# 게임 프로그래밍 과제5

201903868 황가은

A* 알고리즘을 사용하여 경로 찾기

<플레이 전의 처음 화면>

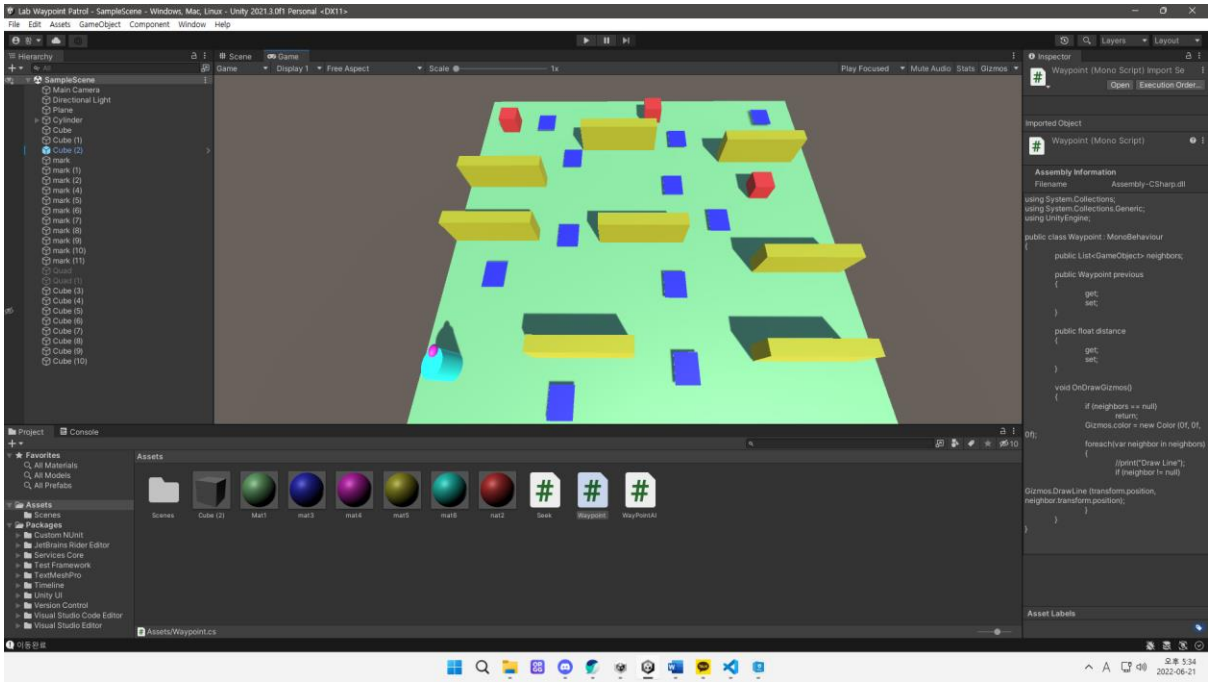

그림 1

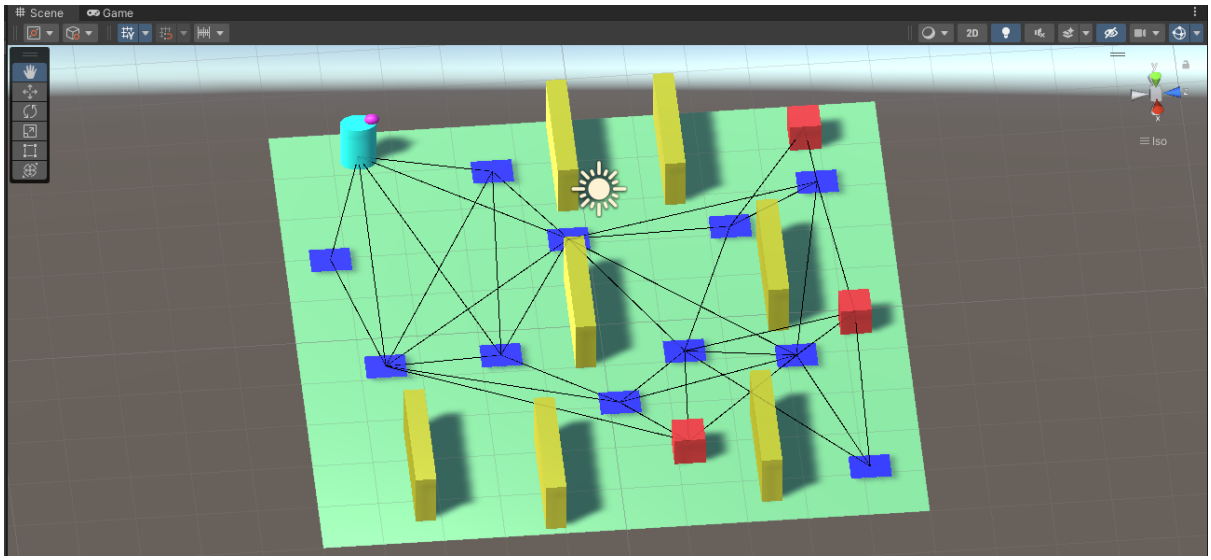

그림 2

그림 2에서 파란색 마크들이 서로 직접적으로 볼 수 있는 관계에 놓여 있다면 직선을 생성하여 Waypoint Graph를 만들었습니다. 서로 직접적으로 볼 수 있는 관계란 노란색 장애물에 가려지지 않고, 서로의 시야에 다른 마크가 있어 가리지 않는다는 경우를 말합니다. 그 과정은 아래와 같습니다.

