

**숙제 5 A star 알고리즘으로 경로 찾기**

**과목명 게임프로그래밍**

**담당교수님 김상철교수님**

**제출일 20220621**

**전공 컴퓨터전자시스템**

**학번 201904458**

**이름 이준용**

1. Patrol.cs와 PatrolCoroutines.cs 라는 스크립트를 새로 만들었습니다. 위의 스크립트는 Character가 mark를 바라 보면서 이동하게 만들어주는 스크립트 입니다.

Patrol.cs

using UnityEngine;

public class Patrol : MonoBehaviour

{

//This array is used to set which points are the waypoints

    //public GameObject[] wayPoints;

    public Transform[] waypoints;

    private int \_currentWaypointIndex = 0;

    private float \_speed = 2f;

    private float \_waitTime = 1f; // in seconds

    private float \_waitCounter = 0f;

    private bool \_waiting = false;

    //public Transform goal;

    private void Update()

    {

        if (\_waiting)

        {

            \_waitCounter += Time.deltaTime;

            if (\_waitCounter < \_waitTime)

                return;

            \_waiting = false;

        }

        Transform wp = waypoints[\_currentWaypointIndex];

        if (Vector3.Distance(transform.position, wp.position) < 0.01f)

        {

            transform.position = wp.position;

            \_waitCounter = 0f;

            \_waiting = true;

            \_currentWaypointIndex = (\_currentWaypointIndex + 1) % waypoints.Length;

        }

        else

        {

            transform.position = Vector3.MoveTowards(transform.position, wp.position, \_speed \* Time.deltaTime);

            transform.LookAt(wp.position);

        }

    }

}

PatrolCoroutines.cs

using System.Collections;

using UnityEngine;

public class PatrolCoroutines : MonoBehaviour

{

    public Transform[] waypoints;

    private int \_currentWaypointIndex = 0;

    private float \_speed = 2f;

    private float \_waitTime = 1f; // in seconds

    private Coroutine \_prevCoroutine;

    private void Start()

    {

        \_prevCoroutine = StartCoroutine(\_MovingToNextWaypoint());

    }

    private IEnumerator \_MovingToNextWaypoint()

    {

        Transform wp = waypoints[\_currentWaypointIndex];

        while (Vector3.Distance(transform.position, wp.position) > 0.01f)

        {

            transform.position = Vector3.MoveTowards(transform.position, wp.position, \_speed \* Time.deltaTime);

            yield return null;

        }

        transform.position = wp.position;

        yield return new WaitForSeconds(\_waitTime);

        \_currentWaypointIndex = (\_currentWaypointIndex + 1) % waypoints.Length;

        StopCoroutine(\_prevCoroutine);

        \_prevCoroutine = StartCoroutine(\_MovingToNextWaypoint());

    }

}

Seek.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class Seek : MonoBehaviour

{

    public float Mass = 15;

    public float MaxVelocity = 3;

    public float MaxForce = 15;

    private Vector3 velocity;

    public Transform target;

    private void Start()

    {

        UnityEngine.AI.NavMeshAgent agent = GetComponent<UnityEngine.AI.NavMeshAgent> ();

        agent.destination = target.position;

        velocity = Vector3.zero;

    }

    private void Update()

    {

        var desiredVelocity = target.transform.position - transform.position;

        desiredVelocity = desiredVelocity.normalized \* MaxVelocity;

        var steering = desiredVelocity - velocity;

        steering = Vector3.ClampMagnitude(steering, MaxForce);

        steering /= Mass;

        velocity = Vector3.ClampMagnitude(velocity + steering, MaxVelocity);

        transform.position += velocity \* Time.deltaTime;

        transform.forward = velocity.normalized;

        Debug.DrawRay(transform.position, velocity.normalized \* 2, Color.green);

        Debug.DrawRay(transform.position, desiredVelocity.normalized \* 2, Color.magenta);

    }

}

WalkToGoal.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class WalkToGoal : MonoBehaviour {

    public Transform goal;

    // Use this for initialization

    void Start () {

        NavMeshAgent agent = GetComponent<NavMeshAgent> ();

        GameObject[] targets = GameObject.FindGameObjectsWithTag("destination");

        // GameObject randomTarget = targets [Random.Range(0, targets.length())];

        agent.destination=goal.position;

    }

    // Update is called once per frame

    void Update () {

    }

}

Waypoint.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

public class Waypoint : MonoBehaviour

{

//  public List<Node> GetNeighbours(Node node)

// {

//     List<Node> neighbours = new List<Node>();

//     int[,] temp = { { 0, 1 }, { 1, 0 }, { 0, -1 }, { -1, 0 } };

//     bool[] walkableUDLR = new bool[4];

//     //상하좌우의 노드 먼저 계산

//     for (int i = 0; i < 4; i++)

