同濟大學
TONGJI UNIVERSITY

面向大型全自动泊车系统的优化控制方法

指导老师：杜豫川教授

团队成员：阴丹宁 樊相宜 蔡秀挺 吴正浩 杨扬

所在学院：交通运输工程学院

论文提交日期：2017/03/24

摘要

城市汽车保有量节节攀升的同时城市土地开发几近饱和，停车难成为各国的大中城市面临的问题；与此同时，人工动态指引、建设多层停车场和使用停车机器人等智能化停车场开始出现，但运营过程中存在指引信息滞后、车辆能耗高、取车等待时间过长诸多问题，亟需进一步优化。

本文以大型全自动泊车系统为重点研究对象，首先以嘉亭荟地下停车场为参考构建无优化的自动驾驶泊车模型，对无优化自动泊车系统进行仿真模拟，在分析存在的问题后从设计停车场结构、出库入库方案、冲突规则和优化路径选择等方面对目前的自动车泊车系统进行创造性优化。

本文提出多路泊车模型、内部多路外部环路泊车模型和内部单路外部环路模型三种大型全自动泊车系统，在CAD中设计停车场结构，在Tiled地图编辑区中对停车场结构进行简化，将简化结构导入COCOS游戏开发平台进行泊车系统仿真的同时用Matlab计算用户平均取车等待时间、停车场利用效率系数和车辆能耗等评价指标；综合仿真模拟与评价指标对上述三个模型分别从驾驶者、停车场和车辆的角度对自动泊车模型进行评价分析，推选出综合较优模型。

通过综合评价，本文认为内部单路外部环路模型综合运营效果最优。虽然此泊车系统有较高的车辆能耗，但它在减少用户取车等待时间和增加停车场利用效率方面远大于另外两个模型。在未来发展中，新能源汽车特别是太阳能汽车的开发与普及将会大大缩小此模型的弊端，模型优势更加凸显，发展空间将会很大。

关键词：自动泊车，路径规划，结构设计，层次分析法 AHP

ABSTRACT

Urban car ownership is climbing at the same time urban land development is almost saturated, "parking difficult" has a particular problem which the country's large and medium-sized cities face. At the same time, manual dynamic guidance, the construction of multi-storey parking, the use of parking robots and other intelligent parking lot began to appear. However, there exist several difficulties during the operation: the lag of guidance information, high vehicle energy consumption and long waiting time to fetch the car. There is particular need for further optimization.

In this paper, a large-scale automatic parking system is the focus of research object. Firstly, use Jiatinghui underground parking lot as a reference to build no optimal automatic driving parking model and simulate no optimization of automatic parking system. After analyzing the existing problems, refer to the design of the parking lot structure, the library storage program, conflict rules and optimized path selection to optimize the current automatic car parking system creatively.

This paper proposes a multi-channel parking model, internal multi-channel external loop parking model and internal single external loop model three large automatic parking systems. Design the parking structure in the CAD and edit simplified area on the parking field structure in the Tiled map. The simplified structure is introduced into the COCOS game development platform for parking system simulation. At the same time, the simulation and evaluation index: the average waiting time of the vehicle, the utilization factor of the parking lot and the energy consumption of the vehicle are calculated by Matlab. The above three models are used to evaluate the automatic parking model from the perspective of driver, car park and vehicle respectively, and the comprehensive model is selected.

Through the comprehensive evaluation, this paper holds that the internal single external loop model has the best operation effect. Although this parking system has a higher vehicle energy consumption, it is much better than the other two models at reducing user waiting time and increasing parking lot utilization efficiency. In the future, the development and popularization of new energy vehicles, especially solar vehicles, will greatly reduce the drawbacks of this model, which advantage is more prominent and the development space will be great.