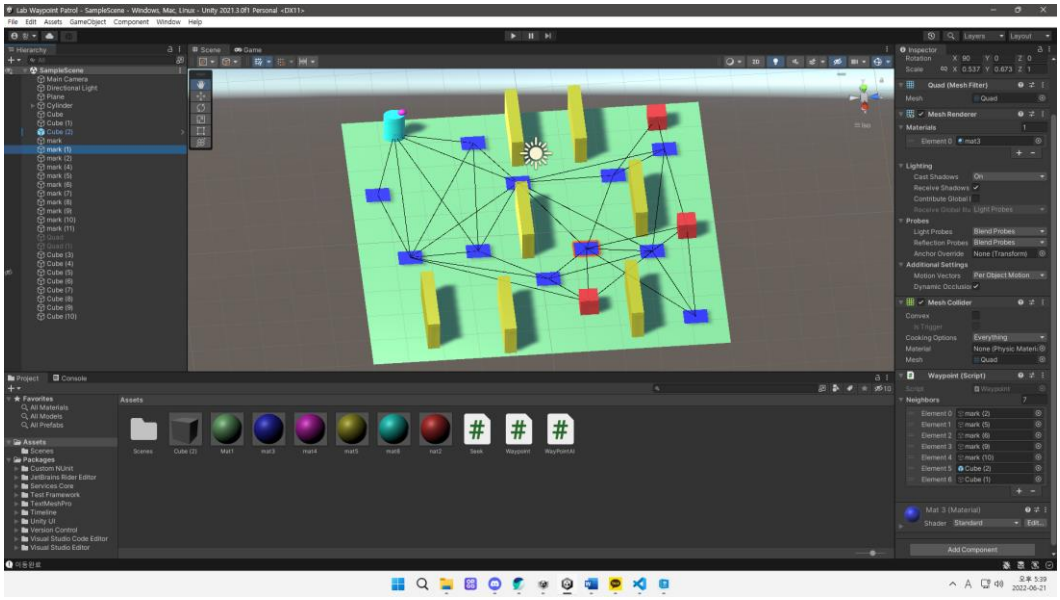

그림 3

예를 들어 mark(1)의 Inspector 창을 보도록 하겠습니다. Waypoint를 연결될 수 있는 마크를 직접 지정하여 넣어 만들었습니다. 또한 mark(1)에서 mark(2)를 지정해주면 Waypoint에서 그 사이의 직선을 만들어주는데, mark(1)만 mark(2)에 접근할 수 있는 게 아닌 mark(2)도 mark(1)에 접근할 수 있다고 판단되면 서로의 Neighbors에 추가해주었습니다. 그렇게 각 마크마다 연결을 해준 결과가 <그림 3>과 같은 Waypoint Graph 형태입니다.

