

VR을 활용한 자동차 운전 시뮬레이션 최종발표

문벼리 이승민 정동원 최동우

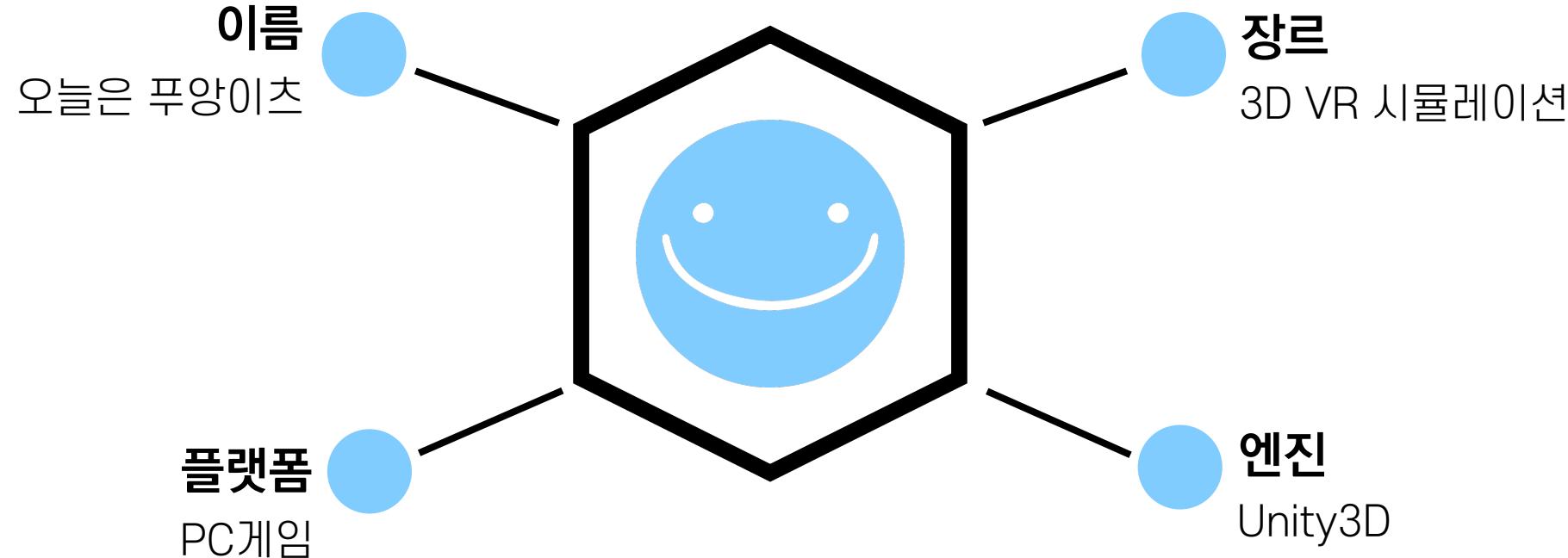


가상증강현실
박경주 교수님

01

게임 개요

01 게임 개요



02

게임 컨텐츠

02 게임 컨텐츠

운전 시스템

- 플레이어는 양손 컨트롤러로 운전대를 조종한다.
- 컨트롤러의 버튼으로 액셀, 브레이크, 등 기타 조작을 할 수 있다.
- 운전자의 현재 속도는 상단 UI에 숫자로 표시된다.
- 운전자의 현 위치는 하단 UI의 미니맵에 표시된다.



사용 예셋:

Unity의 XR Hands package

02 게임 컨텐츠

배달 시스템

- UI에 배달 장소의 집 모양과 제한 시간 스톱워치를 배치한다.
- 제한시간 안에 배달한다면 보상(코인) 획득, 신뢰도가 상승한다.
- 제한시간 안에 배달하지 못하면 신뢰도가 떨어진다.
- 신뢰도는 월급에 영향을 미친다.
- 고객의 집 앞 트리거에 자동차를 주차해야 최종적으로 고객님께 음식을 전달할 수 있다.

02 게임 컨텐츠

경찰차 시스템

- 경찰차는 기본 플레이어 속력보다 10정도 빠르게 설정하고, 초반에는 빨리 잡히더라도 업데이트를 어느정도 하면 플레이어 차량 속력이 더 빨라질 수 있도록 한다.
- 경찰차에 닿으면 일정 코인을 뺏기고, 시작지점으로 리스폰되고 신뢰도가 낮아진다.

상점 시스템

- 월급과 배달로 얻은 코인을 이용해서 자동차를 업그레이드(색 변경, 속도 상승, 자동차 내구성 높이기) 할 수 있다.
- 염소 회피를 위한 당근 구매를 할 수 있다.

02 게임 컨텐츠

도로 시스템

- 도로에는 차선, 신호등, 과속방지턱 등 교통 법규 관련 요소들이 포함되어 있고, 장애물에 닿을 때마다 일정 확률로 차량이 손상되고, 차 속력이 낮아진다. 상점에서 수리 가능하다.

신호등 시스템

- 빨간 불이 켜졌을 때 정지선을 넘었다면 뒤에서 경찰차가 쫓아온다.

코인 시스템

- 길 중간중간에 놓여있는 코인 아이템을 먹으면 코인을 얻을 수 있다.

