Make A Port Builder

항만 빌더 매뉴얼

이현재

목차

CHAPTER.1

클래스

목적, 연관성, 클래스 종류 및 기능

목적

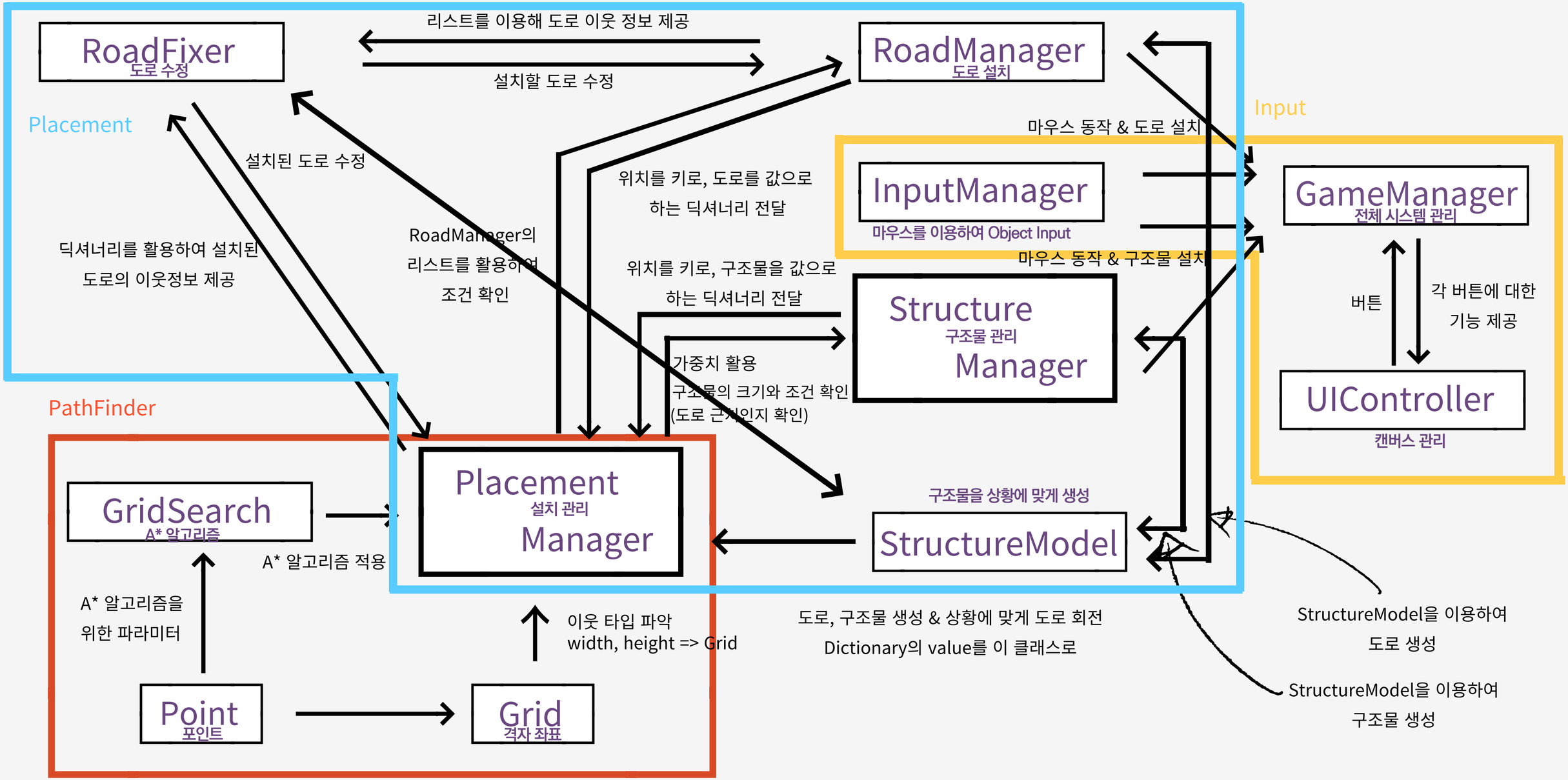
기존 유니티

타이쿤 게임 한 장면

기존의 문제점 : 오브젝트들을 하나씩 설치하는 번거로움

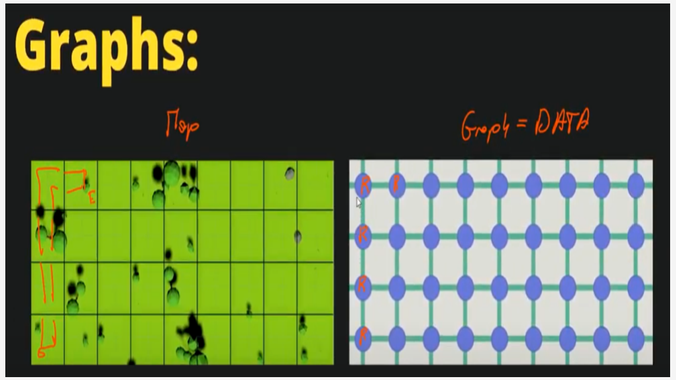
타이쿤 게임처럼 마우스 동작만으로 오브젝트를 설치하고 제거할 수 있다면 작업의 편리성 향상

클래스 연관성(도로 설치)


RoadFixer
도로 수정
리스트를 이용해 도로 이웃 정보 제공
설치할 도로 수정
RoadManager
도로 설치
Placement
설치된 도로 수정
위치를 키로, 도로를 값으로
하는 딕셔너리 전달
마우스 동작 & 도로 설치
Input
InputManager
마우스를 이용하여 Object Input
GameManager
전체 시스템 관리
딕셔너리를 활용하여 설치된
도로의 이웃정보 제공
RoadManager의
리스트를 활용하여
조건 확인
위치를 키로, 구조물을 값으로
하는 딕셔너리 전달
마우스 동작 & 구조물 설치
Structure
구조물 관리
Manager
버튼
각 버튼에 대한
기능 제공
가중치 활용
구조물의 크기와 조건 확인
(도로 근처인지 확인)
UIController
캔버스 관리
PathFinder
Placement
설치 관리
Manager
구조물을 상황에 맞게 생성
StructureModel
GridSearch
A* 알고리즘
A* 알고리즘 적용
A* 알고리즘을
위한 파라미터
이웃 타입 파악
width, height => Grid
도로, 구조물 생성 & 상황에 맞게 도로 회전
Dictionary의 value를 이 클래스로
StructureModel을 이용하여
도로 생성