여기에서 쓰인 Waypoin.cs 에서는 바꾼 코드가 없습니다. 다음은 플레이 후의 화면을 보도록 하겠습니다.

<플레이 후의 화면>

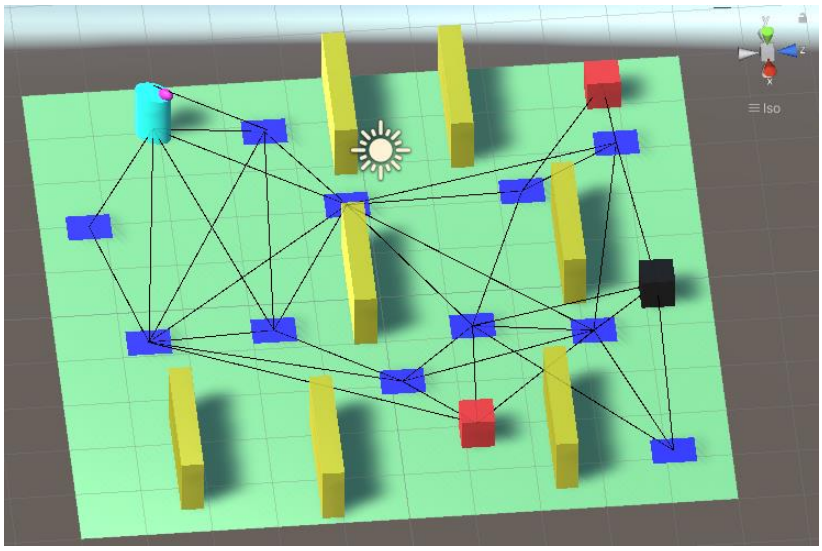

그림 4

도착 목표인 3개의 빨간색 마크 중에서 랜덤으로 하나가 선택되고 검은색으로 변합니다. 또한 청록색 실린더 즉, 캐릭터가 선택된 도착 목표를 향해 이동하기 시작합니다. 청록색 실린더 위에 있는 분홍색 구는 방향을 표시합니다.

