**小组项目报告**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 项目名称 | 四个猛男停车场管理系统 | | | | | | |
| 完成时间 |  | 指导教师 |  | | | | |
| 小组自评  （A/B/C） |  | 其他  说明 |  | | | | |
| 小组成员  （姓名学号） | 成员分工  （角色及任务） | | 组内  排序 | 个人成绩 | | | |
| 3-③ | 4-③ | 5-② | 总分 |
| 蔡廷漳 201892053 | 组长：算法板块与整合工作、与组员合理沟通 | | 1 |  |  |  |  |
| 蒋元龙201892226 | 组员：与组长共同编写源代码，负责部分类的实现，项目二的策划 | | 2 |  |  |  |  |
| 石皓翔201892464 | 组员：QT学习与可视化界面实现 | | 3 |  |  |  |  |
| 蔡远哲201892005 | 组员：QT学习与项目图表的绘制，项目二的策划 | | 4 |  |  |  |  |
|  |  | |  |  |  |  |  |

大连理工大学软件学院

评分细则及标准

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 考察项目 | 总分 | 评分细则 | 分数 | 得分 |
| 问题规模 | 5分 | 创新超额完成指定任务，工作量饱满 | 4-5分 |  |
| 基本完成小组指定任务，工作量一般 | 2-3分 |
| 大量指定任务未完成，工作量不足 | 0-1分 |
| 技术难度 | 5分 | 模型设计合理优化 | 4-5分 |  |
| 模型设计基本正确 | 2-3分 |
| 模型设计存在一些问题 | 0-1分 |
| 实现程度 | 5分 | 系统设计及实现完整，界面友好，没有错误 | 4-5分 |  |
| 系统设计及实现正确，存在少许错误 | 2-3分 |
| 系统设计及实现不够完整，存在一些错误 | 0-1分 |
| 个人工作 | 5分 | 团队核心成员，积极参与项目设计与实现 | 4-5分 |  |
| 团队主要成员，参与项目部分的设计或实现 | 2-3分 |
| 团队其他成员，参与项目不够积极 | 0-1分 |

目 录

第一部分 学习内容及总结 2

1 成员1学习内容及体会 2

2 成员2学习内容及体会 2

3 成员3学习内容及体会 3

4 成员4学习内容及体会 3

第二部分 项目开发报告 4

1 项目简介 4

2 需求分析 5

3 总体设计 7

4 详细设计与实现 10

4.1 背景 10

4.1.1 定位为APP 10

4.1.2 立项 10

4.1.3 停车场的建立 11

4.2四个猛男停车场的运营与维护 13

4.2.1 四个猛男停车场的维护 13

4.2.1.1 bfs(广度优先搜索) 15

4.2.1.2 dijkstra算法 18

4.3 对于用户 19

4.3.1选择入场 20

4.3.1.1 succeed\_in()函数 20

4.3.2 选择退场 20

4.3.3 选择游戏 20

4.4 对于管理者 21

4.4.1 查看四个猛男停车场的信息 21

4.4.2 对外营业 21

4.4.2 查看四个猛男停车场的客户评价 21

4.4.3 查看二维地图 22

4.4.3.1绘图所涉及到的算法 23

4.4.4 抽选幸运车位 23

4.4.4.1抽取幸运车位涉及的算法 23

4.4.5 营业额显示 23

4.4.6 修改收费系数 23

4.4.7 测试函数 24

4.4.8 停车场的连边功能 24

5 系统测试 24

6 设计总结 25

总结项目总体完成情况： 25

解决办法： 25

不足： 25

写项目的反思： 25

对于项目的展望： 25

第一部分 学习内容及总结

1. 成员1学习内容及体会

蔡廷漳：作为校ACM组的一员，这次做项目跟以往写算法题目的感受截然不同。原本一道题目对应的算法也就一两个，整个题目也用着面向过程的形式去编写。而作为项目的组长，即使一开始分工明确，开始着手编写代码也会担心如何将自己与他人完成的板块整合在一起。但就是这份令人忧惧但富有挑战的项目中，我接触到了软件设计的雏形，我交予组员的任务，他们也能积极地配合着我的节奏，不慌不乱地推进项目的进行。与蒋元龙一起编写源代码的合作很愉快，他能很快地将我发给他的STL库的知识学会并应用到自己编写的类中，对建造者模式也有清晰的认识，这给我的工作减少了很多负担。同时蔡远哲制作的图表与我提供给他的背景与项目功能十分符合，我拿到图表时甚至惊讶于他是否穿越时空，看到了两周后完成的源代码。石皓翔则在四员中最先开始学习QT知识，为我们的界面实现以及相关工作衔接做足准备。一个好项目需要一个好指挥，但更重要的是队员的积极回应与行动。他们的行为给予我正面的反馈，这是我最后能上交好的项目的动力。

1. 成员2学习内容及体会

蒋元龙：在这几天里，我从第一次接触项目时的迷茫，慢慢变得能够明确自己应该完成哪一部分的工作和任务，这让我初步了解到项目是怎么一回事以及分工合作的重要性。在学习新知识方面，一开始，我通过老师给的PPT和组长给的网页学习了UML的相关知识，对于用例图，类图和系统结构图等有了一个初步的了解和认识。之后，由于我负责停车位类，停车场类，奖金系统，游戏系统和评分系统的代码，所以我去学习了vector类以及string类的相关知识，以便能够编写这部分的程序。总的来说，这两天我通过实践学到了很多平时比较难接触到的有用的知识。

1. 成员3学习内容及体会

蔡远哲：在这短暂的两天的学习中，首先遇到了非常多的困难，建造者模式、接口、UML、用例图、类图、系统功能模块图、QT等等大量不了解的名词扑面而来，感到毫无头绪，无从下手。在组内分工之后，我目前负责的是用例图、类图、系统功能模块图的描绘以及项目简介和需求分析的撰写。在阅读了相关ppt、书籍以及上网寻求更细致解答之后，对建造者模式以及UML有了初步的认识，并在组内同学提供的相关资料后提炼绘制出了用例图、系统功能模块图以及类图的草图。希望在这三周内都能像这两天一样学到各种各样的新知识。

1. 成员4学习内容及体会

石皓翔：这两天特别的有收获,由于一开始并不知道有关于C++的统筹思想,对于用例图,系统思维图,ns图之类的都是一知半解,这两天我都专注于学习有关于图解的内容以及初步认识有关于qt的基本知识点,虽然还有很多都不认识,但是已经有相关的见解.而在关于停车场的大作业中,我也主要参与在图解以及qt的界面制作,因此最近都在大量阅读此类内容,查阅了有关书籍以及资料,并进行编码训练,增加对qt的操作的熟练度,,尤其在qt的界面制作中的控制有一定的难度,需要花费许多精力去学习,而用例图之类的,相比较起来会没有那么困难,因此对于这次的大作业需求,我深有感触,期待能够在接下来几天对于qt的掌握度能更加熟练!

第二部分 项目开发报告

1 项目简介

本项目是面向多区域大型停车场的管理者与司机开发的，多区域大型停车场是指由多个小停车场之间由道路连通而形成的大型停车场；软件提供了停车场管理者关于停车场地大小、停车场数量、各停车场之间道路、价格、优惠活动的管理、大型停车场的图形绘制以及查询车辆信息的功能，并且司机在使用后能给停车场打分，同时管理者也能管理评分数据。

