3D콘텐츠 이론 및 활용

13주(1). 네비게이션

- Tag처리
- Navigation 처리

학습개요

학습목표

- Collider 클래스의 tag멤버 변수를 활용할 수 있다.
- NavMesh 설정하여 플레이어가 이용하는 경로를 설정할 수 있다.
- 지금까지 배운 내용을 활용하여 게임콘텐츠를 제작할 수 있다.

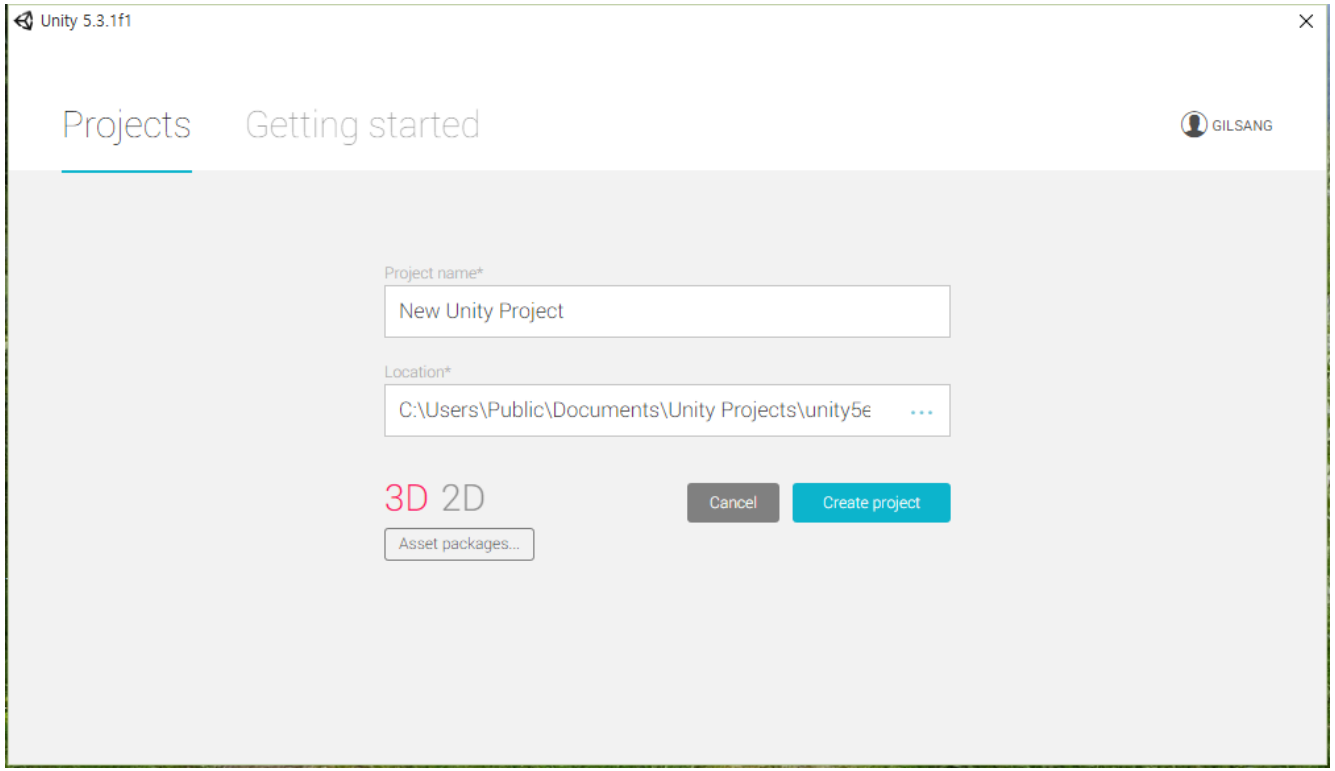
학습내용

- Collider 클래스의 tag멤버 변수
- NavMeshAgent 컴포넌트 설정

1. 씬 준비하기

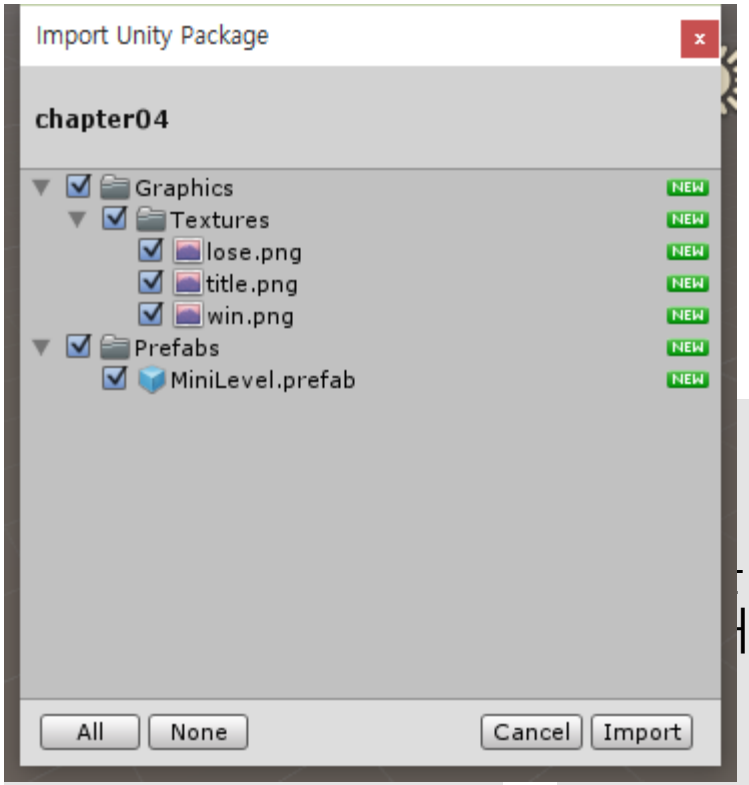
1) 새프로젝트 생성

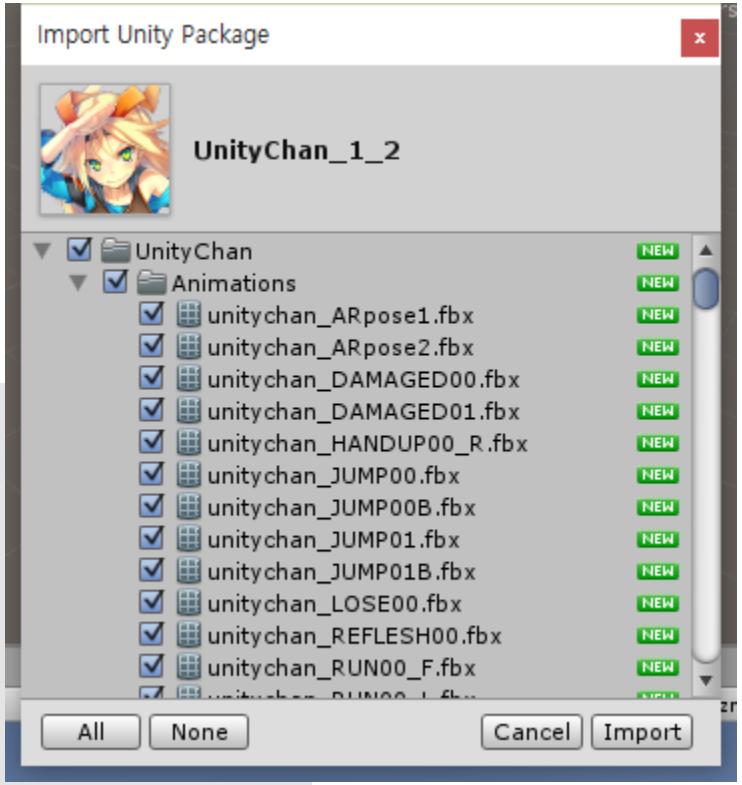
- 프로젝트 만들기



1. 씬 준비하기

2) 데이터 불러오기

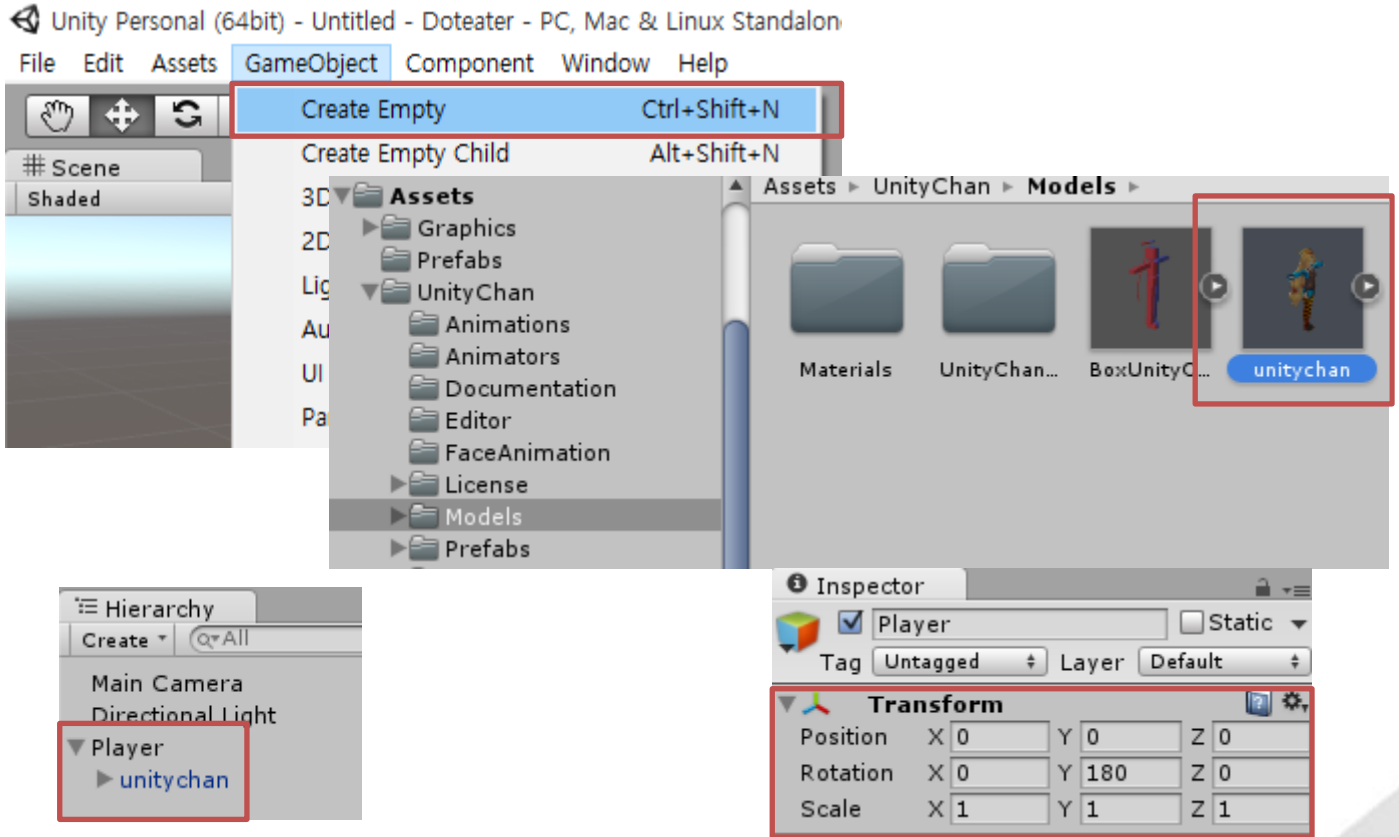




1. 씬 준비하기

3) 씬 준비

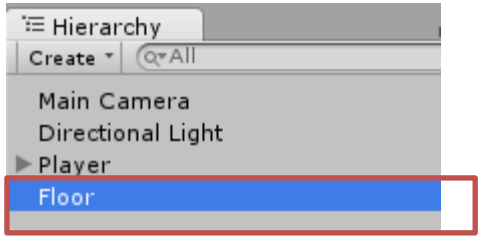
- 플레이어 만들기
  - 플레이어 게임오브젝트 생성을 위해 빈 게임오브젝트 생성
  - 유니티짱 모델을 플레이어의 자식 모델로 설정
  - 정면을 보도록 Player의 Rotation값을 (0,180,0)으로 설정