02 게임 컨텐츠

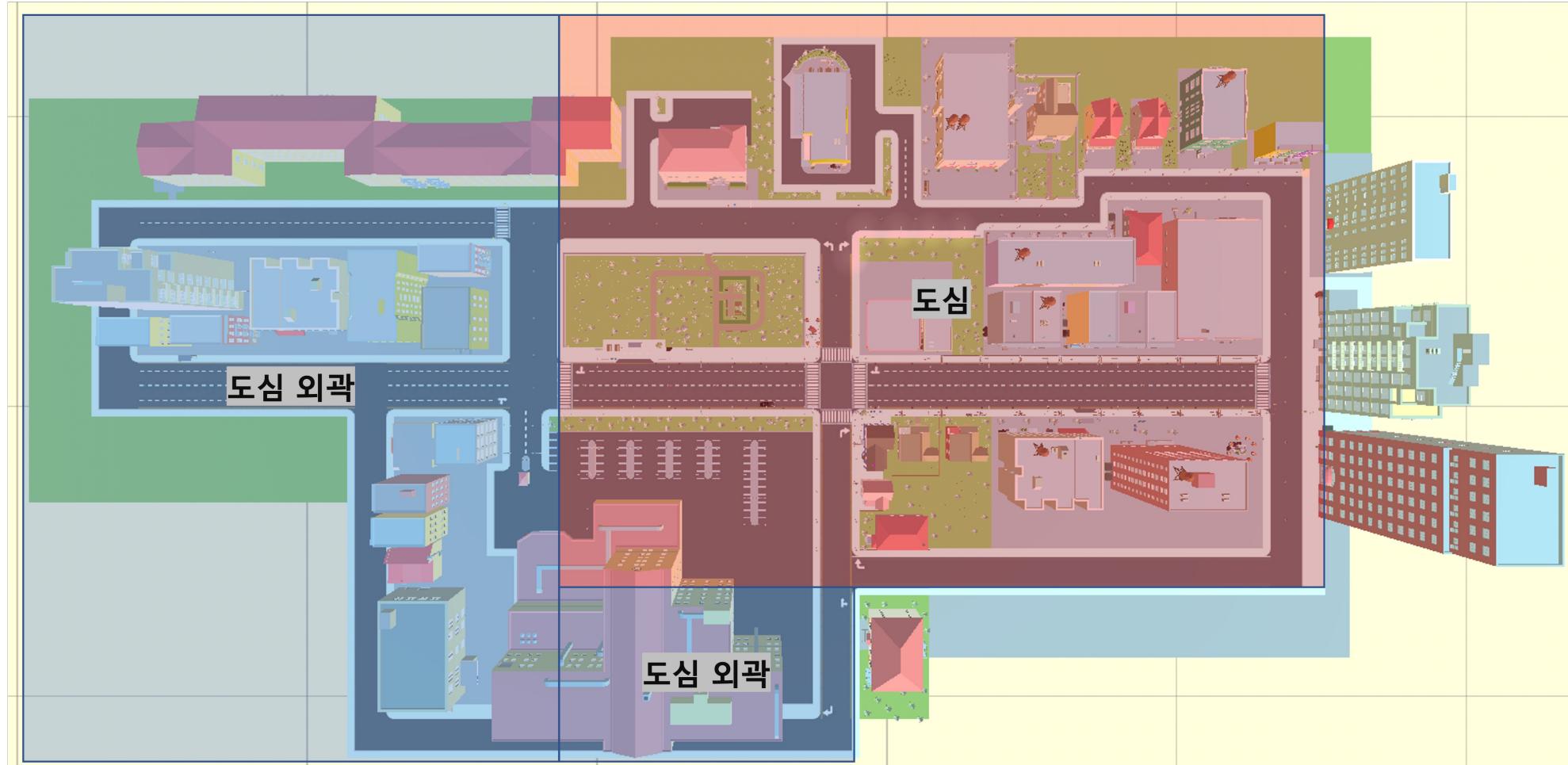
NPC - 염소

- 길가다가 염소가 나오면 당근을 던져주어야 한다.
- 당근을 던지면 염소가 길을 비켜준다.

NPC - 아이

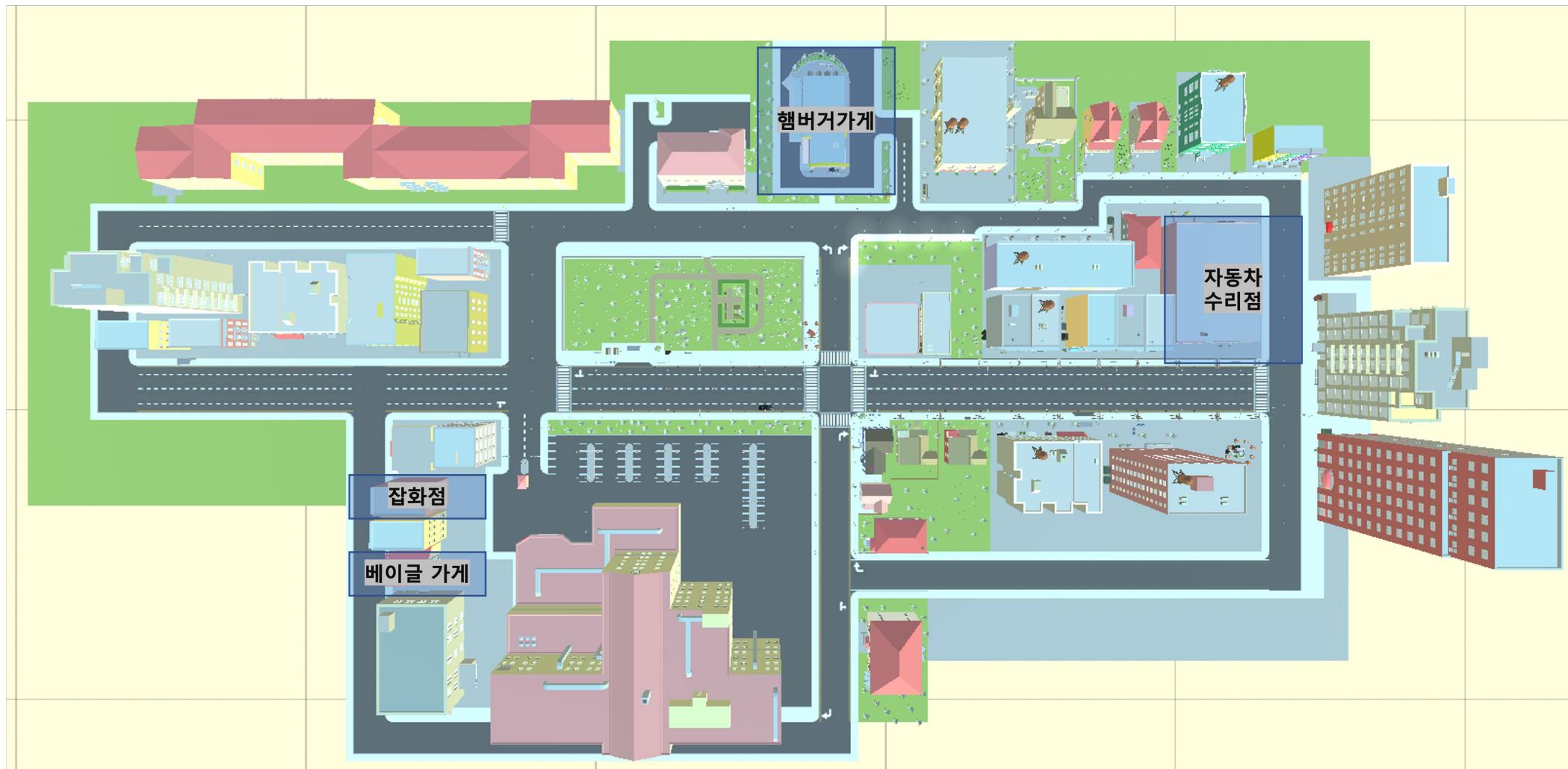
- 도심에는 신호등을 건너는 아이가 있다.
- 혹시나 아이를 치면 경찰차가 소환된다.