目录

一．绪论

1.1 研究背景及意义

目前，随着我国国民经济的持续稳定发展以及生活水平的提高，尤其是汽车工业的快速发展，“买车容易停车难”成为了城市生活的新烦恼。如何缓解越来越突出的停车难矛盾，实现人与车、车库与城市空间之间的和谐，已经成为各地政府关心的社会问题。

城市停车难是每个国家的大中城市都要面临的问题。发达国家日本中的东京、大阪，美国的纽约、华盛顿，中国香港、台湾等城市四十多年前。新面临了停车难的问题。中国的北京、上海、广州、成都等城市二十年前部分地区已出现了停车难的问题。为了解决该问题，发达国家四十多年积累的经验是：建立停车场（库）；发展机械式立体车库，节约有限的土地资源。日本东京、大阪等城市，因为土地资源十分紧张，优先发展高密度电梯式智能化塔式立体车库。东京、大阪目前已有 2000 多座电梯是智能化塔式立体车库。欧美国家发展立体车库以仓储式的大型立体停车场居多。这是适合欧美国家的国情，欧美国家停车场的服务半径是其他国家的一倍有余，达 600 米甚至更远。但也配置了占立体车库总量 5%的电梯式智能化塔式立体车库。中国的香港、台湾也以电梯式智能化塔式立体车库为主，配以封闭式或地下式升降横移动立体车库。其他发达国家如德国、韩国、新加坡都是以发展高密度电梯式智能化他是立体车库来解决停车难矛盾，而我国目前以发展多段式（敞开）升降横移地面立体车库为主。这种立体车库的优点是：价格低、投资小。但其存在的主要问题是，其与城市的建筑景观不协调；其增加的泊位有限；占用的城市土地和空间相对较大，土地资源没有得到充分开发，从而导致停车难的矛盾没有从根本上得到解决。

在这样的情况之下，发展大型全自动泊车系统有着极大的时代意义和需求。该泊车系统旨在基于不占用更多城市土地和空间，不增添更多泊位的情况之下提高车辆在停车库中的存放以及提取效率，减少驾驶员的等待延误。在现有的设施规模下通过效率的提高从而在相同时间内解决更多的车辆停放问题。

1.2 国内外发展现状

北京王府井地区停车诱导系统于 2001 年 12 月开通运行，是我国第一套正式投入运行的

智能停车诱导系统。作为“数字王府井”系统工程的组成部分，停车诱导系统要由智能停车场、控制中心和网络通信电子系统等组成，人工收费停车场改造成计算机管理和控制的智能化停车后，通过卫星通信和光纤通信技术构成系统网络，控制中心对各停车场空车位数据实时采集处理，并将信息发布到所有电子显示牌上。共设立 3 块大牌，48 块小牌，显示着附近停车场位置及剩余空车位信息，涵盖附近 10 多个停车场的区域，连通了王府井地区停车场 800k 以上的停车位，该系统与北京交通台直接连接，引导进入王府井地区的司机实现就近、尽快停车。

上海自 2002 年起，在全市率先开始引入停车诱导系统，2003 年 8 月 16 日，上海“停车诱导系统”黄浦区示范工程一期工程正式建成开通。示范工程选择停车问题最为突出、影响面积极大的中央商务区(CBD)地区，东至黄浦江、西至西藏中路、北至苏州河、南至延安东路，共 18 个停车场、3000 多个车位。到 2003 年底，黄浦区建成了覆盖中央商务区(CBD 地区)、广场地区、豫园地区的智能停车诱导系统，该系统为 28 个停车场(库)进行停车诱导黄浦区停车诱导系统运行后，平均减少了 10%的交通流量，加入系统的停车场(库)的停车泊位利用率平均上升了 15ry0。上海 2010 年前将建成全市停车信息服务平台，同时实现停车诱导系统的全市覆盖。

广州市 20 世纪 90 年代后期已经建立了较为先进的静态交通管理信息系统,这使其走在了全国停车诱导系统发展应用的前列。广州根据城市交通小区划分了 274 个 TR 工 PS 分区，采用简单的顺序码编码方式表达 TR 工 PS 分区，对停车场采用区间码，形成了编码规则;结合数据库设计完成了功能结构配置和从系统流程到对象结构树的转换。登录广州市静态交通管理信息系统，用户可以对数字化地图实行放大、缩小、平移等基本操作，并利用它实施多项辅助数据查询功能。不过该系统偏重于静态交通管理功能,尚未实现出行途中的停车诱导，不属于严格意义上停车诱导系统。