2.1.1项目目标：

本系统希望能实现对一个多区域大型停车场进行较为便捷的管理，并希望能实现以下目标：

1）减少人力成本和管理费用

2）简洁的人机交互界面，操作响应迅速

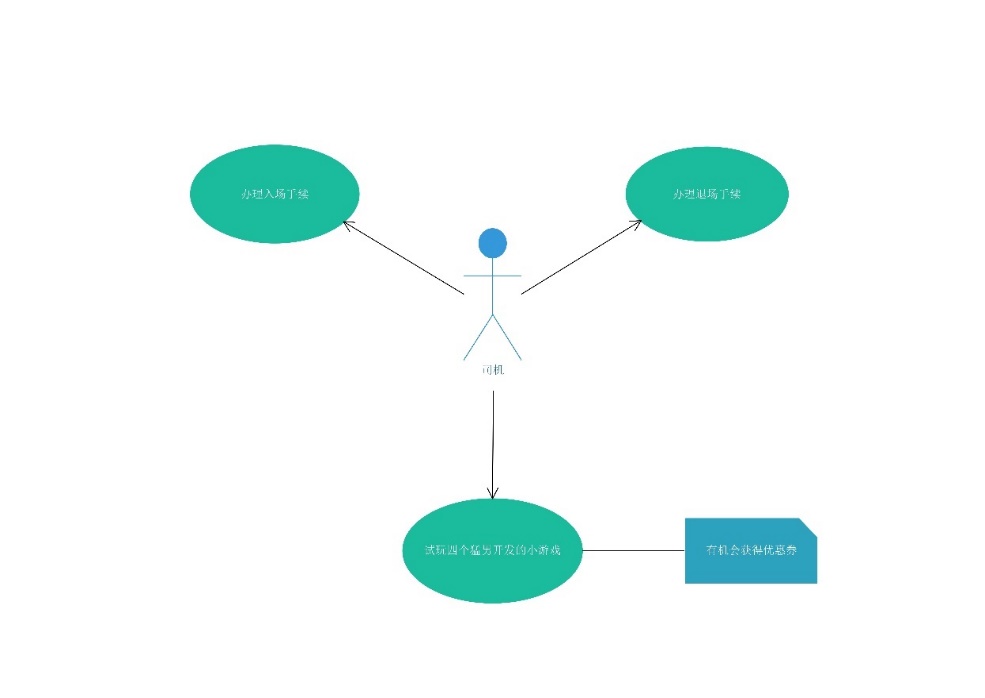
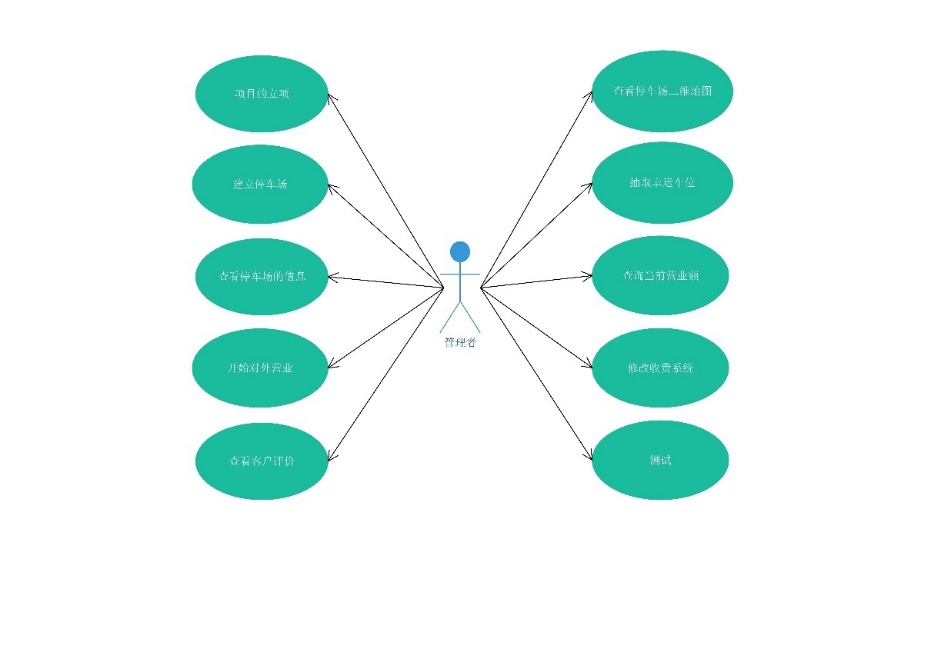
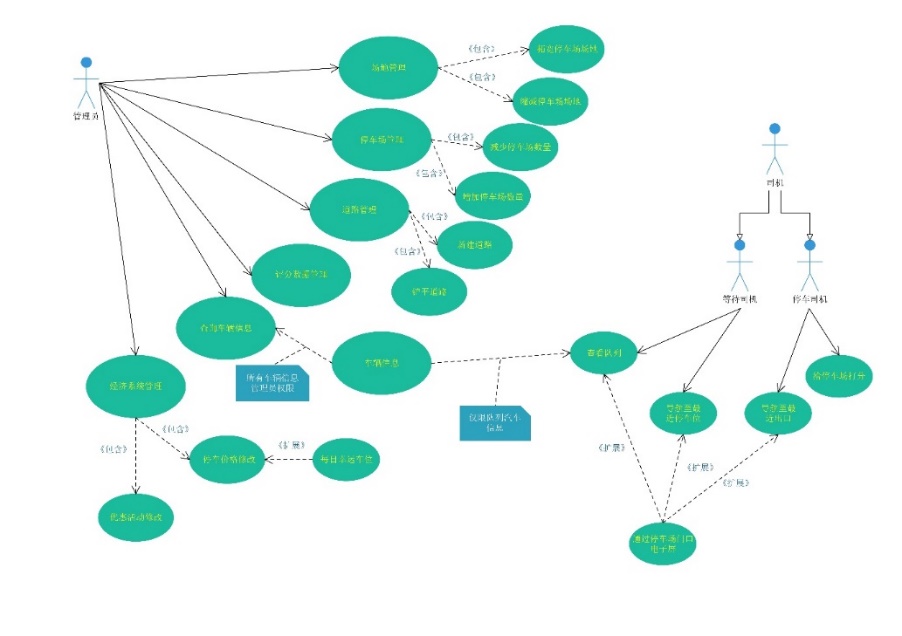
3）显示给司机们的界面内容清晰有用

4）管理者易于操作

2 需求分析

对于一个多区域的大型停车场，

1. 为了方便日后场地的扩建以及以备不时之需的缩减，需要对场地大小进行增减的功能。
2. 为了方便日后在场地空旷地区进行额外停车场的建设以及对于经济效益低的停车场的拆除，需要对新建与拆除停车场的功能。
3. 为了方便日后铺设新道路以及铲除利用率低的道路，需要新建与铲除道路的功能。
4. 为了更好地管理大型停车场系统，发现平时管理者发现不了的问题漏洞，需要司机离开停车场后评论的功能，以便给管理者反馈。
5. 为了更细致地管理整个大型停车场，需要车辆信息查询的功能，允许管理者对场地内任何位置的车辆进行信息查询，以便发现和排查问题。
6. 需要修改优惠活动和停车费用的功能，以便管理者能根据情况决定相应的营销策略，提供了可维护性。
7. 需要给予等待车辆的司机显示队列车辆系统，给予车主们一个明确的心理预期。
8. 为了节省司机们的时间，需要导航系统来指引车辆的入场和退场，以防在大型停车场内绕远路或迷路。
9. 提供大型停车场的二维地图，包括了停车场以及道路信息，这个对于管理者可以直观地看到大型停车场的停车场分布，以及其道路的连接，同时也可以给司机明确的导航。



