가상세계 **HW #2. Feeding Animals**

제 출 일	**2024. 5. 22.(**수**)**	전 공	소프트웨어학부
과 목	가상세계	학 번	**2017203082**
		이 름	김찬진

목차

1. Fulfil table

Easy1	Vertical Player Movement	O
Easy2	Animals increasing proportionally	O
Easy3	The food no longer moves	O
Easy4	Camera switcher	O
Medium	Animal Hunger Bar	O
Hard	Move to food	O
Expert1	Move freely	O
Expert2	Collision avoidance	O

- 과제로 제출한 패키지를 빈 프로젝트에 **import**해서 보니, **Food** 레이어가 계속 빠져 있습니다**. Layer**에 **Food** 를 추가한 후, 인스펙터에서 음식 프리팹**(**바나나, 샌드위치, 뼈다귀**)** 에 **Food** 레이어를 설정해야 합니다**….**

2. 각 요구사항에 대한 구현방법

(1) Vertical Player Movement

- PlayerControl 스크립트, Player에 연결됨

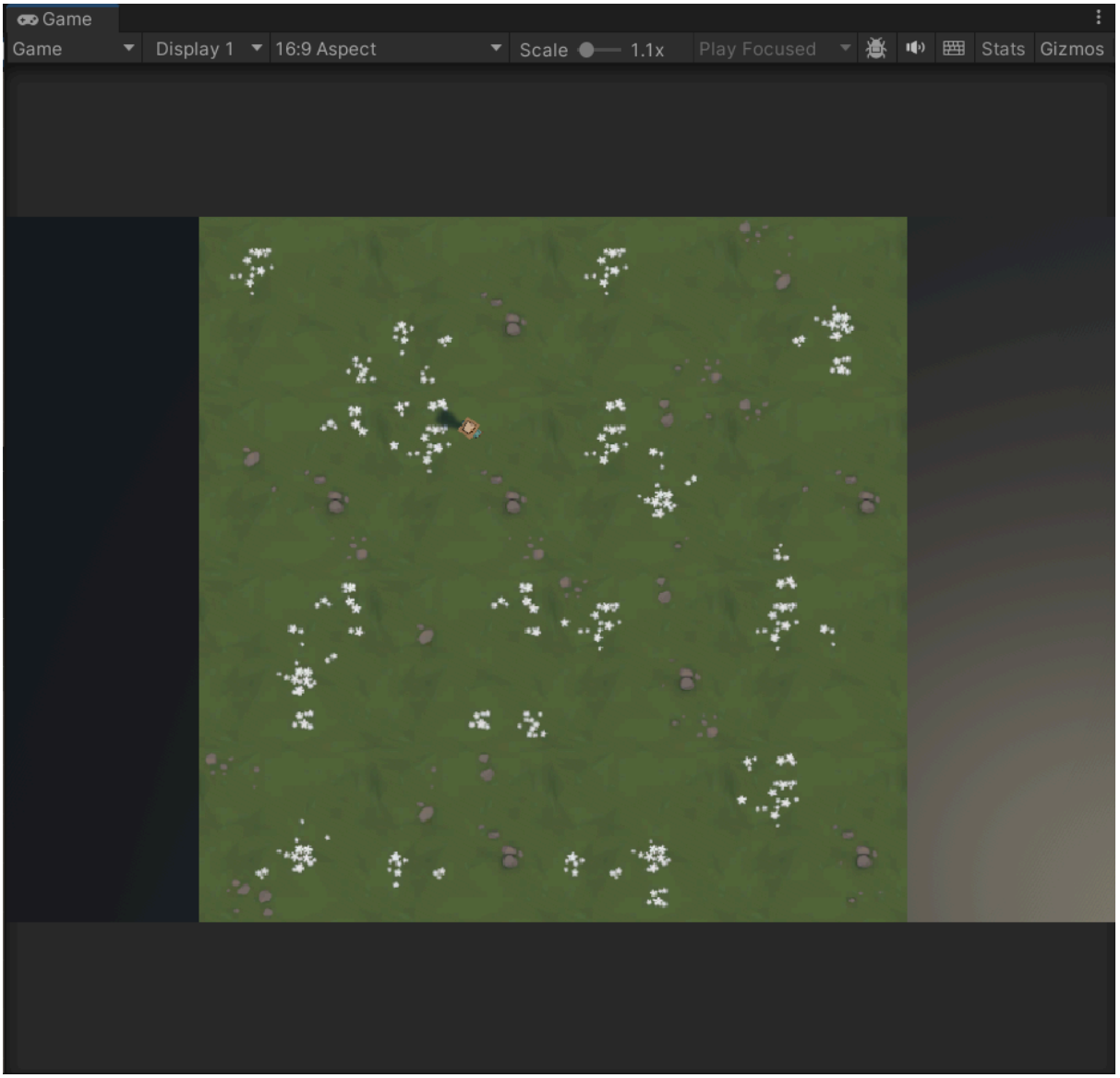
- 키보드의 좌우 방향키로 Player 오브젝트의 회전을 입력하며, 상하 방향키로 Player 오브젝트의 전진,후진을 입력합니다.

```
public void MovePlayer()
{
    horizontalInput = Input.GetAxis("Horizontal");
    verticalInput = Input.GetAxis("Vertical");
    transform.Translate(Vector3.forward * verticalInput * Time.deltaTime * speed);
    transform.Rotate(Vector3.up, Time.deltaTime * turnspeed * horizontalInput);

    if (transform.position.x < -xRange)
    {
        transform.position = new Vector3(-xRange, transform.position.y, transform.position.z);
    }
    if (transform.position.x > xRange)
    {
        transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
    }

    if (transform.position.z < zRangeDown)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y,zRangeDown);
    }
    if (transform.position.z > zRangeUp)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y,zRangeUp);
    }

}
```

- MovePlayer 함수를 매 프레임마다 호출하여, 사용자의 키보드 입력을 받고 Player 오브젝트가 움직이게 합니다.

- xRange, zRangeUp, zRangeDown 변수를 통해 플레이어가 주어진 맵 공간 내에서 움직이도록 제한합니다.