02 맵 구성



도심외곽: 야생동물 출현 // 도심 : 길건너는 사람 & 신호등

02 맵 구성



03

구현 방법

03 플레이어(자동차)

- Oculus 컨트롤러를 통해 훨을 조작할 수 있도록 함
- Oculus 컨트롤러를 통해 기어를 변경할 수 있도록 함 (전진, 후진 2가지 조작만)
- 훨의 상태와 Oculus 컨트롤러의 입력 값을 통해 자동차가 움직일 수 있도록 함
- 자동차의 동작에 따라 적절한 사운드가 발생할 수 있도록 함
- 미니맵(네비게이션)을 통해 목표 지점까지의 방향을 알 수 있도록 함



03 플레이어(자동차)

- 컨트롤러의 HandTrigger(중지에 위치한 버튼)을 통해 압셀, 브레이크를 밟을 수 있음.
- PrimaryHandTrigger(왼손에 드는 컨트롤러에 위치한 버튼)을 통해 압셀, SecondaryHandTrigger(오른손에 드는 컨트롤러에 위치한 버튼)을 통해 브레이크를 밟을 수 있음.
- 해당 버튼의 입력 값을 받아와 후방 WheelCollider의 MotorTorque값 (브레이크 시에는 brake Torque 값)을 결정함.
- Gear.cs 컴포넌트의 state 값을 통해 MotorTorque의 부호(+ or -)가 결정됨.
- SteeringWheel.cs 컴포넌트의 steering 값을 통해 전방 Wheelcollider의 SteerAngle값이 결정됨.
- 해당 SteerAngle값은 애커만 조향법의 공식에 따라 회전 시의 기울어짐을 최소로 하도록 결정됨

03 플레이어(자동차)

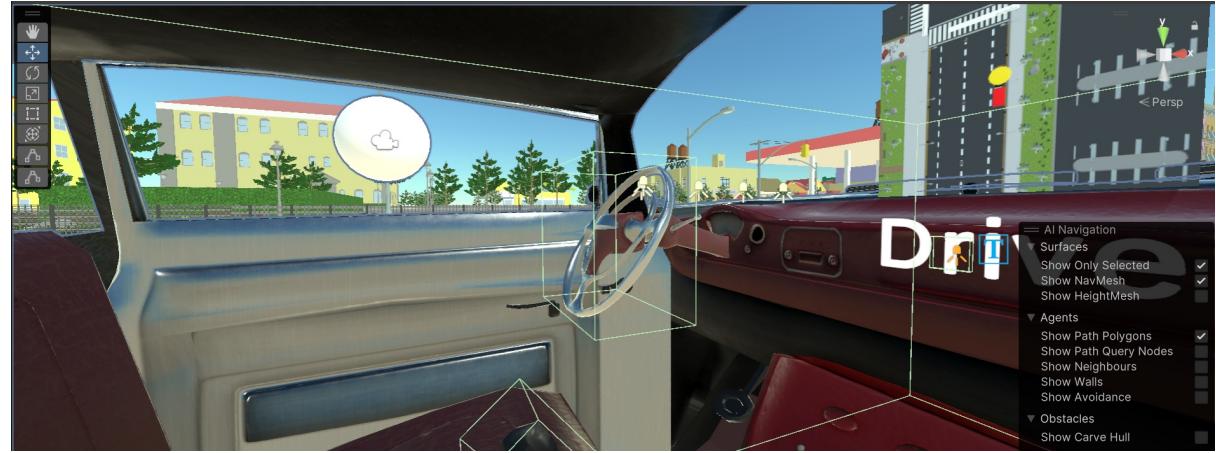
```
private void Steer()
{
    float steering = _steeringWheel.steering;
    if (steering > 0)
    {
        // rear tracks size is set to 1.5f          wheel base has been set to 2.55f
        wheels[0].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (steeringRadius + (1.5f / 2))) * steering;
        wheels[1].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (steeringRadius - (1.5f / 2))) * steering;
    }
    else if (steering < 0)
    {
        wheels[0].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (steeringRadius - (1.5f / 2))) * steering;
        wheels[1].steerAngle = Mathf.Rad2Deg * Mathf.Atan(2.55f / (steeringRadius + (1.5f / 2))) * steering;
    }
    else
    {
        wheels[0].steerAngle = 0;
        wheels[1].steerAngle = 0;
    }
}
```

03 플레이어(자동차) - 사운드

- 자동차의 속도 값에 따라 엔진음의 pitch값을 조정하여 엔진 rpm이 올라가는 것을 소리로 표현함.

```
➊ Unity 메시지 | 참조 0개
private void FixedUpdate()
{
    carSpeed = _rigidbody.velocity.magnitude;
    if (_audioSource.clip != _carIdle)
    {
        _audioSource.clip = _carIdle;
        _audioSource.volume = 0.5f;
        _audioSource.Play();
    }
    _audioSource.pitch = Mathf.Lerp(1f, 2f, carSpeed / carMaxSpeed);
}
```

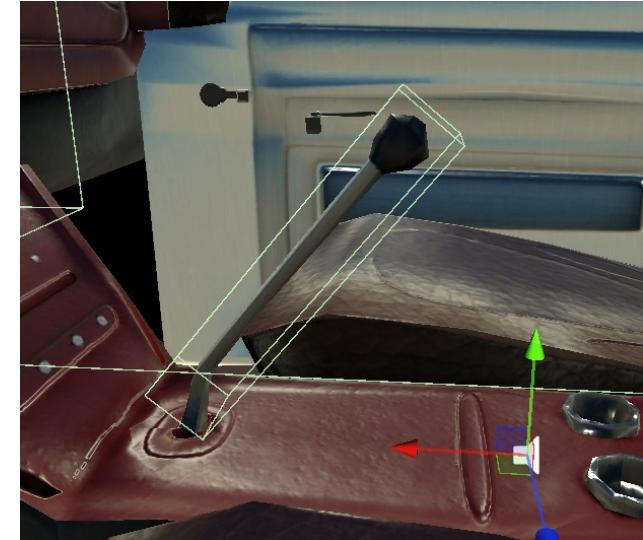
03 플레이어(자동차)-휠 조작



- SteeringWheel.cs 스크립트를 통해 구현
- OVR Camera Rig의 자식 오브젝트로 존재하는 Hand Object로부터 트리거 이벤트를 받아 동작함
- 컨트롤러를 핸들의 Box Collider 내부에 위치시킨 상태에서, 컨트롤러의 IndexTrigger(검지에 위치한 버튼)를 누르면, 핸들을 잡을 수 있음
- 핸들을 잡은 상태에선, 잡은 쪽 손 컨트롤러의 회전 값에 따라 핸들이 회전하게 됨
- 양손 모두 잡은 경우, 두 컨트롤러의 회전 값의 중간을 계산하여 핸들이 회전함.
- 핸들의 회전 값에 따라 steering 변수 값이 -1 ~ 1 사이로 정해지게 됨.
- 해당 steering 값에 따라 자동차가 회전함.