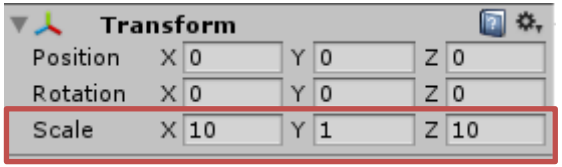


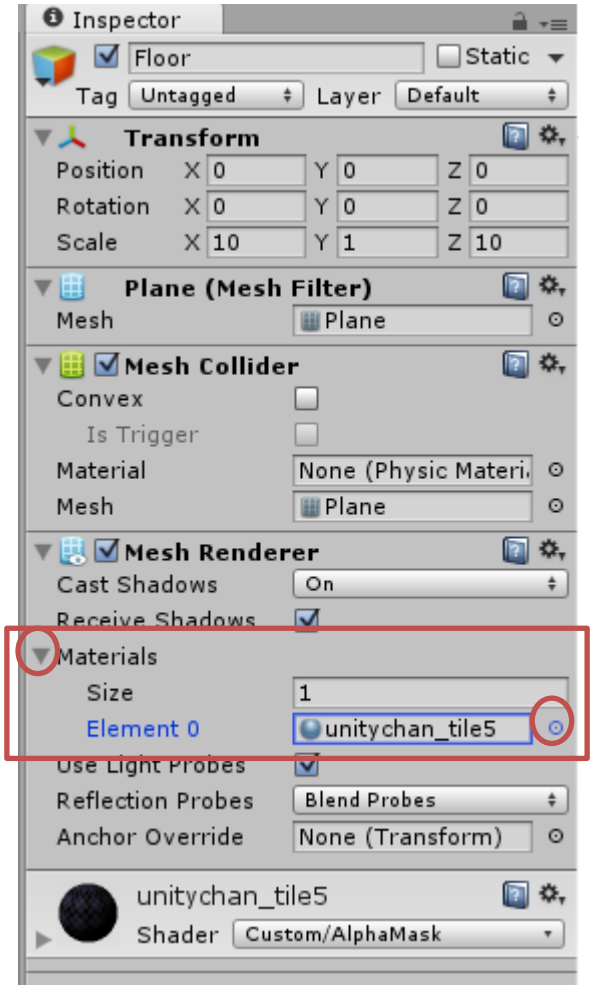
1. 씬 준비하기

3) 씬 준비

- 바닥 만들기
  - [GameObject]- [3D Object] – [Plane] 추가 및 이름변경(Floor)
  - Transform의 스케일 값을 변경
  - 바닥 매트리얼 입히기





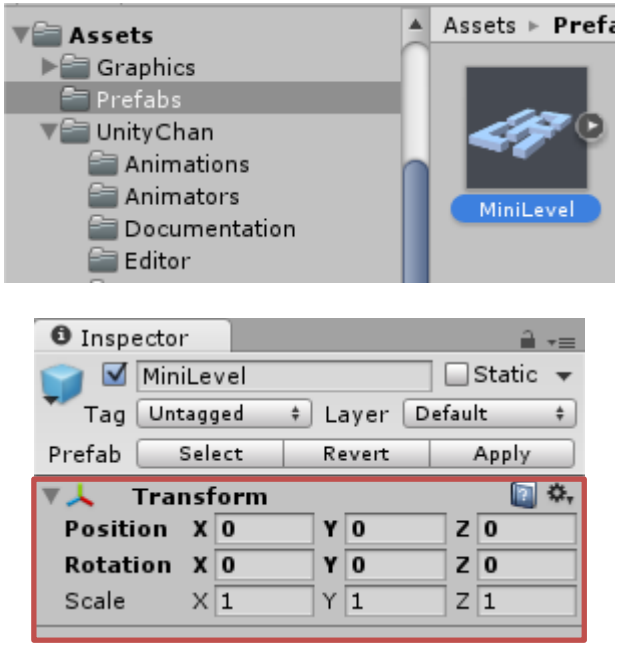


1. 씬 준비하기

3) 씬 준비

- 벽 만들기

  - [GameObject]- [3D Object] – [Cube]를 이용한 벽 만들기
  - Transform 값 확인

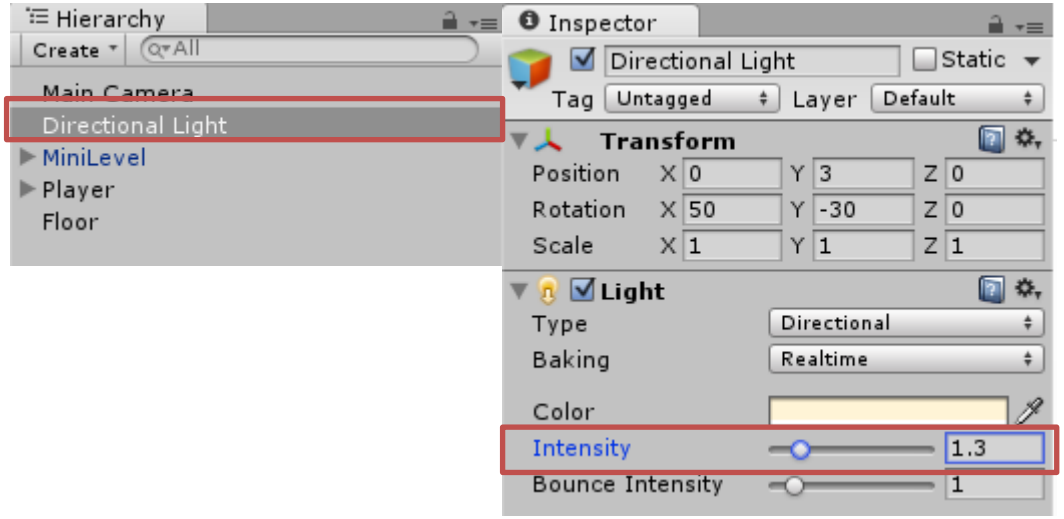


1. 씬 준비하기

3) 씬 준비

- 광원 조정하기

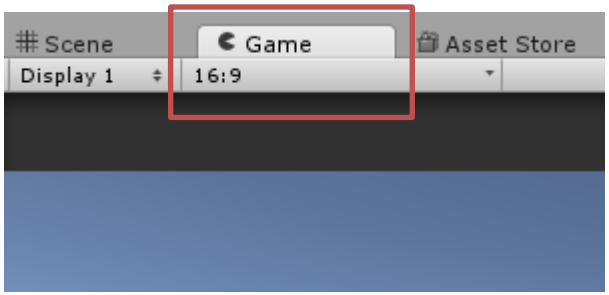
  - Directional Light 광원을 취향에 따라 조절 (1~1.3)



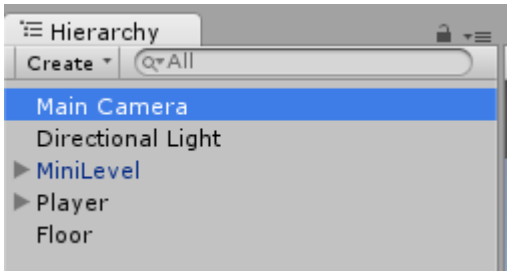
1. 씬 준비하기

3) 씬 준비

- 게임 뷰 화면 비율 고정 하기
  - Game 뷰의 화면비 16:9 선택



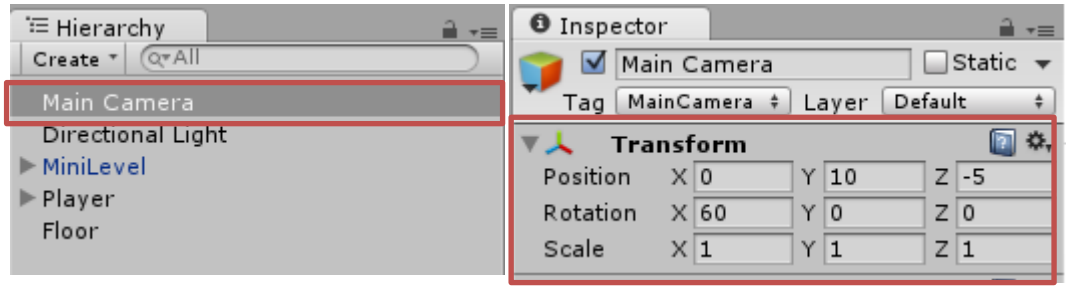
- 메인 카메라 조정하기
  - Game 뷰의 화면비 16:9 선택



1. 씬 준비하기

3) 씬 준비

- 카메라 조정 (Ctrl + shift + f)
  - 메인 카메라의 높이, 앞뒤거리,상하좌우 값을 조절하여 메인 화면을 설정

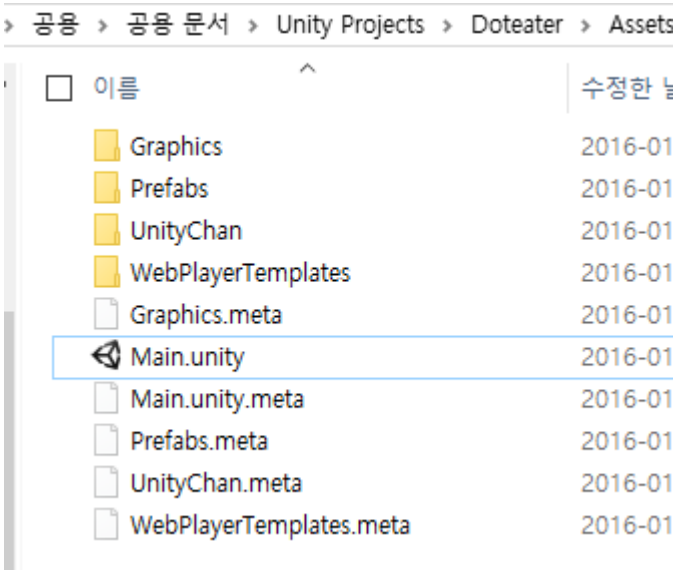


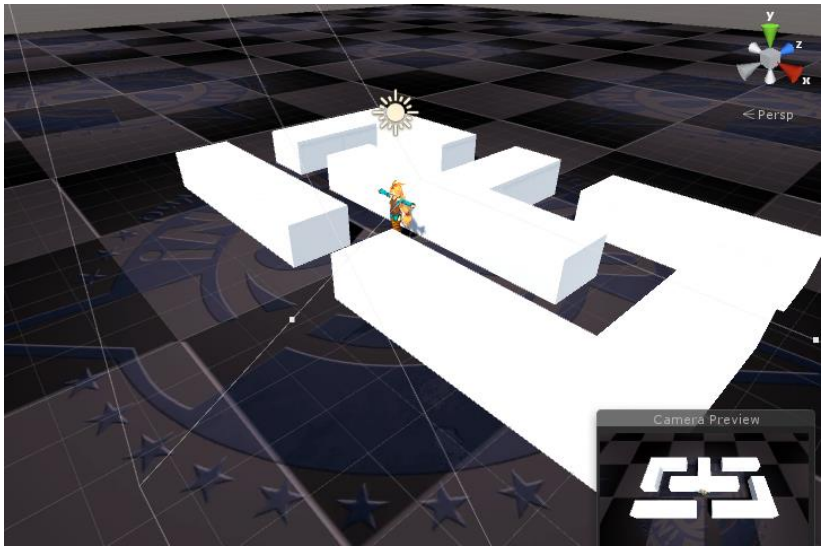
1. 씬 준비하기

3) 씬 준비

- 씬 저장

  - [Save Scene] 메뉴에서 대화창이 나오면 ₩Assets 경로에 Main.unity 로 저장
  - 씬 파일내에는 게임오브젝트의 정보와 속성정보 등이 저장된다.





