温州肯恩大学
WENZHOU-KEAN UNIVERSITY

The Simulator of WKU Carpooling Application

In Partial Fulfillment of the Requirements

for the Bachelor of Science in Computer Science

by

Zhu Yunfeng (Student Name)

1098757 (Student ID Number)

June, 2022

Catalog

Abstract

In the CPS*4951 Senior Capstone course, I participated in the development process of three projects in total. In the first project, my teammates and I wrote a proposal and SRS (Software Requirements Specification) for a graduate school admission information sharing platform. In the second project, my team worked on the two main levels of a game called Room. In the third project, I developed a taxi order simulator for carpooling software. Since I only took part in the practical development of Project 2 and Project 3, I will not elaborate on the relevant content of Project 1 and Project 3 in this report. For both Project 2, I used Unity and C# for development because the game engine can develop not only games but also develop more utilitarian applications. In addition, these project experiences have helped me accumulate a lot of valuable knowledge and experience for my game development path in the future. Next, I will elaborate on my work on Projects 2.

Chapter I

1.1 Introduction

In this chapter, the content of Project 2 will be elaborated on. For Project 2, our group developed a carpooling system for WKU students. Due to our campus' location in a remote suburb and the insufficient public transportation, WKU students mainly travel by taxi or car-hailing. WKU students can carpool with other students to the same destination through our carpooling system, which is more convenient, safe and economical for WKU students. However, our software has a constraint that is the absence of API from the third-party platform, which allows our application to access the actual taxi function (such as Uber, Didi, CaoCao, etc.), and the route planning in the real map (Gode, Baidu, etc.). The major online car-hailing platforms do not provide the corresponding APIs in the market, and the map applications do not provide fully functional and free APIs. Therefore, to improve our application, I used Unity and C# to build a simulator for online car-hailing and realistic route calculation. The simulator can read the trip orders established by the user in the application in real-time (you can also simulate the starting point of the order by clicking), display the starting point of the order, compute the shortest path of the trip, and simulate the driving process of the taxi. In the following content, I am going to elaborate on the algorithm, logic and code I applied when implementing this simulator.

1.2 A* Pathfinding Algorithm

In the simulator, I applied the A* pathfinding algorithm to calculate the shortest path between the starting points set by the user. Because of excellent performance of the A* pathfinding algorithm, it is widely used in the character guidance function of video games. Therefore, I decided to use A* algorithm here to serve our carpooling application and also accumulate experience for my future game development career.

1.2.1 Algorithmic Logic

The program can use the A* Pathfinding Algorithm to calculate the shortest path for objects to avoid obstacles from the starting grid node to the target (end) grid node.

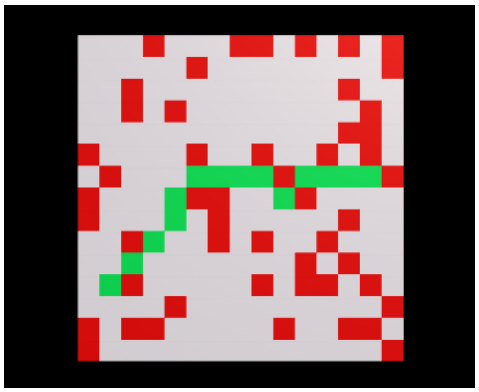

Image 1.1

The basic principle of A* algorithm is to traverse all the accessible nodes around the current node and select an optimal node among them to traverse repeatedly until the end node is reached. A* pathfinding algorithm has an important formula:

$f = g + h$

where f is Pathfinding consumption; g is the distance from the current node to the starting node. For example, the four grid nodes from the starting node to the top, bottom,

left, and right are adjacent to the starting point. The distance can be represented by 1. The top left, bottom left, top right, and bottom right vertices adjacent grids, according to the Pythagorean theorem, it can be judged that the distance is √2, which is approximately equal to 1.4. Through this rule, the g value of the current node can be obtained by accumulating the distance between the current node and the previous node and the g value of the previous node.

In the equation, h is the distance from the current node to the end node. The calculation of h value is determined by the Manhattan block algorithm, that is, how many cells are needed to walk from this cell to the end. It follows the equation:

$h = abs\ (A.x - B.x) + abs(A.y - B.y)$

where A is the current position, B is the final position, and abs() calculates the absolute value of formula in the brackets.

In addition, the A* pathfinding algorithm requires two lists: an openList and a closedList. The openList is used to store considered accessible nodes, and the closedList is used to store nodes that are no longer considered (the nodes in the complete shortest path found are all in the closed list).

The general processes of the A* pathfinding algorithm are:

1. Record the starting point (or node) as the current point **a**

2. Put the current point **a** into the close list and set the parent to null

3. Put all accessible grid nodes around the current point **a** (not out of bounds and not blocking) into the open list, if the surrounding point is already in the openList or closedList, ignore it

4. Record the f value and parent node of all the grid nodes that can travel around the current node a (the parent node is a)

5. Find the best point (the node with the smallest f value in the openList) and put it in the closedList. Every time you put a point in the closedList, you must judge whether this node is the end node. If the node is the end not, skip to step 7. If not continue to step 6

6. Assign the optimal node with least f in the previous step to the current point **a** and jump back to step 3

7. In the closeList, find all nodes in the path through the parent node of the end point until the parent node of the last node is empty. Finally, reverse the obtained list of paths to get the paths in normal order

1.2.2 Classes

In order to implement the basic frame of the simulator, it needs

AStarNode class: represents the grid node in the A* algorithm

AStarMgr class: manages A Star nodes and implements pathfinding logic

TestAStar class: visualizes the inputs and results of A* pathfinding in the Unity viewpoint

TaxiMove class: controls the taxi model in the Unity viewport

I drew the contents and relationships of these classes into a simple UML class diagram:

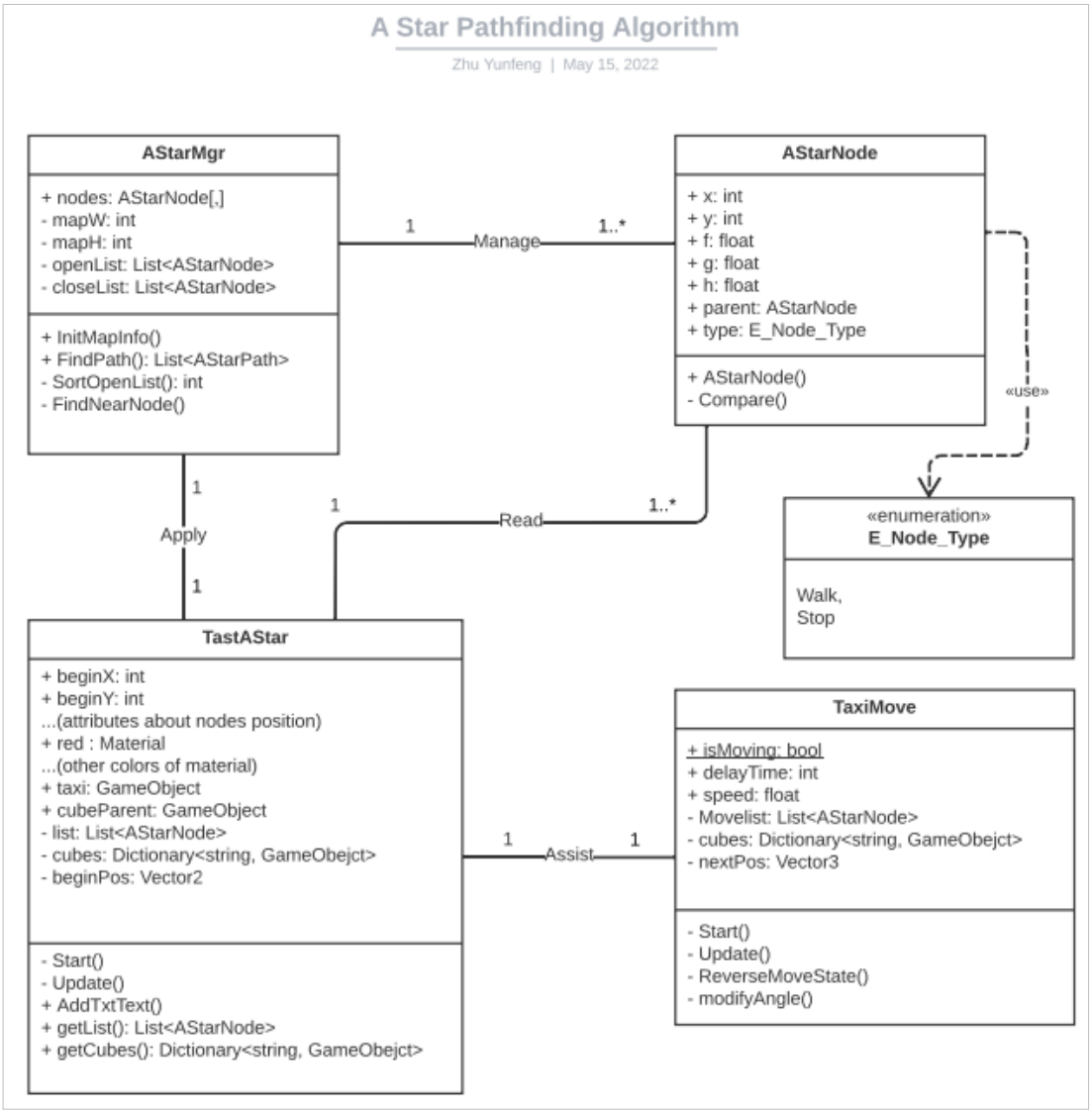

Diagram 1.1

1.2.3 Implementation

According to the UML class diagram above, I program the demo for the A* pathfinding algorithm with C#. The version of editors I used are Unity 2020.3.25f1c1 and Visual Studio 2019. Here is the scripts' catalog of my project (some of these will not be exhibited until Optimization part):

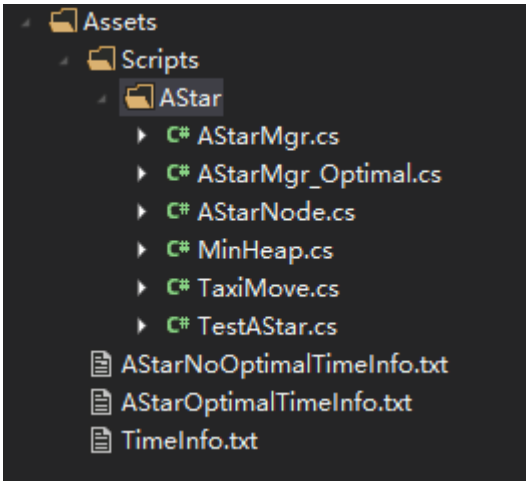

Image 1.2

And here are detailed codes I wrote for the A* Pathfinding Algorithm demo.

AStarNode.cs: represents the grid node in the A* algorithm

System.Collections namespace includes interfaces and classes that define various collections of objects, such as lists, queues, bit arrays, hash tables, and dictionaries.

UnityEngine namespace integrate every useful part of game development, such as events, rendering, UI, AR, and Scripting.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public enum E_Node_Type
6   {
7       Walk, // walkable place
8       Stop, // unwalkable place
9   }
10  /// <summary>
11  /// A Star box/ node Class
12  /// </summary>
13  public class AStarNode : IComparer
14  {
15      // coordinate of the node
16      public int x;
```