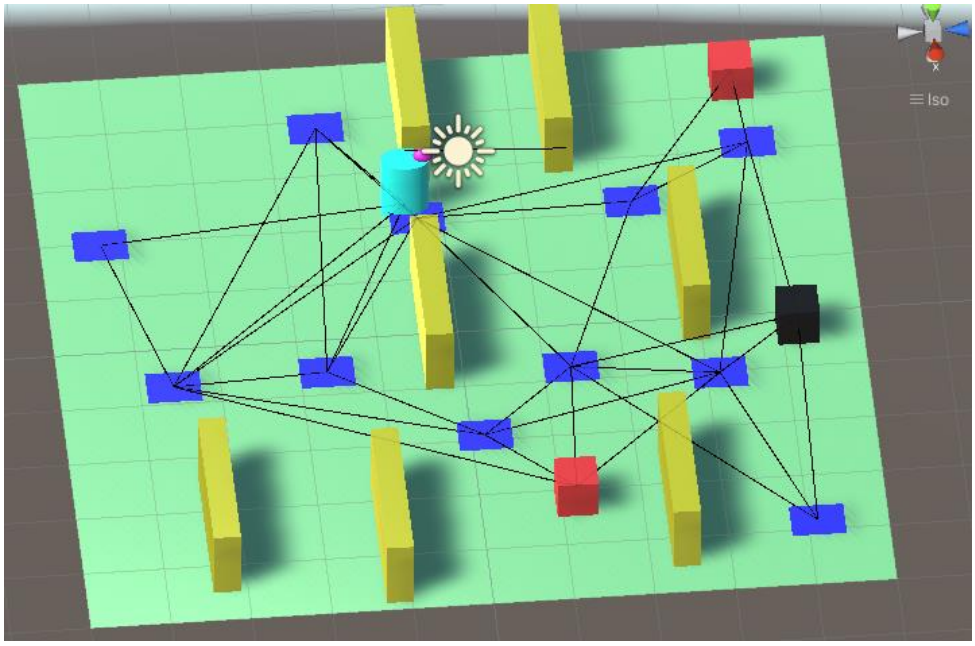


**그림 5**

A* 알고리즘을 적용하여 목표지점에 도착하기 위해 거리를 계산하면서 적절한 mark로 이동합니다. 분홍색 구는 다음 mark를 향해 방향을 업데이트합니다. 이를 반복하여 목표지점에 도착하게 되면 콘솔창에 “이동완료”라는 문구를 표시하여 이동을 종료합니다.

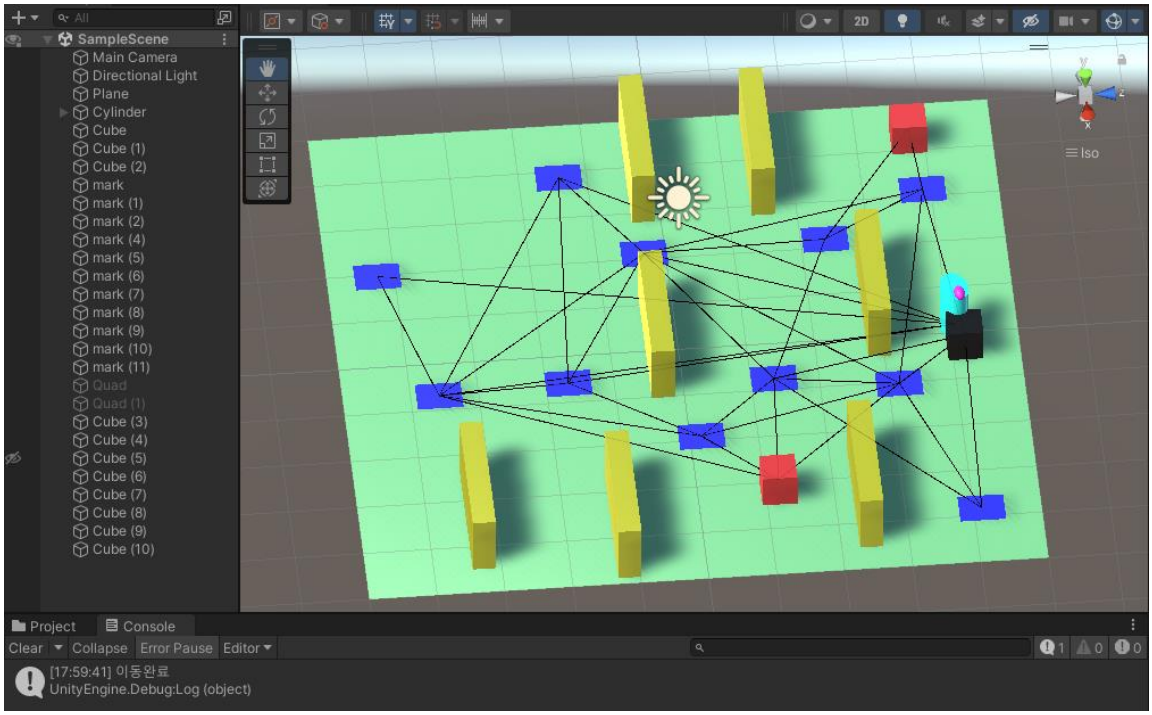


**그림 6**

<코드: WayPointAI.cs>

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Linq;
public class WayPointAI : MonoBehaviour
{
    //This array is used to set which points are the waypoints
    public GameObject[] wayPoints;
    public GameObject[] GoalPoints;
    public GameObject Cylinder;
    Waypoint WP;
    MeshRenderer mr;