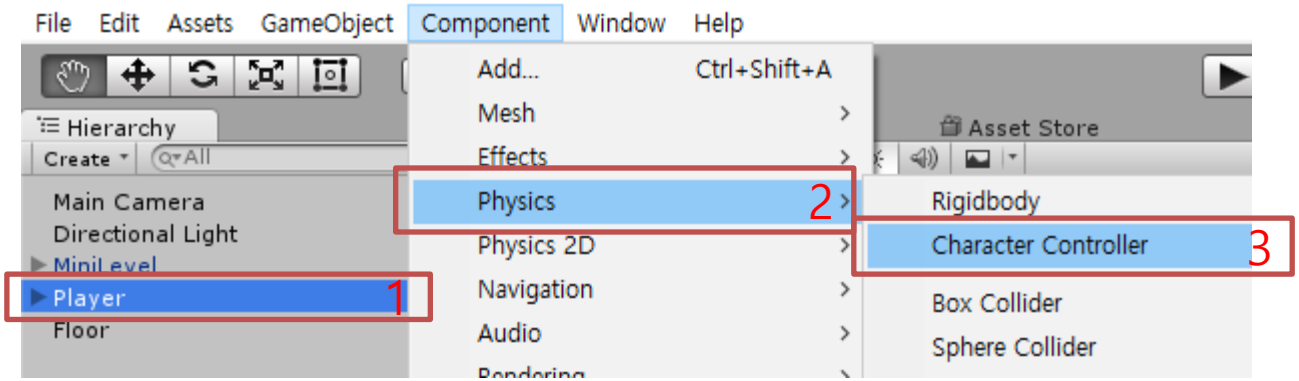
2. 플레이어 이동 처리

1) 플레이어 이동

- 포지션 이동
- 충돌 후 포지션이동
- 장애물 벽과의 충돌
- 기울어진 발판에서 움직임 처리
- 게임세계에서 충돌처리는 Character Controller 컴포넌트를 이용

2) Character Controller 컴포넌트 사용하기

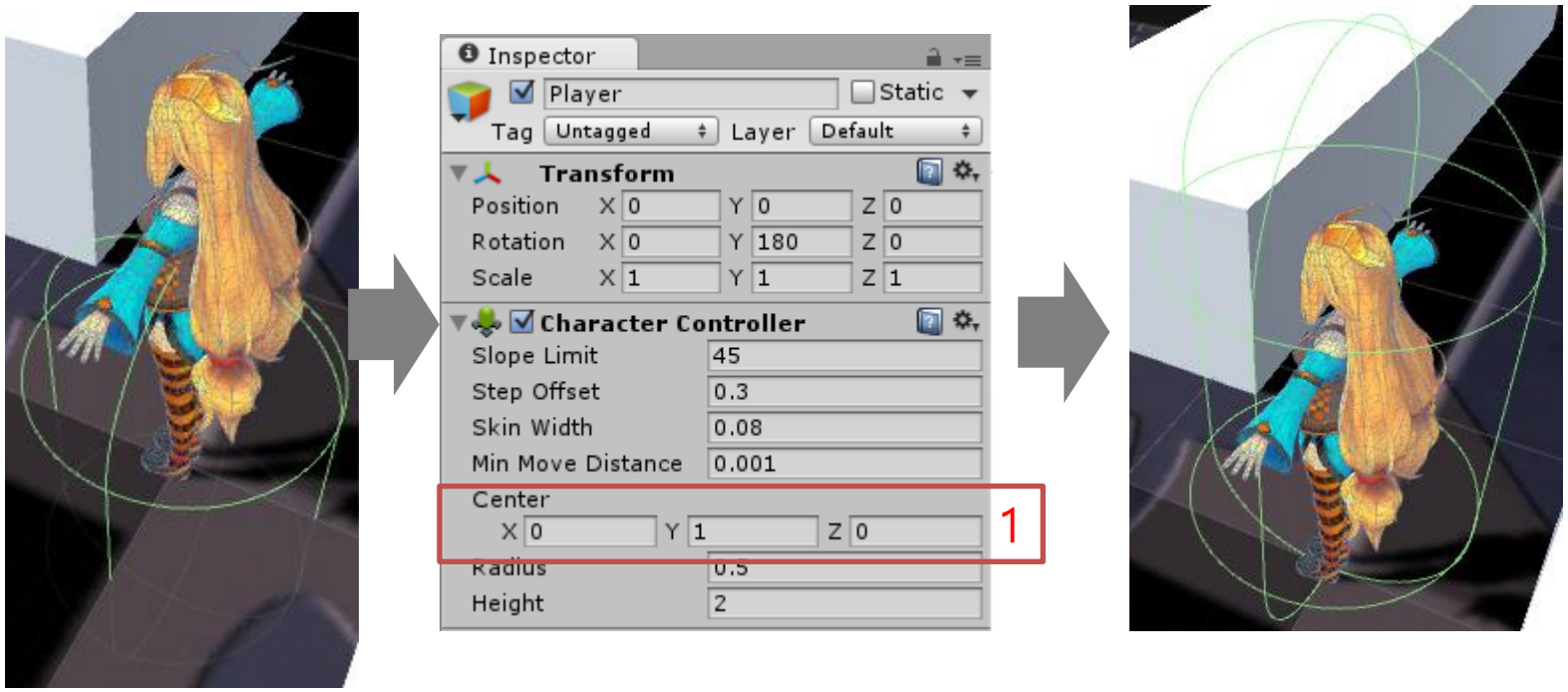
- Character Controller 컴포넌트 추가하기
- 컴포넌트가 추가되면 캐릭터에 캡슐형태의 녹색선이 표시됨



2. 플레이어 이동 처리

2) Character Controller 컴포넌트 사용하기

- Character Controller 위치 조절
  - 센터의 높이 값을 1로 주어 위치를 조절한다.

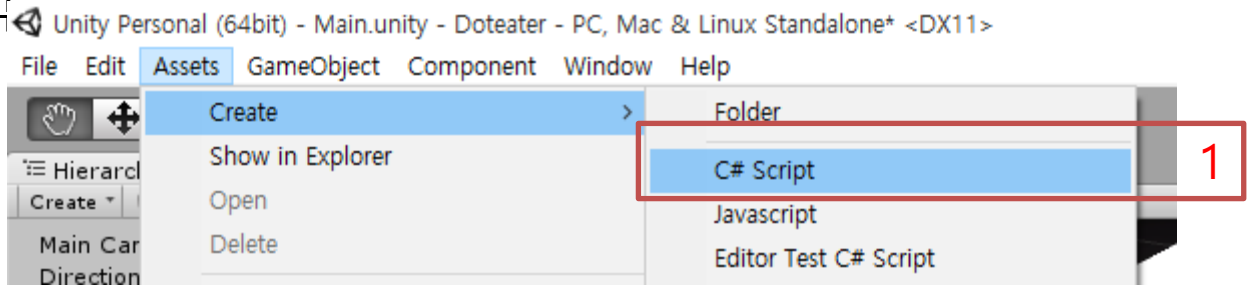


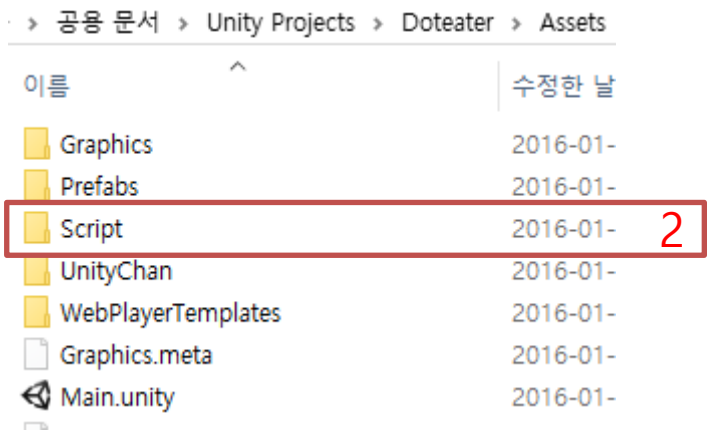
2. 플레이어 이동 처리

3) 플레이어 조작하여 이동하기

- 스크립트 만들기

  - 메뉴 또는 프로젝트 뷰에서 생성가능
  - 생성 후 파일이름은 Player로 변경
  - 생성된 파일은 Assets 폴드에 Script폴드를 만들어 관리하는 것이 효율적

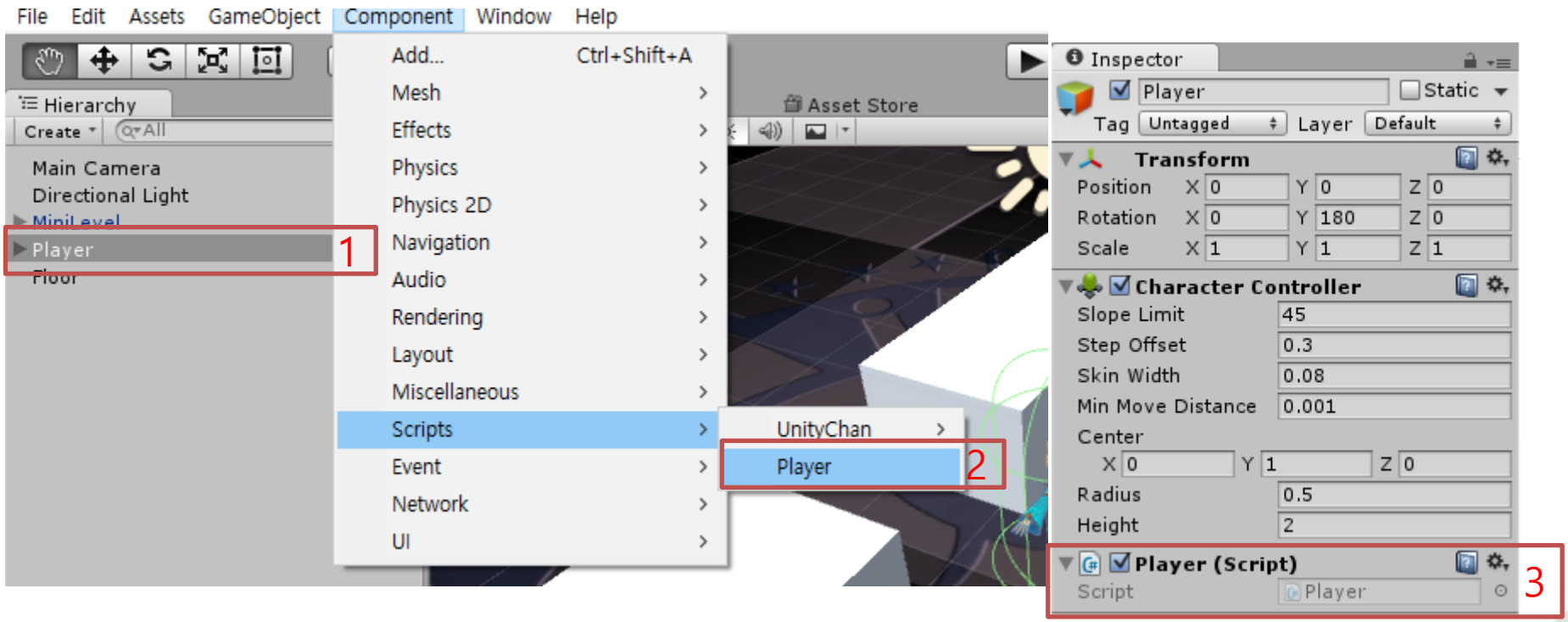




2. 플레이어 이동 처리

3) 플레이어 조작하여 이동하기

- 플레이어 게임오브젝트에 스크립트 적용하기

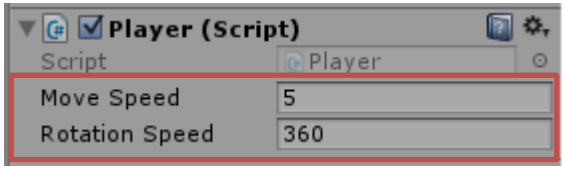


2. 플레이어 이동 처리

3) 플레이어 조작하여 이동하기

- Player.cs 스크립터 작성 (1/2)

```
using UnityEngine;
using System.Collections;

public class Player : MonoBehaviour {

public float moveSpeed = 5f; //이동속도
public float rotationSpeed = 360f; //방향전환속도

CharacterController characterController;

// Use this for initialization
void Start () {
characterController = GetComponent<CharacterController>();
}
```



2. 플레이어 이동 처리

3) 플레이어 조작하여 이동하기

- Player.cs 스크립터 작성 (2/2)

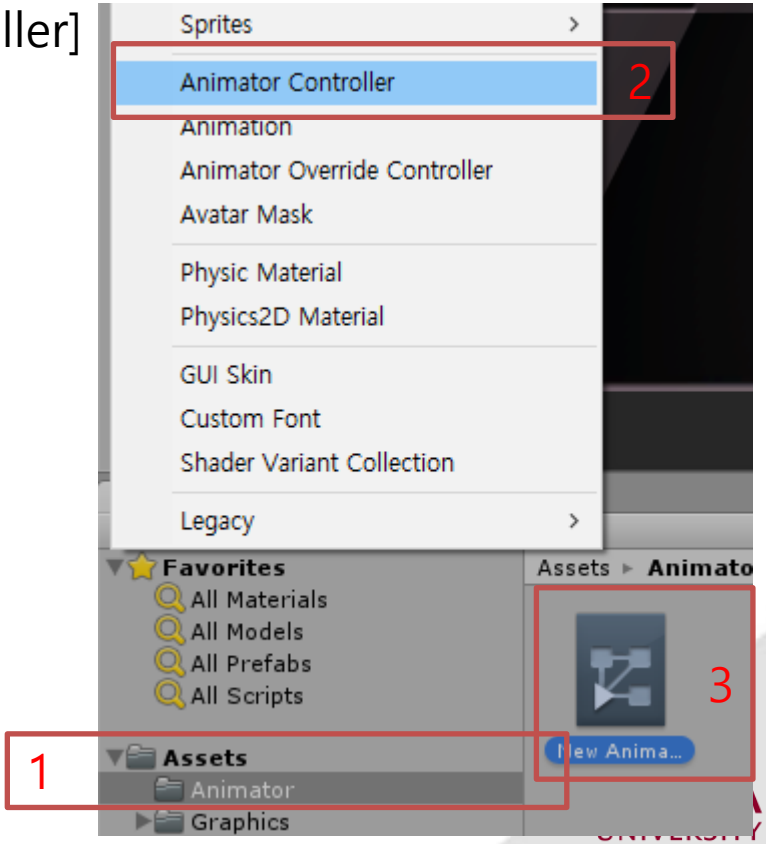
```
// Update is called once per frame
void Update () {
Vector3 direction = new Vector3(Input.GetAxis("Horizontal"), 0, Input.G
etAxis("Vertical"));
if (direction.sqrMagnitude > 0.01f) {
Vector3 forward = Vector3.Slerp(transform.forward,direction,
rotationSpeed * Time.deltaTime / Vector3.Angle(transform.forward, dire
ction)
);
transform.LookAt(transform.position + forward);
}
characterController.Move(direction * moveSpeed * Time.deltaTime);
}
}
```

3. 플레이어 모션 설정

1) Animator 컴포넌트 설정하기

캐릭터의 동작 상태를 전환하기 위해서는 Animator Controller 파일로 관리한다.