- speed 변수를 통해 전진,후진 속도를 제어합니다
- turnSpeed 변수를 통해 회전 속도를 제어합니다.



- 플레이어의 위치는, 이미 주어진 맵의 크기 내에서만 변경시킬 수 있습니다.

(2) Animals increasing proportionally

- AnimalSpawn 스크립트, AnimalSpawnManager에 연결됨

- AnimalSpawnManager 오브젝트를 통해 미리 Prefab으로 만들어 놓은 4가지 동물 오브젝트를 랜덤으로 생성합니다.

- currentAnimalCount 변수를 사용하여, 현재 spawn된 동물의 수를 확인합니다. currentAnimalCount 변수는 2로 초기화되어, 처음 시작 시에는 2마리의 동물이 spawn됩니다.

```
public void SpawnAnimal()
{

    if (start ==0)
    {
        for (int i = 0; i < currentAnimalCount; i++)
        {
            RandomSpawn();
        }
        start++;
    }
    else
    {
        count = IncreasingProportion(currentAnimalCount);
        for (int i = 0; i < count; i++)
        {
            RandomSpawn();
        }

        currentAnimalCount += count;
    }
    Debug.Log(currentAnimalCount);
}
```

- SpawnAnimal 함수에서는, 맨 처음에는 2마리의 동물만 생성하기 위해 start변수를 사용하여, start ==0인 경우 첫번째 조건문이 실행되도록 하였습니다. 이후에는 항상 else 문이 실행됩니다.

- 시작 시 첫번째 조건문이 실행되고 난 이후, else 문에서는 현재 생성된 동물 오브젝트의 수(currentAnimalCount)에 비례하여 새롭게 생성될 동물의 수를 결정하는 IncreasingProportion 함수가 사용됩니다. IncreasingProportion함수의 리턴값으로 받아온 수는 count를 갱신시켜 count 값 만큼 새롭게 동물이 생성됩니다.

- 이후, currentAnimalCount 에는 count가 더해져 현재 생성된 동물 수가 올바르게 갱신됩니다.

```
public int IncreasingProportion(int n)
{
    float proportion = 0.5f;
    return Mathf.FloorToInt(n * proportion);
}
```

- IncreasingProportion 함수는, 현재 생성되어 있는 동물 수의 50%를 리턴하는 함수입니다. 이를 통해 현재 생성된 동물 수의 50%에 해당하는 동물이 새롭게 생성됩니다.(ex 현재 생성된 동물 수가 10이면 5마리 생성, 현재 생성된 동물 수가 20이면 10마리 생성)

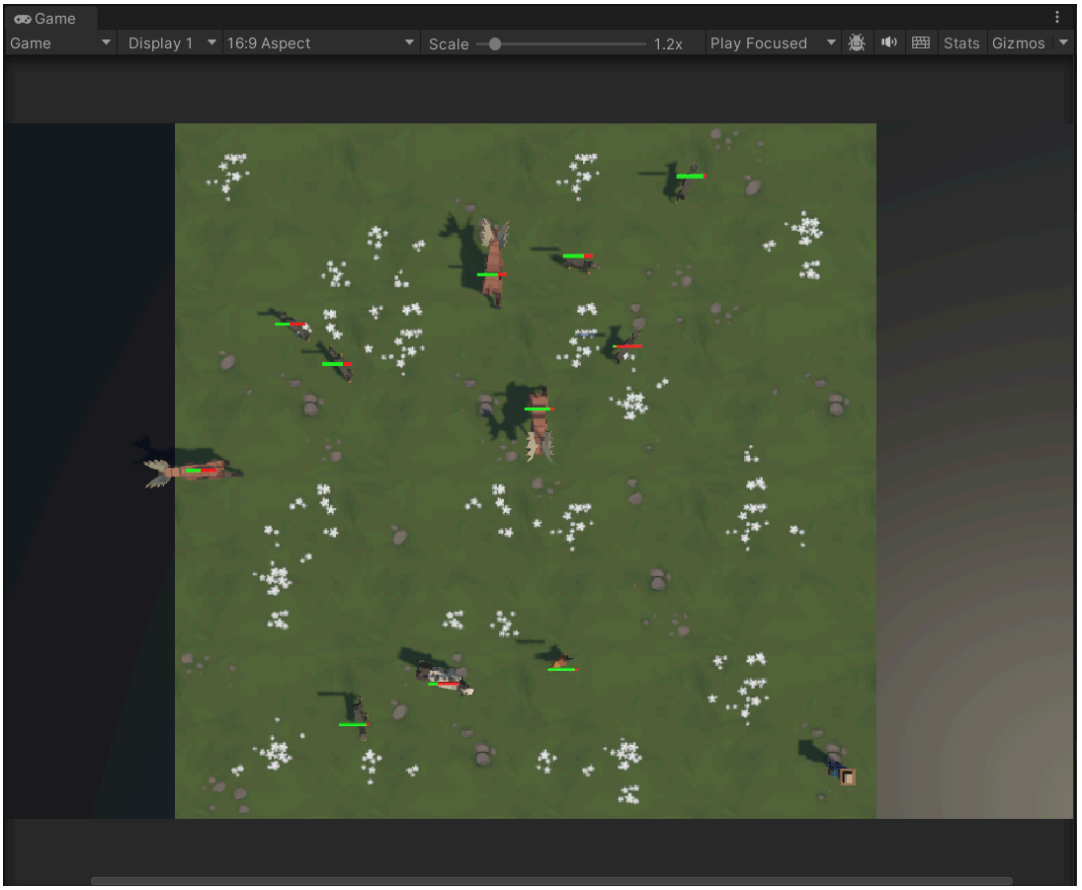
```
public void RandomSpawn()
{
    float randomX = UnityEngine.Random.Range(-22.0f, 22.0f);
    float randomZ = UnityEngine.Random.Range(-11.0f, 30.0f);
    Vector3 spawnPoint = new Vector3(randomX, 0f, randomZ);
    int animalIndex = UnityEngine.Random.Range(0, animalPrefabs.Length);

    Instantiate(animalPrefabs[animalIndex], spawnPoint, animalPrefabs[animalIndex].transform.rotation);
}
```