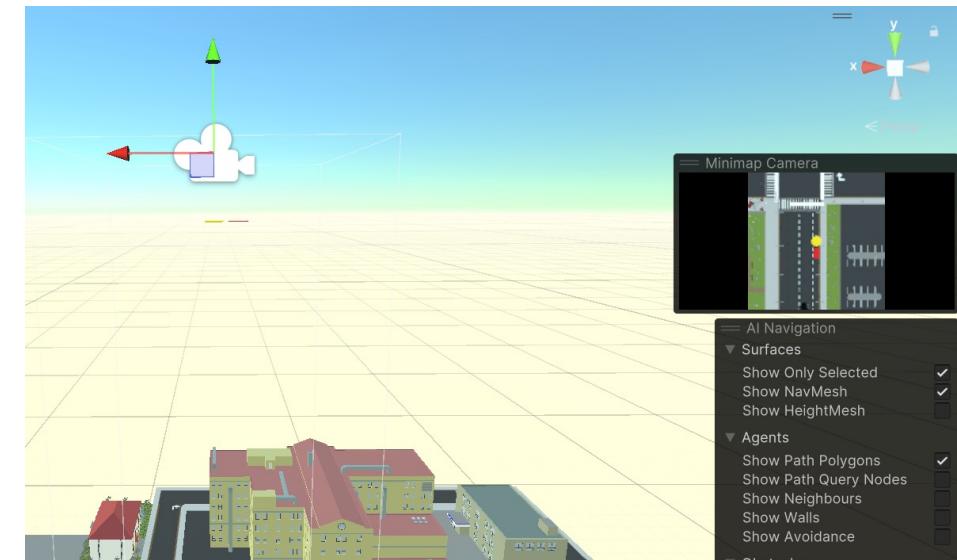
03 플레이어(자동차)-기어 변경

- Gear.cs 스크립트를 통해 구현
- OVR Camera Rig의 자식 오브젝트로 존재하는 Hand Object로부터 트리거 이벤트를 받아 동작함
- 컨트롤러를 기어의 box Collider 내부에 위치시킨 상태에서, 컨트롤러의 IndexTrigger(검지에 위치한 버튼)를 누르면, 기어가 변경됨
- 기어의 상태는 미니맵 아래의 Drive, Reverse 텍스트를 통해 알 수 있음(기어 변경 시 효과음도 존재)



03 플레이어(자동차) - 미니맵

- 자동차의 child object에 미니맵을 위한 Icon Mesh 와 미니맵용 Camera를 배치함
- 하늘에서 아래방향을 찍도록 Camera를 회전한 후, Culling Mask에서 미니맵에 표시할 Layer를 선택함.



03 배달 시스템

- 배달주문 시작, 음식 받아오기, 배달지까지 운반하기의 순서로 배달을 진행할 수 있다
- 배달 시작, 진행, 결과에 대한 정보를 UI를 통해 얻을 수 있다.
- 주문 시작 함수가 호출되면 주문에 관한 데이터를 파라미터로 받아 아래의 Data에 반영함
- 코루틴을 이용해 OrderData의 변화를 체크하는 것으로 배달 진행 과정을 구현함

참조 1개

```
private void StartOrder(DeliveryPoint food, DeliveryPoint dest, float limitTime = 120)
{
    _inProgressOrderData.foodPoint = food;
    _inProgressOrderData.destPoint = dest;
    _inProgressOrderData.progressTime = 0;
    _inProgressOrderData.limitTime = limitTime;
    _inProgressOrderData.progress = OrderData.Progress.TakeFood;
    _inProgressOrderData.reward = PlayerDataHelper.CalculateSalary(_inProgressOrderData.reliability);
    _inProgressOrderData.reliability = 3;
    _inProgressOrderData.result = OrderData.Result.NotDecided;
    StartCoroutine(OrderCoroutine(_inProgressOrderData));
}
```

```
IEnumerator OrderCoroutine(OrderData order)
{
    order.foodPoint.EnablePoint();
    UM.SetMinimapTarget(order.foodPoint.transform);
    //Debug.Log("음식 받으러 가기");
    yield return new WaitWhile(() => order.progress == OrderData.Progress.TakeFood && order.result != OrderData.Result.Canceled);

    order.destPoint.EnablePoint();
    UM.SetMinimapTarget(order.destPoint.transform);
    //Debug.Log("배달하기");
    yield return new WaitWhile(() => order.progress == OrderData.Progress.DeliverFood && order.result != OrderData.Result.Canceled);

    CalculateOrderResult();
    FinishOrder();
    //Debug.Log("배달완료");
}
```

03 배달 시스템

- 배달이 종료되면 걸린 시간과 같은 OrderData의 변수를 체크하여 OrderData의 결과관련 변수 값을 갱신함

```
private void CalculateOrderResult()
{
    if(_inProgressOrderData.result == OrderData.Result.Canceled)
    {
        _inProgressOrderData.reward = 0;
        _inProgressOrderData.reliability = -3;
    }
    else
    {
        bool isSuccess = _inProgressOrderData.limitTime >= _inProgressOrderData.progressTime;
        if (isSuccess)
        {
            _inProgressOrderData.result = OrderData.Result.Success;
        }
        else
        {
            _inProgressOrderData.result = OrderData.Result.Fail;
            _inProgressOrderData.reliability = -1;
        }
    }
}
```

```
private void FinishOrder()
{
    _playerData.coin += _inProgressOrderData.reward;
    _playerData.reliability += _inProgressOrderData.reliability;
    UM.DisableNavigation();
    UM.EnableOrderResultUI();
    isProgessOrder = false;
}
```