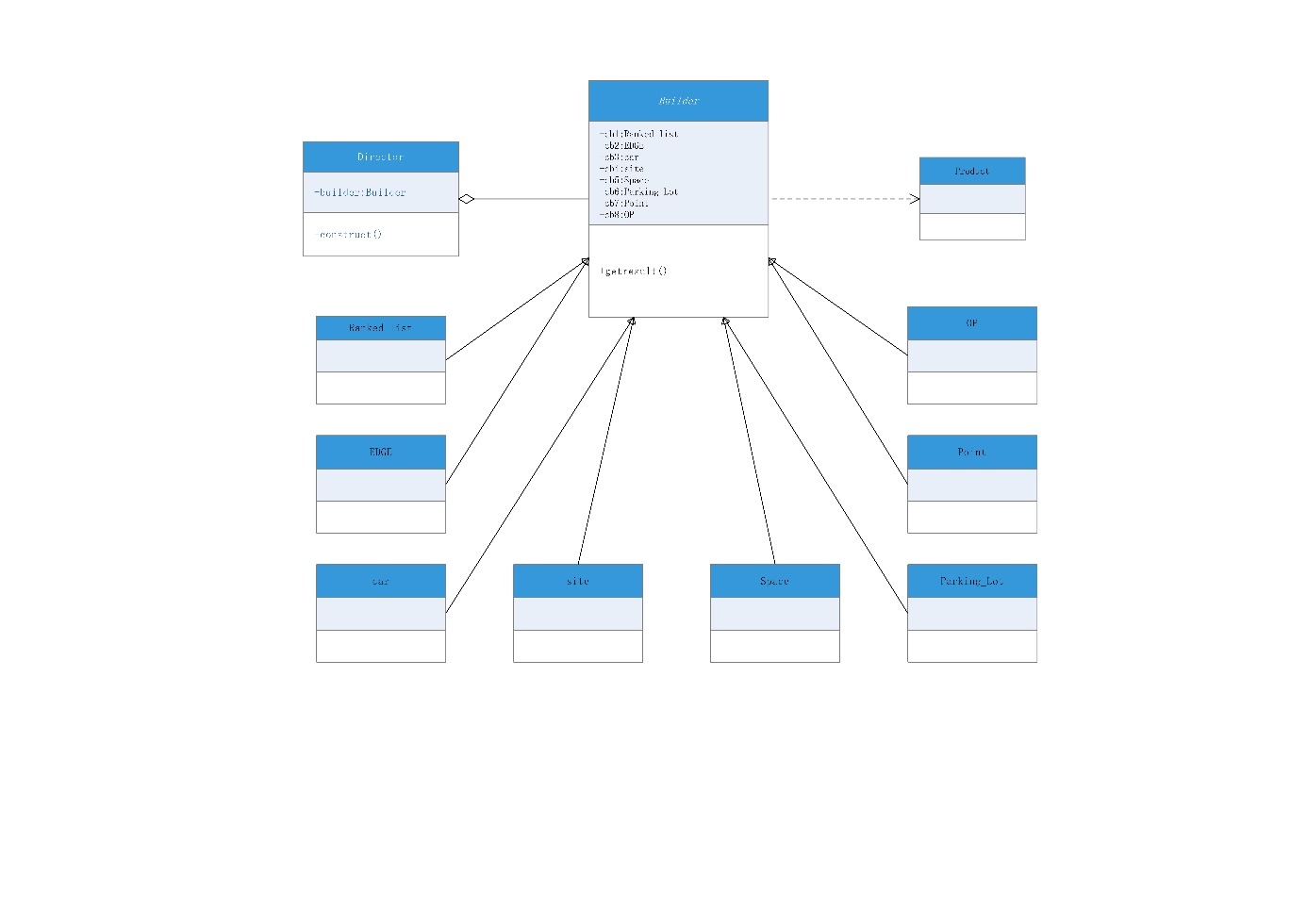
# 3 总体设计

2.3.1 设计方案

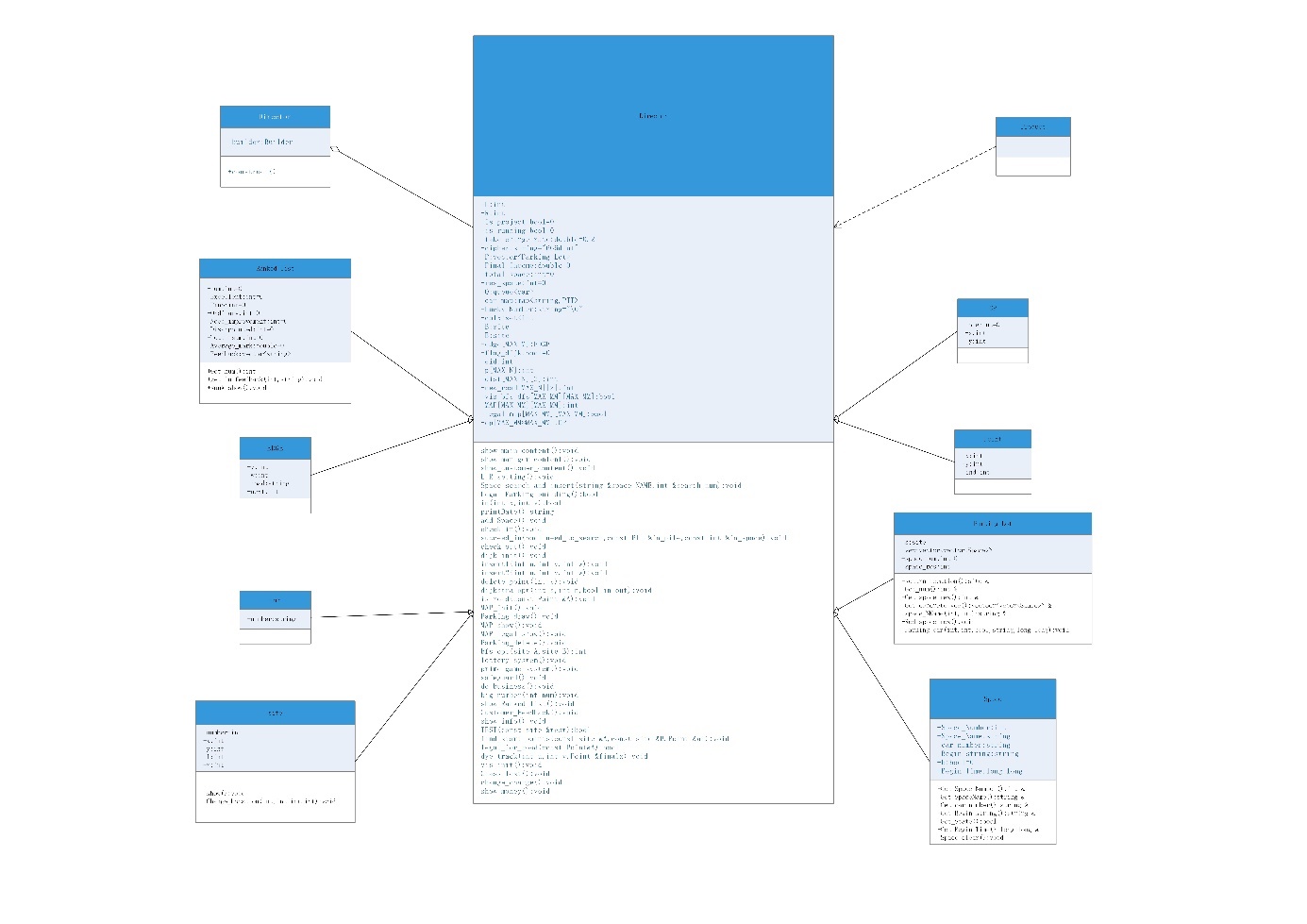
根据需求分析，以及结合该停车场为多区域的大型停车场的情况，本项目决定使用建造者模式来进行设计，原因如下：

1. 各个小停车场分布在场地各处，相互独立，利于增减停车场，且除了车位数量与编号之外性质基本一致，利于抽象为一个类。
2. 管理者不必知道产品内部细节，便于控制细节风险。
3. 在基础的板块完成后，如果有新增的功能，对接也十分方便。