StructureModel을 이용하여
구조물 생성
Point
포인트
Grid
격자 좌표

세부 정보

Grid and Graph



오브젝트 설치와 A* 알고리즘 적용을 위해 Graph 사용

Structure은 도로 주변에만 & 캔버스를 이용

캔버스에 있는 버튼 클릭하여 원하는 구조물 설치

CellType

```
public enum CellType
{
    Empty,
    Road,
    Structure,
    SpecialStructure,
    None
}
```

각 Grid의 한 공간은 CellType 중 하나의 속성

클래스 종류 및 기능(도로 설치)

Grid의 좌표 X,Y

· Class Point를 이용해서
Grid 생성
· 이웃 Point Type 파악

· A* 알고리즘

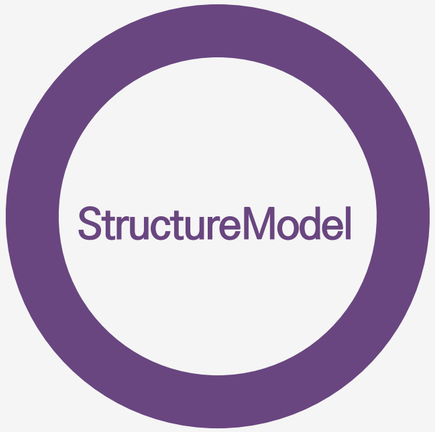
· OnMouseClick
· OnMouseHold
· OnMouseUP
· RayCast

클래스 종류 및 기능(도로 설치)

· InputManager로부터 정보를 가져와
Object들을 Grid에 놓을 수 있도록
다른 Class들에게 정보를 전달

· RoadFixer를 이용해서 도로를 수정
· A* 알고리즘을 적용하여 도로 설치
· List 2개 이용 :
- 배치된 도로의 이웃의 정보를 담은 리스트
- 특정 조건을 확인하기 위한 임시 리스트

· Grid Class를 이용하여 폭과 높이를
입력받아 실질적인 Grid 생성
· 제약 조건 확립
(Grid 안에 있는지, 비어있는지, 도로근처인지)
· Dictionary 2개 이용 :
- Key를 Position, Value를 StructureModel
로 하여 조건을 확인하는 Dictionary
- Gird에 설치한 것을 영구적으로
보관하는 Dictionary