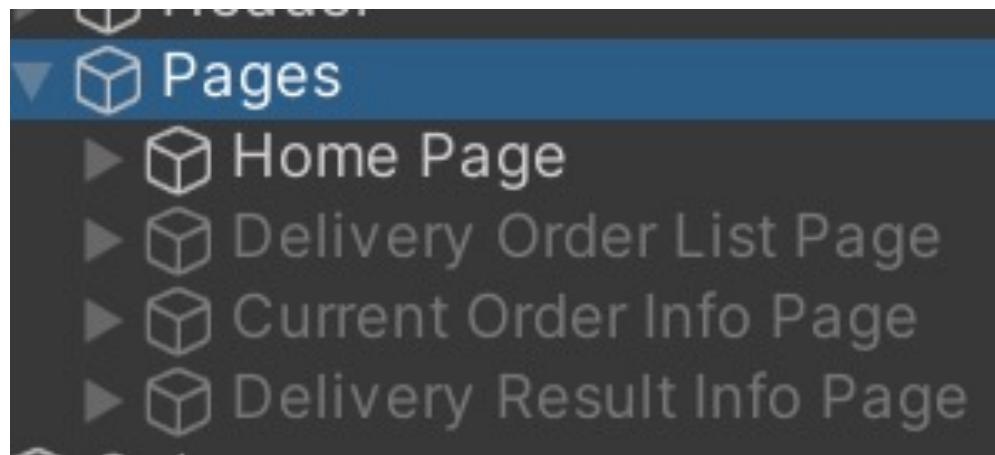
03 배달 시스템 - UI

- UI구현을 위해 유니티의 Canvas의 Render Mode를 World Space로 설정하여 공간좌표계 상에 배치할 수 있도록 함



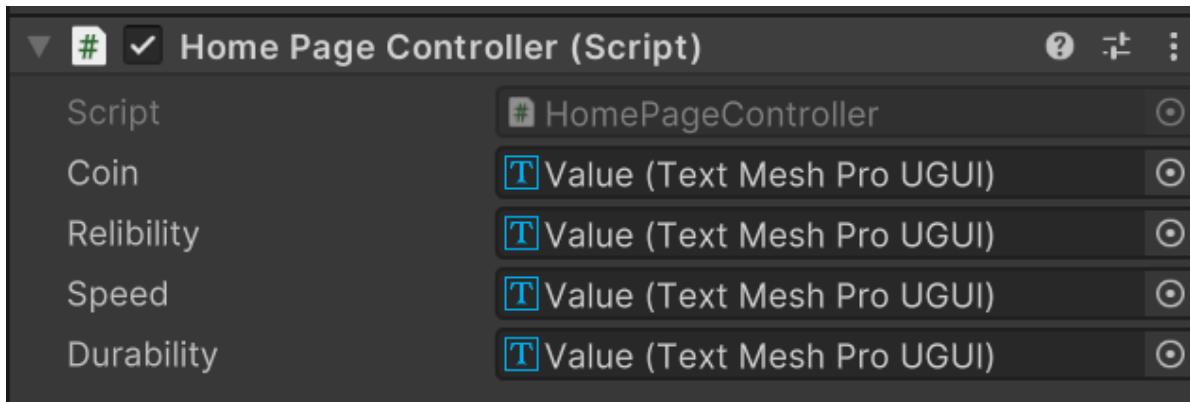
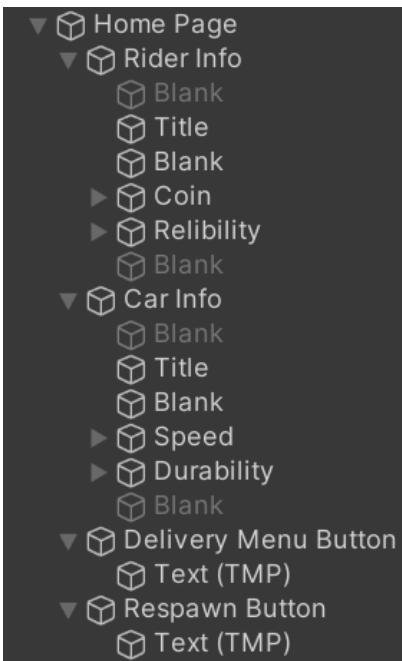
03 배달 시스템 - UI

- UI구현을 위해 유니티의 Canvas의 Render Mode를 World Space로 설정하여 공간좌표계 상에 배치할 수 있도록 함
- 주로 게임 내에서 모두가 공유하는 데이터인 Player Data와 Order Data등을 UI로 표현함
- 아래와 같이 하이라리를 설정하였고, 각 GameObject를 하나만 키는 것을 통해 페이지가 바뀌는 것을 구현함



03 배달 시스템 - UI

- 각 페이지는 아래와 같이 UI의 집합을 자식으로 가지고 있고, 각 UI중 데이터에 따라 바뀌어야 하는 UI 오브젝트를 페이지마다 불어있는 컴포넌트에 (inspector창을 이용하여) 직접 하여 할당함



03 빼달 시스템 - UI

- 각 페이지에 붙어있는 컴포넌트들은 아래와 같은 함수를 통해 Player Data (or Order Data)의 값을 UI에 반영함
- 데이터가 갱신되는 시점은 Object가 활성화 될 때 (= Page가 바뀔 때) 혹은 참조하는 데이터에 갱신이 생길 때

```
Unity 메시지 | 참조 0개
private void OnEnable()
{
    LoadData();
}

참조 3개
private void LoadData()
{
    if (_data == null) return;
    coin.text = _data.coin.ToString();
    reliability.text = _data.reliability.ToString();
    speed.text = _data.speed.ToString();
    durability.text = _data.durability.ToString();
}
```

```
private void Awake()
{
    Addressables.LoadAssetAsync<PlayerData>(playerDataAddress).Completed
        += (handle) =>
    {
        _data = handle.Result;
        _data.AddListener(LoadData);
        LoadData();
    };
}
```

03 빼달 시스템 - UI

- 각 컴포넌트에서 최초에 데이터를 불러올 때, 데이터 갱신시마다 호출되어야 할 함수를 data에 등록함
- 각 data에서는 데이터가 변경될 때마다(각 변수들의 setter가 호출될 때마다) 등록된 함수들을 모두 한번씩 실행함

```
private List<Action> listeners = new List<Action>();
```

```
public void AddListener(Action listener)
{
    listeners.Add(listener);
}

참조 0개

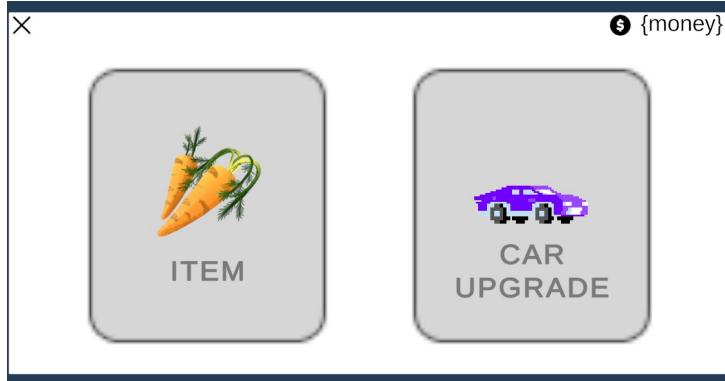
public void RemoveListener(Action listener)
{
    listeners.Remove(listener);
}
```

```
private void OnDataChanged()
{
    foreach(var listener in listeners)
    {
        listener.Invoke();
    }
}
```

```
public int speed
{
    get
    {
        return _speed;
    }
    set
    {
        _speed = value;
        OnDataChanged();
    }
}
```

03 상점 시스템

- 상점 trigger에 도달하게 되면 다음과 같은 UI가 나타남
- vr 기기를 통하여 item을 살 것인지 차량을 업그레이드 할 것인지를 고를 수 있음
- 사용자의 data에 아이템 갯수, 차량정보가 더해짐



```
_triggerBox = transform.GetChild(0).gameObject;
_triggerBox.SetActive(true);

private void Start()
{
    UM = UIManager.Instance;

}

private void OnTriggerEnter(Collider other)
{
    if (other.CompareTag("Player")) return;
    UM.EnableShopUI();
}

private void OnTriggerExit(Collider other)
{
    if (other.CompareTag("Player")) return;
    UM.DisableShopUI();
}

public void EnablePoint()
{
    _triggerBox.SetActive(true);
}

public void DisablePoint()
{
    _triggerBox.SetActive(false);
}
```

```
private void UpgradeSpeed()
{
    if (_playerData.coin < _shopData.speedUpgradePrice) return;
    _playerData.speed += 10;
    _playerData.coin -= _shopData.speedUpgradePrice;
}

private void UpgradeDurability()
{
    if (_playerData.coin < _shopData.durabilityUpgradePrice) return;
    _playerData.durability += 10;
    _playerData.coin -= _shopData.durabilityUpgradePrice;
}

private void BuyCarrot()
{
    if (_playerData.coin < _shopData.carrotPrice) return;
    _playerData.carrotNum += 1;
    _playerData.coin -= _shopData.carrotPrice;
}
```

Shop Data (Mono Script)

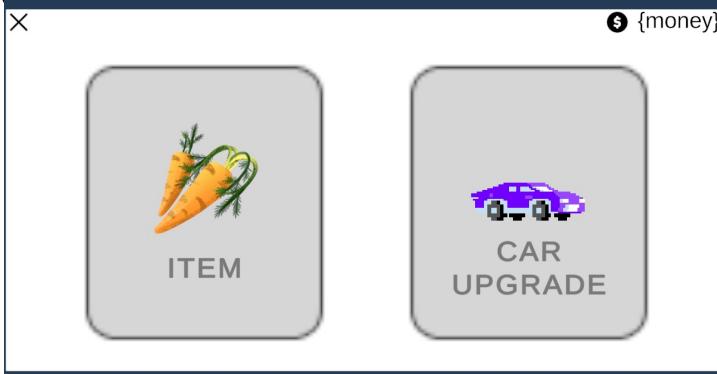
Assembly Information
Filename Assembly-CSharp.dll