    List<GameObject> OpenList = new List<GameObject>();
    List<GameObject> ClosedList = new List<GameObject>();

    //Speed and distance. Distance is used to determine how close you
    //want the object to get before going to next point.
    public float spd;
    public float distance;

    //Holds the current waypoint it is going to


    private bool change = false;
    List<float> F = new List<float>();
    List<float> G = new List<float>();
    List<float> H = new List<float>();

    //Holds the position of the waypoint it's heading towards.
    private Vector3 targetPosition;  //목표 지점
    private Vector3 StartPosition;  //시작 지점
    private Vector3 currentPoint;  //현재 지점
    private Vector3 N_Position;  //다음 지점
    private GameObject NextPos;  //다음 지점에 있는 mark

    // Use this for initialization
    void Start()
    {
        Cylinder = GameObject.Find("Cylinder");
        int Ppos = Random.Range(0,3);  //도착 목표 지점을 랜덤 선택
        mr = GoalPoints[Ppos].GetComponent<MeshRenderer>();
        targetPosition = GoalPoints[Ppos].transform.position;  //목표 지점 설정
        StartPosition = Cylinder.transform.position;  //시작 지점 설정
        mr.material.color = Color.black;  //도착 mark 색 변경
```

리스트 내에서 Min/Max 기능을 사용하기 위해서 Linq를 사용했습니다.

거쳐가는 지점인 mark와 목표 지점인 mark를 구분하기 위해 오브젝트를 나눠주었다.

Waypoint 스크립트를 사용하기 위해 선언

도착 목표의 색을 바꾸기 위해 선언

갈 수 있는 mark를 모은 OpenList와 갈 수 없는 mark를 closedList로 나눴다.

F = G + H 를 만들기 위해 각 마크들의 값을 넣기 위해 리스트를 생성함.

```
        SetArr("Cylinder");
        NextPos = NextNode();
        N_Position = NextPos.transform.position;
        ClosedList.Add(Cylinder);
        StartCoroutine(MoveNode(N_Position)); //코루틴 시작
    }

    public void SetArr(string a)
    {
        OpenList.Clear();
        WP = GameObject.Find(a).GetComponent<Waypoint>();
        foreach(var neighbor in WP.neighbors)
        {
            OpenList.Add(neighbor);
        }

        if(ClosedList.Count != 0 && OpenList.Count != 0)
        {
            for(int q = 0; q < OpenList.Count; q++)
            {
                for(int b = 0; b < ClosedList.Count; b++)
                {
                    if(OpenList[q] == ClosedList[b])
                    {
                        OpenList.Remove(OpenList[q]);
                        q=0;
                    }
                }
            }
        }
    }

    public GameObject NextNode()
    {
        currentPoint = Cylinder.transform.position;
        for(int i = 0; i < OpenList.Count; i++)
        {
            float gval =
Mathf.Sqrt(Mathf.Pow((OpenList[i].transform.position.x - currentPoint.x), 2) +
Mathf.Pow((OpenList[i].transform.position.z - currentPoint.z), 2));
            G.Add(gval);

            float hval = Mathf.Sqrt(Mathf.Pow((targetPosition.x -
OpenList[i].transform.position.x), 2) + Mathf.Pow((targetPosition.z -
OpenList[i].transform.position.z), 2));
            H.Add(hval);
```

갈 수 있는 지점을 OpenList에 넣는 함수이다. 여기에서 Waypoint의 스크립트를 사용해서 mark의 이웃을 찾아 먼저 Openlist에 넣고 ClosedList와 검사하여 만약 ClosedList 안의 요소가 Openlist에 있다면 그 요소를 제거함.

Remove를 통한 인덱스 접근 오류를 해결하는 q 초기화

다음 마크를 정하는 함수이다. 이 함수에서 A* 알고리즘을 사용하여 F, G, H 값을 알아내어 각각의 리스트에 저장한다.