· 오브젝트 인스턴스 담당
· 도로를 회전

클래스 종류 및 기능(도로 설치)

RoadFixer

· 현재 이웃에 따라서 도로를
상황에 맞게 수정

Structure
Manager

· 가중치를 사용하여 랜덤하게
구조물 설치
· 구조물 크기를 확인한 뒤에 설치

UIController

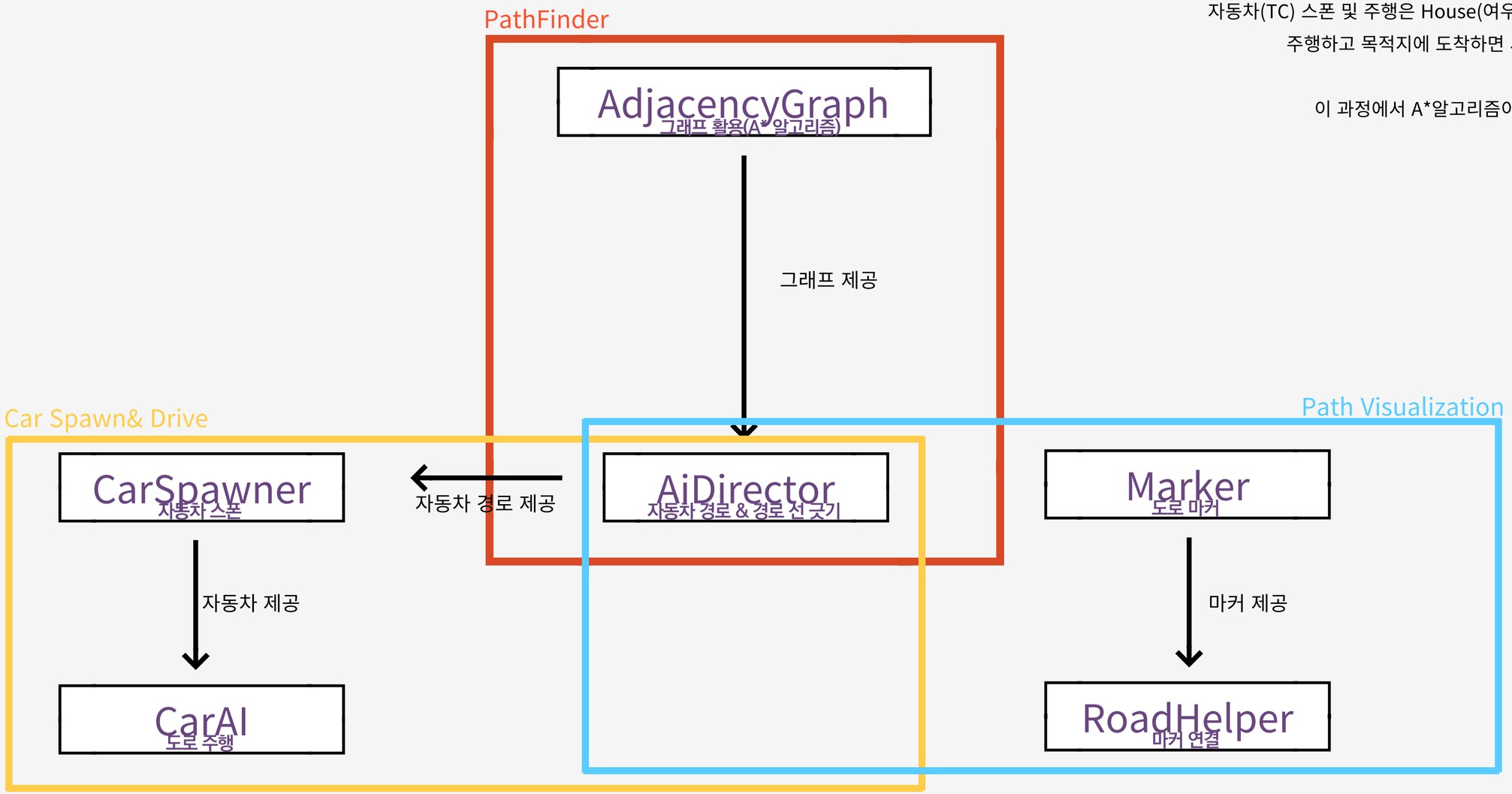
· 버튼을 클릭했을 때, 적절한
구조물을 설치할 수 있게 함
· 버튼을 클릭했을때 테두리 색 변화