```
17      public int y;
18      // comsumption of finding way: f = g + h
19      public float f;
20      // distance to the start position
21      public float g;
22      // distance to the target position
23      public float h;
24      // parent class
25      public AStarNode parent;
26
27      public E_Node_Type type;
28
29      public AStarNode(int x, int y,  E_Node_Type type)
30      {
31          this.x = x;
32          this.y = y;
33          this.type = type;
34      }
35
36      int IComparer.Compare(System.Object a, System.Object b)
37      {
38          if (((AStarNode)a).f== ((AStarNode)b).f) return 0;
39          else if (((AStarNode)a).f > ((AStarNode)b).f) return 1;
40          else return -1;
41      }
42  }
```

Source code 1.1 AStarNode.cs

AStarMgr.cs: manages A Star nodes and implements pathfinding logic

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   /// <summary>
6   /// A Star way finder manager
7   /// </summary>
8   public class AStarMgr
9   {
10      private static AStarMgr instance;
11      public static AStarMgr Instance
12      {
13          get
14          {
```

```
15              if (instance == null)
16                  instance = new AStarMgr();
17              return instance;
18          }
19      }
20
21      private int mapW;
22      private int mapH;
23
24      //The container of all nodes class about the map
25      public AStarNode[,] nodes;
26
27      private List<AStarNode> openList = new List<AStarNode>();
28      private List<AStarNode> closeList = new List<AStarNode>();
29
30      /// <summary>
31      /// Initalize map info
32      /// </summary>
33      /// <param name="w"></param>
34      /// <param name="h"></param>
35      public void InitMapInfo(int w, int h)
36      {
37          this.mapW = w;
38          this.mapH = h;
39
40          nodes = new AStarNode[w, h];
41          // create nodes according to the wight and height; random
set block since we do not have map info now
42          for (int i = 0; i < w; i++)
43              for (int j = 0; j < h; j++)
44              {
45                  if (Random.value < 0.2)
46                      nodes[i, j] = new AStarNode(i, j,
E_Node_Type.Stop);
47                  else
48                      nodes[i, j] = new AStarNode(i, j,
E_Node_Type.Walk);
49              }
50      }
51
52      /// <summary>
53      /// Find Path Function
54      /// </summary>
55      /// <param name="startPos"></param>
```

```
56      /// <param name="endPos"></param>
57      /// <returns></returns>
58      public List<AStarNode> FindPath(Vector2 startPos, Vector2
endPos)
59      {
60          AStarNode start = nodes[(int)startPos.x, (int)startPos.y];
61          AStarNode end = nodes[(int)endPos.x, (int)endPos.y];
62
63          // judge whether the inputted positions are valid: 1. in
the range of map; 2. not block node
64          if ((startPos.x < 0 || startPos.x >= mapW || startPos.y <
0 || startPos.y >= mapH) || start.type == E_Node_Type.Stop)
65          {
66              Debug.Log("Start position or end position is out of
range");
67              return null;
68          }
69          if ((endPos.x < 0 || endPos.x >= mapW && endPos.y < 0 ||
endPos.y >= mapH) || end.type == E_Node_Type.Stop)
70          {
71              Debug.Log("Start position or end position is
unwalkable");
72              return null;
73          }
74          // clear the data from the last time
75          // clear open and close list
76          closeList.Clear();
77          openList.Clear();
78          // add start node into closeList
79          start.parent = null;
80          start.f = 0;
81          start.g = 0;
82          start.h = 0;
83          closeList.Add(start);
84
85          while (true)
86          {
87              // find nodes around and judge whether they are in the
open or close list
88              // up-left
89             FindNearNode(start.x - 1, start.y - 1, 1.4f, start,
end);
90              //up
91              FindNearNode(start.x, start.y - 1, 1f, start, end);
```

```
92              //up-right
93              FindNearNode(start.x + 1, start.y - 1, 1.4f, start,
end);
94              //left
95              FindNearNode(start.x - 1, start.y, 1f, start, end);
96              //right
97              FindNearNode(start.x + 1, start.y, 1f, start, end);
98              //down-left
99              FindNearNode(start.x - 1, start.y + 1, 1.4f, start,
end);
100             //down
101             FindNearNode(start.x, start.y + 1, 1f, start, end);
102             //down-right
103             FindNearNode(start.x + 1, start.y + 1, 1.4f, start,
end);
104
105             if (openList.Count == 0)
106             {
107                 Debug.Log("Dead way");
108                 return null;
109             }
110
111             // select the node with least f in the openList out and
put it into the closeList
112             openList.Sort(SortOpenList);
113             /*for(int i = 0; i < openList.Count; i++)
114             {
115                 Debug.Log(openList[i].f + " , ");
116             }
117             Debug.Log(" ____________________________________");*/
118             closeList.Add(openList[0]);
119             start = openList[0];
120             openList.RemoveAt(0);
121
122             if (start == end)
123             {
124                 //have found the way
125                 List<AStarNode> path = new List<AStarNode>();
126                 path.Add(end);
127                 while (end.parent != null)
128                 {
129                     path.Add(end.parent);
130                     end = end.parent;
131                 }
```

```
132                 path.Reverse();
133                 return path;
134             }
135         }
136 
137     }
138 
139     private int SortOpenList(AStarNode a, AStarNode b)
140     {
141         if (a.f > b.f)
142             return 1;
143         else if (a.f == b.f)
144             return 1;
145         else
146             return -1;
147     }
148 
149     // find the node  and judge whether it is in the open or close
list
150     private void FindNearNode(int x, int y, float g, AStarNode
parent, AStarNode end)
151     {
152         // judge whether the node is valid
153         if (x < 0 || x >= mapW || y < 0 || y >= mapH)
154             return;
155         AStarNode node = nodes[x, y];
156         if (node == null || node.type == E_Node_Type.Stop ||
closeList.Contains(node) || openList.Contains(node))
157             return;
158         // compute the value of f
159         node.parent = parent;
160         node.g = parent.g + g;
161         node.h = Mathf.Abs(x - end.x) + Mathf.Abs(y - end.y);
162         node.f = node.g + node.h;
163         openList.Add(node);
164     }
165 }
```

Source code 1.2 AStarMgr.cs

TestAStar.cs: visualizes the inputs and results of A* pathfinding in the Unity viewpoint

System.IO contains the types of files and data streams allowed to be read and

written. It mainly manages read and write operations.

System.Text namespace includes a series of character encodings, such as ASCII, Unicode, UTF-7, and UTF-8.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using System.IO;
5   using System.Text;
6
7   public class TestAStar : MonoBehaviour
8   {
9       // the first box on the top left
10      public int beginX = -3;
11      public int beginY = 5;
12      // offset between every box
13      public float offsetX = 2;
14      public float offsetY = -2;
15      //the wight and height of every map node
16      public int mapW = 5;
17      public int mapH = 5;
18
19      List<AStarNode> list;
20
21      public Material red;
22      public Material yellow;
23      public Material green;
24      public Material normal;
25      public GameObject taxi;
26
27      private Dictionary<string, GameObject> cubes = new
Dictionary<string, GameObject>();
28      private Vector2 beginPos = Vector2.right * -1;
29      private bool clickable = true;
30
31      // Start is called before the first frame update
32      void Start()
33      {
34          AStarMgr.Instance.InitMapInfo(mapW, mapH);
35          for (int i = 0; i < mapW; i++)
36              for(int j = 0; j < mapH; j++)
37              {
```

```
38                //create cubes
39                GameObject obj =
GameObject.CreatePrimitive(PrimitiveType.Cube);
40                obj.transform.position = new Vector3(beginX + i *
offsetX, beginY + j * offsetY, 0);
41                obj.name = i + "_" + j;
42
43                cubes.Add(obj.name, obj);
44
45                // get nodes and judge whether it is block
46                AStarNode node = AStarMgr.Instance.nodes[i, j];
47                if(node.type == E_Node_Type.Stop)
48                {
49                    obj.GetComponent<MeshRenderer>().material = red;
50                }
51            }
52        //modify the postion of the camera
53        float cameraX = (mapW / 5 - 1) * 3.5f;
54        float cameraY = 2 - (mapH / 5 -0.8f) * 3.5f;
55        float cameraZ = (0 - 9) * mapW / 5;
56        transform.position = new Vector3(cameraX,cameraY,cameraZ);
57    }
58
59    // Update is called once per frame
60    void Update()
61    {
62        if ( Input.GetMouseButtonDown(0))
63        {
64            RaycastHit info;
65            Ray ray =
Camera.main.ScreenPointToRay(Input.mousePosition);
66            // ray test
67            if(Physics.Raycast(ray, out info, 1000) &&
TaxiMove.ismoving == false)
68            {
69                if(beginPos == Vector2.right * -1)
70                {
71                    // clear path last time
72                    if (list != null)
73                    {
74                        for (int i = 0; i < list.Count; i++)
75                        {
76                            cubes[list[i].x + "_" +
list[i].y].GetComponent<MeshRenderer>().material = normal;
```