- 랜덤한 위치에 동물을 생성하는 함수입니다. randomX,randomY는 주어진 필드 범위 내의 값을 랜덤하게 설정하고, 이 값들로 spawnPoint를 생성하여 동물이 랜덤한 위치에 생성되게 됩니다.

```
void Update()
{
    if(currentAnimalCount < maxAnimalCount) {
        if (Time.time >= nextSpawnTime)
        {
            SpawnAnimal();
            nextSpawnTime = Time.time + spawnInterval;
        }
    }
}
```

- 마지막으로 Update 함수에서는 현재 동물 수가 최대 동물 수(maxAnimalCount) 보다 작을 경우, spawnInterval 간격으로 계속해서 SpawnAnimal 함수를 호출하게됩니다.

- 아래는 예시 사진입니다.



(3) The food no longer moves

- FoodSpawn 스크립트, FoodSpawnManager에 연결됨

- FoodSpawn 오브젝트는, Prefab으로 준비되어 있는 Food 오브젝트를 주어진 필드 범위 내에 무작위로 생성합니다. 생성된 Food 오브젝트는 생성된 자리에 고정됩니다.

```
public void SpawnFood()
{
    if (foodCount < maxFoodCount)
    {
        float randomX = UnityEngine.Random.Range(-22.0f, 22.0f);
        float randomZ = UnityEngine.Random.Range(-11.0f, 30.0f);

        int foodIndex = UnityEngine.Random.Range(0, foodPrefabs.Length);

        Vector3 spawnPoint = new Vector3(randomX, 0.2f, randomZ);

        Instantiate(foodPrefabs[foodIndex], spawnPoint, foodPrefabs[foodIndex].transform.rotation);
        foodCount++;
    }
    else
    {
        CancelInvoke("SpawnFood");
        isSpawning = false;
    }
}
```

- SpawnFood 함수를 통해 Food가 생성되며, 현재 음식 수(foodCount)가 최대 음식 수(maxFoodCount) 보다 작을 경우에만 Food가 생성됩니다. Food는 1개씩 생성됩니다.

- 이번에는 수업중 배운 InvokeRepeating 함수를 활용했습니다.

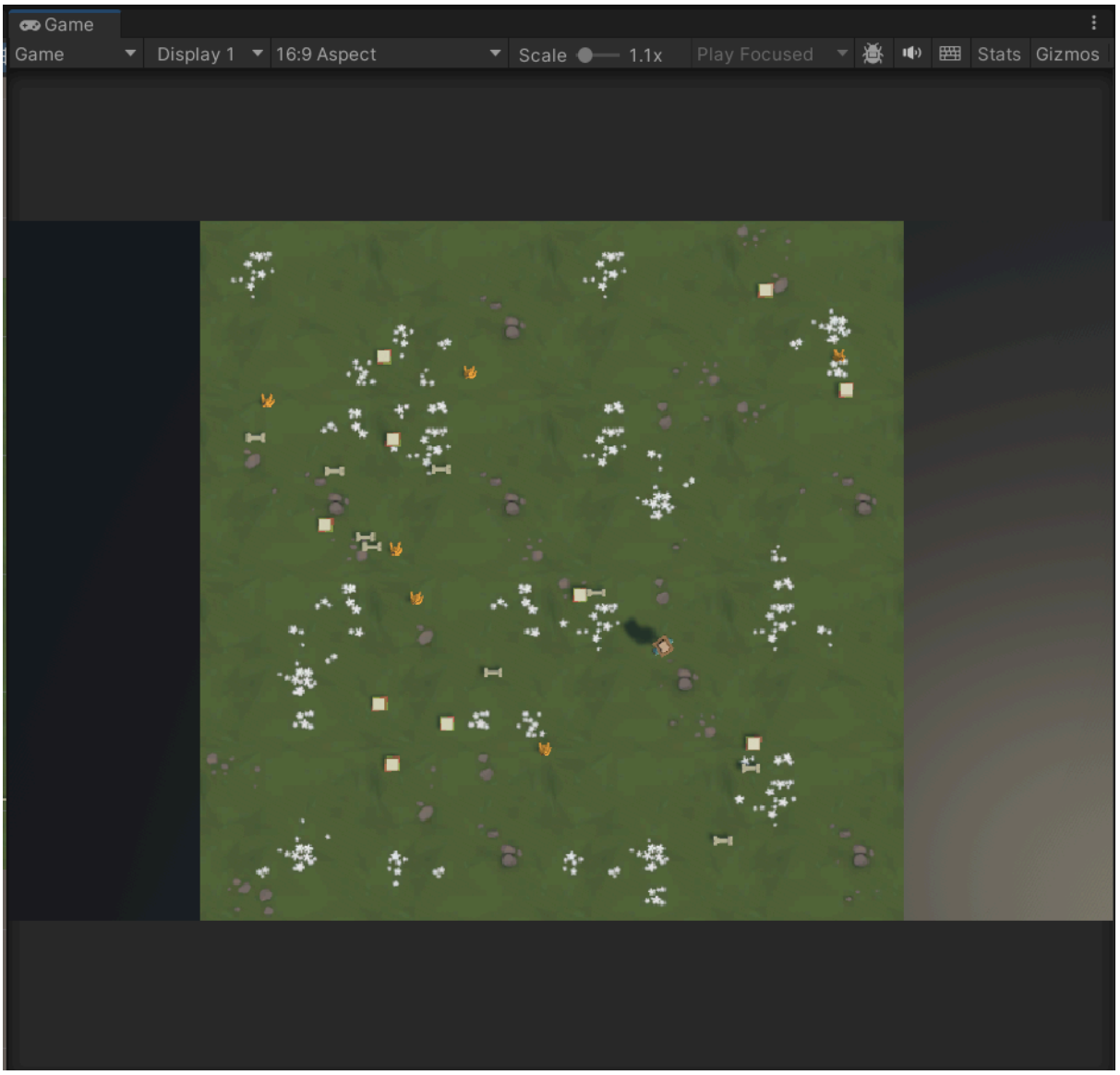
```
void Start()
{
    InvokeRepeating("SpawnFood", 2, 0.5f);
}

// Update is called once per frame
Unity 메시지 | 참조 0개
void Update()
{
    if (foodCount < maxFoodCount && !isSpawning)
    {
        InvokeRepeating("SpawnFood", 0, 0.5f);
        isSpawning = true;
    }
}
```

- 현재 음식 수가 최대 음식 수를 넘어 CancelInvoke를 통해 반복호출이 중단될 경우, 이후에 동물이 음식을 먹어 현재 음식 수가 최대 음식 수보다 줄어들어도 이미 InvokeRepeating 함수가 중지되어 음식이 생성되지 않는 문제가 있었습니다.