클래스 연관성(자동차 주행)

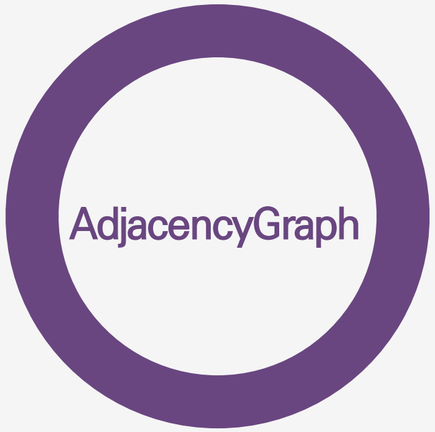
Graph(Edge, Vertex) 사용

자동차(TC) 스폰 및 주행은 House(여우)에서 Special(닭)까지 주행하고 목적지에 도착하면 사라지게 구현

이 과정에서 A*알고리즘이 2번 적용

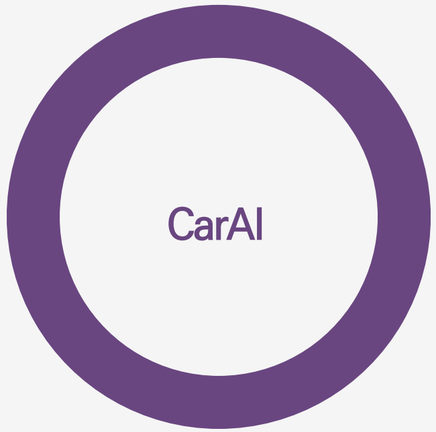


클래스 종류 및 기능(자동차 주행)



· Graph(Edge, Vertex) 제공
· Grid 좌표를 이용하여 주행 경로 생성
(A* 알고리즘 사용)

· 자동차 경로 생성
· 주행 경로에 선으로 시각화



· 자동차 주행
· 자동차 간의 충돌 여부 감지

· 자동차 스폰

클래스 종류 및 기능(자동차 주행)

· 마커 제공

(마커는 도로에 표시된 점, 자동차 주행 경로)