欧洲、美国、日本可以说是国外城市 PGIS 发展应用的“三大巨头”。 德国的 PGIS 因起步早、发展持续稳定、建设完善最为突出，堪称典范。早在 1971 年，德国的亚琛市就建立了被公认是世界上最早的 PGIS。当时在亚琛市主要的交叉路口对市内的 12 处停车场设置了光电显示的停车诱导标志；1980 年增加到了 40 处，诱导标志采用远距离控制。20 世纪 70 年代末英国也开始采用 PGIS。在此后的一个时期内，法国、瑞士等欧洲国家也相继建立了类似系统。

1996 年 2 月,PGIS 在美国首次应用于圣保罗市商业区,曾进行为期 12 个月的运行测试。

这个耗资 120 万美元的系统管理 7 个停车库和 3 个停车场，使用了 56 块标志牌来显示停车设施的位置和车位占用情况，其中 46 块是静止引导志，10 块是 VMS。一台中央计算机控制数据采集并把 10 个停车场（库）实时、更新的停车信息通过 VMS 传送发布。驾车者根据这些信息选择了目的停车地点后，就可跟随静止引导标志抵达他所选择的停车场（库）。

1973 年日本在柏市建立了亚洲最早的 PGIS。1993 年 4 月东京建造了第一个 PGIS，位于日本东京中心的新宿区，PGIS 结构完整，诱导功能完善，应用十分成功。新宿区 1995 年以前已经采用 PGIS 管理 29 个小汽车停车场的近 6000 个按时租用的泊位，在主要商业区配备了 3 类 VMS 提供动态诱导信息服务。到 1995 年，日本已有 40 个城市引进了基于 VMS 的 PGIS，如大阪卫星城 Ibamki 的 PGIS 管理着 6 个公共停车场和 3 个私人停车场，使用两类 VMS：一类称为“图形类信息”，是在一幅可变形地图上显示各停车场的动态信息，安装在通往中心商务区（Central Business District，简称为“CBD”）的主要道路上，显示“满”、“空闲”或“关闭”的状态信息；另一类称为“文字类信息板”，是通过显示停车场的名称和使用状态来引导驾车者，若某个停车场已饱和，它就在该停车场的名称后显示“满”，若某个停车场空闲，它就显示一指向该停车场名称的绿色箭头。

与先进国家相比，我国停车诱导系统的发展应用由于基础薄弱、经验欠缺、资金不足和管理体制不尽完善等方面的原因，存在明显的差距。可喜的是，从 20 世纪 90 年代末开始，国内少数经济发达、科技实力强的大城市在各级政府大力支持下，经过专家、学者和研究人员的艰苦攻关，在较短的时间内已经取得了城市停车诱导系统领域的一定成果，为继续开发奠定了基础，相信国内停车场诱导系统在今后的发展中还有更大的上升空间。

1.3 课题来源、研究内容

1.3.1 课题来源

随着城市汽车保有量的节节攀升，城市动态交通与静态交通所承受的压力不断增长，行车难、停车难成为了大城市交通的普遍状况。为了应对上述问题，本组对国内外现有的大型停车场的运行现状进行了分析和比对，发现我国的停车诱导系统基础相对薄弱。随着近年来科技技术的不断创新，我国停车场内的自动停车系统已经取得了一定成果，并为后续的开发提供了一定的成果。

基于以上种种思考，本组拟对面向大型全自动泊车系统的优化控制方法进行理论及实际

研究。通过分析现有停车方式的不足，以及根据现有的不足构建出一个新的优化控制方法的手段，来改善现今停车场内的停车效率问题，从而对停车难等问题进行本质上的解决。

1.3.2 研究内容

本组拟首先对嘉定嘉亭荟地下停车场进行分析，通过对该停车场进行实地分析与考察。对现有的停车场停车秩序有了一个简要分析后，本文将借用嘉亭荟地下停车场中的具体参数规划一个全自动地下泊车系统，在该系统中，车辆的停放与取走都由系统进行统一调控，系统会根据停车场内的实时信息给出相对于车辆的最优车位。