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

[CreateAssetMenu(fileName = "Shop Data",
menuName = "Scriptable Object/Shop Data", order =
int.MaxValue)]
public class ShopData : ScriptableObject
{
    public int carrotPrice = 1000;
    public int speedUpgradePrice = 5000;
    public int durabilityUpgradePrice = 3000;
}
```

03 상점 시스템

- 차량 업그레이드를 선택하면 speed, durability를 업그레이드 가능
- Item을 선택하면 당근을 구매할 수 있음
- 페이지 넘어갈 때, 닫기, 뒤로가기 등 가능



```
private void LoadData()
{
    if (_shopData is null) return;
    speedUpPrice.text =
    _shopData.speedUpgradePrice.ToString();
    durabilityUpPrice.text =
    _shopData.durabilityUpgradePrice.ToString();
}

public void OnClickUpgradeSpeed()
{
    onUpgradeSpeed();
}

public void OnClickUpgradeDurability()
{
    onUpgradeDurability();
}
```

```
private void OnEnable()
{
    LoadData();
}
private void LoadData()
{
    if (_shopData is null) return;
    carrotPrice.text =
    _shopData.carrotPrice.ToString();
}

public void OnClickBuyCarrot()
{
    onBuyCarrot();
}
```

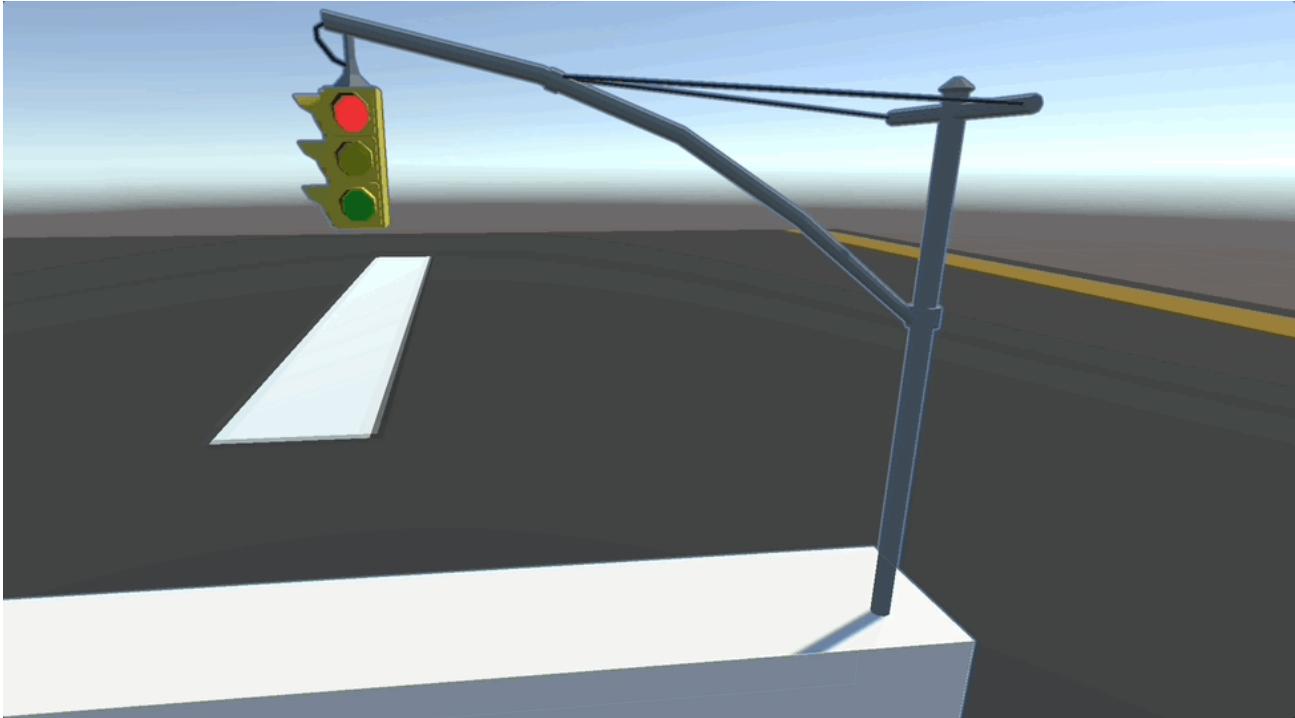
```
public void GoHomePage()
{
    GoPage(homePage.name);
}