- 이를 Update 함수 내에 조건문을 넣어 해결하였습니다.
- 현재 음식 수가 최대 음식 수보다 작고, isSpawning 변수가 false라면, 다시 InvokeRepeating을 호출합니다.

- 매 프레임마다 InvokeRepeating 가 호출되는 것을 방지하기 위해 isSpawning 변수를 사용하였습니다.

- 아래는 음식이 최대로 스폰된 사진입니다.



(4) Camera switcher

- CameraView 스크립트, Main Camera에 연결

- 기본적으로 (0,60,0) 포지션에서 아래를 내려다보는 탑다운 뷰를 사용하며, 왼쪽 시프트를 눌러 1인칭 뷰(Player) 로 전환할 수 있습니다.

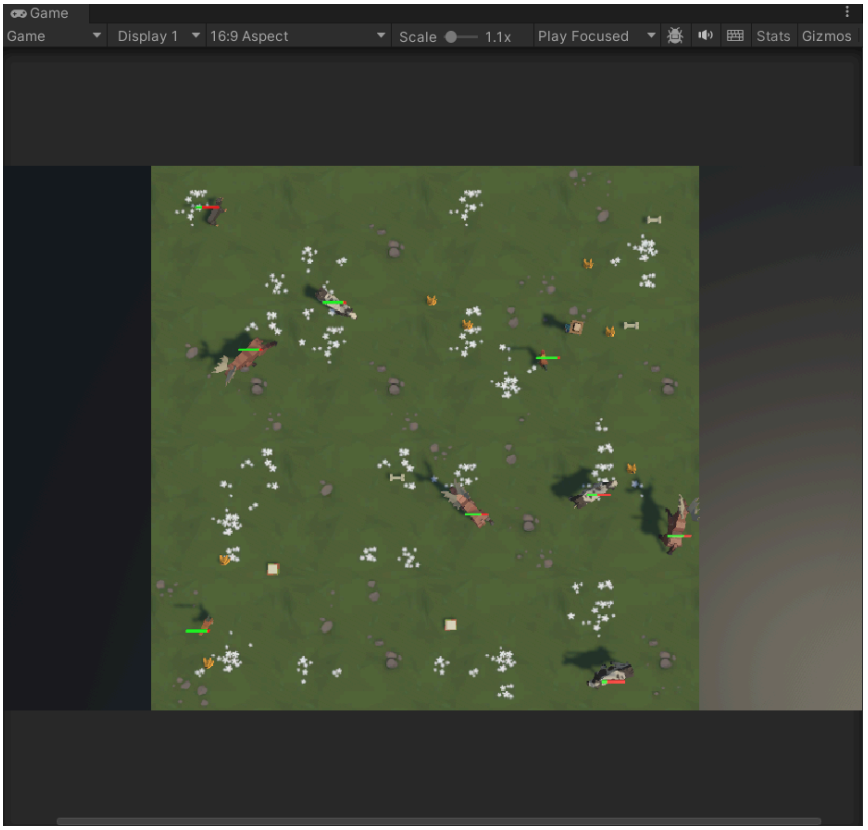
```
public void CameraSwitch()
{
    if (Input.GetKeyDown(KeyCode.LeftShift))
    {
        CameraNumber = (CameraNumber == 0) ? 1 : 0;
    }
}
```

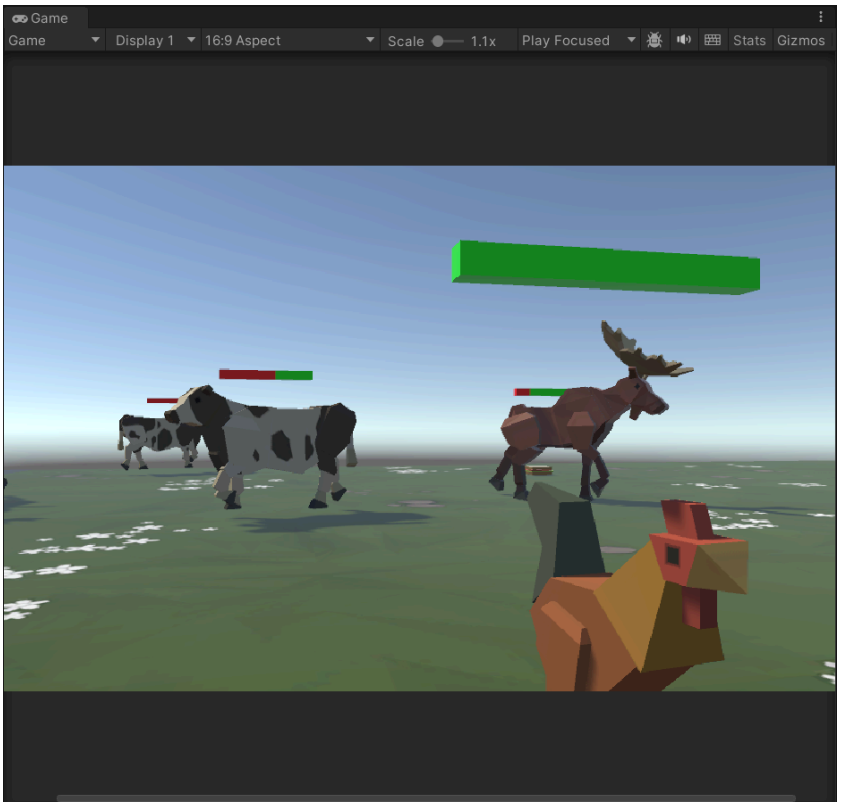
- CameraSwitch 함수를 통해, 왼쪽 시프트를 누를 때마다 CameraNumber의 값을 0,1로 바꿉니다.

```
public void CameraType()
{
    if (CameraNumber == 1)
    {
        Vector3 offset1;
        Vector3 offset2;
        Vector3 lookAtPosition;
        offset1 = Player.transform.forward;
        offset2 = Player.transform.forward*2;
        offset1.y += 2.0f;
        offset2.y += 1.0f;
        lookAtPosition = Player.transform.position + 2 * offset2;

        transform.position = Player.transform.position + offset1;
        transform.LookAt(lookAtPosition);
    }
    else if (CameraNumber == 0)
    {

        transform.position = topView;
        transform.rotation = topRotation;
    }
}
```