· 마커의 시작과 끝 가져오기

· 가장 가까운 마커끼리 연결

CHAPTER.2

구현한 기능

삭제, 저장

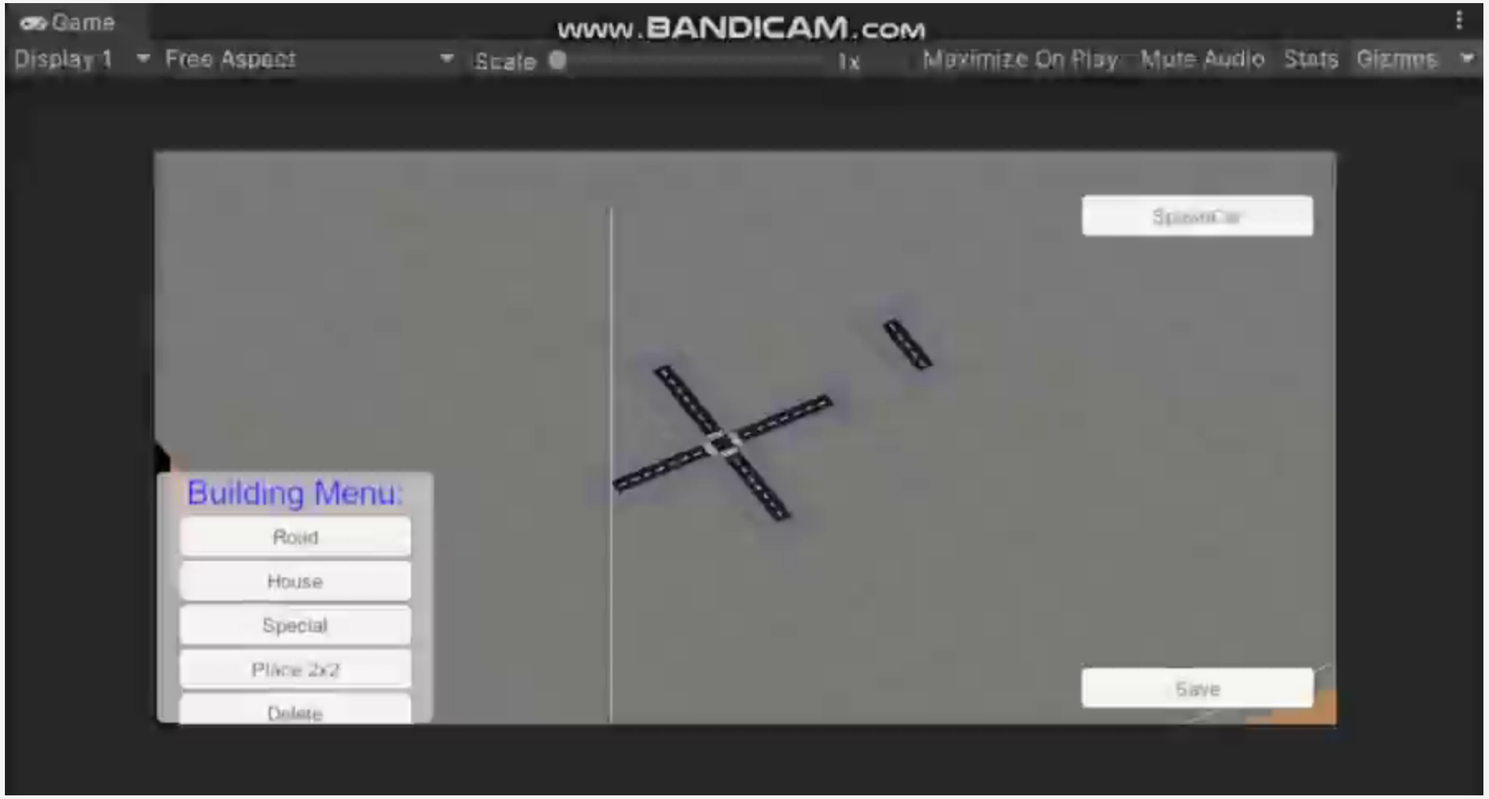
삭제 기능 구현 방법

도로를 설치할 때, 코드를 확인하면 직선 도로의 형태가 default가 되어 클릭하면 그 자리에 설치하고
클릭한 자리의 상하좌우의 위치에 도로의 존재 여부를 확인하여 그 숫자와 위치에 따라 도로의 형태가 변화

이 기능을 참고하여 클릭한 위치의 도로가 삭제될 때도 도로를 삭제(유니티의 Destroy 활용)하고
삭제한 도로 위치의 상하좌우의 도로 존재 여부를 확인하고 클릭한 위치의 도로 삭제 이후에 상황에 맞게 주위의 도로를 변화
삭제 이후의 그 지점의 CellType 또한 변경해주어야 한다.

```
public void FixRoadAfterDelete()
{
    foreach (var positionsToFix in roadManager.roadPositionsToRecheck)
    {
        roadFixer.FixRoadAtPosition(placementManager, positionsToFix);
    }
    placementManager.AddtemporaryStructuresToStructureDictionary();
    roadManager.roadPositionsToRecheck.Clear();
}
```