- Animator Controller 만들기
  - 프로젝트 Assets 폴드에 Animator 폴드를 생성
  - 프로젝트 뷰 – Create – [Animator Controller]
  - 생성된 파일을 Player로 수정

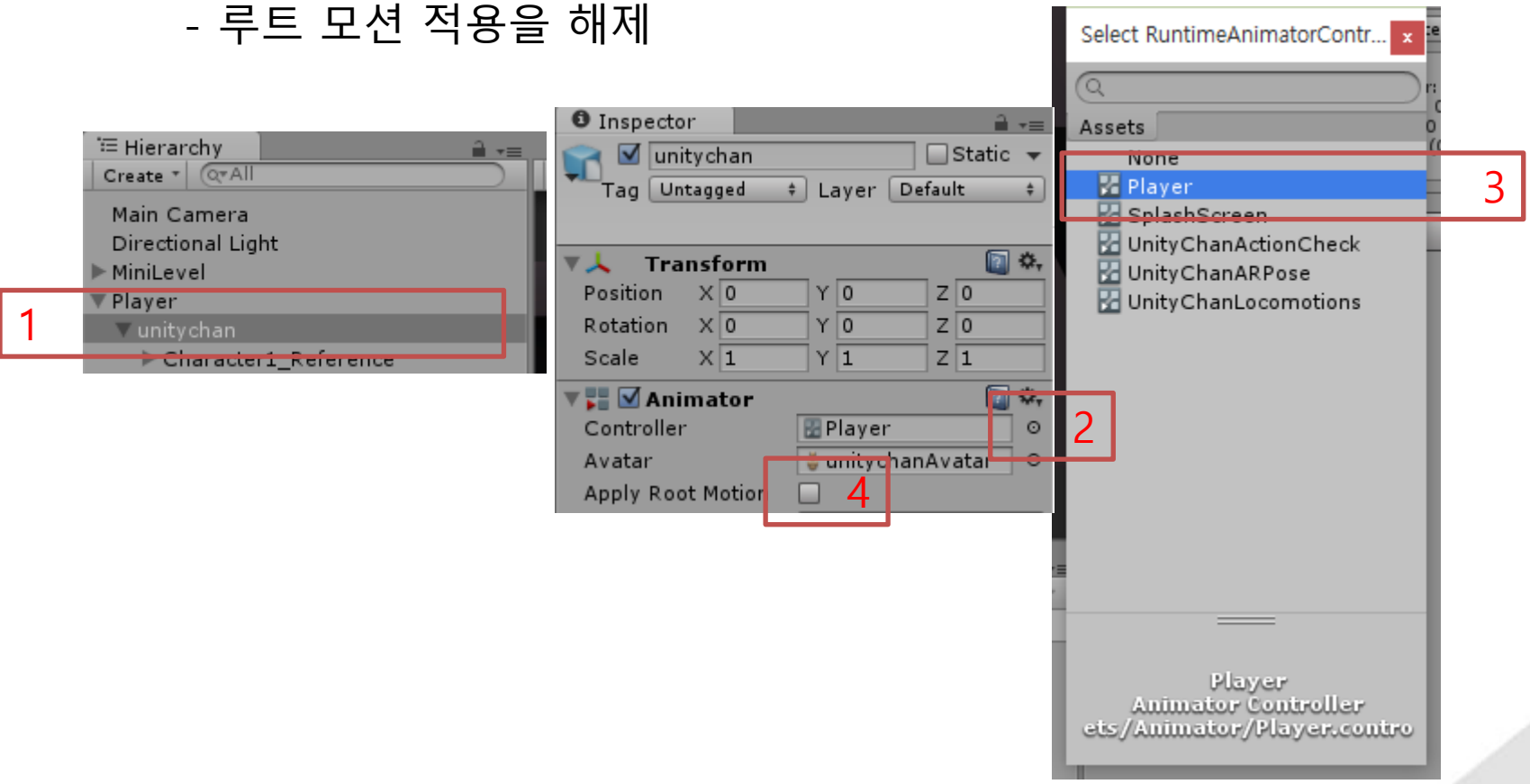


3. 플레이어 모션 설정

1) Animator 컴포넌트 설정하기

- Animator 컴포넌트 설정하기

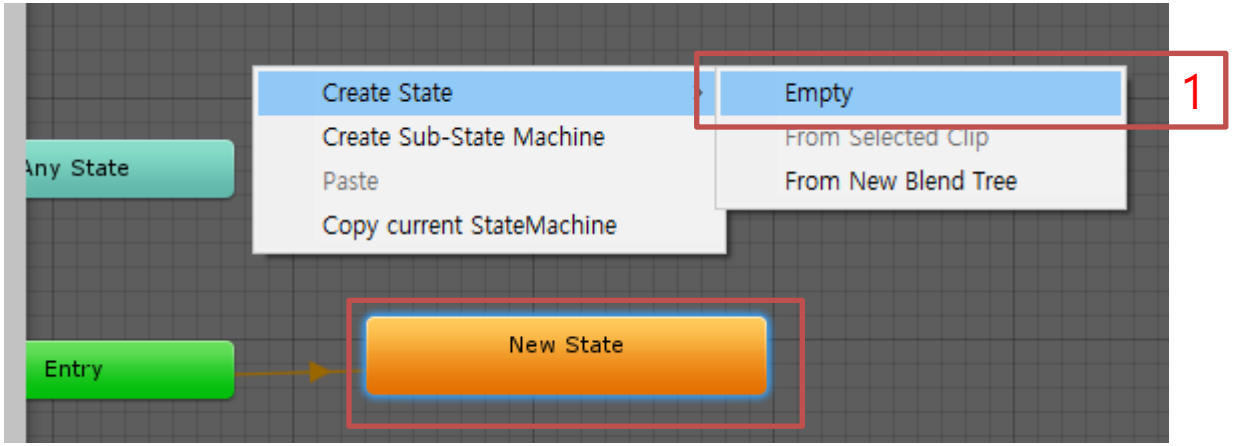
  - 자식 유니티짱 오브젝트를 선택
  - 방금 생성한 컨트롤러로 지정
  - 루트 모션 적용을 해제



3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

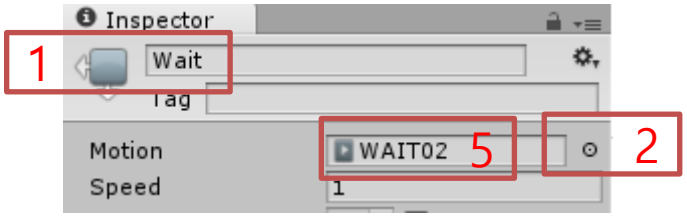
- 애니메이트 뷰 설정
- 빈 스테이트 만들기(우 클릭 팝업창)

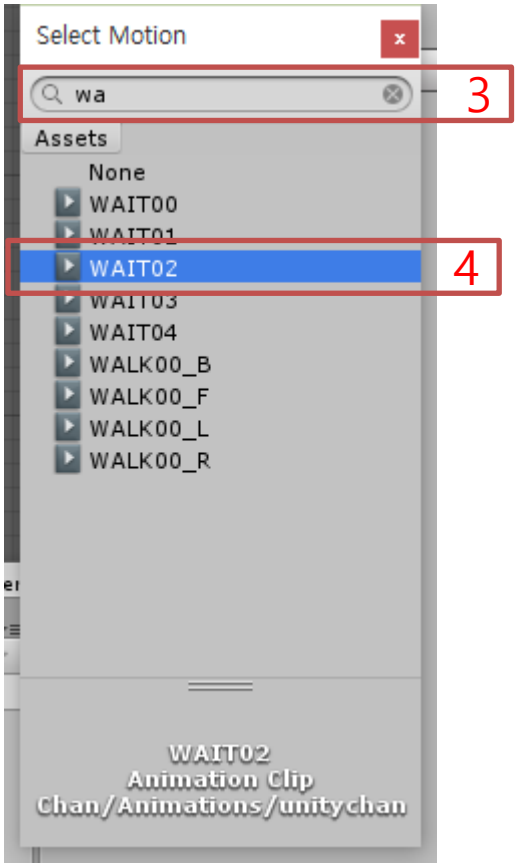


3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

- Wait 상태 만들기 예
  - 생성된 New State를 클릭하여 Inspector에서 이름을 Wait으로 준다.
  - 모션을 찾아 등록 후 확인
  - Run 상태도 추가해 보자



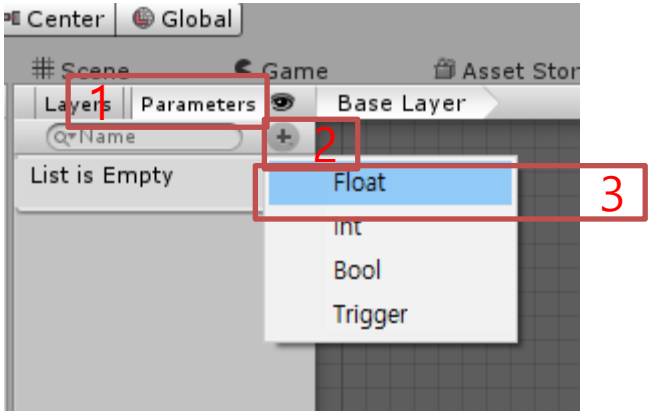


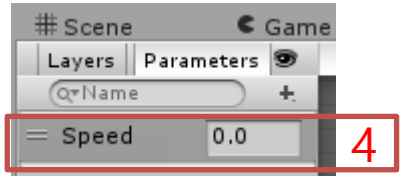
3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

- 상태전환을 위한 기준 설정

  - <조건> 이동 속도 조절을 위해 이동속도가 0이면 대기상태 그 이상이면 달리기 상태로 설정
  - 파라메터를 Float으로 지정하고 new Float이 생성되면 이름을 Speed로 변경하고 0으로 초기화





3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

- 스크립트 코딩

- 플레이어 이동속도를 Animator Controller의 파라미터로 전달하도로 처리

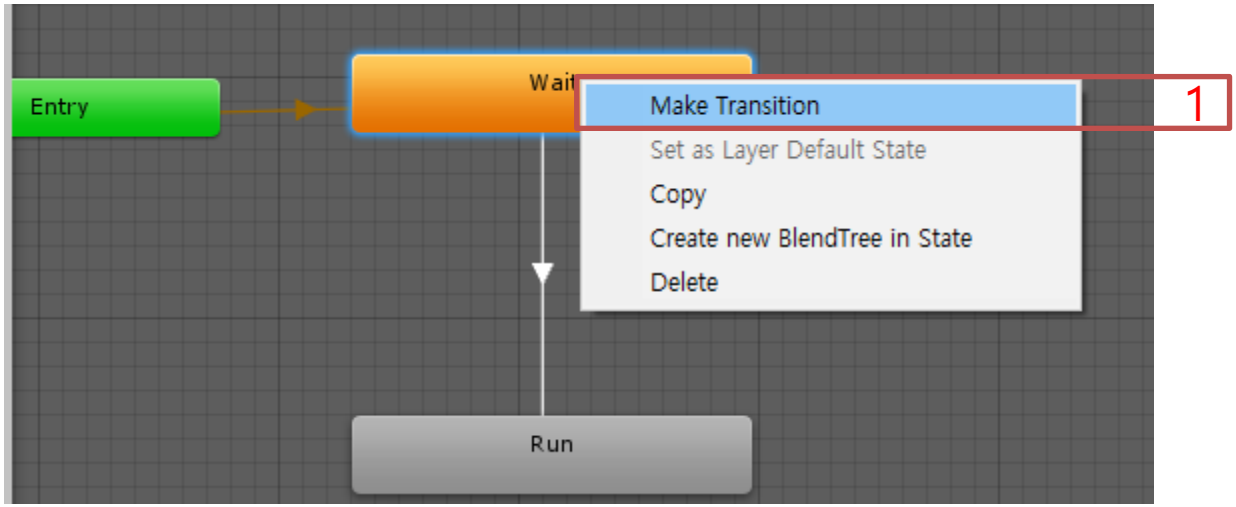
```
Animator animator;
...

void Start () {
        characterController = GetComponent<CharacterController>();
        animator = GetComponentInChildren<Animator>();
}


void Update () {
. . .
characterController.Move(direction * moveSpeed * Time.deltaTime);
animator.SetFloat("Speed", characterController.velocity.magnitude);
}
```