在此过程中，本文将重点探究该全自动泊车系统的可行性，运用 cocos 对其进行模拟，得到最后的相应衡量参数，运用层次分析法对其进行评估，最终得出相应的结论。

1.4 研究方法及技术路线

1.4.1 研究方法

a. 文献资料法

由于全自动泊车系统尚处于探索和应用的初步阶段，笔者查阅国内外相关期刊、论文及书籍，了解全自动泊车系统的运作原理、国内外发展状况以借鉴目前较为优秀的全自动泊车系统的技术原理，并针对目前发展、应用的不足提出本文关于大型化全自动泊车系统的设计，旨在提出自动化泊车系统中路径选择的优化方案。

b. 实地调查法

为更好地了解目前的大型停车场的布局结构以及路线疏导机制，笔者去到上海市嘉定区嘉亭荟商业中心的地下停车场，实地观察和记录嘉亭荟地下停车场的车位、道路、出入口的布局，车道的通行方向限制以及路面的疏导标志。接下来，笔者以此典型停车场为参照，构建出用于建立模型和实施优化方案的停车场，并完善其车位、出入口和道路的细节设计。

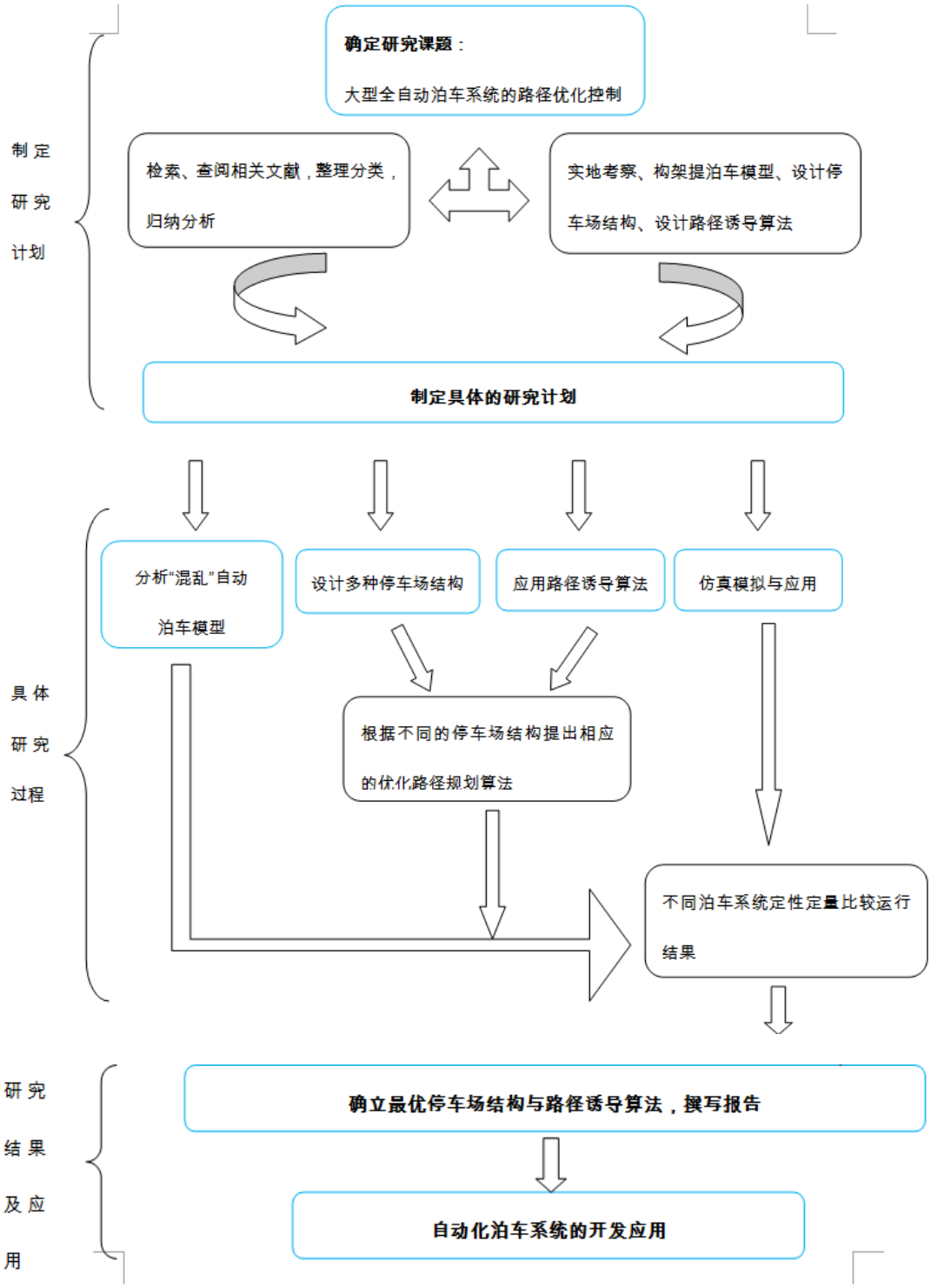
c. 分析归纳法

为了提出更加完善的全自动泊车系统的路线选择优化方案，笔者研究分析查阅的文献资料，归纳总结目前已有的路径诱导算法并合理分类：Dijkstra 算法即一种经典的最短路径算法；A*算法即通过不断的搜索逐渐逼近终点的路径来获取停车位的最短路径；蚁群算法即基于信息素浓度以及路径长度来模拟最优路径；以及改进的 Dijkstra 算法——椭圆限制算法、任意四边形限制分析算法。不同路径诱导算法计算出的不同路径各有利弊，本文借鉴以上算法，提出优化的车辆路径选择规则，在构建出的停车场模型中进行仿真模拟，以检验新提出的路径诱导算法的有效性。

1.4.2 技术路线

首先，笔者查阅国内外相关期刊、论文及书籍，了解全自动泊车系统的运作原理、国内外发展状况以及目前已有的自动化泊车路径诱导算法，进行整理分类和归纳总结。接着，小组成员去到上海市嘉定区嘉亭荟地下停车场进行实地考察，以简化的停车场结构构建现有的“混乱”全自动泊车模型，并为之后构建不同类型停车场模型打好基础 。[1]然后，笔者设计 3 种优化的停车场结构，同时根据每个停车场的车位、道路、出入口布局结构设计相应的路径诱导优化算法。紧接着，找到合适的平台搭建全自动泊车系统的仿真平台，定量分析比较各个全自动停车场模型的优劣，确定最优的大型全自动泊车系统的设计方案。最后，结合目前已有的技术和设备，开发可以投入实际应用的智能停车管理系统，使之在可触屏操作显示终端实现。

