A * Algorithm:

Code:-

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

public class AStar : MonoBehaviour {

    public List<Node> FindPath(Node startNode, Node targetNode, Node[] allNodes) {
        List<Node> openSet = new List<Node>();
        openSet.Add(startNode);

        List<Node> closedSet = new List<Node>();

        while (openSet.Count > 0) {
            Node currentNode = openSet[0];
            for (int i = 1; i < openSet.Count; i++) {
                if (openSet[i].FCost < currentNode.FCost
                    || (openSet[i].FCost.Equals(currentNode.FCost)
                        && openSet[i].HCost < currentNode.HCost)) {
                    currentNode = openSet[i];
                }
            }

            openSet.Remove(currentNode);
            closedSet.Add(currentNode);

            if (currentNode == targetNode) {
                Debug.Log("RETURNING CORRECT NODE!!");
                return RetracePath(startNode, targetNode);
            }

            foreach (Node connection in currentNode.neighbors) {
                if (!closedSet.Contains(connection)) {
                    float costToConnection = currentNode.GCost + GetEstimate(currentNode,
connection) + connection.Cost;

                    if (costToConnection < connection.GCost || !openSet.Contains(connection)) {
                        connection.GCost = costToConnection;
                        connection.HCost = GetEstimate(connection, targetNode);
                        connection.Parent = currentNode;

                        if (!openSet.Contains(connection)) {
                            openSet.Add(connection);
                        }
                    }
                }
```

```
            }
        }
        Debug.Log("RETURNING NULL");
        return null;
    }

    private static List<Node> RetracePath(Node startNode, Node endNode) {
        List<Node> path = new List<Node>();

        Node currentNode = endNode;

        while (currentNode != startNode) {
            path.Add(currentNode);
            currentNode = currentNode.Parent;
        }

        path.Reverse();

        return path;
    }

    private float GetEstimate(Node first, Node second) {
        float distance;

        float xDistance = Mathf.Abs(first.pos.x - first.pos.x);
        float yDistance = Mathf.Abs(second.pos.z - second.pos.z);

        if (xDistance > yDistance) {
            distance = 14 * yDistance + 10 * (xDistance - yDistance);
        } else {
            distance = 14 * xDistance + 10 * (yDistance - xDistance);
        }