3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

- 상태 관계 설정

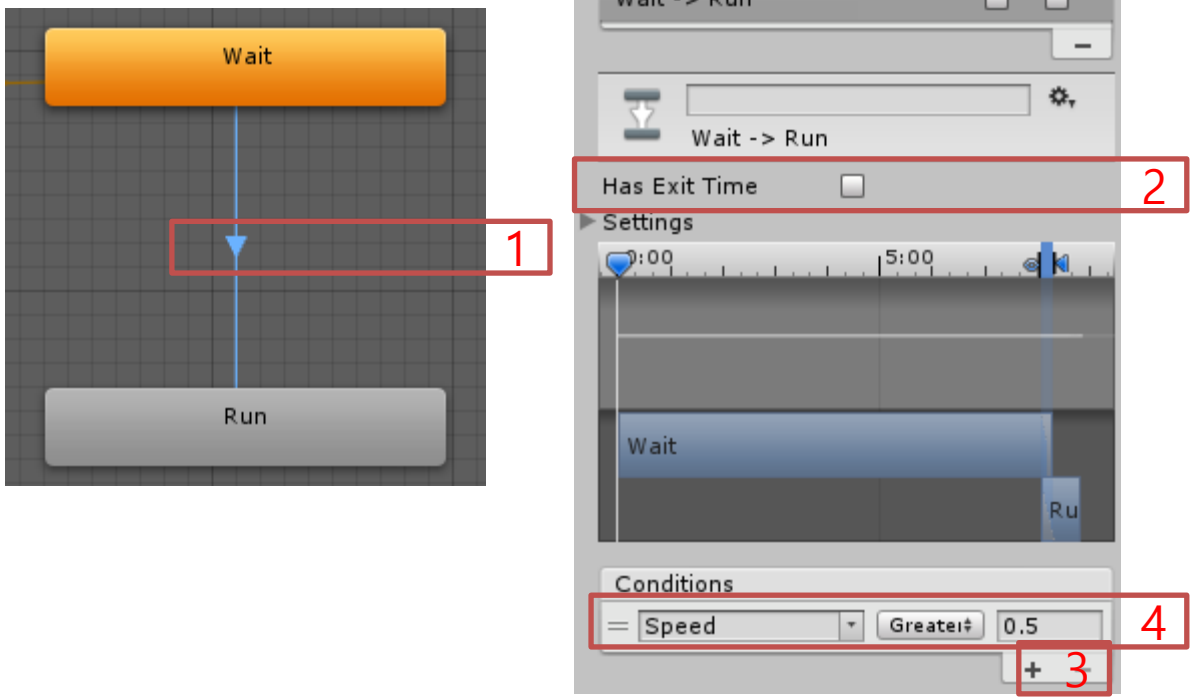
  - Wait 상태를 우 클릭 후 Make Transition 으로 run상태로 변환



3. 플레이어 이동 처리

2) Animator 뷰를 이용한 애니메이션 설정

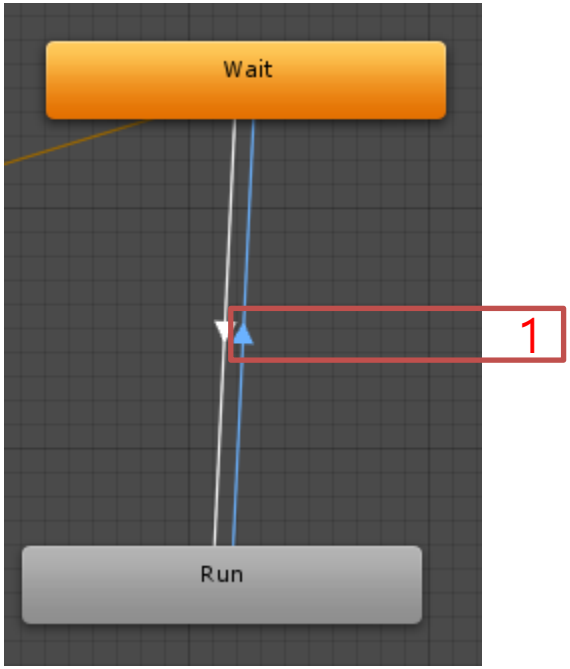
- 상태 전환 조건 설정 (wait → run)
  - Wait 과 run상태 연결선을 클릭하여 조건 설정
  - 스피드가 0.5보다 크면 전환상태로 설정

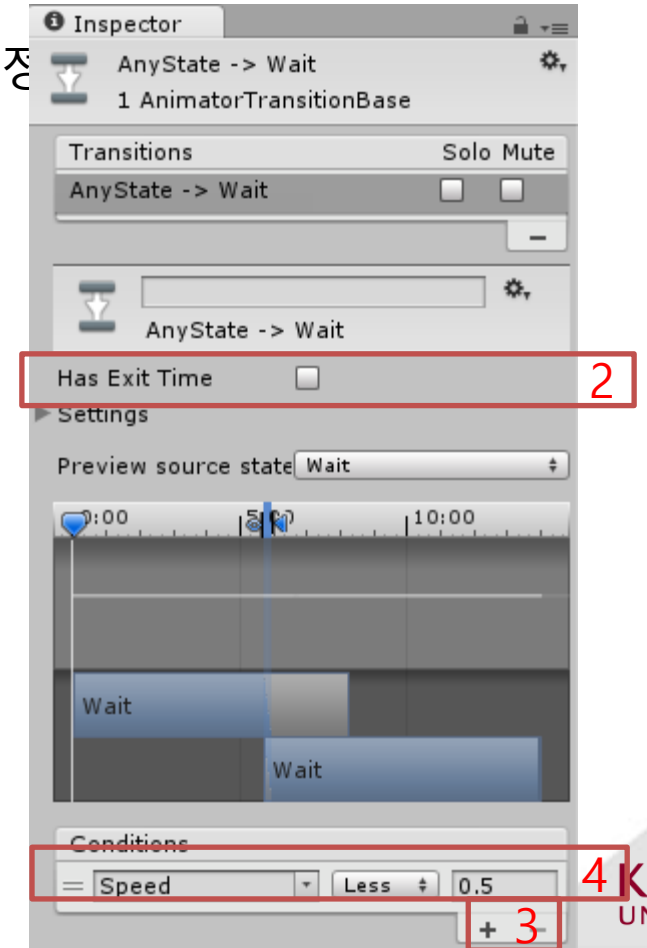


3. 플레이어 이동 처리

3) Animator 뷰를 이용한 애니메이션 설정

- 상태 전환 조건 설정 (run → wait)
  - Run상태를 우클릭하고 Make Transition, 연결선을 wait으로 연결해 줌
  - 연결선을 클릭하여 조건 설정
  - 스피드가 0.5보다 작으면 전환상태로 설정



