 스크립트에 있어야 한다는 점.

Particles system: 작은 이미지나 mesh의 모음으로 smoke, snow 등을 나타낼 때 사용하는 이미지.

* 복잡한 효과가 필요한 게 아닌 경우, unity의 particle system으로도 효과를 만들 수 있다.
  + Smoke particle을 위해 sprite를 준비하려면 project > Art > Sprites > VFX로 가서 ParticleSpriteAtlas(대상 asset)의 sprite mode를 multiple로 설정한다. Pixel per unit 은 100으로 설정하고 sprite editor에서 4 by 4등 알맞은 slice로 자른다.
  + Hierarchy > Add > Effects > Particle System을 추가한다. 해당 object의 이름을 어울리는 것으로 바꾼다.
  + Object 선택 후 inspector에서 확인하면, 여러 propertie들이 조절 가능함을 알 수 있다.
  + Texture Sheet Animation 모듈을 활성화 하고, property에서 mode를 sprite로 바꾼다. 비어있는 sprite property의 옆에 있는 add를 눌러 빈칸을 만들고 원하는 sprite를 assign 한다.
  + Start Frame을 열고 Random Between Two Constants로 선택하고 Start Frame 값을 2로 잡는다.
  + Frame Over Time 를 선택, particle 시스템이 새로운 파티클을 생성할때마다 0,1 둘 중의 한값을 랜덤하게 골라 해당 sprite를 새로운 파티클로 사용할 것이다. 아래의 Particle System Curves가 시간 별 프레임의 변화를 보여주는데, 우리는 애니메이팅이 된 파티클이 아닌 하나의 sprite로 이루어진 파티클이 필요하기 때문에 visualization의 키프레임을 삭제해버릴 것이다.
  + 이제 파티클의 스프레드 모양을 어울리게 바꿀 것이다. Shape > Radius = 0으로 설정하여 모든 파티클이 하나의 점에서 시작하도록 한다.
  + Angle > 5로 설정하여 파티클의 퍼짐을 줄이고 조금 더 반듯한 선에 따라 생성되도록 한다.
  + 이제 randomness를 더해 더 현실적인 효과를 줄 것이다. Start Lifetime property를 찾아 Random Between Two Constants, 1.5와 3으로 설정한다. 이러면 각 particle의 수명이 랜덤하게 정해져 어떤 것은 더 일찍 사라지고 어떤 것은 더 늦게 사라진다.
  + 같은 효과를 speed와 size에도 줄 것이다. 랜덤성이 더해져 더 몰입감이 높아졌다
  + fade효과를 주어 사라지는게 더 어울리도록 고칠 것이다. Color over Lifetime 모듈을 찾아 Color 박스를 연다. Gradient editor은 파티클의 색이 수명동안 어떻게 변하는지를 보여준다. 우측 상단의 마커를 선택하여 alpha를 0으로 설정하면 투명도가 설정된다.
  + Size over Lifetime 모듈을 열어 Size 박스를 열어 원하는 프리셋을 설정하든 커스텀으로 설정하든 하여 원하는 효과를 적용한다. 여기선 5번째 프리셋을 써서 시작시엔 가장 크게 시작하여 수명이 다할수록 작아지는 효과를 줄 것이다. Ctrl을 누르고 시작과 끝점을 누르면 등장하는 tangent를 이용하여 커브를 조절 할 수 있다.
  + 이후 해당 smoke effect를 prefab으로 만들어 enemy개체의 child 요소로 넣으면 enemy에게 붙어서 연기가 나게 된다.
  + 연기의 움직임을 수정하기 위해선 Simulation Space > World로 설정하면 된다. 이러면 Local인 enemy의 움직임에 제한되는 것이 아니라 게임의 세계에 따라 작동하게 된다.
  + 플레이어가 enemy를 고치고 나선 연기가 나지 않도록 하려면, enemycontroller 스크립트의 위에 public ParticleSystem smokeEffect;를 추가한다. 이런식으로 gameobject가 아닌 component 타입을 public으로 선언해주게 되면, unity는 자동적으로 해당 컴포넌트에 대한 레퍼런스를 저장한다. 이는 GetComponent를 사용하지 않아도 됨을 의미하며 해당 컴포넌트가 없는 gameobject를 실수로 assign하지 않는다는 의미도 된다.
  + smokeEffect.stop();를 fix함수에 넣어준다. Destroy가 아닌 stop이기 때문에 있던 파티클은 끝까지 날라가고 새로운 게 생성이 되지 않을 뿐이다.
  + 이제 Unity editor로 돌아가면 enemycontroller 스크립트의 요소에 smoke effect가 추가된 것이 보인다. 여기에 목표한 child gameobject smokeeffect를 넣어주면 된다.
* 한정된 시간 동안만 있는 particle 제작:
  + Particle System 컴포넌트 > Looping 해제 > Duration 프로퍼티 수정 > Stop Action property, Destroy로 설정.
* Burst Emission: hit과 같은 행동 때 한번에 많은 파티클을 내고 싶은 경우를 위한 효과
  + Rate over Time: 시간별로 얼마나 많은 파티클이 만들어지는지를 조절, Rate over Distance: 움직인 거리 별로 얼마나 많은 파티클이 만들어지는지를 조절. 자동차 같은 상황에 적절. 이 둘로 해결할 수 있는 상황이라면 해도 좋다.
  + 만약 그게 아니라면 Particle System > Looping 해제 > Stop Action > Destroy 설정.
  + Emission > Rate over Time > 0으로 설정. 이건 딱 한번 만들어 낼 것이기 때문.
  + Bursts에서 add 하여 Count를 20으로 설정, Time을 0.00으로 설정.
  + 이런 애들은 Instantiate function으로 불러와야 딱 한번 실행되는 상황에 적합하다. Public variable에 prefab을 reference하고, 이후 일어나야 하는 상황에 해당 함수를 넣어주면 된다. Instantiate(smokeEffect, transform.position, transform.rotation); 이런식.

Build your game:

* Edit > project Settings > Player 선택. 이후 원하는 회사 이름, 제품명, 버전 등의 정보를 넣을 수 있다.
* Resolution and Presentation에선 여러가지 변경사항을 적용할 수 있다.
* File > Build Settings > Scenes in Build 아래의 Add Open Scenes를 선택하면 열려있는 Scene이 빌드에 추가된다. Scene을 빼고 싶다면 우클릭하여 remove selection 하면 됨.
* Platform은 Windows, Mac, Linux > Target Platform으로 설정하면 된다. 이후 Build를 누르면 됨.