整体技术路线如下面结构图所示。



二．现有泊车系统分析——以嘉亭荟停车库为例

2.1 现有泊车系统概况

面对日益增长的机动车保有量，有限的停车位已经不能满足停车需求，另外停车场内部的管理不善和外部诱导系统不成熟也加剧了“停车难”的问题，“停车难”不可能单纯地通过增加停车位供给来缓解，有限的道路资源永远无法满足无限的车辆需求，同时，因为停车设施供给增加还将导致交通量进一步增加，进而加剧交通拥堵和空气污染的问题；而土地资源的有限性和稀缺性使之不能大量的用于建设停车设施，因为这些资源还需供给居住、商业和办公这些更为必需、土地价值也更高的城市功能发展。

路外停车占用道路划分平行或侧向停车位，花费成本相对较低，收费主要使用“咪表”系统或者人工收费。路内停车管理通过约束路边停车的行为、地点、时间和时长以确保停车位的高效利用，并且与街道、区域和交通系统的长远目标相辅相成。

在路外停车方面，为了缓解“停车难”问题，现在应用了人工动态指引、建设多层停车场和使用停车机器人等方式提高效率，同时通过收费等手段调节需求。人工动态指引即停车场的管理员工根据当时停车场的停车分布情况通过手势或摆放障碍物指引车辆寻找车位，使停车场内部交通有序，目前大部分大型停车场都通过人工指引来减少无序混乱的情况和交通冲突，减少车辆在停车场内部的无效绕行，然而人工指引难免信息滞后，也会有不服从指引的驾驶员，而指引员长期暴露在汽车尾气中。建设多层停车场提高了单位土地的利用率，但是地下停车场建设层数受限制，而多层停车场中的高层停车位通常使用率较低，车辆行驶上高层停车场时油耗较多并不环保。停车机器人是一种新兴的停车工具，它允许停车场向车位密度更高停车层数越多的方向发展，用户把车停在特定地点后，停车机器人可以把车辆移到某一车位。目前多个城市都出现了智能化立体停车场，利用有限的土地提供大量的停车位，但是由于将汽车停到指定位置时对车辆对准车库的要求太高倒出倒入麻烦以及取车等待时间太长，部分智能化立体停车场使用率并不高。[2]