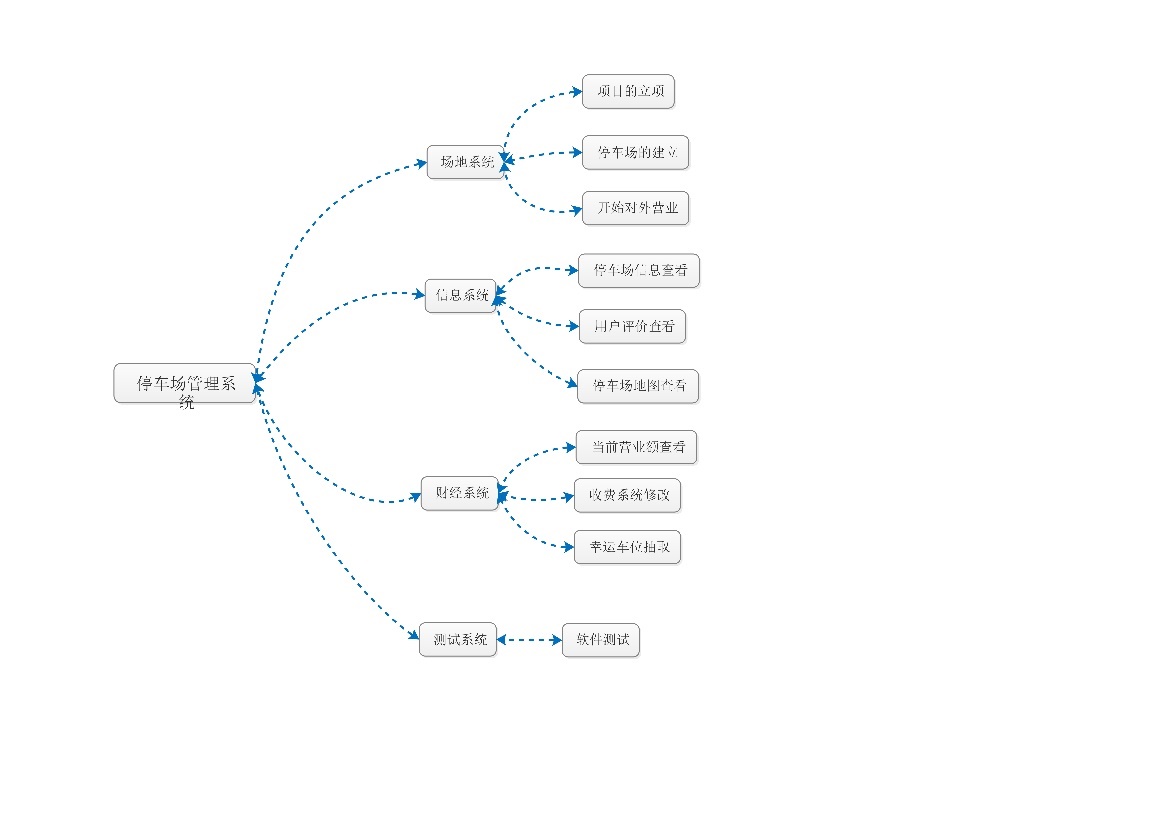
2.3.2 系统类图（草图）：



类图（详细）：



2.3.3 系统功能模块图



# 4 详细设计与实现

## 4.1 背景

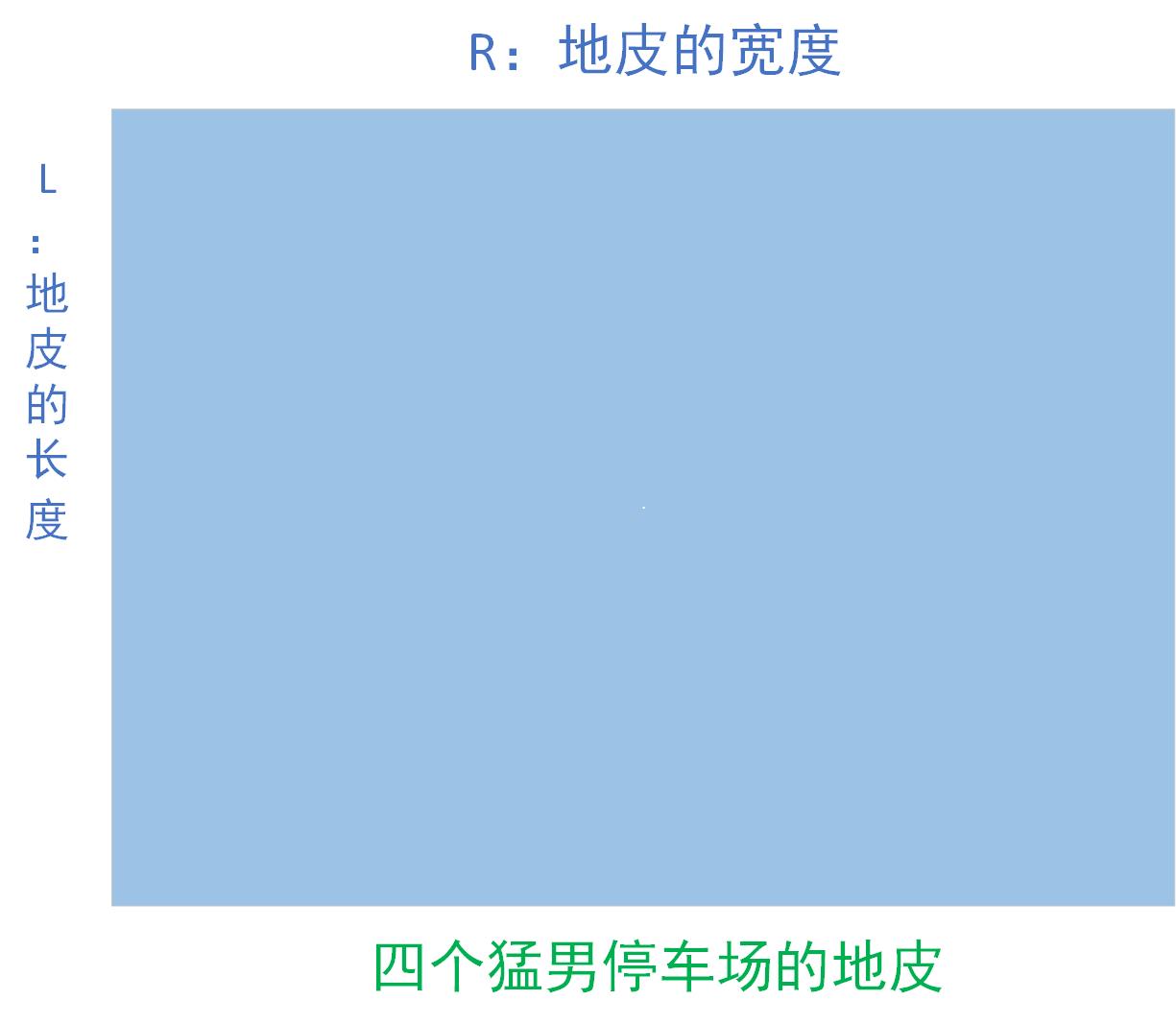
本四个猛男停车场带有较现实主义的色彩，其很多前置功能都是基于**真实停车场**的运营机制下设计的。

### 4.1.1 定位为APP

手机使用在各类平台中占据绝对优势，所以四个猛男停车场项目立足于**APP**的服务理念（虽然为黑框程序，但功能是基于面向app程序所设计的），客户与管理者皆能简单滑动手指就能体验APP的服务。对于管理者的使用，我们力求**智能化**与**轻量化**，管理者能在**提供最少的信息的情况**下，在办公室中就能轻松管理自己的四个猛男停车场。而对于司机来说，**简易的操作**以及**亲和的信息反馈**也能提高用户的体验。

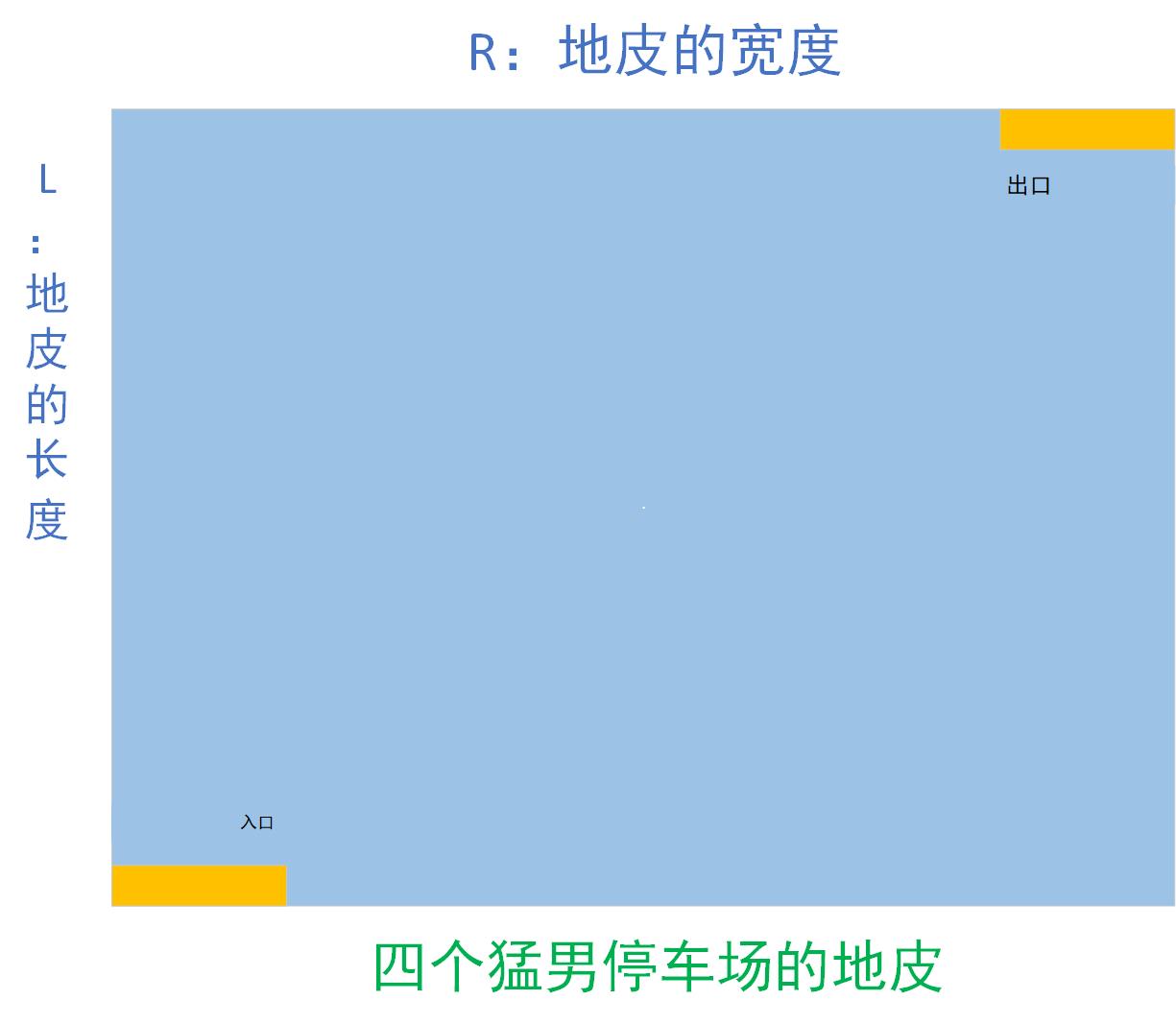
### 4.1.2 立项

对于一个尚未立项的四个猛男停车场来说，我们需先立项，才能建成一个或多个停车场。若连项目的地皮空间都没有，就不可能谈及停车场的建立与运营了。立项需要输入两个值，L与R，分别表示四个猛男停车场的地皮长度。我们预设其L与R不能低于150，不能超出200，以腾出足够的空间建立停车场，同时又不会让地图显得太空旷。



图例4.1

立项成功后，我们的出入口将会自动生成，分别在四个猛男停车场的左下角和右上角。

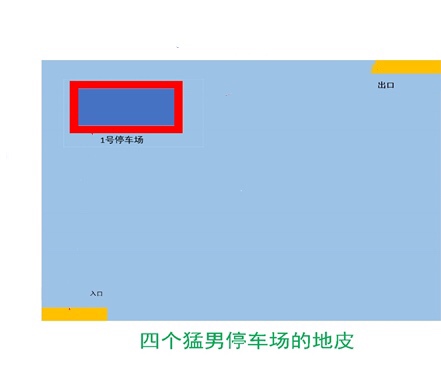


图例4.2

**在立项之后，我们的二维地图将被构建出来**，不管是建立停车场还是增加道路，我们在这个地图上都能看得一清二楚。

### 4.1.3 停车场的建立

立项之后的四个猛男停车场仍处于待营业状态，倘若车主这时候使用该项目的司机app来执行入场操作的时候会提示司机：“四个猛男停车场尚未营业…”等信息提示。因此**拥有一个停车场也是营业的必要条件**。我们在管理者app中执行A-2操作（停车场的建立），输入**停车场中心坐标**以及**长宽半径**（），我们的程序会根据地皮的非法区域来进行停车场是否违规建立的判定。在新建一个停车场的同时，会在其周围绕上一层**非法区域界线**，避免两个停车场相互挨在一起。（如例4.3）



图例4.3

如果停车场的在地图上可以合法建成，那么我们将会开始构建停车场的内部：我们伟大的建筑工人可以为我们的管理者建立1-3层的停车场，每层空间可容纳5-20个停车位，而建立多少层停车场，以及每层车位的个数（每一层的车位数目可以不同），都可由管理者来决定。