public void GoCarUpgradeShopPage()
{
    GoPage(carUpgradeShopPage.name);
}

public void GoItemShopPage()
{
    GoPage(itemShopPage.name);
}

private void TurnOffAllPages()
{
    foreach (Transform page in pageContainer)
    {
        page.gameObject.SetActive(false);
    }
}
```

03 신호등 시스템



- 코루틴을 활용해서 일정 시간마다 신호등의 emission을 true로 설정해서 빛나는 효과를 표현함
- 색 변화가 있을 때 신호등의 isRedLight bool값도 계속 변화함

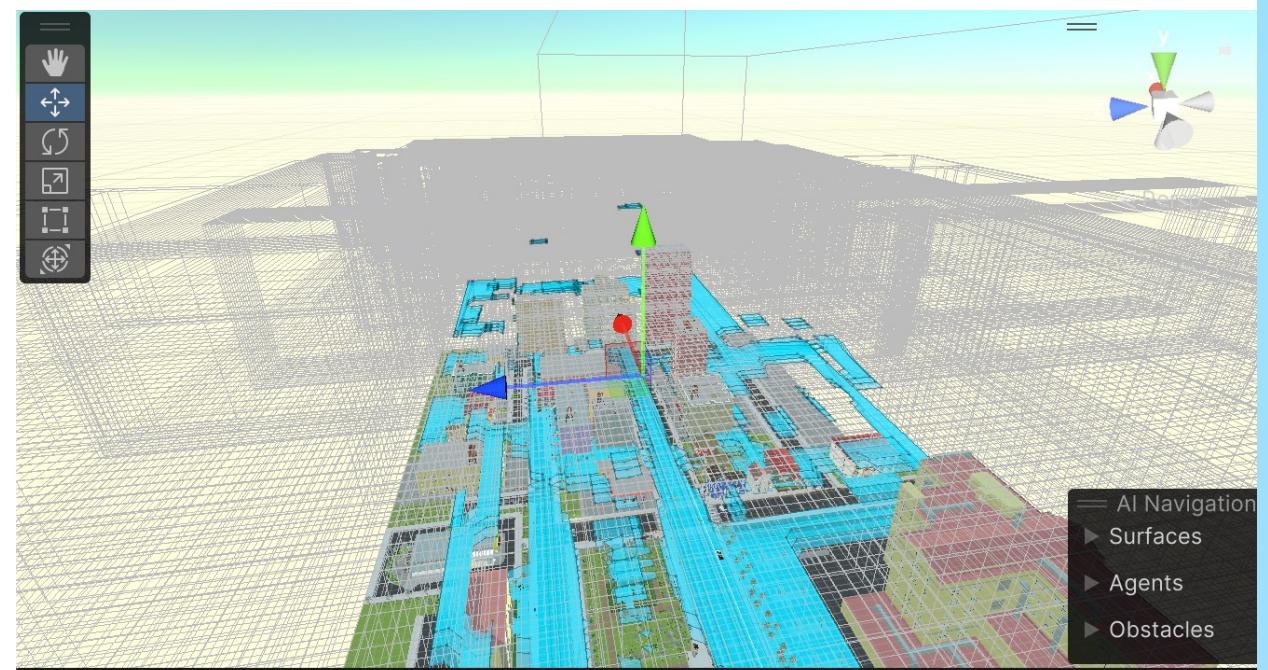
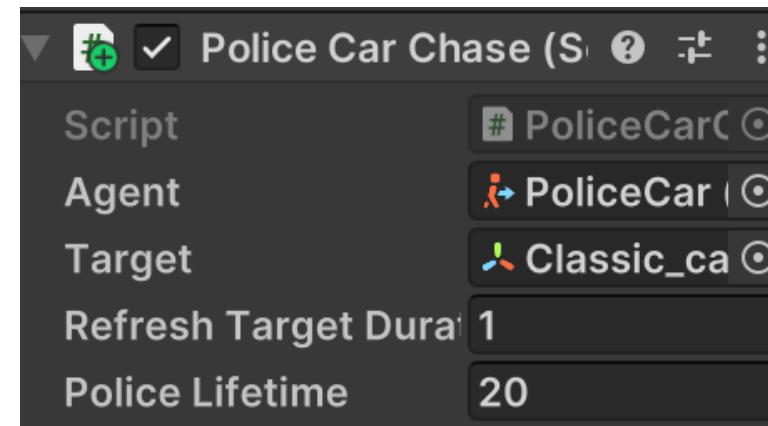
03 경찰차 시스템

- 플레이어가 신호등 빨간불이 켜진 동안 신호등마다 불어있는 TrafficLightTrigger에 닿으면 플레이어의 옆에 경찰차가 소환됨
- 경찰차는 코루틴으로 1초마다 플레이어의 위치를 파악하고, AINavmesh를 사용해 플레이어를 쫓을 경로를 지정하고 플레이어를 쫓아감
- 10초가 지나면 경찰차는 다시 사라짐
- 경찰차에 닿으면 플레이어가 소지한 코인이 깎임



03 경찰차 시스템

- Alnavmesh 베이크 – 하늘색으로 표시된 부분만 경찰차가 이동할 수 있음
- Target duration과 lifetime을 직접 선택 가능



03 코인 줍기

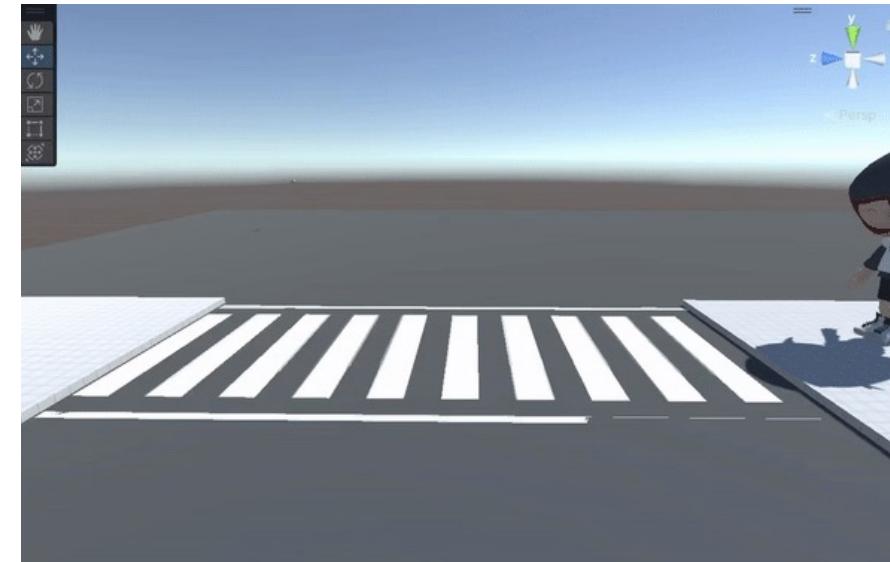
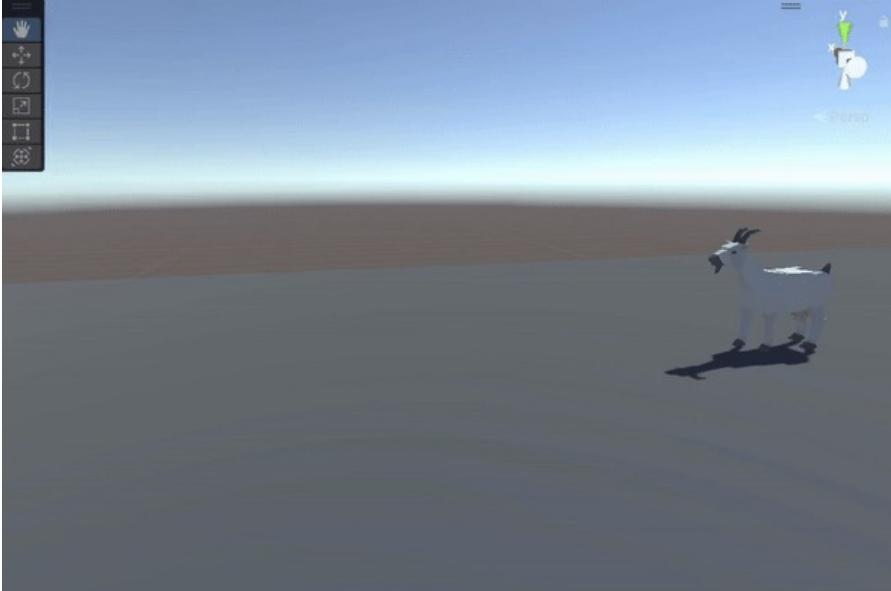
- Dotween 라이브러리 활용하여 코인이 제자리에서 도는 애니메이션 구현
- 플레이어에 닿으면 코인은 Destroy되고 플레이어는 100코인을 얻음