- 지난번 과제와 마찬가지로, 탑다운 뷰는 미리 start함수에서 초기화한 topView, topRatation을 사용합니다. 1인칭 뷰는 먼저 카메라를 player 약간 앞에 고정시키고(offset1 사용), 고정된 카메라가 lookAtPosition(offset2 사용 - offset1좌표의 2배 거리의 위치좌표) 을 계속 바라보게 하여 구현하였습니다.

- 아래는 순서대로 탑다운 뷰, 1인칭 뷰 입니다.





(5) Animal Hunger Bar

- HungerBarController 스크립트, HungerBar 프리팹에 연결
- AnimalBehaivor 스크립트, 모든 동물 프리팹에 연결

- HungerBar는 동물의 체력을 의미합니다. HungerBar는 10개의 큐브 오브젝트(segment)로 이루어져 있습니다. 매쉬 렌더러를 사용하여, 각 세그먼트의 material의 색상을 초록색 -> 빨간색으로 변경하면서 동물의 체력이 줄어드는 듯한 효과를 내었습니다.

- 동물과 함께 스폰되는 HungerBar는, 맨 처음 모든 세그먼트가 초록색으로 변경되었다가, 1.5f 의 간격마다 한칸 씩 빨간색으로 변경됩니다.

- HungerBarController 스크립트

```
void Start()
{

    AnimalBehaivor = GetComponent<AnimalBehaivor>();
    segmentRenderers = GetComponentsInChildren<MeshRenderer>();

    foreach (var renderer in segmentRenderers)
    {
        renderer.material.color = startColor;
    }

    InvokeRepeating("HungerBarState", 2, 1.5f);
}
```

- 맨 처음, HungerBar 의 모든 세그먼트가 startColor(초록색)으로 초기화되고, 이후 HungerBarState 가 1.5f 마다 호출됩니다.

- HungerBarController 스크립트

```
public void HungerBarState()
{
    if (hungerBarNumber < segmentRenderers.Length)
    {
        segmentRenderers[hungerBarNumber].material.color = endColor;
        hungerBarNumber++;
    }

    else { CancelInvoke("HungerBarState"); }
}
```

- HumgerBarState함수는, 모든 세그먼트를 담고 있는 segmentRenderers 배열과 현재 세그먼트의 인덱스를 나타내는 hungerBarNumber를 사용합니다. 현재 세그먼트 인덱스에 해당하는 세그먼트의 색을 endColor(빨간색) 으로 바꾸어, HungerBar 의 색을 점차 빨간색으로 변화시킵니다.

- HungerBar를 이루는 세그먼트는 총 10개입니다.

- AnimalBehaivor 스크립트

```
void Start()
{
    hungerBar = Instantiate(hungerBar, transform.position + hungerBarOffset, Quaternion.identity);
    hungerBarController = hungerBar.GetComponent<HungerBarController>();
```

```
public void UpdateHungerBar()
{

    hungerBar.transform.position = transform.position + hungerBarOffset;

}
```

- 동물의 체력을 의미하는 HungerBar는 모든 동물 오브젝트에 부착되어 있습니다.(Start 함수에서, HungerBar를 스폰하고, 이후 Update함수에서 UpdateHungerBar 함수를 사용하여 매 프레임 HungerBar의위치를 갱신합니다.)

- AnimalBehaivor 스크립트

```
public void AnimalDie()
{
    if(hungerBarController.hungerBarNumber == hungerBarController.segmentRenderers.Length)
    {
        Destroy(gameObject);
        Destroy(hungerBar);
        animalSpawn.currentAnimalCount--;
    }
}
```

- HungerBar가 부착된 동물의 HungerBar가 모두 빨간색으로 변하면 해당 동물은 사라지게 됩니다.
- 또한 AnimalSpawn 스크립트를 참조하여, 동물이 사라지게 될 경우 currnetAnimalCount의 값을 1 내립니다.

- AnimalBehaivor 스크립트

```
public void OnTriggerEnter(Collider other)
{
    if(other.CompareTag("Food"))
    {
        Destroy(other.gameObject);
        foodSpawn.foodCount--;
        if(hungerBarController.hungerBarNumber > 0)
        {
            hungerBarController.segmentRenderers[hungerBarController.hungerBarNumber].material.color = Color.green;
            hungerBarController.hungerBarNumber--;
            Debug.Log("Hb1");
        }
        if (hungerBarController.hungerBarNumber > 5)
        {
            hungerBarController.segmentRenderers[hungerBarController.hungerBarNumber].material.color = Color.green;
            hungerBarController.hungerBarNumber--;
            Debug.Log("Hb2");
        }

    }
```