我们停车场目前可以建设至多9个，所以最终我们能建成类似图4.4的四个猛男停车场：

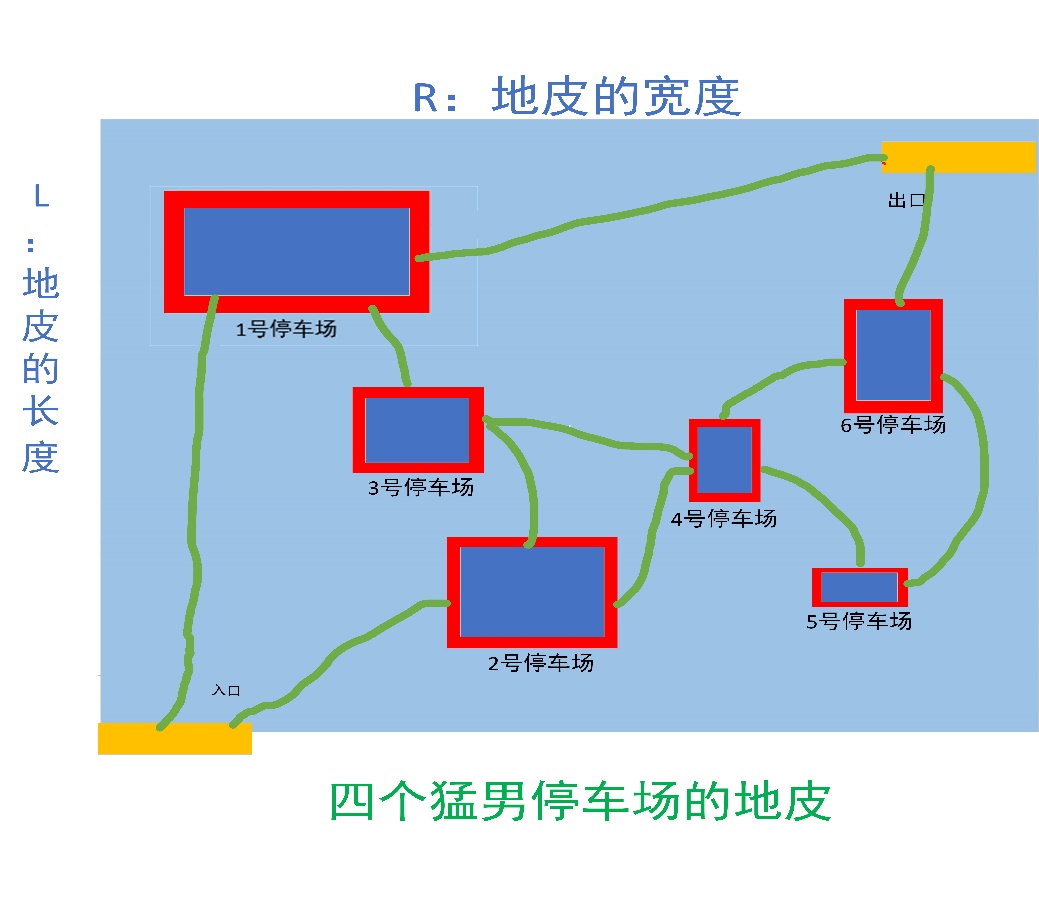


图4.4

※具体使用的数据结构与算法：在多层停车场的信息储存中，我们使用了**二维不定数组**vector<vector<Space> > vec来存储停车场的信息（Space为车位类），较普通数组而言，其空间优势以及信息的方便查询都是不可比拟的。提交完信息后，我们的建筑工人将建成一个停车场，同时根据目前停车场的个数自动为停车场命名，其名为A号，B号…停车场(比如B代表第二个停车场)，**简化了由管理者来命名的操作**。同时每个车位都有对应的名字，比如B号停车场第二层的第三个车位，其名为B-2-003，**方便车主找到车位**。

## 4.2四个猛男停车场的运营与维护

每当建立一次停车场的时候，其都是一个非连通图（从入口或出口一定到不了该停车场），同时停车场的车位总数并没有**向上汇总**（如果车主进入我们大型停车场，我们只需查看四个猛男停车场的剩余车位来判断司机是否需要进入队列等待，而不是遍历全部停车场来寻找车位），信息维护亟待解决，所以我们将会进入一个**维护系统。**