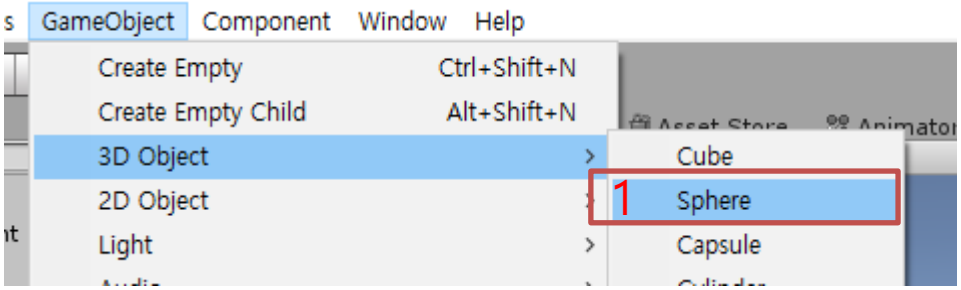


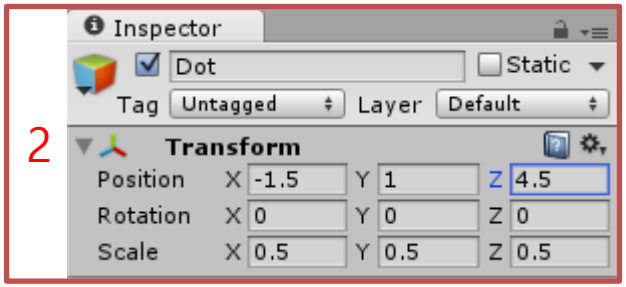
4. 동전 아이템 만들기

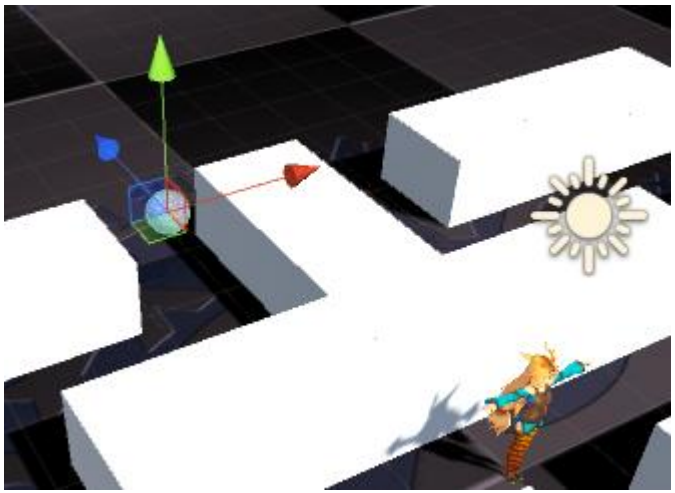
플레이어가 게임을 하면서 먹을 동전 아이템 제작

1) Sphere 만들기

- Sphere 게임오브젝트 생성하고
- Inspector에서 이름을 Dot 로 변경, 위치, 스케일 값 변경





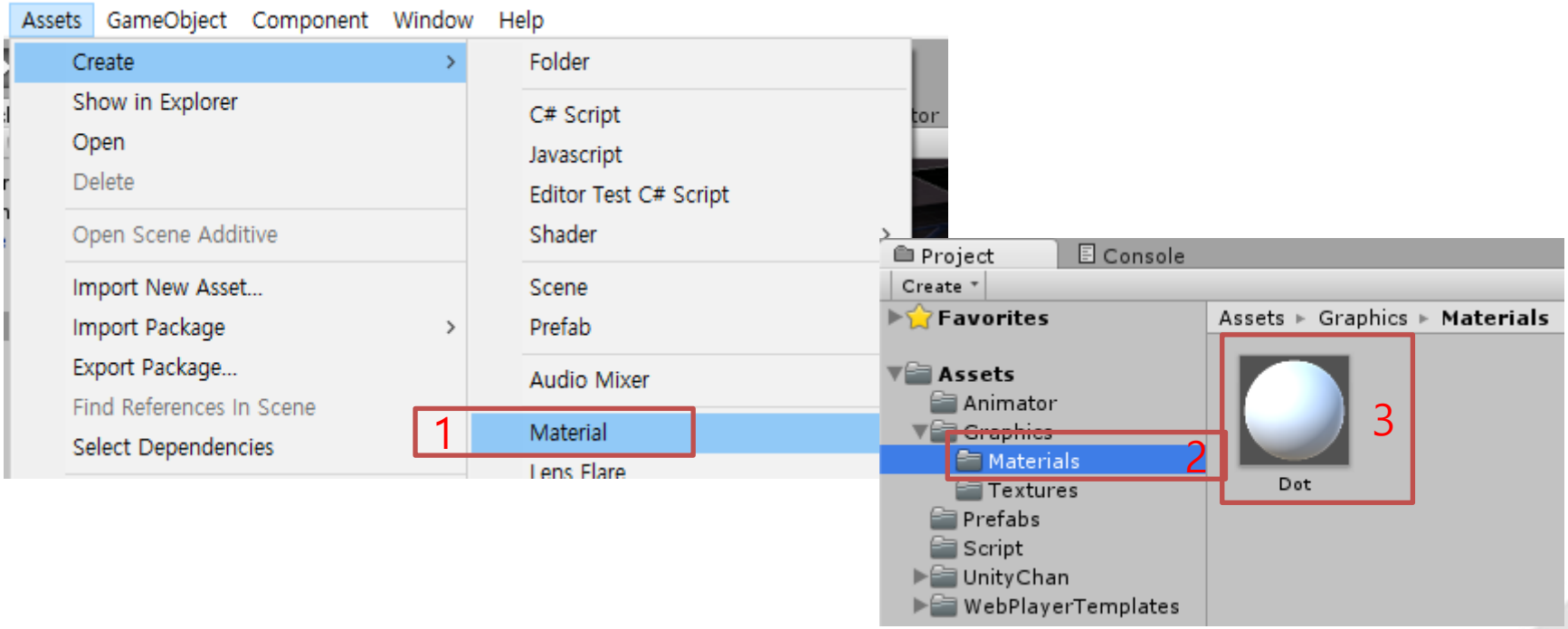


4. 동전 아이템 만들기

플레이어가 게임을 하면서 먹을 동전 아이템 제작

2) Material 설정

- 매트리얼 생성 후 이름을 Dot로 변경
- 폴드 관리

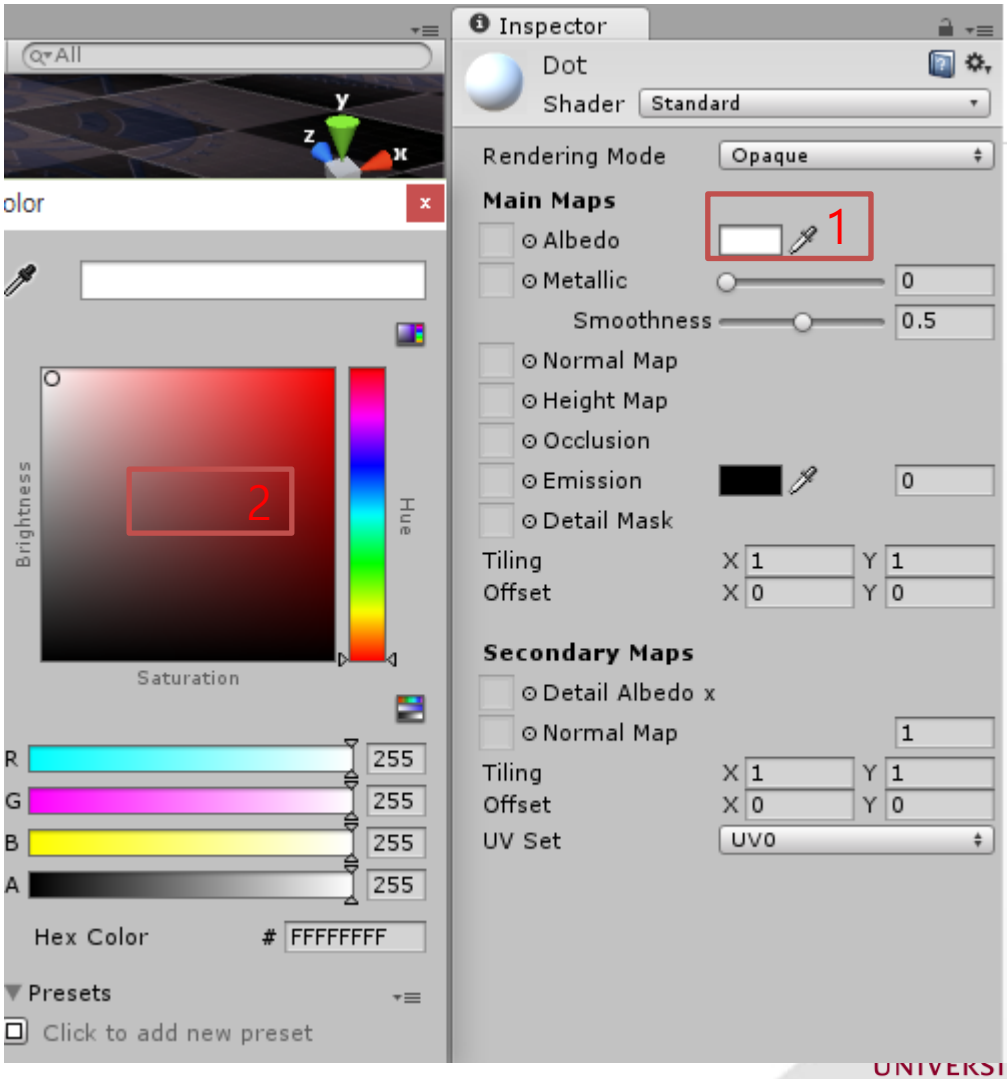


4. 동전 아이템 만들기

플레이어가 게임을 하면서 먹을 동전 아이템 제작

3) 매터리얼 색 바꾸기

- Dot 매터리얼 선택
- Albedo 스포이더 선택
- 원하는 색상으로 선택

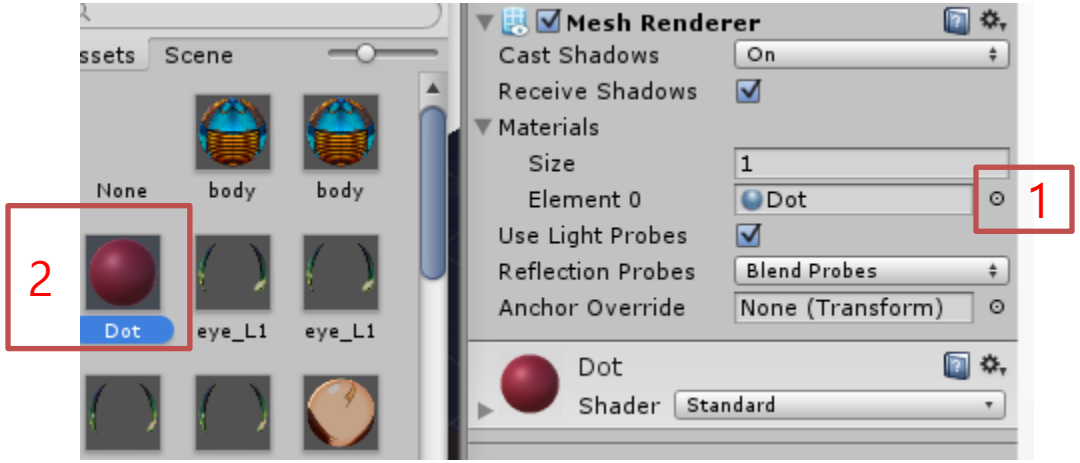


4. 동전 아이템 만들기

플레이어가 게임을 하면서 먹을 동전 아이템 제작

4) 매트리얼 지정

- Dot 오브젝트를 선택하고 Inspector항목에서 Mesh Renderer – Materials – Element 0을 만들어 놓은 Dot로 지정

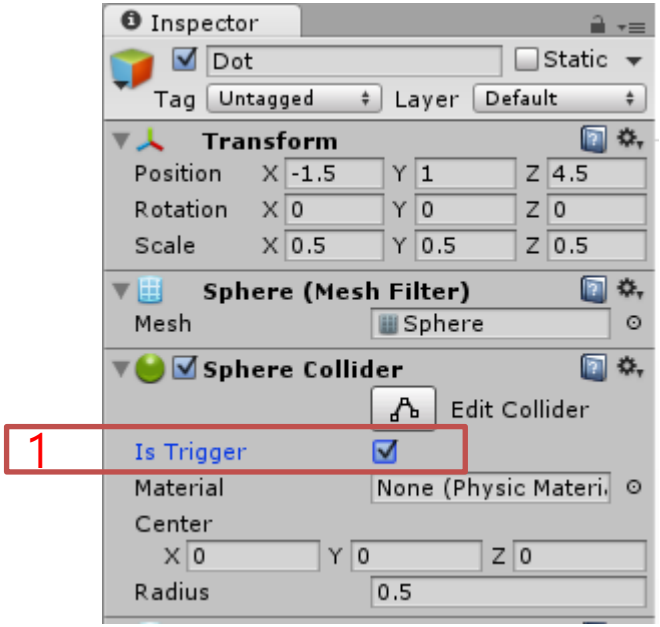


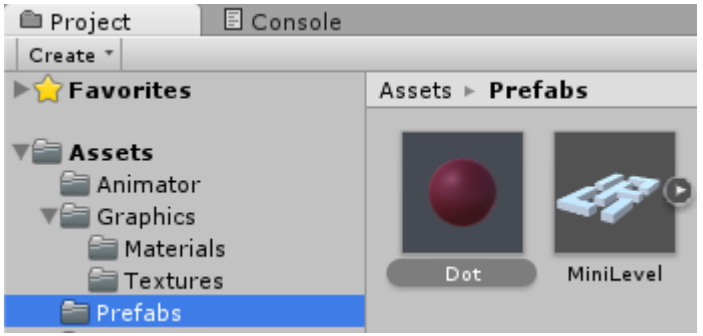
5. 플레이어가 동전을 획득하도록 구현

동전 객체도 하나의 장애물로 플레이어의 진로를 막지 않도록 해야 한다.

1) 콜라이더를 트리거로 지정하기

- Dot 오브젝트의 Sphere Collider에서 Is Trigger 항목을 체크한다.
  - 앞으로 많이 사용되는 Dot객체는 프리팹으로 저장해 두면 생성하기 편리하다.
  - 프리팹 만들기: 계층 뷰의 객체를 Assets₩Prefabs 폴드에 드래그한다.





5. 플레이어가 동전을 획득하도록 구현

2) 동전을 획득하는 과정 코딩

- 플레이어가 트리거 동전과 충돌하면 동전을 씬에서 보이지 않도록 삭제하도록 구현
- Update() 아래에 추가

```
// Update is called once per frame
void Update () {
. . .
}

    void OnTriggerEnter (Collider other) {
        Destroy(other.gameObject);
    }
// 게임오브젝트가 트리거와 충돌하면 OnTriggerEnter()가 호출하
고
// Destroy()의 인수에 동전을 넣어 씬에서 삭제한다.
```

6. 태그로 씐 안에 있는 Dot 개수 세기

게임에 남아 있는 동전 아이템의 개수를 세어 게임 점수를 매김

1) 태그 설정

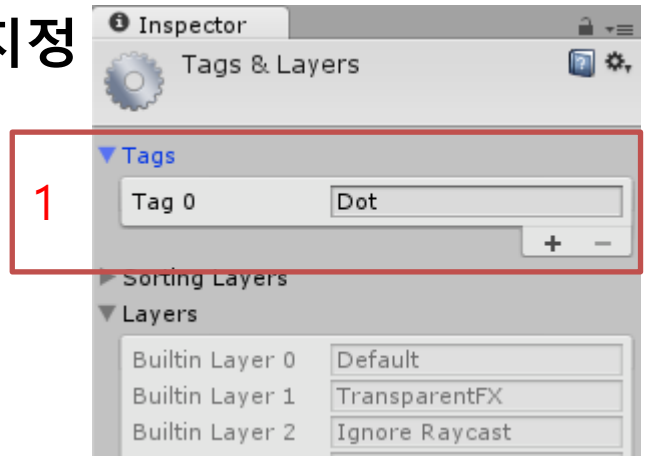
- Edit – Project Setting – Tags and Layers 클릭하여 Inspector에 표시

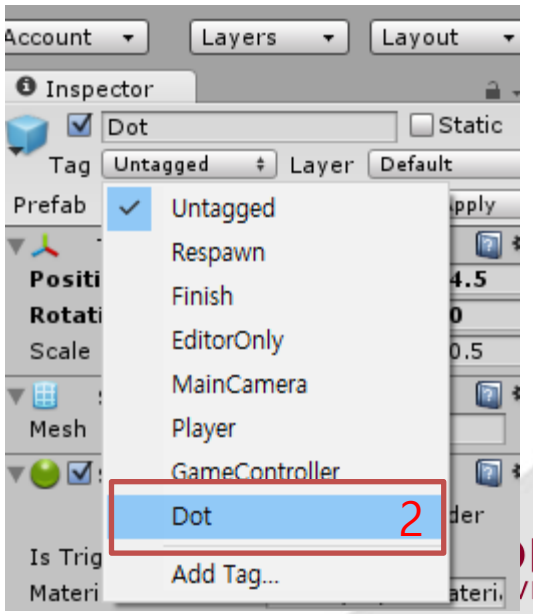
2) 태그 추가

- Tags에 새로운 Tag 0항목 추가
- New Tag를 Dot로 수정

3) Prefabs에 태그 지정

- 생성된 태그 적용





6. 태그로 씬 안에 있는 Dot 개수 세기

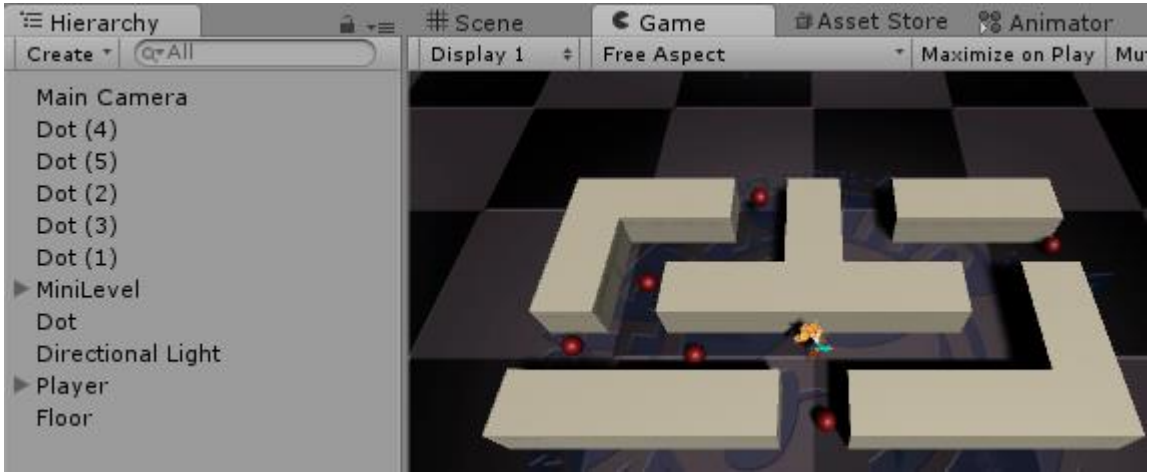
4) 승부 판정하기

- 씬에 남은 동전 아이템의 개수를 세어 게임의 승부를 판정하는 스크립터 작성 추가

```
animator.SetFloat("Speed", characterController.velocity.magnitude);

if (GameObject.FindGameObjectsWithTag("Dot").Length == 0) {
SceneManager.LoadScene("Main");
}
```