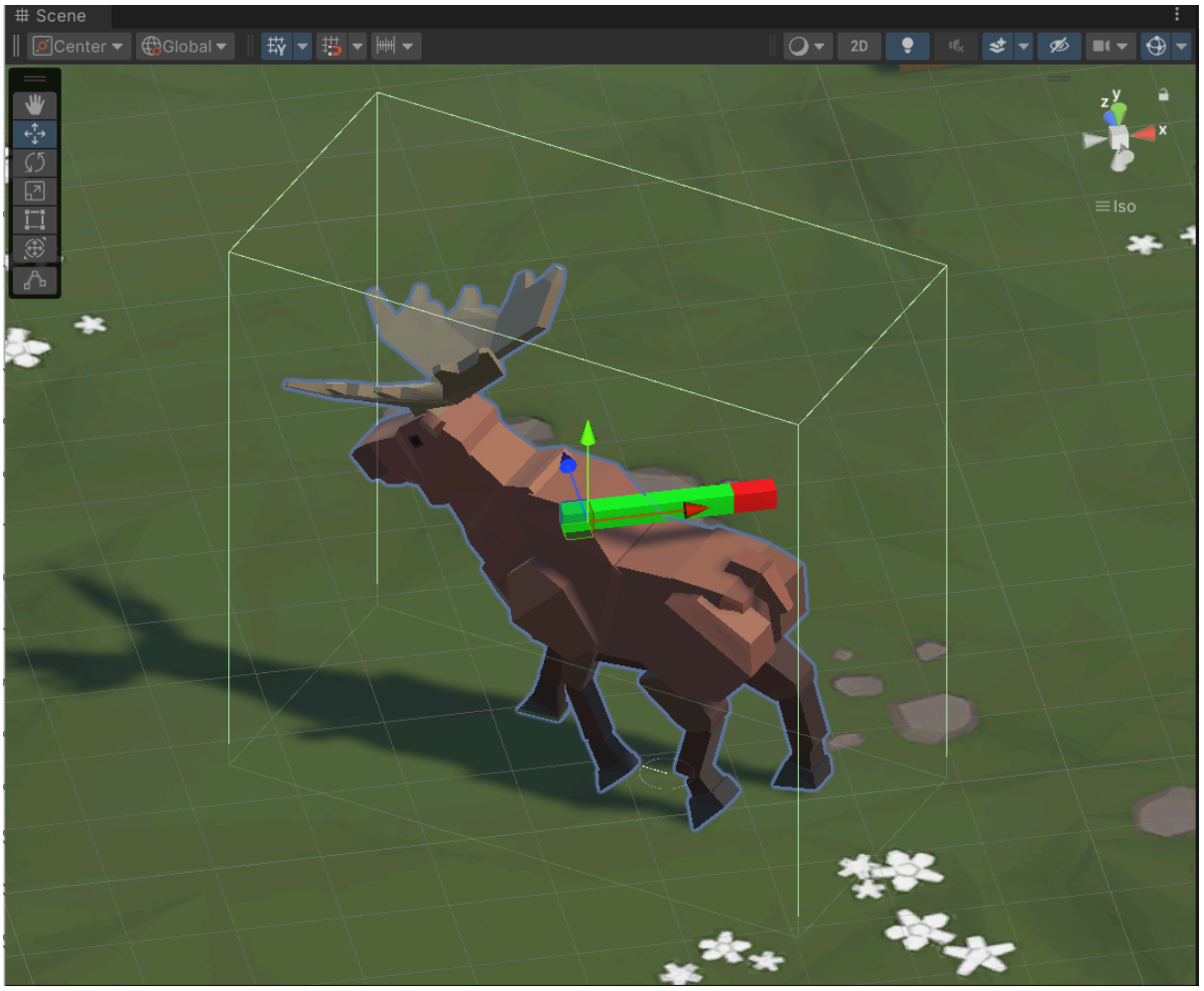
- 만약 동물이 Food 오브젝트와 충돌하면 해당 동물의 HungerBar가 일정 갯수만큼 빨간색 -> 초록색 으로 돌아오게 됩니다.

- 만약, 현재 동물의 HungerBar의 세그먼트에서 빨간색에 해당하는 부분이 7개 이상인 경우, 2칸을 초록색으로 되돌립니다. 그 이외의 경우는 1칸만 초록색으로 되돌림.

- HungerBar가 올바르게 업데이트 되게 하기위하여, 초록색으로 세그먼트 색을 바꿀 때마다 hungerBarNumber의 값을 1 내립니다.

- 디버그 로그를 사용하여 올바르게 작동하는지 확인한 결과, 올바르게 작동합니다.

- 아래는 Hungerbar 의 예시 사진입니다.



(6) Move to food

- AnimalBehaivor 스크립트, 모든 동물 프리팹에 연결

- 동물은 자신의 위치로부터 일정 거리 내에 Food 레이어가 설정된 오브젝트를 탐지하면, 그 오브젝트로 다가가게 됩니다.

```
public void AnimalMoving()
{
    Collider[] foodObjects = Physics.OverlapSphere(transform.position, detectionRange, LayerMask.GetMask("Food"));
    if (foodObjects.Length > 0)
    {
        targetFood = foodObjects[0].transform;
        float closestDistance = Vector3.Distance(transform.position, targetFood.position);

        foreach (Collider foodObject in foodObjects)
        {
            float distance = Vector3.Distance(transform.position, foodObject.transform.position);
            if (distance < closestDistance)
            {
                targetFood = foodObject.transform;
                closestDistance = distance;
            }
        }
        Vector3 direction = targetFood.position - transform.position;
        direction.Normalize();

        Quaternion targetRotation = Quaternion.LookRotation(direction);
        transform.rotation = Quaternion.Slerp(transform.rotation, targetRotation, Time.deltaTime * animalSpeed);
        transform.Translate(Vector3.forward * Time.deltaTime * animalSpeed);


    }
```

- AnimalMoving 함수에서는 먼저, 현재 오브젝트(동물)을 기준으로 반지름이 detectionRange인 원형반경 내의 Layer가 Food인 오브젝트를 탐지하여 foodObjects 배열에 추가합니다.

- 이후, targetFood 변수를 사용하여 foodObject 배열 내의 모든 Food 오브젝트들 중, 현재 오브젝트(동물)와 가장 가까운 Food 오브젝트를 foreach문으로 찾아냅니다.(targetFood를 갱신시킴)

- 이후, 현재 오브젝트(동물)와 targetObject의 위치의 차 를 통해, Direction을 계산하고, 동물 오브젝트가 해당 방향을 바라보게 하면서 targetObject로 다가가게 합니다.