2.2 现有泊车系统模型建立

本文选取了上海嘉定安亭镇嘉亭荟地下停车场为研究对象，研究基于自动驾驶技术的适

用于该停车场的停车组织。该停车场位于曹安公路和墨玉南路的交叉口附近、安亭地铁站的南侧；主要为嘉亭荟城市生活广场及周边的消费者和员工服务；停车场内部以颜色分区；在泊位周转率高（高峰）时，停车场内有保安指挥并封闭某些通道以保持秩序。该停车场有两层，只有小型车停车位，B2 层周转率大大低于 B1 层，本文以其 B1 层为例，B1 层有 385 个停车位，其中包括 2 个伤残车位、8 个 evcard 专用车位和 6 个微型车车位；有三个出入口，收费 6 元/小时。其平面示意图草图如下：

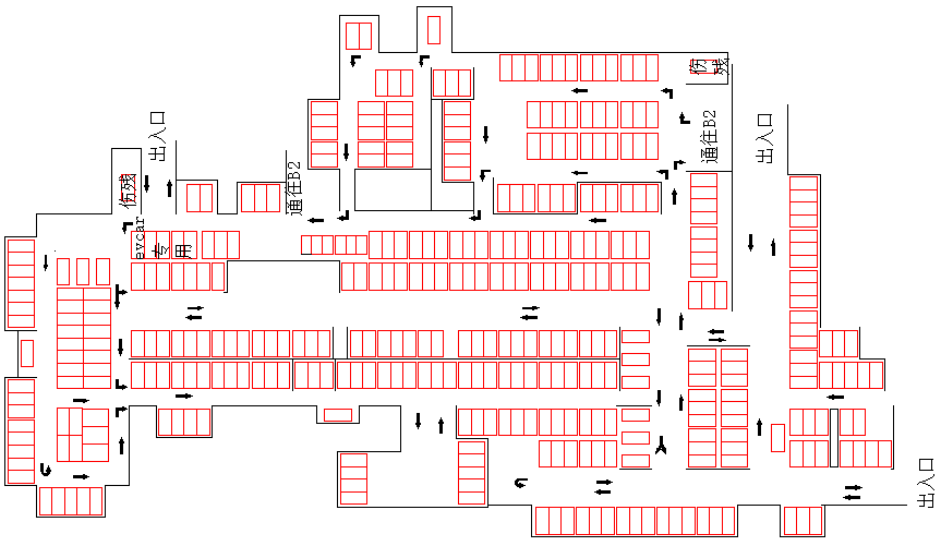


图 2-1 嘉亭荟地下停车场平面示意草图

2.2.1 现有泊车系统参数确定

1）车位数、出入口数

本文对上图进行了修改，简化停车场内部的交通组织为只有一层的地下停车场，简化后停车场平面示意图草图如下：

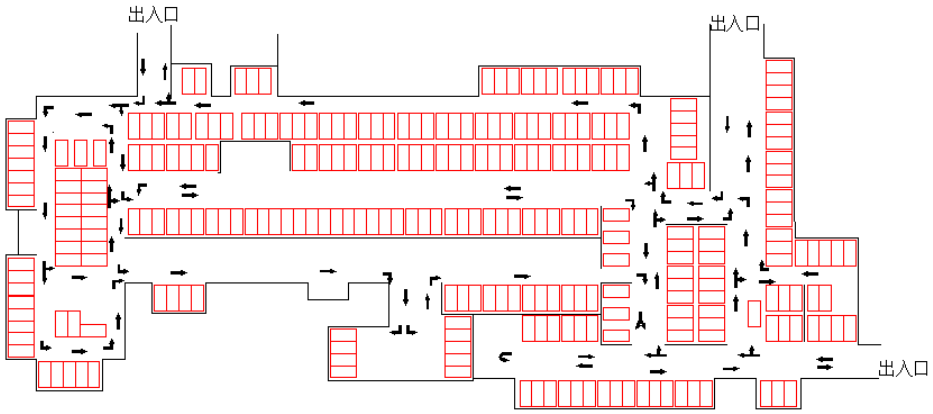


图 2-2 简化后的停车场平面示意图草图

该停车场有 282 个小型车车位，3 个出入口。

2）入口道平均驶入率

根据下表和停车场的车位数、出入口数确定每个入口道平均驶入率。

停车场规模/泊位数	入口道平均驶入率/(veh·h^{-1})		入口道长度/m	
	设 2 个入口道	设 3 个入口道	设 2 个入口道	设 3 个入口道
250	83	55	76	26
300	100	67	156	46
350	117	78	—	56
400	134	89	—	96
450	150	100	—	156
500	168	112	—	—

表 2-1 停车场出入口道参数

每个入口道平均驶入率=（67-55）*（282-250）/（300-250）+55 =63（veh/h）

3）高峰小时驶出率

根据下表和停车场的车位数确定高峰小时驶出率。

停车场 规模/泊位	高峰小时 驶出率/(veh·h^{-1})	3 min 内驶出 车辆/veh
250	83.75	8
300	100.50	9
350	117.25	11

表 2-2 停车位数及高峰小时驶出率

高峰小时驶出率=（100.50-83.75）*（282-250）/（300-250）+83.75 =94.50（veh/h）

2.2.2 现有泊车系统算法选取

现有泊车系统以“混乱”状态运行作为模型的基础——车辆从不同进口道进入后选择行驶路线上第一个空停车位停车。车辆行驶过程中选择车位的算法与车辆确定车位后倒车入库的算法实现如下所示：

```
if (cardSprite[j][i]->direction[X] == 1&& cardSprite[j-X][i]==0)//判断目前车道行驶方向，获取行驶方向前方车道是否为空可供行驶
{
    this->setPosition = (84 + 11 * i, 306 - 11 * (j - 1));      //将车辆向前移动至前方空车道