//     {

//         int checkX = node.gridX + temp[i, 0];

//         int checkY = node.gridY + temp[i, 1];

//         if (checkX >= 0 && checkX < (int)gridWorldSize.x && checkY >= 0 && checkY < (int)gridWorldSize.y)

//         {

//             if (grid[checkX, checkY].walkable)

//                 walkableUDLR[i] = true;

//             neighbours.Add(grid[checkX, checkY]);

//         }

//     }

//     //대각선의 노드를 계산

//     for (int i = 0; i < 4; i++)

//     {

//         if (walkableUDLR[i] || walkableUDLR[(i + 1) % 4])

//         {

//             int checkX = node.gridX + temp[i, 0] + temp[(i + 1) % 4, 0];

//             int checkY = node.gridY + temp[i, 1] + temp[(i + 1) % 4, 1];

//             if (checkX >= 0 && checkX < (int)gridWorldSize.x && checkY >= 0 && checkY < (int)gridWorldSize.y)

//             {

//                 neighbours.Add(grid[checkX, checkY]);

//             }

//         }

//     }

//     return neighbours;

// }

    public List<GameObject> neighbors;

    public Waypoint previous

    {

        get;

        set;

    }

    public float distance

    {

        get;

        set;

    }

    void OnDrawGizmos()

    {

        if (neighbors == null)

            return;

        Gizmos.color = new Color (0f, 0f, 0f);

        foreach(var neighbor in neighbors)

        {

            //print("Draw Line");

            if (neighbor != null)

                Gizmos.DrawLine (transform.position, neighbor.transform.position);

        }

    }

}

WayPointAI.cs

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.AI;

//using UnityEngine;

//using System.Collections;

public class WayPointAI : MonoBehaviour

{

    //This array is used to set which points are the waypoints

    public GameObject[] wayPoints;

    //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

    public int currentPoint;

    private bool change = false;

    //Holds the position of the waypoint it's heading towards.

    private Vector3 targetPosition;

    // Use this for initialization

    void Start()

    {

        //Sets  the first waypointPosition

        targetPosition = wayPoints[currentPoint].transform.position;

    }

    // Update is called once per frame

    void Update()

    {

        //Checks if the distance from the object to the way point is less than distance given

        //Once true it will increment to the next point.

        if (Vector3.Distance(transform.position, targetPosition) < distance && !change)

        {

            nextPoint();

        }

        else

        {

            float dir = 0;

            //Moving and turning

            //Gets the direction to face

            dir = -Mathf.Atan2(transform.position.z - targetPosition.z, transform.position.x - targetPosition.x) - 90 \* Mathf.Deg2Rad;

            //Draws the line to show where the object is going.

            Vector3 sp = new Vector3(transform.position.x,

                1, transform.position.z);

            //Debug.DrawRay(transform.position, new Vector3(Mathf.Sin(dir), 0, Mathf.Cos(dir)) \* 20);

            Debug.DrawRay(sp, new Vector3(Mathf.Sin(dir), 0, Mathf.Cos(dir)) \* 2, Color.black);

            //Rotates the object towards to the waypoint.

            transform.rotation = Quaternion.AngleAxis(dir \* Mathf.Rad2Deg, Vector3.up);

            //Moves the object forward

            transform.Translate(Vector3.forward \* spd \* Time.deltaTime);

            change = false;

        }

    }

    private void nextPoint()

    {

        print("nextPoint()");

        //Checks to see if it reached the max waypoint possible

        if (currentPoint >= wayPoints.Length - 1)

            //If true sets it to 0 so array doesn't go out of bounds.

            currentPoint = 0;

        else

            currentPoint++;

        //Sets the the targetPosition to the next waypoint.

        targetPosition = wayPoints[currentPoint].transform.position;

        change = true;

    }

}

// void FindPath()

// {

//     List<Node> openSet = new List<Node>();          //OPEN

//     HashSet<Node> closedSet = new HashSet<Node>();  //CLOSE

//     openSet.Add(start);                             //OPEN에 시작노드 저장

//     while (openSet.Count > 0)

//     {

//         Node currentNode = openSet[0];

//         //OPEN에 fCOST가 가장 작은 노드를 찾기

//         for(int i = 1; i<openSet.Count; i++)

//         {

//             if (openSet[i].fCost < currentNode.fCost || openSet[i].fCost == currentNode.fCost && openSet[i].hCost < currentNode.hCost)

//             {

//                 currentNode = openSet[i];

//             }

//         }

//         openSet.Remove(currentNode);

//         closedSet.Add(currentNode);

//         //도착지점에 오면 종료

//         if (currentNode == end)

//         {

//             return;

//         }

//         if (currentNode != start)

//             currentNode.ChangeColor = Color.Lerp(Color.cyan, Color.white, 0.2f);