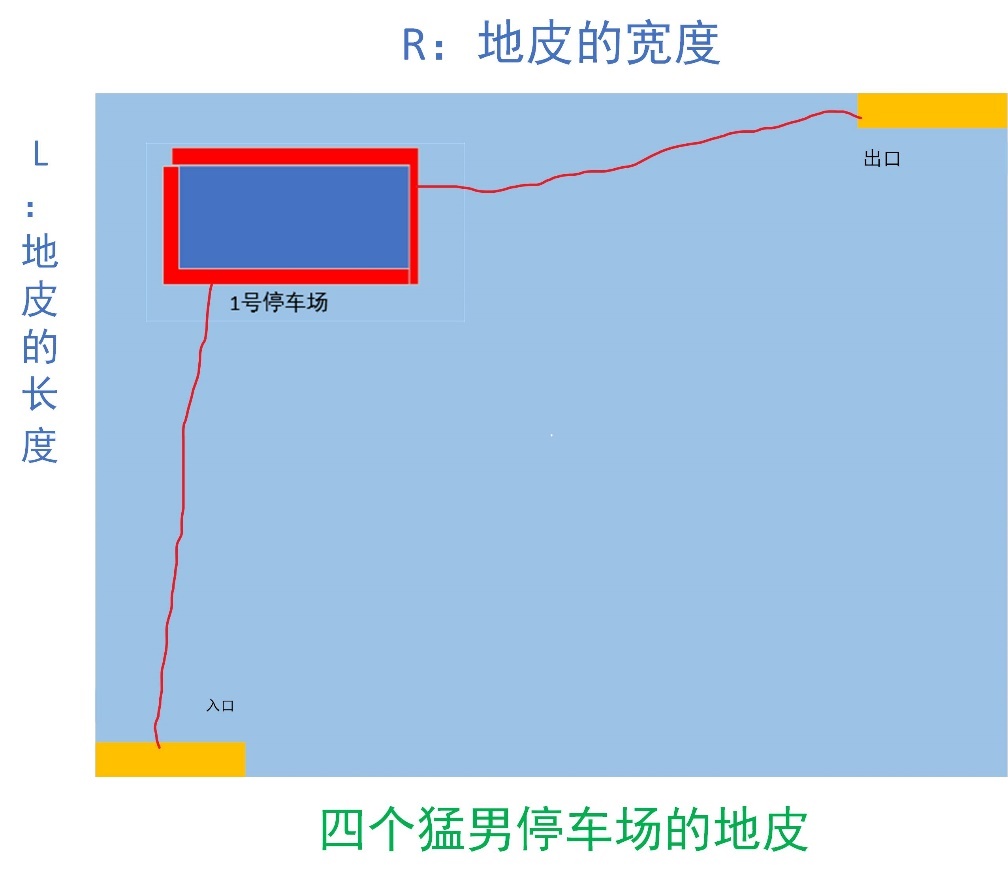
### 4.2.1 四个猛男停车场的维护

我们可以从地图上发现，停车场已经成功建成，可是其与入口出口根本没有路径相连！

这时候我们会进入**维护系统。**

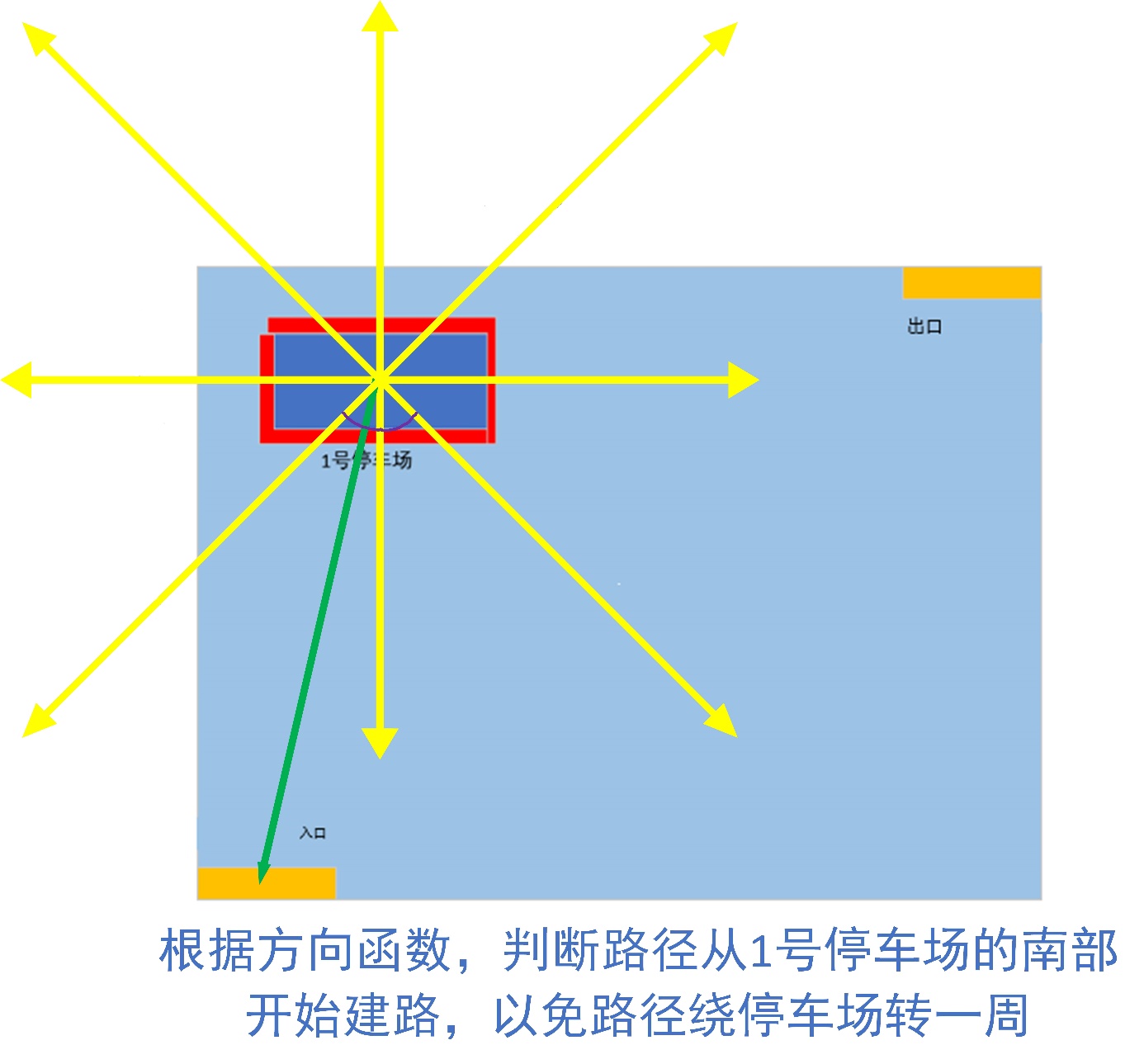
维护系统的功能

维护系统会提醒管理者，当前需将**哪些场地的路径相连**。入口、出口的**编号**分别为0与100，第i号停车场的**编号**为i。而管理者只需要通过查看地图来看需要将哪些场地相连，**输入相连的两个编号**（比如1号停车场要与入口0相连，那么就只用输入：1 0），以构建出一个**从入口以及从出口都能到达每个停车场的连通图**。例如：



图例4.4

而道路怎么修建，**管理者不需要知道**。我们通过**方位函数**来判断相连两点的相对坐标，以此来判定它的道路构建。例如让1号停车场与入口相连，方向函数将会判断从1号停车场**南部的区域**开始建路：



图例4.5

判断好方向后，用**bfs（广度优先搜索）进行最短路径搜索**。路径可覆盖停车场外的非法区域，而不能覆盖停车场与入口所占的区域，这也符合现实中的建设常识。

### 4.2.1.1 bfs(广度优先搜索)

当管理者通过**方位函数**（横纵坐标差的绝对值比较，坐标差的正负等）找到到底在南北哪一侧开始建路（搜索）的时候，我使用bfs来进行二维数组最短路径的查找算法。我们都知道，bfs是一个覆盖型的搜索算法，倘若将访问点都标记上，就算找到了目标也于事无补。此时我对bfs进行了改造（增添功能）：

1. Bfs\_opt应该要是**一个能返回最短步数记录的算法**，通过以下的写法可以将我们的路径长度返回给调用的地方（让路径长度为两点的边权，为dijkstra最短路做准备）
2. **int** bfs\_opt(site A,site B){   //起始点和目标点
3. ...
4. **int** cur = 0,last = 1,step = 0;  //用作路径长度记录，很好理解就不介绍了。cur是现存的点，last是剩余的点，step代表步数
5. **while**(!Q\_bfs.empty()){
6. Point st = Q\_bfs.front();
7. Q\_bfs.pop();
8. last--;
9. **for**(**int** i=0;i<4;i++){
10. **int** tx = st.x + dir[i][0],ty = st.y + dir[i][1];
11. **if**(!in(tx,ty)) **continue**;    //越界区域要跳过
12. **if**(MAP[tx][ty]==B.number){//搜到目标点
13. ...
14. **return** step;
15. }
16. **if**(!vis\_bfs[tx][ty]){
17. vis\_bfs[tx][ty] = 1;
18. cur++;   //cur++
19. op[id] = OP(tx,ty,ID);
20. Q\_bfs.push(Point(tx,ty,id++));
21. }
22. }
23. **if**(last==0){  //说明队列中已经没有step的点了，step++
24. last = cur;
25. cur = 0;
26. step++;
27. }
28. }
29. }

2、其要**有最短路径标识的数据结构**。我们使用链式前向星（节约空间和优化搜索的效率）的方法模拟链表，然后反向查找将最短路的路径给画出来。

完整的bfs\_opt路径算法以及相关数据结构如下：

1. **struct** OP///用反向链式前向星来储存最短回路的路径信息
2. {
3. **int** pre=0;
4. **int** x,y;
5. OP(**int** x=0,**int** y=0,**int** pre=0):x(x),y(y),pre(pre){}
6. };


10. **void** dye\_track(**int** u,**int** v,Point &finalx)    //路径染色，使用哈希映射，同时将染色值存入set集合中，方便地图的颜色绘制
11. {
12. **int** color\_num = (u\*10086+v\*10010)%10000+10001;
13. colx.insert(color\_num);
14. OP sta = op[finalx.ind];
15. **while**(sta.pre!=0){  //当反向链式前向星还没到头
16. MAP[sta.x][sta.y] = color\_num;
17. sta = op[sta.pre];
18. }
19. **return** ;
20. }
22. **int** Director::bfs\_opt(site A,site B)  //bfs铺路算法
23. {
24. vis\_init();
25. **if**(B.number==0) B.number = -100;
26. **if**(B.number==100) B.number = -200;
27. memset(op,0,**sizeof**(op));
28. **for**(**int** i=0;i<L;i++){
29. **for**(**int** j=0;j<R;j++){
30. **if**(legal\_for\_road(Point(i,j))){
31. vis\_bfs\_dfs[i][j] = 0;
32. }
33. **else**{
34. vis\_bfs\_dfs[i][j] = 1;
35. }
36. }
37. }
38. queue<Point> Q\_bfs;
39. **while**(!Q\_bfs.empty()) Q\_bfs.pop();
40. Point start\_p;
41. **int** id=0;       ///为链式前向星做准备
42. find\_start\_points(A,B,start\_p);
43. start\_p.ind = 0;
44. Q\_bfs.push(start\_p);
45. Point finalx;
46. **int** cur = 0,last = 1,step = 0;  ///用作路径长度记录，很好理解就不介绍了。cur是现存的点，last是剩余的点，step代表步数
47. **while**(!Q\_bfs.empty()){
48. // puts("TEST1-----------------");
49. Point st = Q\_bfs.front();
50. **int** ID = st.ind;
51. Q\_bfs.pop();
52. last--;
53. **for**(**int** i=0;i<4;i++){
54. **int** tx = st.x + dir[i][0],ty = st.y + dir[i][1];
55. **if**(!in(tx,ty)) **continue**;    //越界区域要跳过
56. **if**(MAP[tx][ty]==B.number){
57. finalx = Point(st.x,st.y,st.ind);
59. dye\_track(A.number,B.number,finalx);
60. **return** step;
61. }
62. **if**(!vis\_bfs\_dfs[tx][ty]){
63. vis\_bfs\_dfs[tx][ty] = 1;
64. cur++;
65. op[id] = OP(tx,ty,ID);
66. Q\_bfs.push(Point(tx,ty,id++));
67. }
68. }
69. **if**(last==0){
70. last = cur;
71. cur = 0;
72. step++;
73. }
74. }
75. }

### 4.2.1.2 dijkstra算法

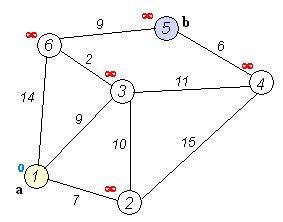


图4.7

虽然dijkstra是图论中最短路的比较基础的算法，但是没有堆优化的dijk的复杂度最差可以达到O(V^2)，而我采用的是**小根堆优化**的dijkstra，假设停车场的规模扩大之后，维护的效率也能保持在较优的水准。

算法目的：

对于此大型停车场，有若干个停车场的管理而言，

找到从入口到最近可停放的停车场位置是十分重要的

我们**算法的目的就是记录从入口到可停放停车场、以及停车场到出口的最短路径**

然后**提供给司机从入口到停车场能最快到达的路径名称**，同时提供预估所需要的时间(预估时间=一个系数×路径长度)

算法优势：

1、摒弃链式stl数据结构（list），采用**数组链式前向星来模拟list<>**，给程序节约了部分空间；

2、此算法用了**小根堆优化**，可以将裸dijkstra算法的o(V^2)时间复杂度降至O((V+E)lgV)