5) 게임 실행

- 6개의 동전 아이템을 추가해서 동작시켜 보자

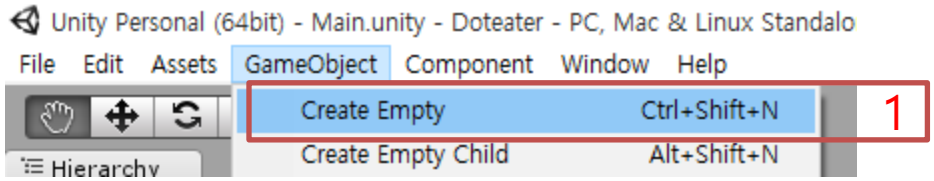


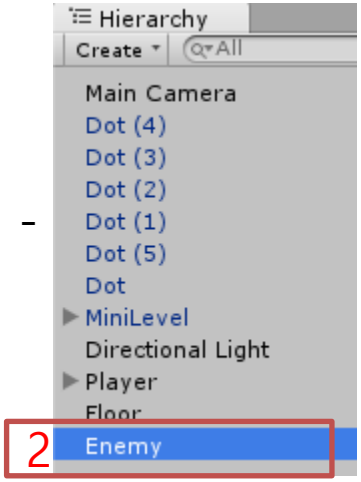
7. 적 만들기

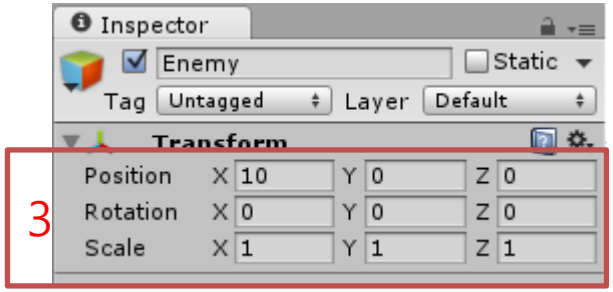
미로 안에서 플레이어를 쫓아오는 적을 생성

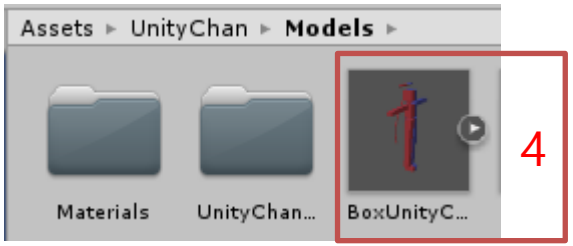
1) 적 Game Object 만들기

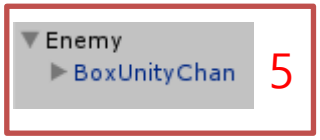
- 적 게임오브젝트 생성 및 이름변경
- 포지션 변경
- 유니티짱의 적 모델을 드래그 하여 Enemy에 적용









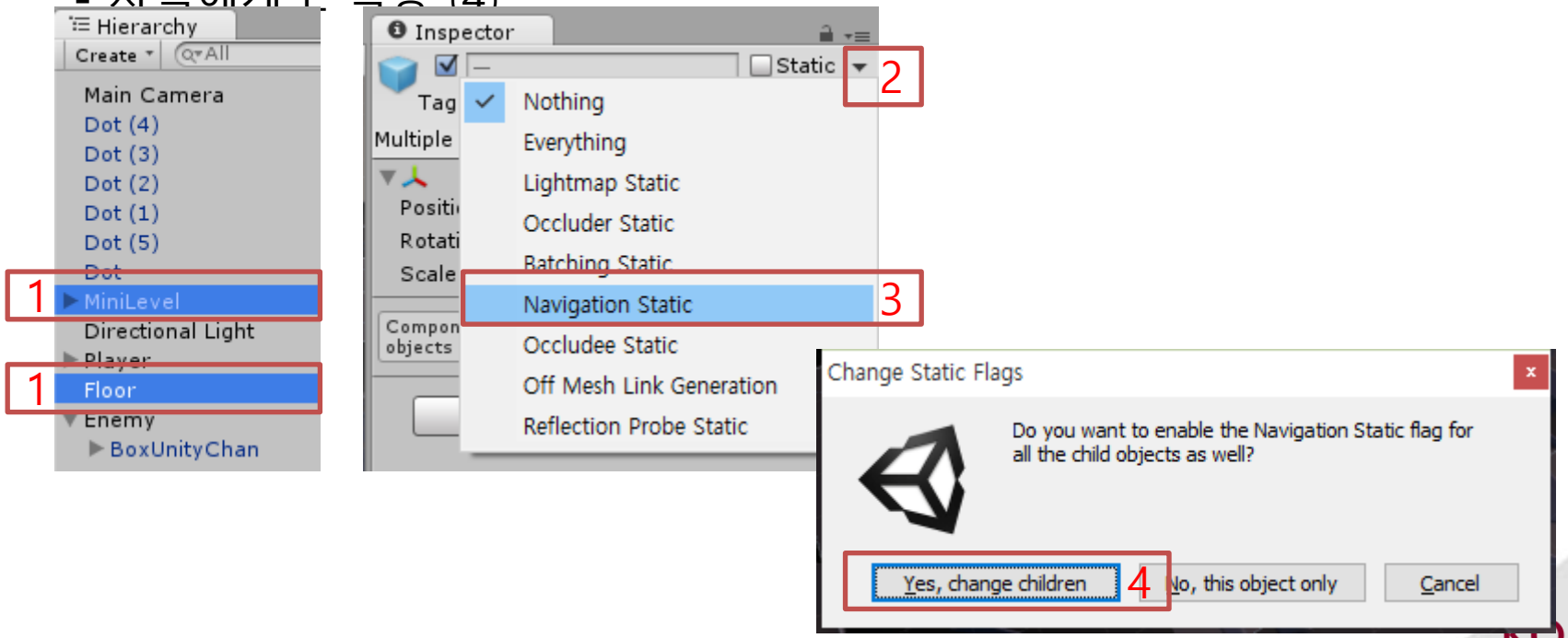


8. NavMesh 설정

플레이어가 이동할 수 있는 이동 범위를 설정

1) Navigation Static 지정

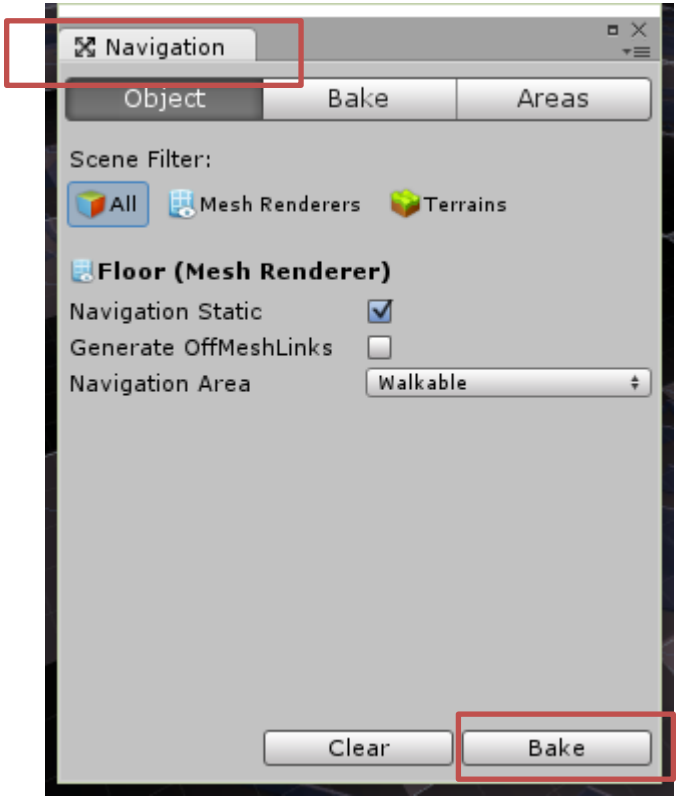
- 계층 뷰의 바닥과 벽 오브젝트를 함께 선택하고 Inspector의 Static를 선택
- Navigation Static을 지정 (3)
- 자신에게도 적용 (4)



8. NavMesh 설정

2) Navigation 뷰 열기

- Window - Navigation

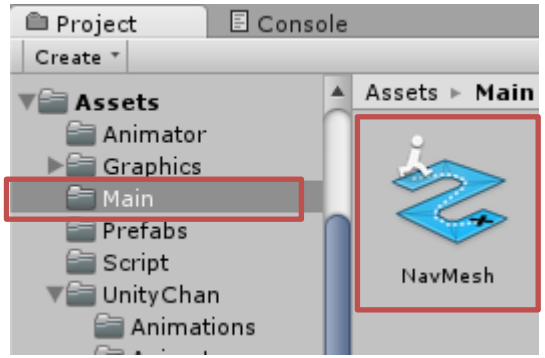


3) NavMesh 만들기

- Bake 버튼 클릭으로 생성
  - 이동 가능한 범위가 하늘색으로 표시됨
  - MavMesh.asset 파일로 저장됨

Keypoint

NavMesh를 만들 때 Navigation Static 으로 설정된 오브젝트만 NavMesh대상임.
오브젝트가 수정된 경우 Bake을 다시 적용해 주어야 함