```
77                      }
78                      list.Clear();
79                  }
80
81                  string[] strs =
info.collider.gameObject.name.Split('_');
82                  beginPos = new Vector2(int.Parse(strs[0]),
int.Parse(strs[1]));
83
   info.collider.gameObject.GetComponent<MeshRenderer>().material =
yellow;
84              }
85              else if (TaxiMove.ismoving == false)
86              {
87                  string[] strs =
info.collider.gameObject.name.Split('_');
88                  Vector2 endPos = new Vector2(int.Parse(strs[0]),
int.Parse(strs[1]));
89
90                  // record time of finding
91                  float strTime = Time.realtimeSinceStartup;
92                  // find way
93                  list = AStarMgr.Instance.FindPath(beginPos,
endPos);
94
95                  float endTime = Time.realtimeSinceStartup;
96                  Debug.Log("Time for finding the route: " +
(endTime - strTime));
97                  //AddTxtText(endTime - strTime + " "); // record
time
98
99                  // in case of dead way
100                 cubes[(int)beginPos.x + "_" +
(int)beginPos.y].GetComponent<MeshRenderer>().material = normal;
101
102                 if (list != null)
103                 {
104                     GameObject car = Instantiate(taxi,
cubes[(int)beginPos.x + "_" + (int)beginPos.y].transform);
105                     for (int i = 0; i < list.Count; i++)
106                     {
107                         cubes[list[i].x + "_" +
list[i].y].GetComponent<MeshRenderer>().material = green;
108                     }
```

```
109                     }
110
111                     // when animation ends, clear the inital
position
112                     beginPos = Vector2.right * -1;
113                 }
114             }
115         }
116     }
117
118     // This function is used to record the time of find-route
method
119     public void AddTxtText(string txtText)
120     {
121         string path = Application.dataPath +
"/AStarNoOptimalTimeInfo.txt";
122         StreamWriter sw;
123         FileInfo fi = new FileInfo(path);
124
125         if (!File.Exists(path))
126         {
127             sw = fi.CreateText();
128         }
129         else
130         {
131             sw = fi.AppendText();   //append text to the document
132         }
133         sw.WriteLine(txtText);
134         sw.Close();
135         sw.Dispose();
136     }
137
138     public List<AStarNode> getList()
139     {
140         return list;
141     }
142
143     public Dictionary<string, GameObject>  getCubes()
144     {
145         return cubes;
146     }
147 }
148
```

Source code 1.3 TestAStar.cs:

TaxiMove.cs: controls the taxi model in the Unity viewport

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   public class TaxiMove : MonoBehaviour
6   {
7       public static bool isMoving = false;
8       public int delayTime = 1;
9
10      private List<AStarNode> Movelist;
11      private Dictionary<string, GameObject> cubes;
12      public float speed = 150.0f;
13      Vector3 nextPos;
14      int i; // counter for movelist
15      // Start is called before the first frame update
16      void Start()
17      {
18          Movelist = GameObject.Find("Main
Camera").GetComponent<TestAStar>().getList();
19          cubes = GameObject.Find("Main
Camera").GetComponent<TestAStar>().getCubes();
20          i = 1;
21          nextPos = cubes[Movelist[i].x + "_" +
Movelist[i].y].transform.position;
22          modifyAngle(nextPos);
23
24      }
25
26      // Update is called once per frame
27      void Update()
28      {
29          isMoving = true;
30
31          if (this.transform.position != nextPos)
32          {
33              this.transform.position =
Vector3.MoveTowards(this.transform.position, nextPos, speed *
Time.deltaTime);
34          }
35          else if (i + 1 < Movelist.Count)
36          {
```

```
37              i++;
38              nextPos = cubes[Movelist[i].x + "_" +
Movelist[i].y].transform.position;
39              modifyAngle(nextPos);
40          }
41
42          else
43          {
44              Invoke("ReverseMoveState", delayTime);
45              Destroy(gameObject, delayTime);
46          }
47      }
48
49      private void ReverseMoveState()
50      {
51          isMoving = false;
52      }
53
54      // Modify angle of the object and make its back up toward
screen and face to the next position
55      private void modifyAngle(Vector3 nextPos)
56      {
57          this.transform.rotation = Quaternion.LookRotation(nextPos
- this.transform.position);
58          Vector3 dirction = (nextPos -
this.transform.position).normalized;
59          if (dirction.x == 0 && dirction.y > 0) { }
60          else if (dirction.y == -1)
this.transform.Rotate(Vector3.forward * 180, Space.Self);
61          else if (dirction.x < 0)
this.transform.Rotate(Vector3.forward * 90, Space.Self);
62          else this.transform.Rotate(Vector3.back * 90, Space.Self);
63      }
64
65  }
66
```

Source code 1.4 TaxiMove.cs

1.2.4 Running Result

With the scripts above, when the user clicks the starting and the ending grids, the

program can correctly display the path and demonstrate the process of taxi driving.

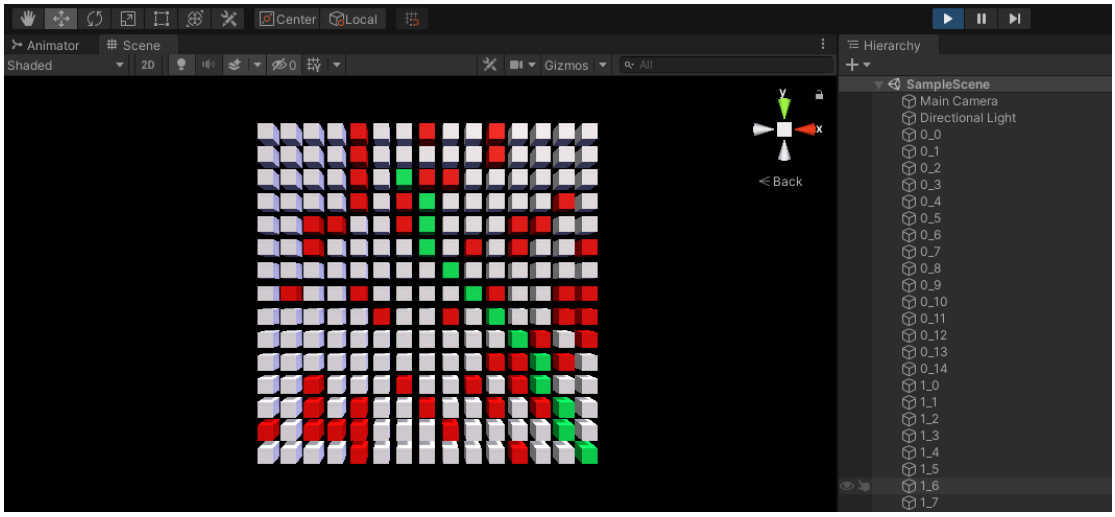

Image 1.3

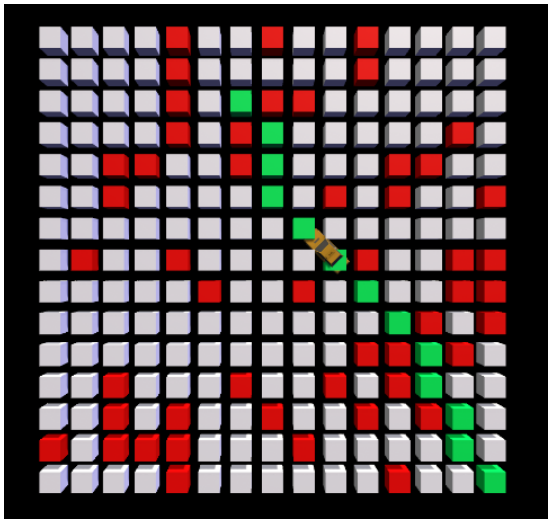

Image 1.4

However, there are still two problems with such the program. There is a logical error in the current algorithm. Since the value of h is the Manhattan path distance that does not consider blocks, when the A* algorithm encounters the following conditions (current node is cubes[2,0]).