//         //이웃 노드를 검색

//         foreach (Node neighbour in grid.GetNeighbours(currentNode))

//         {

//             //이동불가 노드 이거나 이미 검색한 노드 제외

//             if (!neighbour.walkable  || closedSet.Contains(neighbour))

//             {

//                 continue;

//             }

//             int newMovementCostToNeighbour = currentNode.gCost + GetDistance(currentNode, neighbour);

//             if (newMovementCostToNeighbour < neighbour.gCost || !openSet.Contains(neighbour))

//             {

//                 neighbour.gCost = newMovementCostToNeighbour;

//                 neighbour.hCost = GetDistance(neighbour, end);

//                 neighbour.parent = currentNode;

//                 if (!openSet.Contains(neighbour))

//                 {

//                     openSet.Add(neighbour);

//                     if (neighbour.walkable && !neighbour.end)

//                         neighbour.ChangeColor = Color.Lerp(Color.green, Color.white, 0.2f);

//                 }

//             }

//         }

//     }

// }

// //노드간의 거리 계산

// int GetDistance(Node nodeA, Node nodeB)

//     {

//         int dstX = Mathf.Abs(nodeA.gridX - nodeB.gridX);

//         int dstY = Mathf.Abs(nodeA.gridY - nodeB.gridY);

//         if (dstX > dstY)

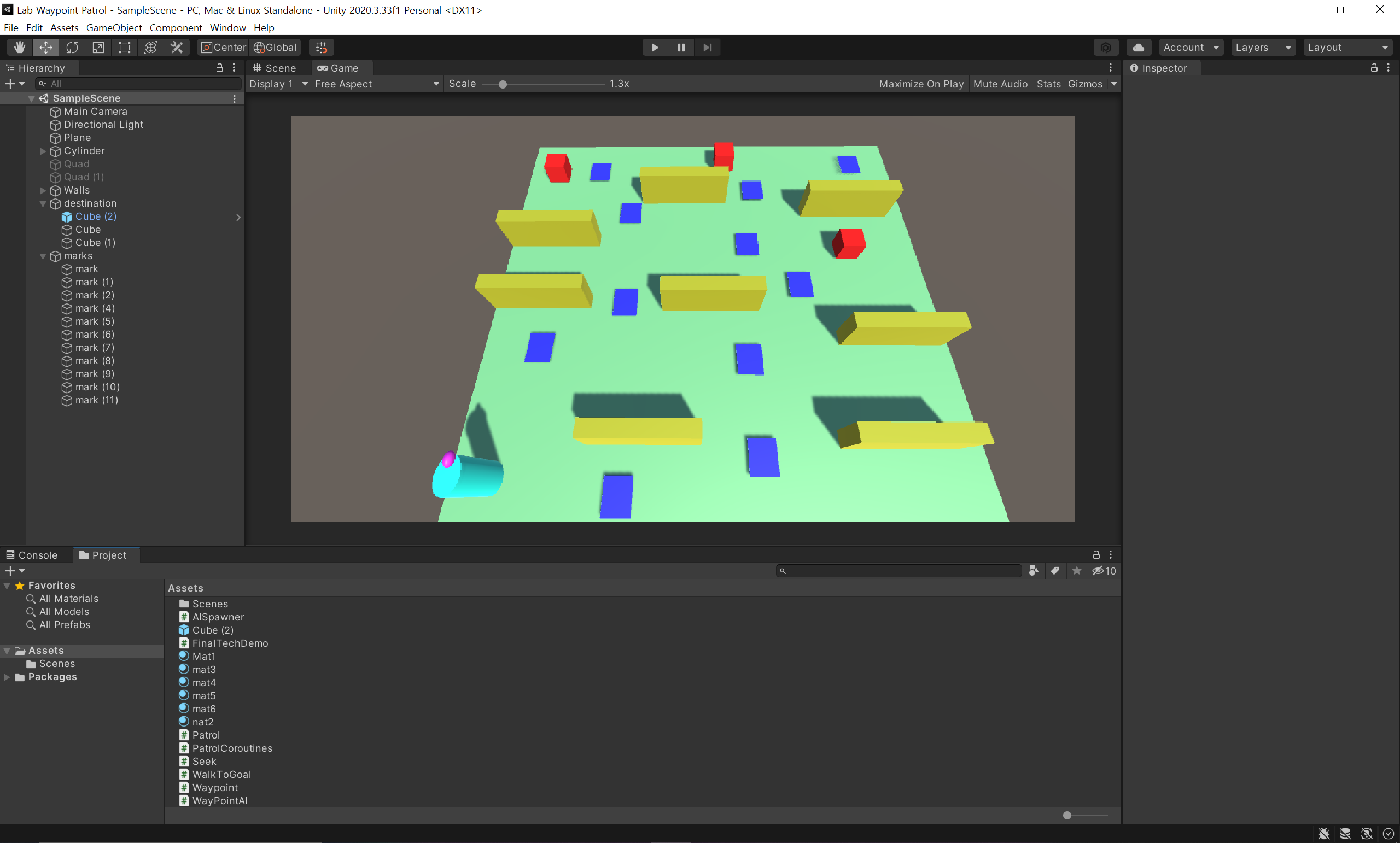
//             return 14 \* dstY + 10 \* (dstX - dstY);