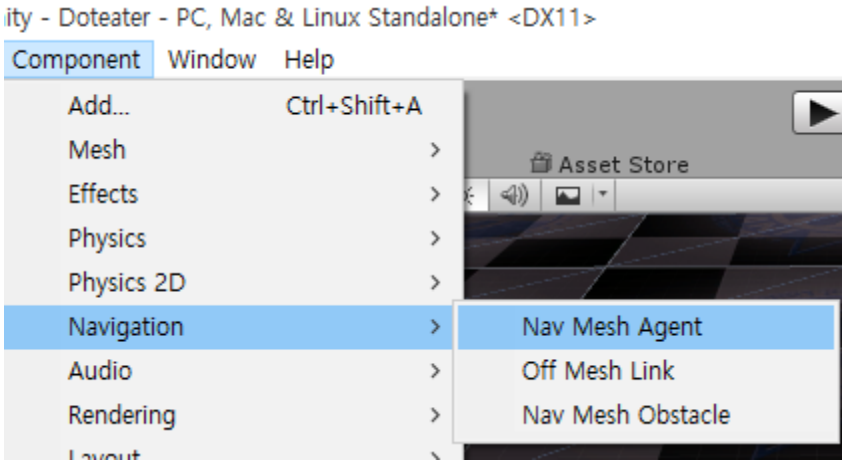


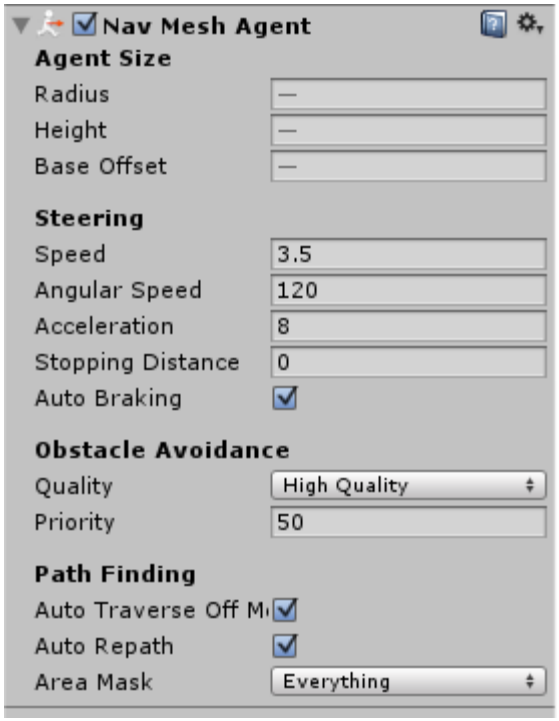
8. NavMeshAgent 설정

NaviMesh를 따라 이동하는 컴포넌트가 NavMeshAgent

1) NavMeshAgent 컴포넌트 추가하기

- Enemy를 선택하고 NavMeshAgent컴포넌트 추가





8. NavMeshAgent 설정

2) 적 스크립트

- Enemy.cs

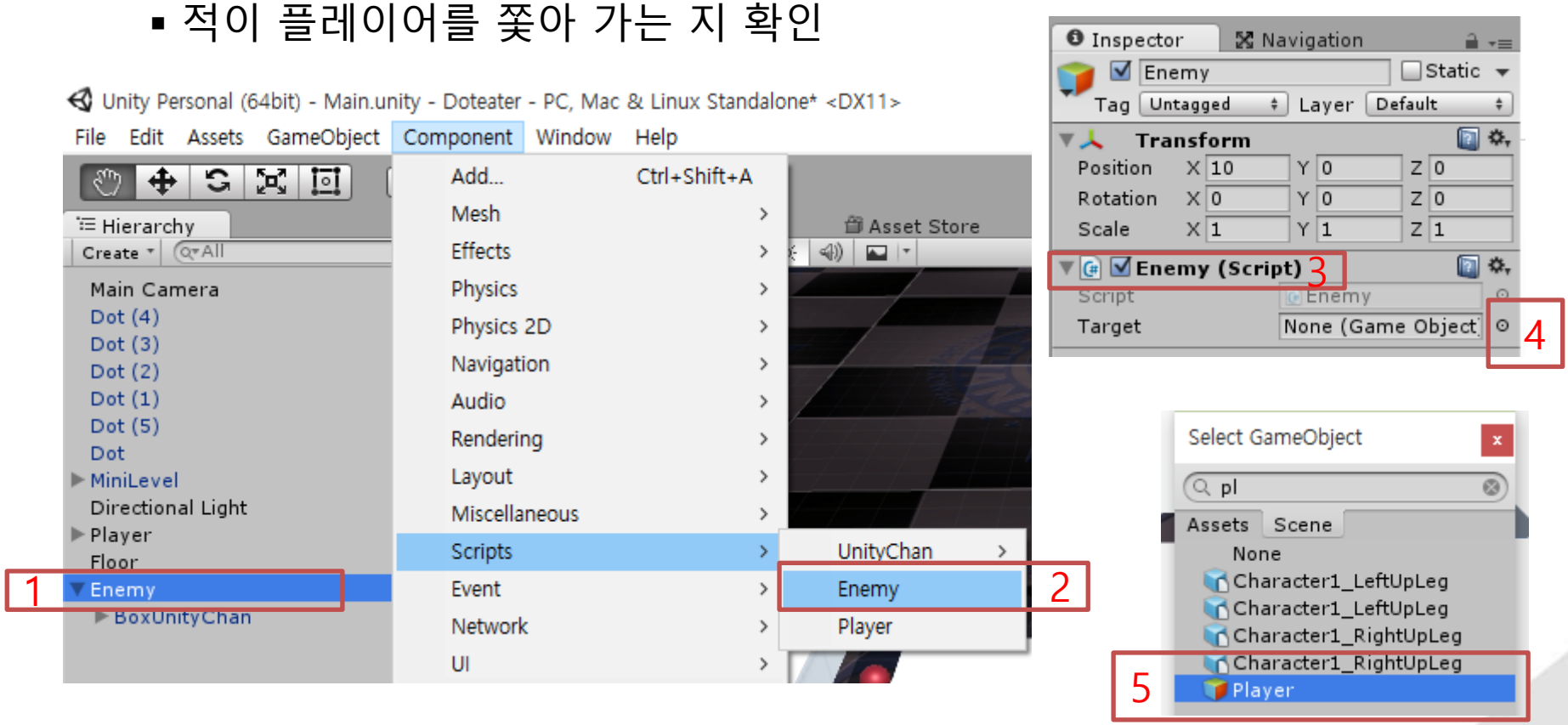
```
using UnityEngine;
using System.Collections;
Using UnityEngine.AI; // 추가해야 함

public class Enemy : MonoBehaviour {
        public GameObject target;
        NavMeshAgent agent;
        // Use this for initialization
        void Start () {
                agent = GetComponent<NavMeshAgent>();
        }
        // Update is called once per frame
        void Update () {
                agent.destination = target.transform.position;
        }
}
```

8. NavMeshAgent 스크립트 사용 설정

적이 플레이어를 쫓아 가도록 설정

1) 적 스크립트 사용 설정

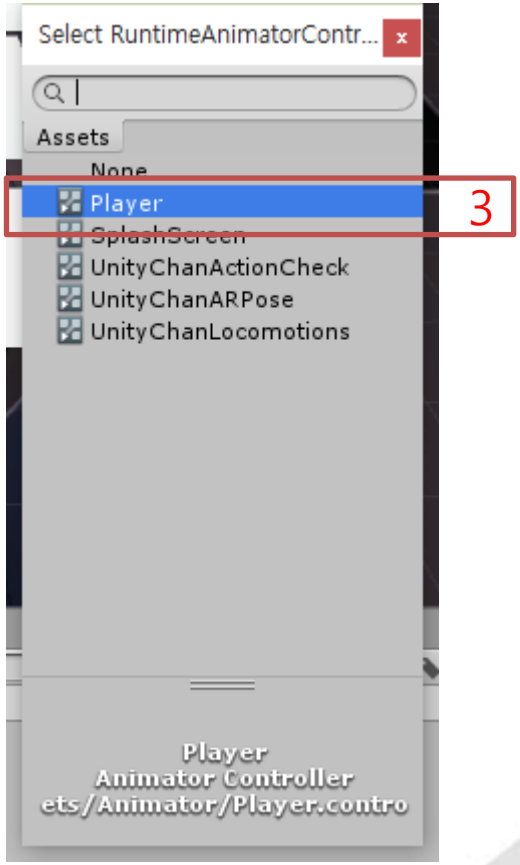
- Enemy 오브젝트 선택 후 Enemy 스크립트 적용 및 확인
- 적이 플레이어를 쫓아 가는 지 확인



8. 적 캐릭터에 애니메이션 추가

1) Animator 컴포넌트 설정

- 적 캐릭터에 달리는 모션 적용하기
  - 적 캐릭터에 플레이어의 모션을 적용
  - 루트 모션은 체크 해제



8. 적 캐릭터에 애니메이션 추가

2) 파라미터 전달

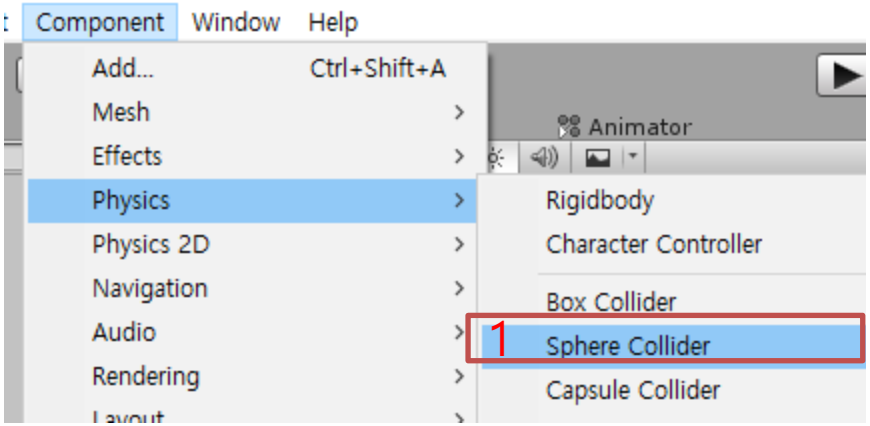
- NavMeshAgent의 이동속도를 유니티짱 Animator에 있는 속도값으로 전달
- Enemy.cs

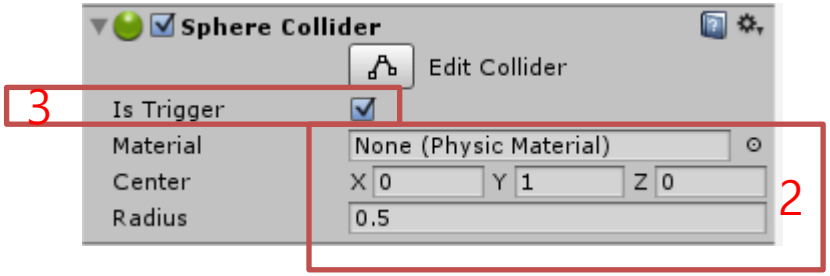
```
public GameObject target;
NavMeshAgent agent;
Animator animator;

// Use this for initialization
void Start () {
agent = GetComponent<NavMeshAgent>();
animator = GetComponentInChildren<Animator>();
}

// Update is called once per frame
void Update () {
agent.destination = target.transform.position;
animator.SetFloat("Speed", agent.velocity.magnitude);
}
```

9. 플레이어와 적 충돌 구현

1) 적에게 콜라이더 추가하기

- 공 모양의 콜라이더를 추가하고 공 위치 변경
- 트리거로 충돌처리 하도록 체크

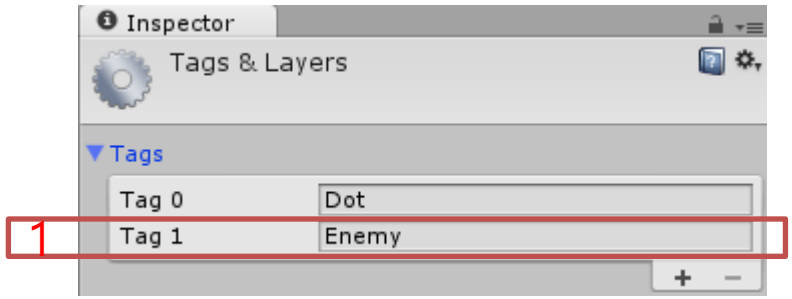


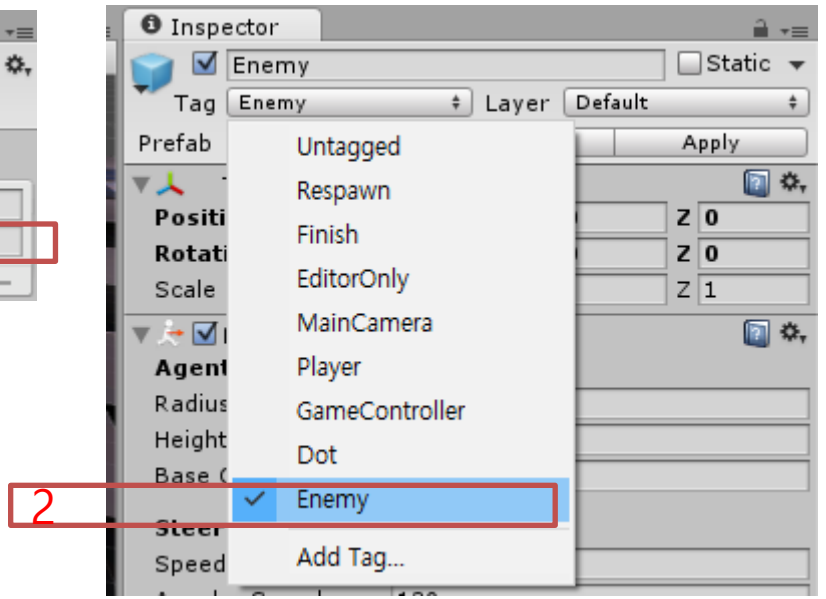


9. 플레이어와 적 충돌 구현

2) 적에게 태그 추가하기

- 충돌 대상이 동전인지 적인지 판별하기 위해 태그 사용
- Edit – Project Settings – Tags and Layers
- Tag 1을 추가 후 Enemy 태그를 설정 (1)
- 계층 뷰에서 Enemy를 선택한 후 Inspector에서 Enemy 선택 (2)





9. 플레이어와 적 충돌 구현

3) 플레이어 스크립트 편집 후 실행

- Player.cs에 다음 내용을 추가
- 플레이어와 충돌하면 씬을 다시 읽어들이도록 코드 수정

```
void OnTriggerEnter (Collider other) {
        if (other.tag == "Dot") {
                Destroy(other.gameObject);
        }
        if (other.tag == "Enemy") {
        SceneManager.LoadScene("Main");
        }
}
```