```
            F.Add(gval + hval);
        }
        float minF = F.Min();
        int minF_idx = 0;
        for(int j = 0; j < F.Count; j ++)
        {
            if(F[j] == minF)
            {
                minF_idx = j;
            }
        }

        return OpenList[minF_idx];
    }


    IEnumerator MoveNode(Vector3 N_Position)
    {
        while(true)
        {
            float dir = 0;
            dir = -Mathf.Atan2(currentPoint.z - N_Position.z, currentPoint.x -
N_Position.x) - 90 * Mathf.Deg2Rad;
            Vector3 sp = new Vector3(Cylinder.transform.position.x, 1,
Cylinder.transform.position.z);
            Debug.DrawRay(sp, new Vector3(Mathf.Sin(dir), 0, Mathf.Cos(dir)) *
2, Color.black);
            Cylinder.transform.rotation = Quaternion.AngleAxis(dir *
Mathf.Rad2Deg, Vector3.up);
            Cylinder.transform.Translate(Vector3.forward * spd *
Time.deltaTime);
            change = false;

            if(Vector3.Distance(Cylinder.transform.position, N_Position) <
distance && !change)
            {

                F.Clear();
                G.Clear();
                H.Clear();

                ClosedList.Add(NextPos);
                string str = NextPos.ToString();
                str = str.Substring(0, str.LastIndexOf(' '));

                if(NextPos.ToString().Substring(0,
NextPos.ToString().IndexOf(' ')) != "Cube")
```

순차적으로 넣어졌으므로 F 리스트 내의 최솟값을 가진 인덱스를 찾는다. 그리고 OpenList에서 해당 인덱스 값을 반환한다. 그 값이 바로 다음 지점이다.

인자 N_Postion, 다음 지점의 Vector3 위치 값을 받아서 코루틴을 통해 캐릭터를 움직이는 함수이다.

만약 캐릭터의 위치와 마크의 거리가 가까워진다는 조건문인데, 이는 이동하려는 마크로의 도착을 의미한다.

그래서 F, G, H의 리스트를 초기화한다. 그리고 ClosedList에 도착한 마크를 추가한다.

도착한 마크가 목표 지점이 아니라면 다음 마크를 탐색하기 위해 SetArr 함수를 호출합니다.

```
                {
                    SetArr(str);
                    /*
                    for(int n = 0; n<OpenList.Count; n++)
                    {
                        Debug.Log("열린 리스트 : "
+OpenList[n].ToString().Substring(0, OpenList[n].ToString().LastIndexOf(' '))
+ " " + n + "번째 인덱스");

                    }

                    for(int m = 0; m<ClosedList.Count; m++)
                    {
                        Debug.Log("닫힌 리스트 : " +
ClosedList[m].ToString().Substring(0, ClosedList[m].ToString().LastIndexOf('
')) +" " + m + "번째 인덱스");
                    }
                    */
                    NextPos = NextNode();
                    //Debug.Log(NextPos.ToString());
                    N_Position = NextPos.transform.position;
                }
                else
                {
                    Debug.Log("이동완료");
                    break;
                }
            }
            yield return null;
        }
    }
}
```

다음 마크의 위치 정보를 저장합니다.

도착한 곳이 목표 지점이라면 콘솔에 이동완료 문구를 출력하고 반복문을 종료합니다.

WayPoint.cs와 Seek.cs에는 변경 사항이 없습니다. Update문을 사용하지 않고 코루틴을 사용하여 캐릭터를 움직였습니다. 또한 A* 알고리즘을 적용하기 위해서 두점 사이의 거리 공식을 이용했습니다. 휴리스틱 함수는 목표지점과 이동할 지점의 사이 거리를 사용했습니다. G 함수는 현재 지점과 이동할 지점 사이의 거리입니다.