        return distance;
    }
}
```

CODE FOR CREATE MAP:-

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.UI;
using UnityEngine.XR.iOS;
using System.Runtime.InteropServices;
using System.IO;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

[RequireComponent(typeof(CustomShapeManager))]
public class CreateMap : MonoBehaviour, PlacenoteListener {

    public Text debugText;

    private const string MAP_NAME = "GenericMap";

    private CustomShapeManager shapeManager;

    private bool shouldRecordWaypoints = false;
    private bool shouldSaveMap = true;
    private bool mARInit = false;

    private UnityARSessionNativeInterface mSession;

    private LibPlacenote.MapMetadataSettable mCurrMapDetails;

    private BoxCollider mBoxColliderDummy;
    private SphereCollider mSphereColliderDummy;
    private CapsuleCollider mCapColliderDummy;

    // Use this for initialization
    void Start() {

        shapeManager = GetComponent<CustomShapeManager>();

        Input.location.Start();

        mSession = UnityARSessionNativeInterface.GetARSessionNativeInterface();
        StartARKit();
        FeaturesVisualizer.EnablePointcloud();
        LibPlacenote.Instance.RegisterListener(this);
    }
```

```
    void OnDisable() {
    }

    // Update is called once per frame
    void Update() {
        if (!mARInit && LibPlacenote.Instance.Initialized())
        {
            Debug.Log("Ready To Start!");
            mARInit = true;

            return;
        }

        if (shouldRecordWaypoints) {
            Transform player = Camera.main.transform;
            //create waypoints if there are none around
            Collider[] hitColliders = Physics.OverlapSphere(player.position, 1f);
            int i = 0;
            while (i < hitColliders.Length) {
                if (hitColliders[i].CompareTag("waypoint")) {
                    return;
                }
                i++;
            }
            Vector3 pos = player.position;
            Debug.Log(player.position);
            pos.y = -.5f;
            shapeManager.AddShape(pos, Quaternion.Euler(Vector3.zero), false);
        }
    }

    public void CreateDestination() {
        shapeManager.AddDestinationShape();
    }

    private void StartARKit() {
        Debug.Log("Initializing ARKit");
        Application.targetFrameRate = 60;
        ConfigureSession();
    }


    private void ConfigureSession() {
#if !UNITY_EDITOR
                ARKitWorldTrackingSessionConfiguration config = new
ARKitWorldTrackingSessionConfiguration ();
```

```
                if (UnityARSessionNativeInterface.IsARKit_1_5_Supported ()) {
                        config.planeDetection = UnityARPlaneDetection.HorizontalAndVertical;
                } else {
                        config.planeDetection = UnityARPlaneDetection.Horizontal;
                }

                config.alignment = UnityARAlignment.UnityARAlignmentGravity;
                config.getPointCloudData = true;
                config.enableLightEstimation = true;
                mSession.RunWithConfig (config);
#endif
    }

    public void OnStartNewClick()
    {
        ConfigureSession();

        if (!LibPlacenote.Instance.Initialized())
        {
            Debug.Log("SDK not yet initialized");
            return;
        }

        Debug.Log("Started Session");
        LibPlacenote.Instance.StartSession();

        //start drop waypoints
        Debug.Log("Dropping Waypoints!!");
        shouldRecordWaypoints = true;
    }

    public void OnSaveMapClick() {
        OverwriteExistingMap();
    }

    void OverwriteExistingMap() {
        if (!LibPlacenote.Instance.Initialized()) {
            Debug.Log("SDK not yet initialized");
            return;
        }

        // Overwrite map if it exists.
        LibPlacenote.Instance.SearchMaps(MAP_NAME, (LibPlacenote.MapInfo[] obj) => {
            bool foundMap = false;
            foreach (LibPlacenote.MapInfo map in obj) {
```

```
            if (map.metadata.name == MAP_NAME) {
                foundMap = true;
                LibPlacenote.Instance.DeleteMap(map.placeId, (deleted, errMsg) => {
                    if (deleted) {
                        Debug.Log("Deleted ID: " + map.placeId);
                        SaveCurrentMap();
                    } else {
                        Debug.Log("Failed to delete ID: " + map.placeId);
                    }
                });
            }
        }

        if (!foundMap) {
            SaveCurrentMap();
        }
    });
}

void SaveCurrentMap() {
    if (shouldSaveMap) {
        shouldSaveMap = false;

        if (!LibPlacenote.Instance.Initialized()) {
            Debug.Log("SDK not yet initialized");
            return;
        }

        bool useLocation = Input.location.status == LocationServiceStatus.Running;
        LocationInfo locationInfo = Input.location.lastData;

        Debug.Log("Saving...");
        debugText.text = "uploading...";
        LibPlacenote.Instance.SaveMap(
            (mapId) => {
                LibPlacenote.Instance.StopSession();

                LibPlacenote.MapMetadataSettable metadata = new
LibPlacenote.MapMetadataSettable();
                metadata.name = MAP_NAME;
                Debug.Log("Saved Map Name: " + metadata.name);

                JObject userdata = new JObject();
                metadata.userdata = userdata;

                JObject shapeList = GetComponent<CustomShapeManager>().Shapes2JSON();
```

```
                userdata["shapeList"] = shapeList;

                if (useLocation) {
                    metadata.location = new LibPlacenote.MapLocation();
                    metadata.location.latitude = locationInfo.latitude;
                    metadata.location.longitude = locationInfo.longitude;
                    metadata.location.altitude = locationInfo.altitude;
                }
                LibPlacenote.Instance.SetMetadata(mapId, metadata);
                mCurrMapDetails = metadata;
            },
            (completed, faulted, percentage) => {
                if (completed) {
                    Debug.Log("Upload Complete:" + mCurrMapDetails.name);
                    debugText.text = "upload complete!!";
                } else if (faulted) {
                    Debug.Log("Upload of Map Named: " + mCurrMapDetails.name + "faulted");
                } else {
                    Debug.Log("Uploading Map Named: " + mCurrMapDetails.name + "(" +
percentage.ToString("F2") + "/1.0)");
                }
            }
        );
    }
  }

  public void OnPose(Matrix4x4 outputPose, Matrix4x4 arkitPose) { }

  public void OnStatusChange(LibPlacenote.MappingStatus prevStatus,
LibPlacenote.MappingStatus currStatus) {
    Debug.Log("prevStatus: " + prevStatus.ToString() + " currStatus: " +
currStatus.ToString());
    if (currStatus == LibPlacenote.MappingStatus.RUNNING && prevStatus ==
LibPlacenote.MappingStatus.LOST) {
      Debug.Log("Localized");
      //                    GetComponent<ShapeManager> ().LoadShapesJSON
(mSelectedMapInfo.metadata.userdata);
    } else if (currStatus == LibPlacenote.MappingStatus.RUNNING && prevStatus ==
LibPlacenote.MappingStatus.WAITING) {
      Debug.Log("Mapping");
    } else if (currStatus == LibPlacenote.MappingStatus.LOST) {
      Debug.Log("Searching for position lock");
    } else if (currStatus == LibPlacenote.MappingStatus.WAITING) {
      if (GetComponent<CustomShapeManager>().shapeObjList.Count != 0) {
        GetComponent<CustomShapeManager>().ClearShapes();
```

```
        }
      }
    }
}
```