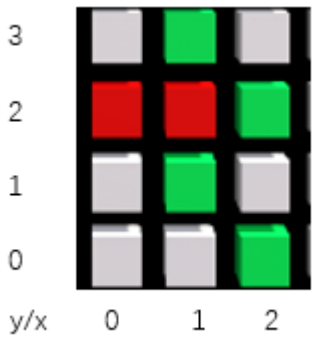

Image 1.5

It would determine cubes[1,1] as the next current node, and then the cubes[2,2] (cubes[2,1] is ignored since it has already in the openList). However, such a path is obviously not the shortest. The shortest path should go through cubes[2,1] instead of cubes[1,1]. It requires me to optimize the logic of the code. On the other side, the program needs to find the node with the smallest f value in the openList as the next current node after the search for eight nodes surrounding the current node. The openList in the current version of the program is implemented based on the list, and the openList needs to be sorted. Such an operation takes a relative long time. It requires me to optimize the performance of the code.

1.2.4 Optimization

· Logic Optimization

In order to find the shortest path accurately, the A* pathfinding algorithm in the code needs some adjustments on the logic. Specifically, we need to add a piece of code to the FindNearNode() function to search the nodes already in the openList (the original logic is to ignore the nodes already in the openList),

```
if (openList.Contains(node))
        {
            float gThis = parent.g + g;
            if (gThis < node.g)
            {
                node.g = gThis;
                node.f = node.g + node.h;
                node. parent = parent;
                return;
            }
            else
            {
                return;
            }
        }
```

This piece of code compares the g value of the node being checked to the sum of the g value of the current node and the distance to the node being checked. If the latter is less than the former, the parent of the node being checked is changed to the current node, and the corresponding f value would be also changed. For example, in the case above, the parent node of cubes[2,2] would be changed to cubes[2,1] from cubes[1,1]. This would guarantee the path found by the A* algorithm is the shortest.

· Performance Optimization

Because the openList in the original code is implemented with data structure of list, each sorting and finding the minimum f value will cost a long time. Since one of the most frequent operations in A* algorithm is to extract the minimum, I decided to implement openList with the data structure of Min-heap. Although it will increase the time complexity of the add operation from O(1) to O(logn), Min-Heap extracts the minimum in O(1) that is much faster than the original time complexity-O(n2) with list.

In general, using the min-heap to implement openList is better than list. In the code, I added a MinHeap class (see in Appendix A), changed some contents of the AStarMgr class, and saved it as the AStarMgr_Optimal class. For instance, line 27 is changed to

```
private MinHeap<AStarNode> openList = new MinHeap<AStarNode>();
```

Logical optimized code is inserted in line 157.

Furthermore, the IComparer.Compare function overridden in the AStarNode class is for AStarNode to be compared according to its f value in MinHeap.

Eventually, all AStarMgrs in TestAStar.cs are also changed to AStarMgr_Optimal to invoke the correct script.

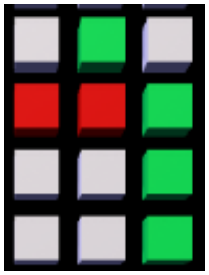

Image 1.6

1.2.5 Performance Analysis

To compare the performance of the original A* pathfinding algorithm and the optimized A* pathfinding algorithm, I printed and recorded the running times of the two at different data sizes (AStarNoOptimalTimeInfo.txt & AStarOptimalTimeInfo.txt). Then I averaged the running time for each size of the data and plotted it as graphs:

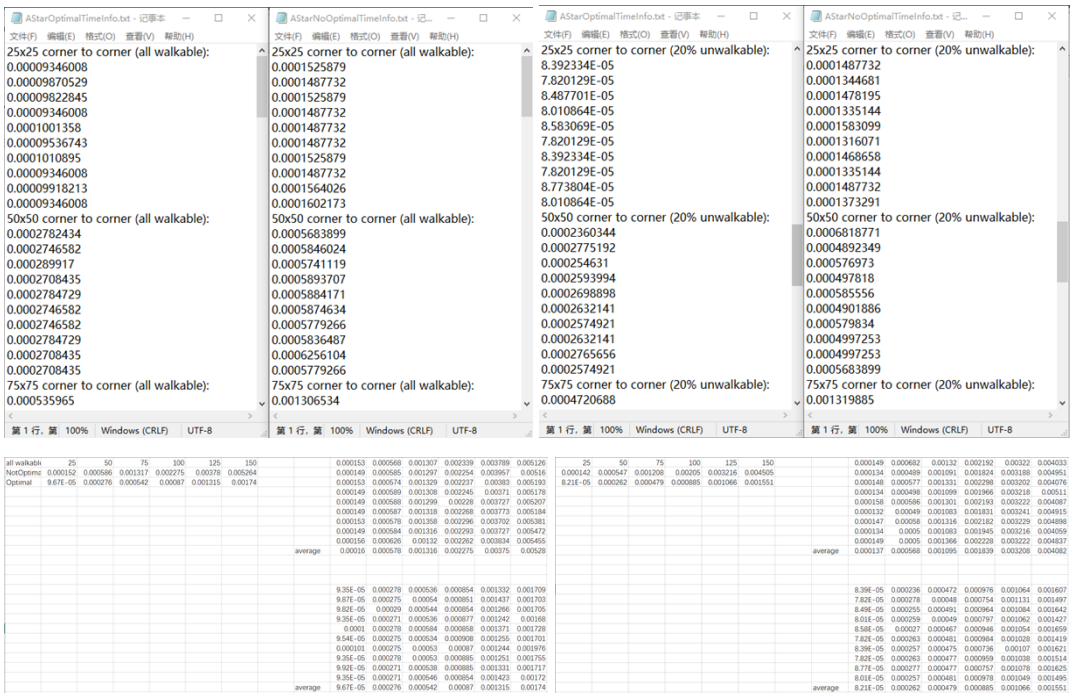

Image 1.7

The following figure show the case without block nodes in the map.