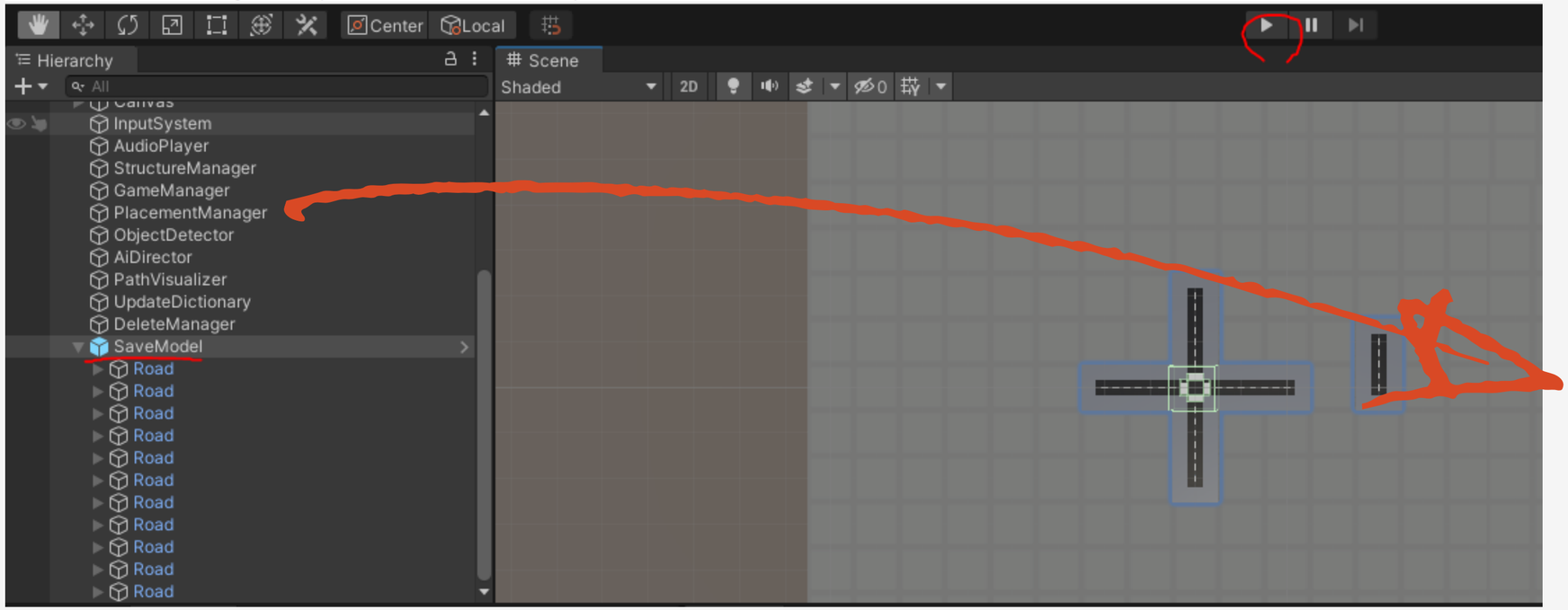
삭제 기능 구현 영상



마우스 클릭하면 해당 위치 도로 삭제 / 마우스를 떼면 삭제 도로의 상하좌우 위치의 도로를 적절하게 수정

삭제하더라도 해당 위치에 도로를 다시 지을 수 있음을 확인 / 삭제하고 다시 지은 도로 주변에도 구조물 생성 가능 확인

저장 기능 구현 방법



게임을 시작하고 나서 도로를 설치할 때는 placementManager 오브젝트의 자식으로 객체 생성

게임 시작전에는 설치했던 도로들이 다 SaveModel 오브젝트의 자식으로 저장

게임이 시작하면 SaveModel 오브젝트 자식으로 있던 도로들의 위치에 새로운 도로를 설치하고 도로를 수정 그러면

총 도로가 2개 있으니 SaveModel의 자식 도로를 다 삭제해서 도로를 한 개만 설치하기

=〉 이렇게 하면 placementManager 오브젝트 자식으로는 도로들이, 최종 딕셔너리에도 도로의 정보들이 저장(작업을 이어서 가능)

저장 기능 구현 방법

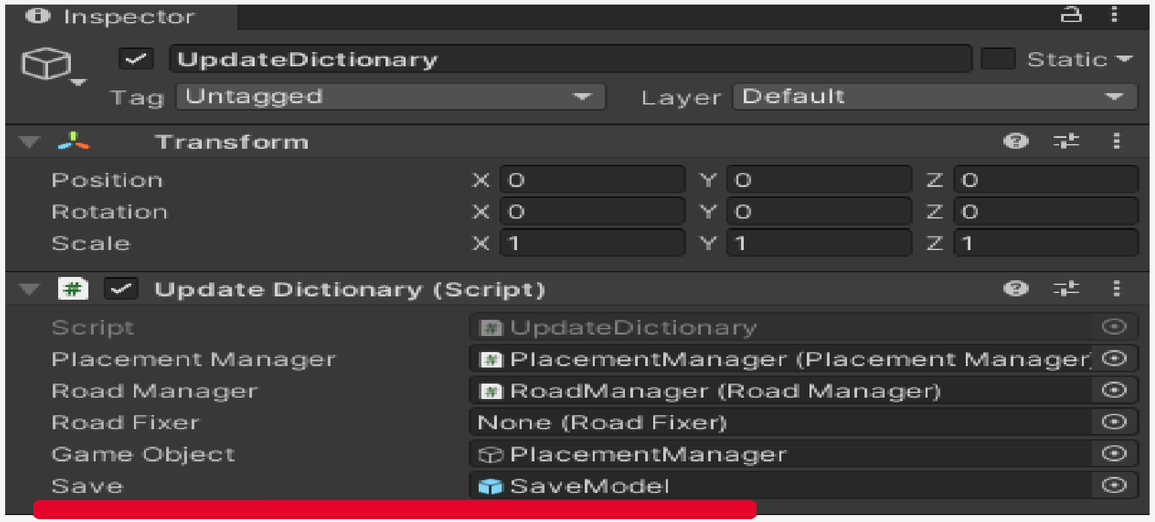
PrefabUtility.SaveAsPrefabAsset

Declaration

public static GameObject **SaveAsPrefabAsset**(GameObject **instanceRoot**, string **assetPath**);