CODE FOR READ MAP:-

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System;
using UnityEngine.UI;
using UnityEngine.XR.iOS;
using System.Runtime.InteropServices;
using System.IO;
using Newtonsoft.Json.Linq;
using Newtonsoft.Json;

public class ReadMap : MonoBehaviour, PlacenoteListener {

    private const string MAP_NAME = "GenericMap";

    private UnityARSessionNativeInterface mSession;
    private bool mARInit = false;

    private LibPlacenote.MapMetadataSettable mCurrMapDetails;

    string currMapID = String.Empty;

    private LibPlacenote.MapInfo mSelectedMapInfo;
    private string mSelectedMapId {
        get {
            return mSelectedMapInfo != null ? mSelectedMapInfo.placeId : null;
        }
    }

    // Use this for initialization
    void Start() {
        Input.location.Start();

        mSession = UnityARSessionNativeInterface.GetARSessionNativeInterface();
        StartARKit();
        FeaturesVisualizer.EnablePointcloud();
        LibPlacenote.Instance.RegisterListener(this);
```

```
}

void OnDisable() {
}

// Update is called once per frame
void Update() {
    if (!mARInit && LibPlacenote.Instance.Initialized())
    {
        Debug.Log("Ready to Start!");
        mARInit = true;

        // Load Map
        FindMap();
    }
}

void FindMap() {
    //get metadata
    LibPlacenote.Instance.SearchMaps(MAP_NAME, (LibPlacenote.MapInfo[] obj) => {
        foreach (LibPlacenote.MapInfo map in obj) {
            if (map.metadata.name == MAP_NAME) {
                mSelectedMapInfo = map;
                Debug.Log("FOUND MAP: " + mSelectedMapInfo.placeId);
                LoadMap();
                return;
            }
        }
    });
}

void LoadMap() {
    ConfigureSession(false);

    LibPlacenote.Instance.LoadMap(mSelectedMapInfo.placeId,
        (completed, faulted, percentage) => {
            if (completed) {
                Debug.Log("Loaded ID: " + mSelectedMapInfo.placeId + "...Starting session");

                LibPlacenote.Instance.StartSession();

            } else if (faulted) {
                Debug.Log("Failed to load ID: " + mSelectedMapInfo.placeId);
            } else {
                Debug.Log("Map Download: " + percentage.ToString("F2") + "/1.0");
            }
```

```
            }
        );
    }
private void StartARKit() {
        Debug.Log("Initializing ARKit");
        Application.targetFrameRate = 60;
        ConfigureSession(false);
    }
private void ConfigureSession(bool clearPlanes) {
#if !UNITY_EDITOR
                    ARKitWorldTrackingSessionConfiguration config = new
ARKitWorldTrackingSessionConfiguration ();
                    config.planeDetection = UnityARPlaneDetection.None;
                    config.alignment = UnityARAlignment.UnityARAlignmentGravity;
                    config.getPointCloudData = true;
                    config.enableLightEstimation = true;
                    mSession.RunWithConfig (config);
#endif
    }
    public void OnPose(Matrix4x4 outputPose, Matrix4x4 arkitPose) { }

    public void OnStatusChange(LibPlacenote.MappingStatus prevStatus,
LibPlacenote.MappingStatus currStatus) {
        Debug.Log("prevStatus: " + prevStatus.ToString() + " currStatus: " +
currStatus.ToString());
        if (currStatus == LibPlacenote.MappingStatus.RUNNING && prevStatus ==
LibPlacenote.MappingStatus.LOST) {
            Debug.Log("Localized: " + mSelectedMapInfo.metadata.name);

GetComponent<CustomShapeManager>().LoadShapesJSON(mSelectedMapInfo.metadata.userd
ata);
            FeaturesVisualizer.DisablePointcloud();
        } else if (currStatus == LibPlacenote.MappingStatus.RUNNING && prevStatus ==
LibPlacenote.MappingStatus.WAITING) {
            Debug.Log("Mapping");
        } else if (currStatus == LibPlacenote.MappingStatus.LOST) {
            Debug.Log("Searching for position lock");
        } else if (currStatus == LibPlacenote.MappingStatus.WAITING) {
            if (GetComponent<CustomShapeManager>().shapeObjList.Count != 0) {
                //GetComponent<CustomShapeManager>().ClearShapes();
            }
        }
    }
}
```