- 여기서 Slerp 함수를 사용하여, 동물의 현재 회전값이 목표 회전값(targetRotation)까지 부드럽게 도달하게 합니다.

- 이를 통해, 동물은 자신의 음식 탐지 범위(detectionRange) 내에 음식이 있다면, 탐지 범위 내의 모든 음식을 탐지하고 그 중 가장 가까운 음식을 향해 다가가게 됩니다.

- 아래는 예시 사진입니다.

(7) Move freely

- AnimalBehaivor 스크립트, 모든 동물 프리팹에 연결

- 모든 동물오브젝트는 자신의 탐지 범위 내에 음식이 존재하지 않는 경우, 주어진 필드 영역 내에서 자유롭게 움직입니다.

- 자유로운 움직임을 구현하기 위해, 동물 오브젝트의 Y축 기준 회전값을 매 프레임 랜덤하게 설정하였습니다. 또한 매 프레임 동물은 자신의 전방 방향(forward)를 향해 일정한 속력(animalSpeed)으로 나아갑니다. 새롭게 설정된 회전값에 도달하기 위한 회전 속도(animalTrunSpeed)는 70.0f 로 초기화하였습니다.

```
    else
    {
        transform.Rotate(Vector3.up, Time.deltaTime * animalTrunSpeed * randomDirection);
        transform.Translate(Vector3.forward * Time.deltaTime * animalSpeed);

    }

    if (transform.position.x < -xRange)
    {
        transform.position = new Vector3(-xRange, transform.position.y, transform.position.z);

    }
    if (transform.position.x > xRange)
    {
        transform.position = new Vector3(xRange, transform.position.y, transform.position.z);
    }

    if (transform.position.z < zRangeDown)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y, zRangeDown);
    }
    if (transform.position.z > zRangeUp)
    {
        transform.position = new Vector3(transform.position.x, transform.position.y, zRangeUp);
    }
}
```

- 위 코드는, AnimalMoving함수의 else 문 입니다. 해당 함수의 if 문에서는 음식을 탐지하면 다가가는 부분이 작성되어 있습니다.

- 만약, 탐지 범위 내에 음식이 없을 경우 else문이 실행되어 동물은 랜덤한 값으로 회전하며, 앞으로 나아가게 됩니다.

- 또한 주어진 필드 범위 내에서만 움직이게 하기 위해 x,z축으로 범위 제한을 두어 해당 범위 내에서만 움직이게 하였습니다.

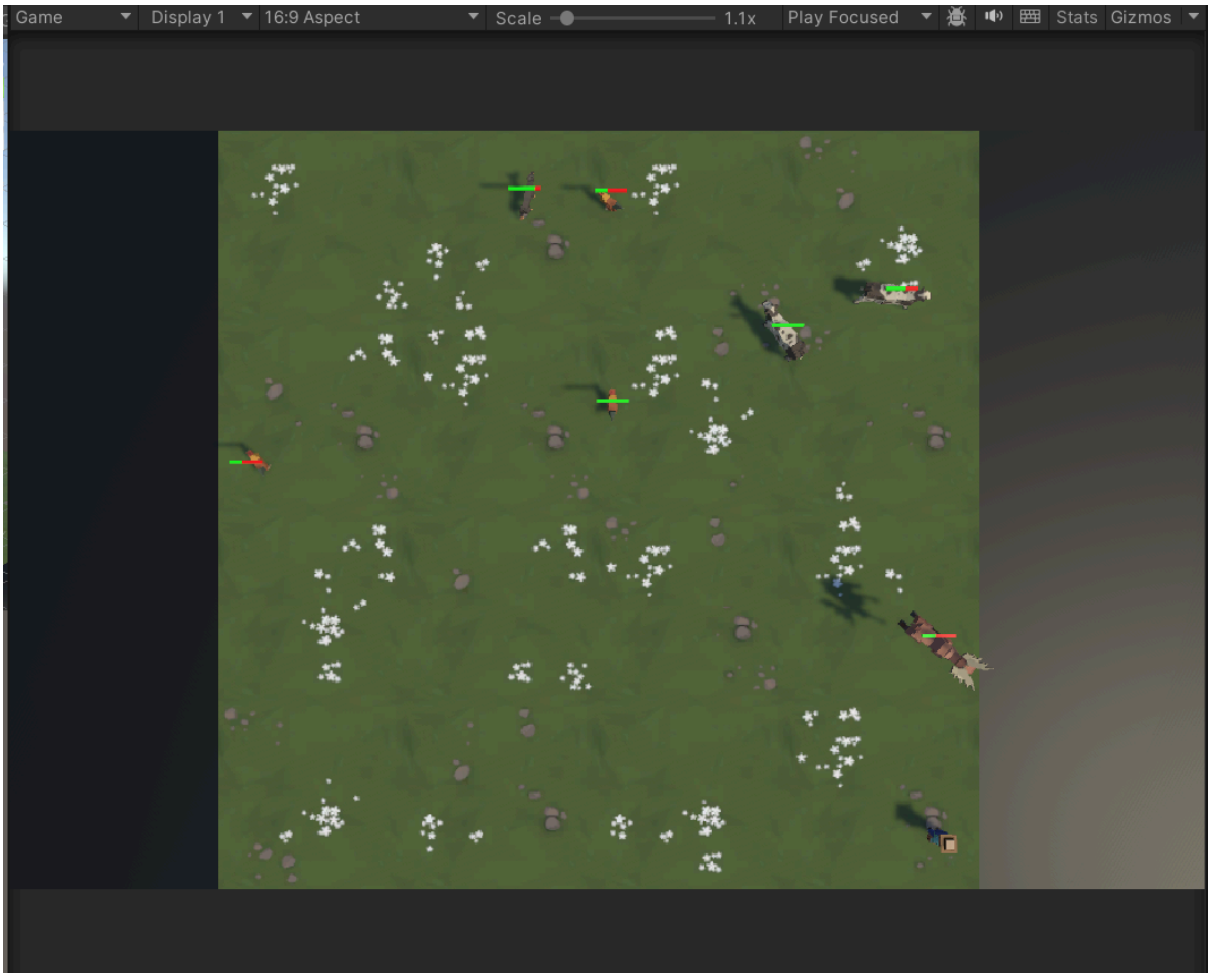
```
void Update()
{
    UpdateHungerBar();
    if(Time.time >= nextDirectionChangeInterval)
    {
        RandomDirection();
        nextDirectionChangeInterval = directionChangeInterval + Time.time;
    }


    if (!isColliding)
    {
        AnimalMoving();
    }
    AnimalDie();
}
```