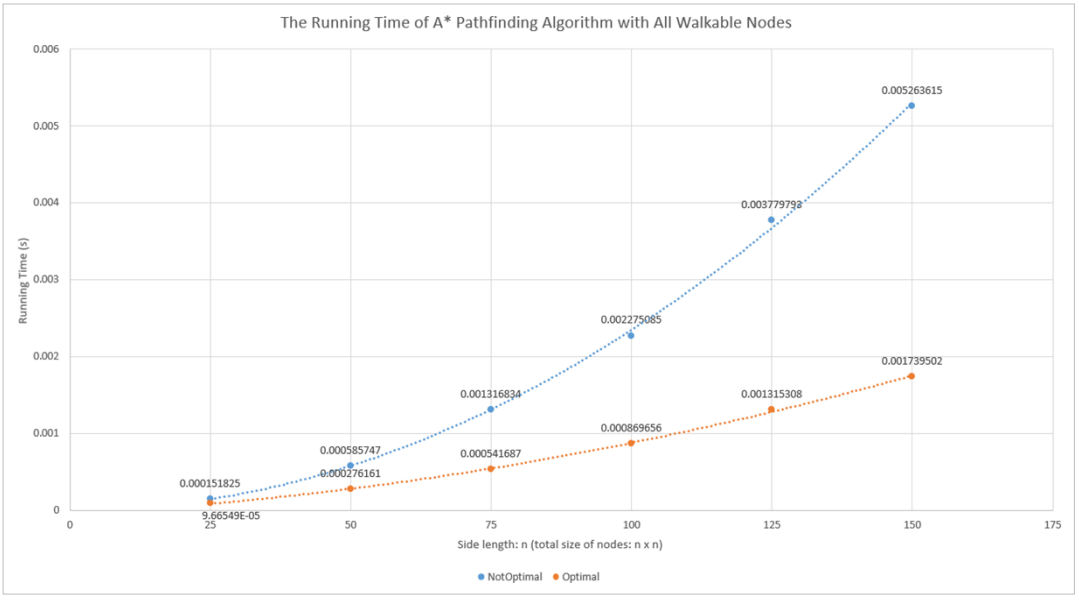

Figure 1.1

The following figure show the case with 20% block nodes in the map.

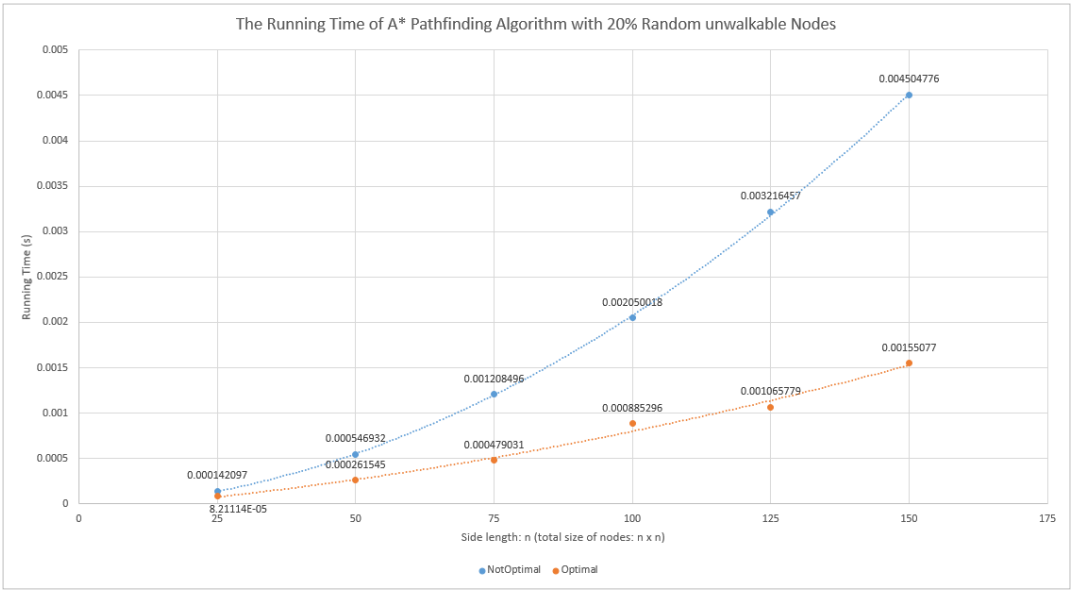

Figure 1.2

From the two graphs, we can find that the efficiency of the optimized A* pathfinding algorithm is significantly higher than that of the unoptimized one, and the advantage of the optimized algorithm is more obvious when the size of the node becomes larger. Moreover, I also conducted a theoretical analysis about the time complexity of the A* algorithm before and after optimization (see the table below):

Step / A* Algo	Original (List)	Optimized (Min-heap)
1. Record start node as current node	O(1)	O(1)
2. (in the loop) Put current node into closeList, and set the parent node	O(1) * O(n)	O(1) * O(n)
3. (in the loop) Put all surrounding accessible nodes into openList	O(1) * O(n)	O(logn) * O(n)
4. (in the loop) Record f and parent node of the surrounding nodes	O(1) * O(n)	O(1) * O(n)
5. (in the loop) Find the optimal node with least f value, (if the	$O(n^2)$ * O(n)	O(1) * O(n)

node equals to end node, then finish the finding and jump to the step 7)		
6. (in the loop) Assign the optimal node from the previous step to the current point a, and jump back to the step 3	O(1) * O(n)	O(1) * O(n)
7. Get the shortest path	O(n)	O(n)
Total time complexity	$O(n^3)$	O(nlogn)

Table 1.1

In conclusion, the time complexity of O(nlogn) is much better than O(n3), so the optimization of the A* algorithm is necessary and effective.

1.3 Real Location Mapping

After implementing the essential logic of the A* pathfinding algorithm, to apply it in carpooling project, we need to map the grid nodes above to locations in reality. In this part, I will introduce the method, code and effect of mapping.

1.3.1 Method

The scripts in the Configuration folder below are used to edit the map.

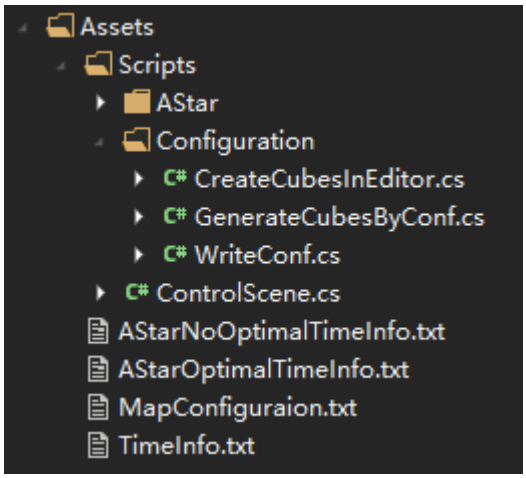

Image 1.8

CreateCubesInEdior.cs can be executed directly in Unity's editing mode, which generates grid nodes in the editor viewport. Meanwhile, I pasted the actual map of parts of Wenzhou onto a vertical plane. Then I edited the accessibility of each grid node (with different textures attached to it) based on the roads and blocks in the map. WriteConf.cs can read the material of each grid node and write its coordinates and properties into a configuration file called MapConfiguration.txt.