Declaration

public static GameObject **SaveAsPrefabAsset**(GameObject **instanceRoot**, string **assetPath**, out bool **success**);



Save 버튼을 클릭하면 placementManager의 자식 도로들이
모두 SaveModel의 자식으로 돌아가고
이 SaveModel을 Prefab으로 만들어서 저장!

SaveModel 오브젝트 자체를 Inspector에 올려버리면
자동으로 Load된다

번거롭게 다시 올리지 않아도 버튼을 누르면 prefab이 만들어지고
그 Prefab이 인스펙터에 올라간 것이니 게임이 종료돼도
기존에 했던 작업들이 저장이 된다
(일반적으로 저장하는 방법으로 구현 완료)

CHAPTER.3

일방 통행 구현 실패 이유

구현 방법, 실패 이유

일방 통행 구현 방법(실패)

```
17 references
public enum CellType
{
    1 reference
    Empty,
    3 references
    Road,
    1 reference
    Structure,
    1 reference
    SpecialStructure,
    4 references
    None
}
```

CellType에 일방 통행 도로 속성 추가
(일반적인 도로와 일방 통행 도로가 구별돼야 하므로)

시작점부터 목적지까지 이동하면서 도로의 속성이 일방 통행 속성이라면 그 쪽으로 가는 것을 최우선으로 할 수 있게
if문을 넣거나 가중치를 조절하는 방법으로 구현하고자 하였습니다.
(A* 알고리즘을 기반으로 그 조건을 조절해 일방 통행을 구현하고자 하였습니다.)

실패 이유

```
public static List<Point> AStarSearch(Grid grid, Point startPosition, Point endPosition, bool isAgent = false)
{
    List<Point> path = new List<Point>();

    List<Point> positionsTocheck = new List<Point>();
    Dictionary<Point, float> costDictionary = new Dictionary<Point, float>();
    Dictionary<Point, float> priorityDictionary = new Dictionary<Point, float>();
    Dictionary<Point, Point> parentsDictionary = new Dictionary<Point, Point>();

    positionsTocheck.Add(startPosition);
    priorityDictionary.Add(startPosition, 0);
    costDictionary.Add(startPosition, 0);
    parentsDictionary.Add(startPosition, null);

    while (positionsTocheck.Count > 0)
    {
        Point current = GetClosestVertex(positionsTocheck, priorityDictionary);
        positionsTocheck.Remove(current);
        if (current.Equals(endPosition))
        {
            path = GeneratePath(parentsDictionary, current);
            return path;
        }

        foreach (Point neighbour in grid.GetAdjacentCells(current, isAgent))
        {
            float newCost = costDictionary[current] + grid.GetCostOfEnteringCell(neighbour);
            if (!costDictionary.ContainsKey(neighbour) || newCost < costDictionary[neighbour])
            {
                costDictionary[neighbour] = newCost;

                float priority = newCost + ManhattanDiscance(endPosition, neighbour);
                positionsTocheck.Add(neighbour);
                priorityDictionary[neighbour] = priority;

                parentsDictionary[neighbour] = current;
            }
        }
    }
    return path;
}
```

이 코드에서 사용하는 A* 알고리즘

이 코드로 구현한 A* 알고리즘에 대한 이해 부족

가중치 함수도 조절하고
if문으로 CellType이 일방통행 속성일 때 최우선으로 가게 구현하려고 했지만 A* 알고리즘에 대한 이해가 부족한 것 같아 계속 오류가 발생
(priorityDictionary를 중점적으로 활용하여 구현하려고 하였음)

코드가 다양한 함수로 객체지향형 프로그래밍으로 구현되어 있어 일방 통행을 구현하다가 중간에 특정 함수를 빼먹어 오류가 발생한 것 같기도 합니다.

감사합니다