- 랜덤한 방향 설정은, RandomDirection 함수를 통해 이루어집니다. 또한 동물의 움직임은 이후 설명할 충돌 회피 구현을 위해, isColliding이 false 인 경우(충돌이 없는경우) 에만 움직이도록 하였습니다.

```
public void RandomDirection()
{

    randomDirection = UnityEngine.Random.Range(-1.0f, 1.0f);
}
```

- 위 함수는 랜덤방향 설정 함수입니다.

- 아래는 예시 사진입니다.


Game
Display 1
16:9 Aspect
Scale
1.1x
Play Focused
Stats
Gizmos

(8) Collision avoidance

- AnimalBehaivor 스크립트, 모든 동물 프리팹에 연결

- 충돌 회피를 구현하기 위해, 각 동물의 전방벡터(바라보는 방향벡터) 를 사용하였습니다.

- 아래는 OnTriggerEnter 함수의 일부입니다.

```
if (other.CompareTag("Animal"))
{
    Vector3 thisFoward = transform.forward;
    Vector3 otherFoward = other.transform.forward;
    thisFoward.y = 0.0f;
    otherFoward.y = 0.0f;
    dotProductAnimal = Vector3.Dot(thisFoward,otherFoward);

    if(dotProductAnimal < 0 && dotProductAnimal > -1)
    {
        Debug.Log("dotproduct1");
        Vector3 oppositeDirection = -transform.forward;
        Quaternion newRotation = Quaternion.LookRotation(oppositeDirection);
        transform.rotation = newRotation;
    }

     else if(dotProductAnimal >= 0)
    {
        /* isColliding = true;
         Debug.Log("dotproduct2");
         //transform.position += Vector3.zero;

         StopMovement();*/
        Quaternion newRotation = Quaternion.Euler(0, 30, 0) * transform.rotation;
        transform.rotation = newRotation;
        Debug.Log("dotproduct2");
    }


}
```

- 모든 동물 오브젝트에 Boxcollider를 설정하여, 만약 Tag가 Animal인 오브젝트와 충돌이 발생한 경우, 현재 동물 오브젝트의 전방벡터와 충돌한 동물 오브젝트(other) 의 전방벡터의 내적을 계산하여, 그 값이 -1~0의 범위인 경우 (동물 오브젝트들의 진행방향이 반대라서 마주치는 상황이라고 생각함)
  즉시 현재 동물 오브젝트가 자신의 전방벡터 방향에서 180도에 해당하는 방향을 바라보게 하였습니다.

- 만약, 각 동물의 전방벡터의 내적이 0 이상인 경우(동물 오브젝트의 진행 방향이 비슷해서, 옆에서 부딪히는 상황이라고 생각함 ex -> A동물의 옆구리에 B동물의 머리가 부딪힘), 원래는 충돌판정이 사라질 때까지 B 동물을 정지시키려고 하였으나, A,B 두 동물 오브젝트가 영원히 정지하는 버그가 발생하여, 내용을 수정하였습니다.

- 수정된 내용은, 현재 오브젝트가 자신의 전방벡터가 바라보는 방향에서 Y축으로 30도 만큼 돌아간 방향을 바라보게 하는 것입니다.

- 수정된 내용을 바탕으로 실행 후 상황을 본 결과, 70%정도의 비율로 동물 오브젝트 간 충돌 회피가 구현되었습니다. 이루어지지 않은 30%의 경우, 동물오브젝트가 20마리 이상으로 늘어나 필드에 빽빽하게 채워지게 되자 일부 동물 오브젝트들끼리 충돌회피가 잘 이루어지지 않았습니다.

- 아래는 충돌 회피 예시입니다.

- 현재 상황은, 두 Moose 동물 오브젝트가 서로 머리부분에 콜라이더 충돌이 일어나자, 즉시 반대방향을 바라보고 있는 상황입니다.

3. 과제에 대한 나의 의견

(1) 가장 구현하기 힘들었던 부분

- 충돌 회피를 구현할 때, 두 동물 오브젝트가 서로 마주보며 다가오는 경우(내적이 -1~0 인 경우)는 생각보다 쉽게 해결했는데, 이외의 경우에 충돌 회피구현이 잘 이루어지지 않아 어려웠던 것 같습니다.

(2) 향후 이루고 싶은 목표

- 현재 충돌 회피를 하는 부분에서 동물 오브젝트가 완벽하게 충돌을 회피하고 있지 않는데, 이후에는 완벽하게 충돌을 회피하는 내용을 구현할 수 있는 수준에 이르고 싶습니다.

4. 완성본 동영상 링크

- 계속 진행하다 보니, 동물이 너무 많아 음식을 먹었을 때, HungerBar가 채워지는 것을 확인하기 힘들어, 영상 마지막부분에 AnimalSpawnManager를 해제하여 동물을 몇마리만 남긴 후, HungerBar가 채워지는 부분을 확인하였습니다.

- https://youtu.be/8b0s9pxVGe4

- 과제로 제출한 패키지를 빈 프로젝트에 **import**해서 보니, **Food** 레이어가 계속 빠져 있습니다. **Layer**에 **Food** 를 추가한 후, 인스펙터에서 음식 프리팹(바나나, 샌드위치, 뼈다귀) 에 **Food** 레이어를 설정해야 합니다.