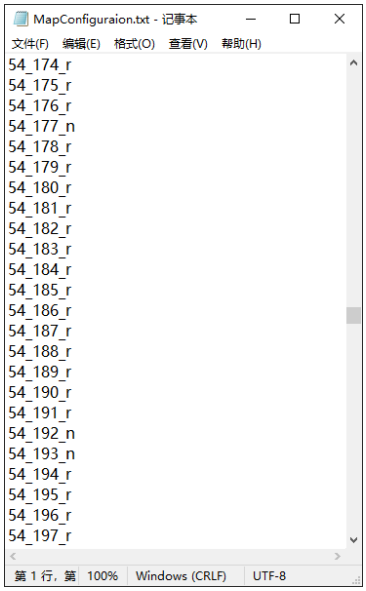

Image 1.9

GernerateCubesByConf.cs can return the corresponding AStarNode[,], two-dimensional array, according to the specified configuration file. In AStarMgr_Optimal.cs, we changed the random block generation code in the original InitMapInfo() function to

```
string filename = " /MapConfiguraion.txt";
nodes = GenerateCubesByConf.GetMapConf(mapW, mapH, filename);
```

In this way, AStarMgr_Optimal class can read the map accessibility information and generate grid nodes that conform to the actual map attributes in the runtime scene through TestAStar.cs

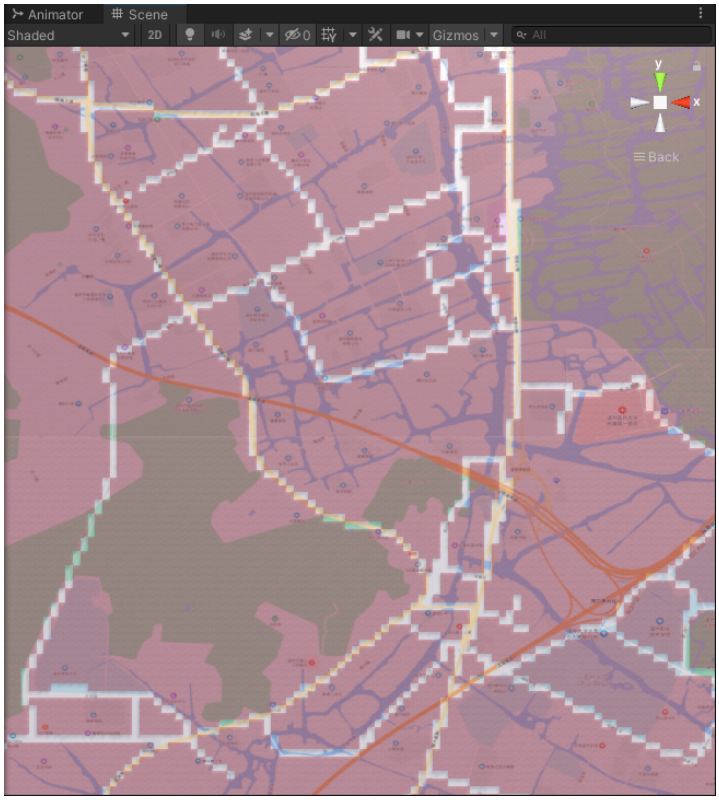

Image 1.10

In addition, ControlScene.cs is used to control the zoom in, zoom out and move of the camera.

1.3.2 Implementation

Here are detailed codes I wrote for the map configuration.

CreateCubesInEditor.cs: generates cubes in the editor viewport with particular size

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
5   [ExecuteInEditMode]
6   public class CreateCubesInEditor : MonoBehaviour
7   {
8       // The first box on the top left
9       public int beginX = -3;
10      public int beginY = 5;
11      // Offset between every box
12      private float offsetX = 1;
```

```
13      private float offsetY = -1;
14      // The wight and height of every map node
15      public int mapW = 103;
16      public int mapH = 210;
17
18      List<AStarNode> list;
19
20      public Material red;
21      public Material yellow;
22      public Material green;
23      public Material normal;
24      public GameObject taxi;
25      public GameObject cubeParent;
26
27      private Dictionary<string, GameObject> cubes = new
Dictionary<string, GameObject>();
28      private Vector2 beginPos = Vector2.right * -1;
29
30      // Start is called before the first frame update
31      void Awake()
32      {
33          AStarMgr_Optimal.Instance.InitMapInfo(mapW, mapH);
34
35          for (int i = 0; i < mapW; i++)
36              for (int j = 0; j < mapH; j++)
37              {
38                  // Create cubes
39                  GameObject obj =
GameObject.CreatePrimitive(PrimitiveType.Cube);
40                  obj.transform.position = new Vector3(beginX + i *
offsetX, beginY + j * offsetY, 0);
41                  obj.GetComponent<MeshRenderer>().material = normal;
42                  obj.name = i + "_" + j;
43                  obj.transform.parent = cubeParent.transform;
44                  obj.GetComponent<MeshRenderer>().material = red;
45
46                  cubes.Add(obj.name, obj);
47
48                  /*// Get nodes and judge whether it is block
49                  AStarNode node = AStarMgr_Optimal.Instance.nodes[i,
j];
50                  if (node.type == E_Node_Type.Stop)
51                  {
52                      obj.GetComponent<MeshRenderer>().material = red;
```

```
53                  }*/
54              }
55          // Modify the postion of the camera
56          float cameraX = (mapW / 5 - 1) * 2.5f;
57          float cameraY = 2 - (mapH / 5 - 1f) * 2.5f;
58          float cameraZ = (0 - 9) * mapW / 5;
59          transform.position = new Vector3(cameraX, cameraY,
cameraZ);
60      }
61  }
62
```

Source code 1.5 CreateCubesInEditor.cs

WriteConf.cs: writes the map configuration into the specific file according to the attribute of the cubes

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using System.IO;
5
6   public class WriteConf : MonoBehaviour
7   {
8       private int mapW;
9       private int mapH;
10
11      public Material red;
12      public Material normal;
13
14      // Start is called before the first frame update
15      void Start()
16      {
17          mapW = 103; //GameObject.Find("Main
Camera").GetComponent<CreateCubesInEditor>().mapW;
18          mapH = 210; // GameObject.Find("Main
Camera").GetComponent<CreateCubesInEditor>().mapH;
19          for(int i = 0; i < mapW; i++)
20          {
21              for (int j = 0; j < mapH; j++)
22              {
23                  GameObject obj = GameObject.Find(i + "_" + j);
24                  char type =
obj.GetComponent<MeshRenderer>().material.name[0];
25                  Debug.Log(i + "_" + j + "_" + type);
```

```
26                  AddTxtText(i + "_" + j + "_" + type);
27              }
28          }
29
30      }
31
32      public void AddTxtText(string txtText)
33      {
34          string path = Application.dataPath +
"/MapConfiguraion.txt"; //"/AStarNoOptimalTimeInfo.txt";
35          StreamWriter sw;
36          FileInfo fi = new FileInfo(path);
37
38          if (!File.Exists(path))
39          {
40              sw = fi.CreateText();
41          }
42          else
43          {
44              sw = fi.AppendText();   // Append text to the document
45          }
46          sw.WriteLine(txtText);
47          sw.Close();
48          sw.Dispose();
49      }
50  }
51
```