```
void Start()
{
    Tween tween = transform.DORotate(new Vector3(0, 360, 0), 2f, RotateMode.FastBeyond360)
        .SetEase(Ease.Linear)
        .SetLoops(-1, LoopType.Restart);
}

private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.tag == "Player")
    {
        Debug.Log("DESTROY!!!");
        other.gameObject.GetComponent<PlayerDataHelper>().coin += 100;
        Destroy(this.gameObject);
    }
}
```



03 NPC시스템

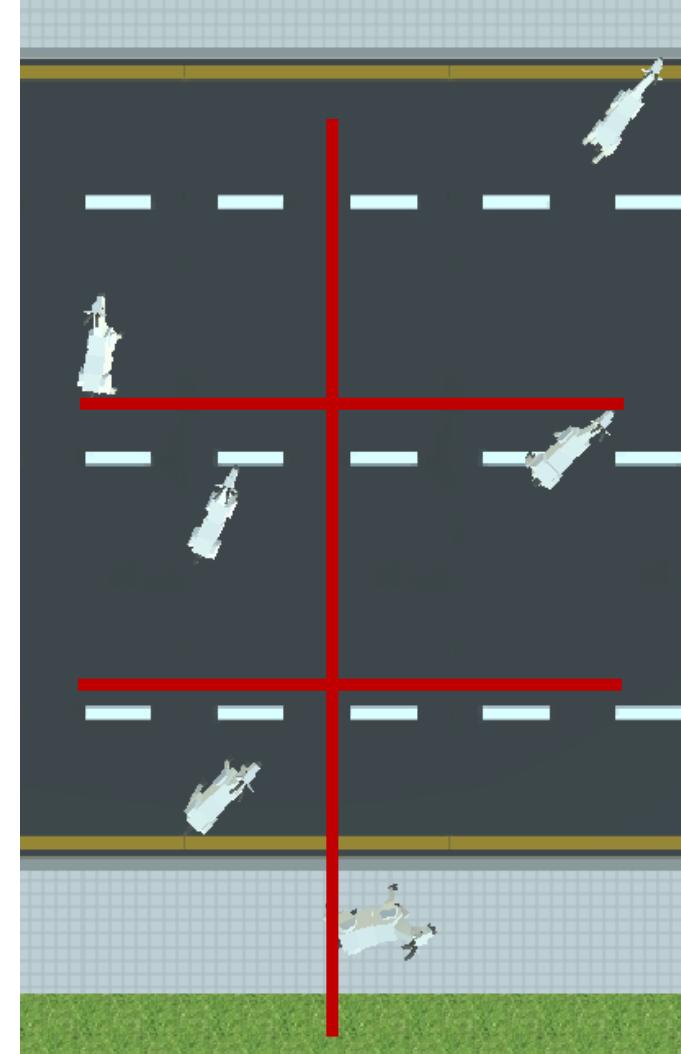


- 염소와 아이 모두 Sphere Collider를 이용하여 자동차가 특정 range 근처로 들어오면 spawn 되도록 함

03 NPC시스템 - 염소



- 도로를 6칸으로 분리하고, 각 칸 안에서 destination 을 랜덤으로 지정
- 염소가 자연스럽게 분포됨과 동시에 도로 전체를 막아 플레이어의 운전을 방해하려는 의도



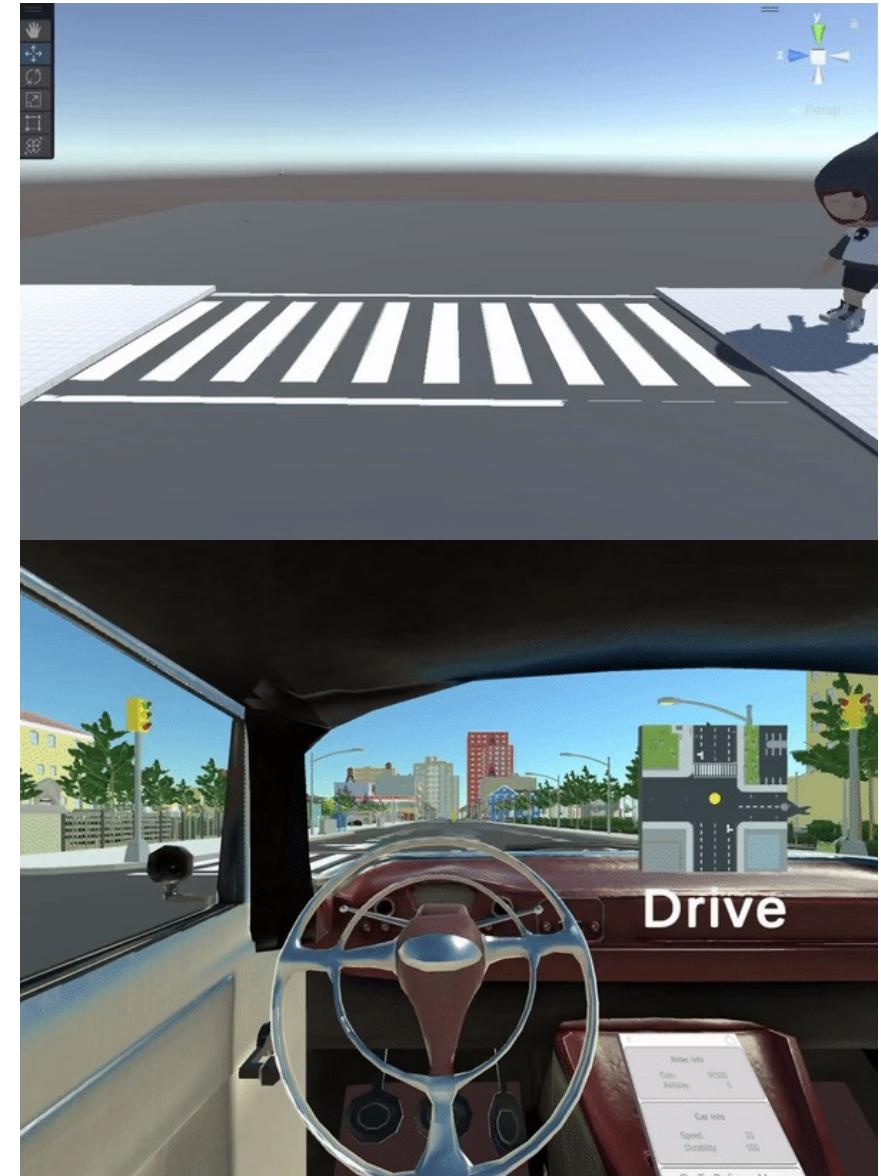
03 NPC시스템 - 염소

- Animator 를 사용하여 뛰어가는 애니메이션 구현
- 당근이 땅에 landing되는 위치로 Translation + Rotation
- 사용자 근처에 떨어진 당근은 Collider 가 해제되어 염소가 인식하지 못하도록 함.
- 염소를 치면 차의 내구도(Durability) 감소



03 NPC시스템 - 어린아이

- Animator 를 사용하여 뛰어가는 애니메이션 구현
- Translation 코루틴을 사용하여 특정 길이만큼 아이가 뛰어가도록 함.
- 아이를 치면 경찰차 출동



04

체험방법

04 체험방법

- 컨트롤러의 HandTrigger(중지에 위치한 버튼)을 통해 액셀, 브레이크를 밟을 수 있음.
- PrimaryHandTrigger(왼손에 드는 컨트롤러에 위치한 버튼)을 통해 액셀, SecondaryHandTrigger(오른손에 드는 컨트롤러에 위치한 버튼)을 통해 브레이크를 밟을 수 있음.
- Oculus 컨트롤러를 통해 기어를 변경할 수 있음
- 미니맵(네비게이션)을 통해 목표 지점까지의 방향을 알 수 있음

감사합니다!