//         return 14 \* dstX + 10 \* (dstY - dstX);

//     }

실행화면

1. 게임 시작전 화면



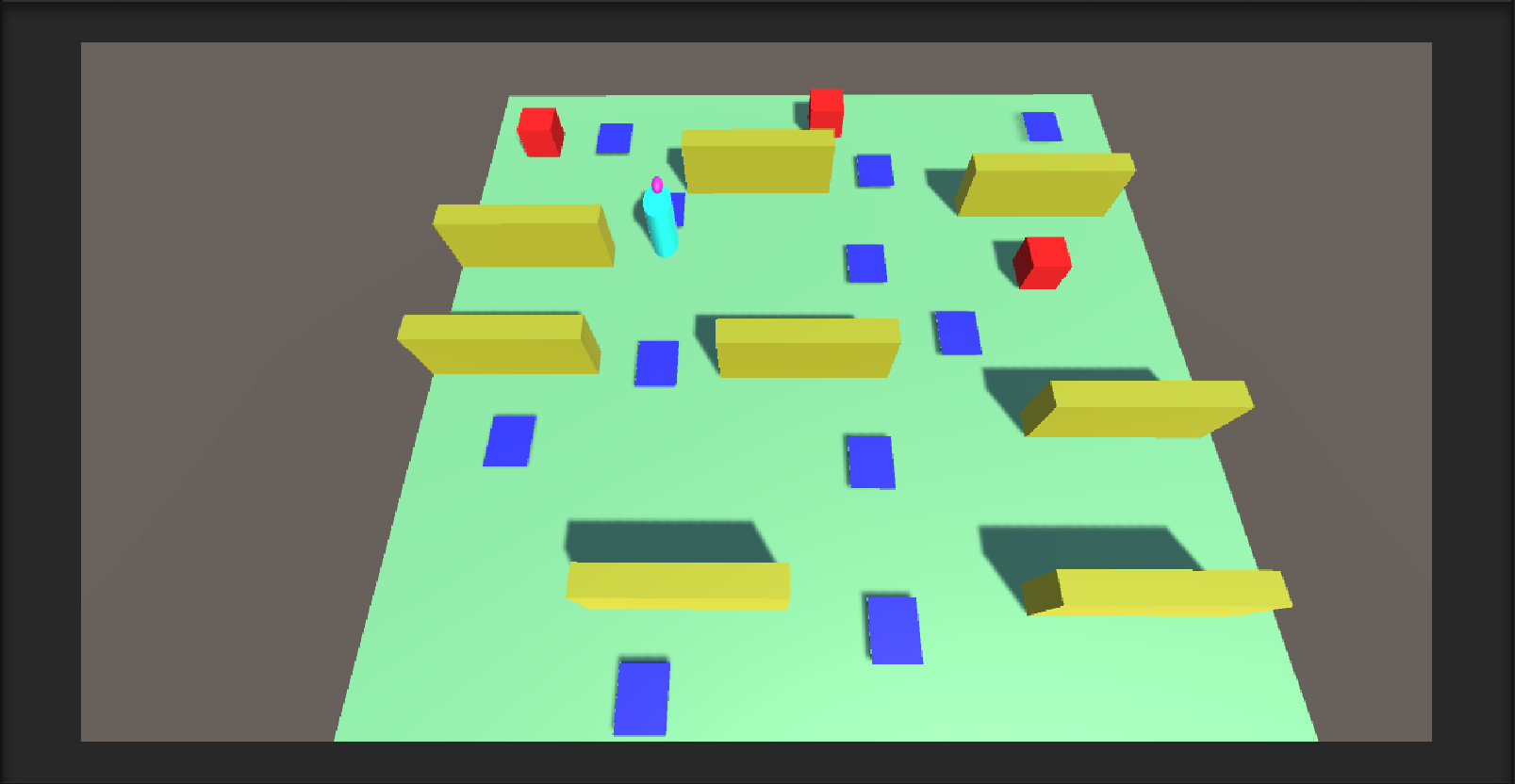
1. Cube라는 GameObject에 이동하는 장면

텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명



텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

1. Cube에 도착한 모습

텍스트, 디스플레이이(가) 표시된 사진

자동 생성된 설명

감사합니다.