Source code 1.5 WriteConf.cs

GenerateCubesByConf.cs: generates cubes set based on the configuration file

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using System.IO;
5   using System.Text;
6
7   public class GenerateCubesByConf : MonoBehaviour
8   {
9       public int mapW = 103;
10      public int mapH = 210;
11      public AStarNode[,] nodes;
12
13      /// <summary>
14      /// Read the configuration file of the map
```

```
15      /// </summary>
16      /// <param name="mapW"></param> The width of the map
17      /// <param name="mapH"></param> The height of the map
18      /// <param name="filename"></param> The name of configuration
file
19      /// <returns> AStarNode[,] </returns> Two-dimensional array of
AStarNode
20      public static AStarNode[,] GetMapConf (int mapW, int mapH,
string filename) // "/MapConfiguraion.txt"
21      {
22          AStarNode[,] nodes = new AStarNode[mapW, mapH];
23          string path = Application.dataPath + filename;
24          StreamReader reader;
25          reader = new StreamReader(path);
26
27          string _text;
28          string[] lineArray;
29          int i, j;
30
31          while ((_text = reader.ReadLine()) != null)
32          {
33              lineArray = _text.Split('_');
34              i = int.Parse(lineArray[0]);
35              j = int.Parse(lineArray[1]);
36              if (lineArray[2] == "r")
37                  nodes[i, j] = new AStarNode(i, j,
E_Node_Type.Stop);
38              else
39                  nodes[i, j] = new AStarNode(i, j,
E_Node_Type.Walk);
40          }
41          reader.Dispose();
42          reader.Close();
43          return nodes;
44      }
45  }
46
```

Source code 1.6 GenerateCubesByConf.cs

ControlScene.cs: control the zoom in or out and the move of the main camera

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
```

```
5   public class ControlScene : MonoBehaviour
6   {
7       public float  moveSpeed = 1000f;
8       public float viewSpeed = 8f;
9
10      // Update is called once per frame
11      void Update()
12      {
13          // Zoom in and out
14          if (Input.GetAxis("Mouse ScrollWheel") > 0 &&
GetComponent<Camera>().fieldOfView > viewSpeed)
15          {
16              GetComponent<Camera>().fieldOfView -= viewSpeed;
17          }
18
19          if (Input.GetAxis("Mouse ScrollWheel") < 0)
20          {
21              GetComponent<Camera>().fieldOfView += viewSpeed;
22          }
23
24          // Moving camera
25          if (Input.GetMouseButton(1))
26          {
27              float h = Input.GetAxis("Mouse X") * moveSpeed *
Time.deltaTime;
28              float v = Input.GetAxis("Mouse Y") * moveSpeed *
Time.deltaTime;
29              this.transform.Translate(-h, -v, 0, Space.World);
30          }
31      }
32  }
```

Source code 1.7 ControlScene.cs

1.3.3 Running Result

Through the script above, the emulator runs correctly. As follows, the program simulates the process of taking a taxi from Wenzhou-Kean University to MixC.

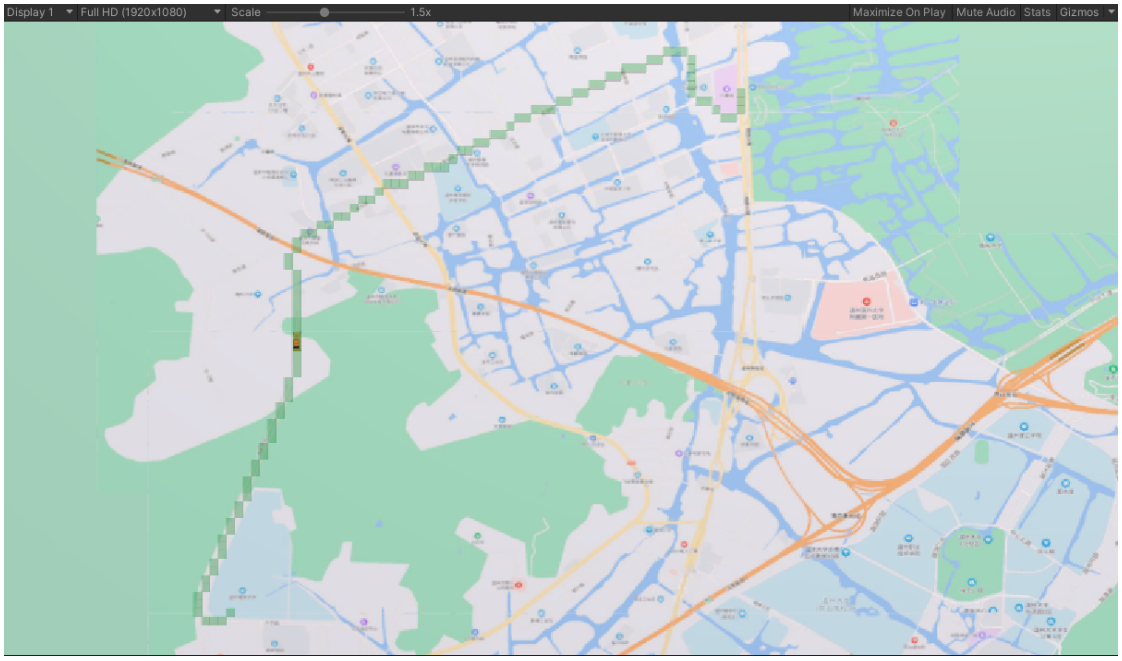

Image 1.11

1.4 Connect PostgreSQL

In order to make the order created by the user displayed on the simulator in real time, the program can establish a connection with the PostgreSQL database of the carpooling software by using the Npgsql package and the following code.

```
NpgsqlConnection conn;
conn = new NpgsqlConnection(strConnec);
conn.Open();
```

Where strConnec store the address and information of the target database. Whenever new order information is added to the form, the database will trigger Unity to read the latest order information and display the itinerary on the simulator in real time.

References

Amitp. (n.d.). Introduction to A* From Amit's Thoughts on Pathfinding. Introduction to A*. Retrieved from http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Koistinen A. (2019). Four Axes of RPG Design. Retrieved from http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html

Xu, Y., Gao, W., Liu, J. (2022). SRS Carpooling App.