且**因边权都为正值，不会形成最短路自环，那么比SPFA算法在时间复杂度更有优势；**

算法的使用：

1、计算**入口到停车场**的最短距离以及相应路径

2、计算**出口到停车场**的最短距离以及相应路径

何时调用：

在进入维护系统的时候调用，以及增加边的时候调用。

具体算法：

1. **bool** Director::dijkstra\_opt(**int** s,**int** n,**bool** in\_out)    ///s代表入口。如果不是连通图，则返回0，作为非法的停车场管理做出判断
2. //n的个数应该是停车场个数+2，算上出口100，入口0
3. {///in\_out=0是入口单源最短路，=1是出口单源最短路
4. set<PII,less<PII> > min\_heap;  ///用集合来伪实现一个小根堆，同时有映射二叉堆的功能。
5. **int** vis[MAX\_N];// 标记每个顶点是否在集合 U 中
6. **for**(**int** i=0;i<MAX\_N;i++){
7. dist[i][in\_out]=INF;
8. new\_road[i][in\_out] = -1; ///为后面更新做好准备
9. vis[i]=0;
10. }
11. min\_heap.clear();
12. ///数据初始化
13. min\_heap.insert(make\_pair(0,s));
14. dist[s][in\_out] = 0;    ///初始点置为0
15. **for**(**int** i=0;i<n;i++){
16. **if**(min\_heap.size()==0){// 如果小根堆中没有可用顶点，说明有顶点无法从源点到达，算法结束
17. **return** **false**;
18. }
19. set<PII,less<PII> >::iterator it = min\_heap.begin();
20. **int** u = it->second;
21. min\_heap.erase(\*it);
22. vis[u] = **true**;
23. // 进行和普通 dijkstra 算法类似的松弛操作
24. **for**(**int** j=p[u];j!=-1;j=edge[j].next){
25. **int** v = edge[j].v;
26. **if**(!vis[v]&&dist[v][in\_out]>dist[u][in\_out]+edge[j].w){
27. min\_heap.erase(make\_pair(dist[v][in\_out],v));
28. dist[v][in\_out] = dist[u][in\_out] + edge[j].w;
29. new\_road[v][in\_out] = u;///记录路径
30. min\_heap.insert(make\_pair(dist[v][in\_out],v));
31. }
32. }
33. }
34. **return** **true**;
35. }

至此，图的构建已经完毕，信息也汇总到了管理者系统中，我们可以选择开始运营了。

## 4.3 对于用户

对于用户而言，有三个服务可供选择：

用户功能：

### 4.3.1选择入场

车主选择入场时，有两种情况：

1. 当我们查看停车场剩余车位>0的时候车主将进入的函数中，成功入场，触发函数succeed\_in()。
2. 当剩余车位等于0时，我们的车主将会被安排到车队里进行等待，同时我们将会开一张小票给车主，显示车主目前前面还有多少个车辆。当有车主办理离场手续的时候，我们就会让队首的车辆进场，同时**直接让车主去往离开车位的车主原本的位置**，**避免遍历寻找车位操作。**

### 4.3.1.1 succeed\_in()函数

当车主是从车队进入停车场的时候，我们不需要搜索空余车位，而是直接告诉司机新腾出的车位的停车场怎么走。

而**当一开始就有车位**的时候，根据预先dijkstra存储的单源最短路径**dist数组**，我们找到**路径最短的有车位的停车场**，再根据find\_shortest\_way()函数来提供给司机路径导航，以此来让司机能最快到达停车场。

### 4.3.2 选择退场

司机输入车牌号后，系统将会为车主办理退场手续。

在这里我们使用了基于map<string,PII> (PII是pair<int,int>)的数据结构来进行树表搜索，可以将我们的搜索摒弃遍历所有的车位，而是在O(logn)的时间复杂度搜到车位的具体位置，就算**把项目扩大，用树表结构储存信息在查找信息方面仍然是十分优秀的。**

当车队不为0时，传递给succeed\_in()一个布尔参数，告诉我们的函数不需要再进行一次搜索，因为已经有车在等待，只需要让等待的车主入驻这个车位即可。

当车主离场的时候，会让车主自愿选择填写停车场的评价反馈。**司机可以给停车场打分，还可以给停车场的管理者留言**（管理者可以查看）。

### 4.3.3 选择游戏

作为APP的程序，游戏肯定是少不了的。司机可以通过游玩游戏来获取优惠券（暂无写到程序中，但可以玩游戏）。

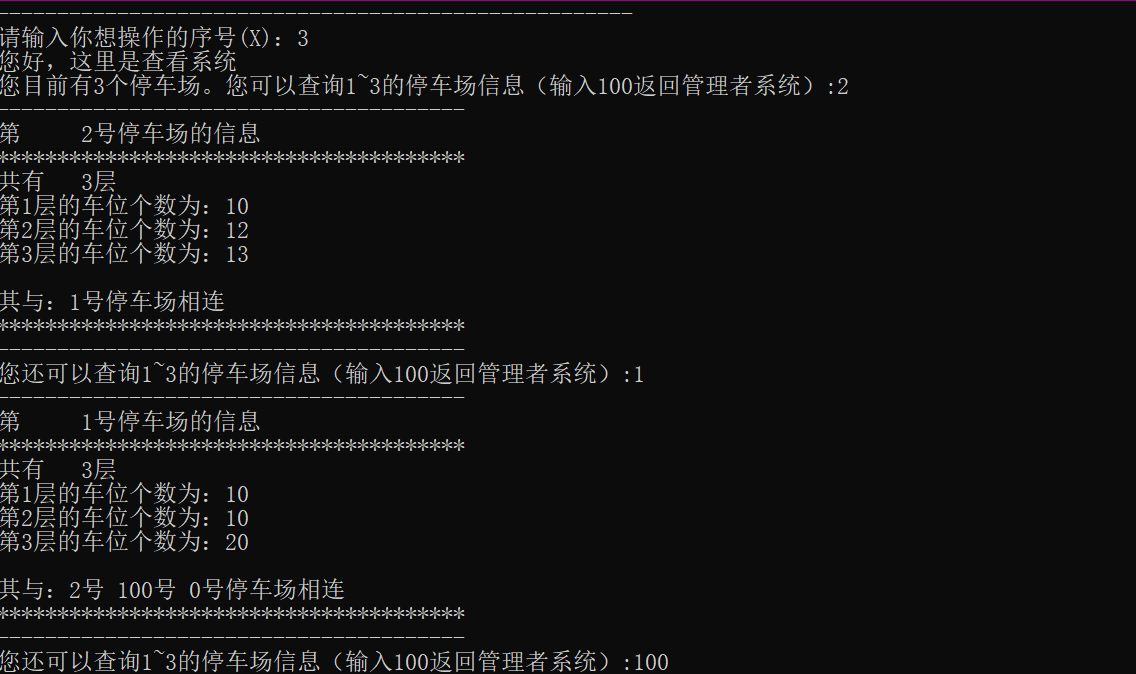
## 4.4 对于管理者

管理者负责管理四个猛男停车场，可操作的功能有不少。而用户和管理者皆能使用这个四个猛男APP，所以我们的管理者需要输入密码：666dlut后才能进入管理者界面。

项目的立项、停车场的建立以及维护系统在前文已介绍的比较周全了，现在介绍**其他功能**：

### 4.4.1 查看四个猛男停车场的信息

对于管理者而言，在办公室就能知道数个停车场的信息，能给管理者很大的帮助。我们A-3的功能就是返回给管理者目前有多少个停车场，**每个停车场的信息**是什么。



只要管理者输入现有停车场对应的编号，其就能查到停车场具体的车位信息，同时也能**知晓其与哪些停车场（出入口）相连**。

### 4.4.2 对外营业

对于A-4的操作其实很简单，当有停车场的时候（此时因为之前必然会进入维护系统，所以四个猛男停车场已经是一个连通图了），函数就将开始营业的布尔值设为1，此时司机就能进入该停车场享受服务了。

### 4.4.2 查看四个猛男停车场的客户评价

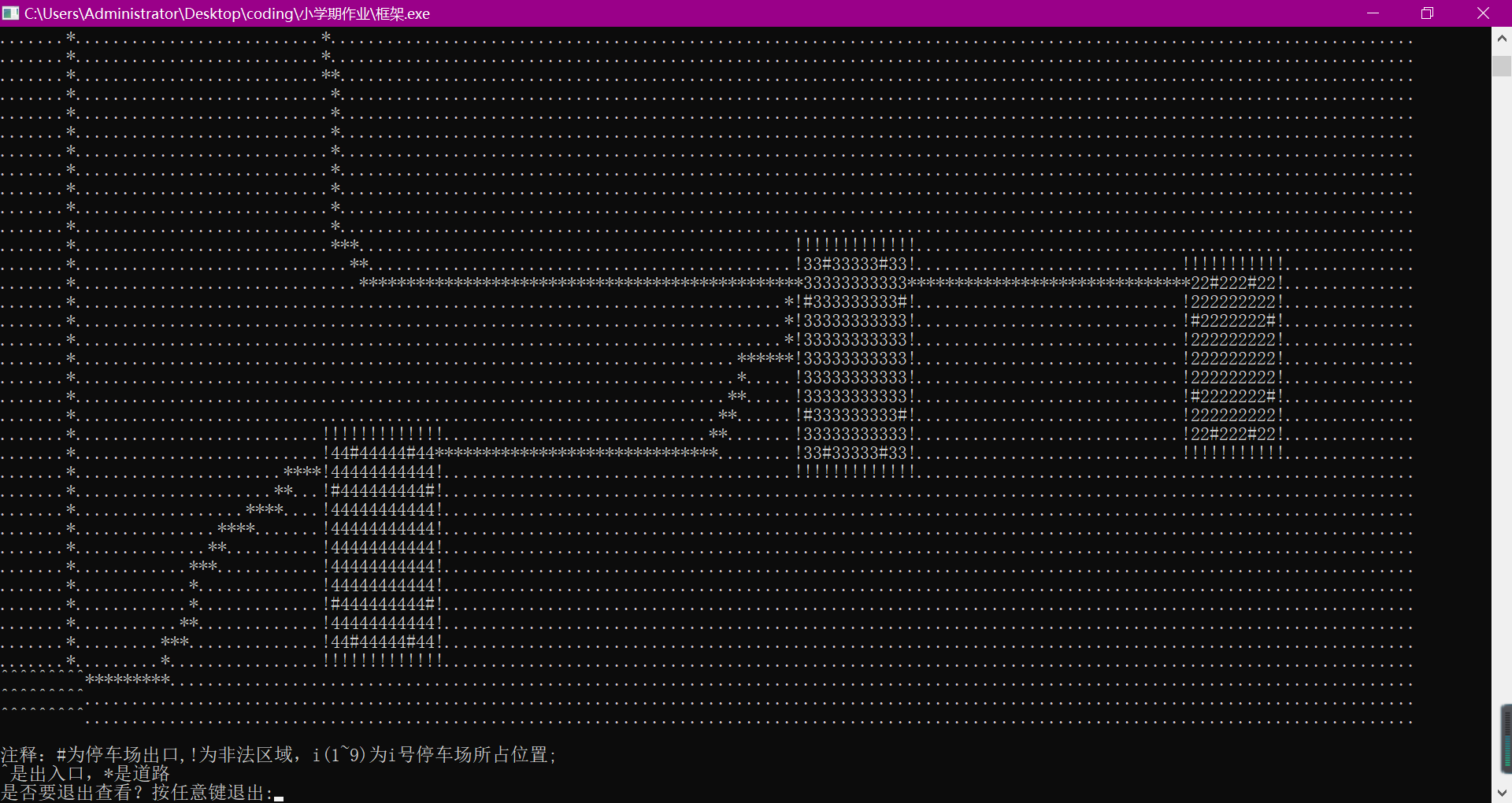
评价类中，我们预先按评分五等化，设立了五个等级：

当用户输入分数（0~100）的时候，系统将会自动将分数按等级划分。

我们可以在A-5操作中看到用户对我们猛男停车场的评价。管理者可以根据用户的评价对停车场进行改进与维护。同时**管理者还能看到用户留下的评价**，比起冷冰冰的评分，用户的评论更来得有价值。

### 4.4.3 查看二维地图

这是**四个猛男停车场的一大特色**，也是该项目花时间和精力最多的地方之一。只要管理者确立了停车场的位置，以及与那些区域相连，地图就能自动生成，且直观简洁。对于大型停车场的布局，不管是对于客户的导航，还是管理者可以直观的看到停车场的位置都是很有价值的。



在地图的下方，我们给出了**注释**，可以让管理者和用户轻松读懂地图的含义。

### 4.4.3.1绘图所涉及到的算法

L\_R\_Setting:负责地图的初始化，地图大小的确立；

MAP\_draw():负责绘制地图的底版（空白区域以及停车场，出入口）；

Bfs\_opt():负责查询目标路径，以及记录最短路的路径。

dye\_track():颜色绘图，依照bfs\_opt()记录的路径，按照反向链式前向星进行路径染色，最终沿目标点到初始点绘制。

### 4.4.4 抽选幸运车位

这也是我们停车场的一个特色。我们会根据目前有多少个车位，以及车位所在的停车场进行抽选。一旦用户从幸运车位离开并结账，**用户将会获取五折优惠**，并通知用户中了奖。

### 4.4.4.1抽取幸运车位涉及的算法

我先获取车位的个数，然后跑一次随机数对车位总个数取模，得到随机数num。

倘若用遍历算法，每遇到一个车位就num--，那么对于大型停车场而言是很低效的算法。我们运用**汇总的思想**：

1. 当num大于一整个停车场的车位个数，让num-=整个停车场的车位个数，换到下一个停车场，直到num小于某停车场的车位总数。
2. 当num大于该停车场的某一层，让num-=该层的车位个数，换到下一层，直到到num小于某层的车位数。
3. 这时才使用num- -的遍历方式找到我们的幸运车位，并将其编号存入我们的“幸运车位”字符串中。

只要可以优化算法的地方，我都进行了较好的优化，抽取幸运车位的功能就体现了这一点。

### 4.4.5 营业额显示

不算特色，输出当前的盈余即可。

### 4.4.6 修改收费系数

我们的停车场是按停车时间来线性收费的。当物价发生变化的时候，只需修改收费系数，收取的费用就能上升或变得优惠。

### 4.4.7 测试函数

（这其实是给程序员编写的，方便调用director里面的类成员变量，以最快查出bug）。

### 4.4.8 停车场的连边功能

这是连通路的额外功能，即不只是为了构成连通图，还能**让一些不必要的一些边相连**，从而改变图的构成，以及**改变出入口的最短路径**。

简要的代码：

1. **void** Director::ADD\_TRACK()
2. {
3. cout<<"(是否要查看地图(0为可以建造，1为不可)？（y=1,n=0）)：";
4. **bool** flagx;
5. cin>>flagx;
6. **if**(flagx){
7. MAP\_legal\_show();
8. }
9. cout<<"请输入连通的两个点（入口是0，出口是100）（u，v）：";
10. **int** u,v;
11. **while**(cin>>u>>v&&(u==v||(u>P.size()&&u!=100)||(v>P.size()&&v!=100))){
12. printf("u与v不能相同且要合法！重新输入：");
13. }
15. site buf1,buf2;
16. **if**(u==0) buf1 = B;
17. **else** **if**(u==100) buf1 = E;
18. **else** buf1 = P[u-1].Return\_Location();
19. **if**(v==0) buf2 = B;
20. **else** **if**(v==100) buf2 = E;
21. **else** buf2 = P[v-1].Return\_Location();
22. //printf("test6:%d %d,%d %d\n",buf1.x,buf1.y,buf2.x,buf2.y);
23. **int** w = bfs\_opt(buf1,buf2); ///返回最短路径的长度（即有向图中的边权）
24. insert2(u,v,w);
25. dijkstra\_opt(0,P.size()+2,0);
26. dijkstra\_opt(100,P.size()+2,1);
27. cout<<"成功连通两地！\n";
28. Sleep(1500);
29. system("cls");
30. }

# 系统测试

（附在文档外的测试txt文件中）

# 设计总结

### 总结项目总体完成情况：

在刚开始的计划功能中，有95%左右的都写出来了，甚至额外增加了一些实用的功能。剩下的5%由于时间不够（而若时间多半周时间还是能做出来的），便暂且不能实现了。另外花了很多时间，而项目还是不能完全对接QT平台，这也是很大的遗憾。

### 解决办法：

将黑框程序的资料写入文件里，供qt工程的程序使用。

### 不足：

完全没有料想到qt与黑框程序的对接如此难。或许一开始应该先去学qt，学完qt后直接在qt的编辑器上写程序，而不应该程序和qt界面分离后去对接。

### 写项目的反思：

1. vector类的写法有问题

问题1：越界访问。倘若vector没有推入数据，直接访问vec[0]会runtime error。

问题2： vector的类包含，比如vector<vector<space> >二维不定数组类中的正确调用方法应该是：先临时建一个vector<space> buf数组，将space数据推入buf中，再将buf整体推入vec中。

总结：调用vec的具体下标值时，需要严谨思考是否会出现越界。

1. 重载函数的原理没有弄清楚

问题1：比如一个类包含的题为vec.push\_back(space(xx,yy,zz));的实现中，先是调用参数构造函数space(int x, int y, int z),然后又调用了space(const space& A)的常引用拷贝构造。

总结：问题出在构造函数的时候，参数构造函数与常引用拷贝构造都要检查。

1. 对于类成员变量和成员函数的**调试效率**很低

解决方案：在类里面写一个测试函数，void Class\_Test()（即后来的调试系统），然后在主界面中调用。在函数里写你要调用的成员函数，以最快解决错误数据。

### 对于项目的展望：

1、关于铲除停车场的操作：

原本想写一个铲除停车场的操作，涉及图论的删边算法。我简单写了一个链式前向星的删边函数，但其时间复杂度最差可达O(N\*M)（最优为O(N)）。奈何时间不够，便就此作罢。假设时间充裕，我们还能运用网络流i xor 1找出反向边的技巧，在最差O(N)的时间复杂度写出铲除停车场的算法

2、关于路径删除

停车场铲除之后，在图论数据中已经删除了相连的边，而二维地图上并没有把染色的路径给清除掉。在我们的1.0项目（即目前完成的项目）上已经存储了路径编号和路径的开端，只要利用dfs（深度优先搜索）就能将路径删除，而避免遍历整个地图，由二维搜索降维成一维搜索，优化了时间的复杂度，同时给原本四个猛男停车场腾出空间。

3、游戏的优惠券

目前游玩游戏后并没有给予优惠券，若能有时间继续做下去，那么可以将通关（布尔参数）与车牌号（string）用map<string,bool>构成关联，只要车主出了停车场缴费的时候，我们在O(logn)就能查到该车主到底有没有通关，从而判定能否给予奖金优惠。