    cardSprite[j][i]->OnParking= 0;                             //将车辆移动前车道位置设为空
    cardSprite[j - X][i]->OnParking = 1;                        //将车道目前所在车道位置设为有车
}
```

图 2-3 选择车位算法实现

```
if (cardSprite[j+X][i]->type == 'p'&&cardSprite[j + X][i]->CarNum == -1)//判断周围是否有空车位
{
    this->setPosition(84 + 11 * i, 306 - 11 * (j + 1));             //将车辆驶入空车位
    cardSprite[j + X][i]->CarNum = CarNum;                          //将车位表示为有车
    cardSprite[j][i]->OnParking = 0;                                //将车辆之前所在车道设为空
}
```

图 2-4 倒车入库算法实现

车辆驶出时则往离停车位最近的出口的最短路径行驶，最短路径算法使用 Dijkstra 算法，得出各个车位的离开停车场的最短路径，车辆驶出时检索所在车位的最短路径，沿该路径驶出。获取车辆最短路径出库行驶的算法实现如下所示：

```
if (time1 > car[i]->time_in&&car[i]->OnRuning==1&&!car[i]->type) {//判断车辆是否在出库过程中
    car[i]->removeFromParent();
    S next = car[i]->route_in.front();                          //从表中获取该车出库路径
    car[i]->x = next.x;                                         //将车辆按照路径进行移动
    car[i]->y = next.y;
    car[i]->route_in.pop();
    car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
}
```

图 2-5 获取最短路径算法实现

发生冲突时，按先到先通行的原则，车辆到达冲突点后判断前方是否有车辆通行，如果前方一直有车辆通行，则该车辆等待前方队列车辆全部通行后再通过。

2.3 现有泊车系统模型运行结果分析

根据上述算法，本组得出了最终“混乱”模型的运行结果，在该运行结果中我们可以发现，在各个交叉口处都出现了较为严重的拥堵，排队长度蔓延到了下游的交叉口处，在第 162 个时间单位处整个系统基本处在瘫痪状态。

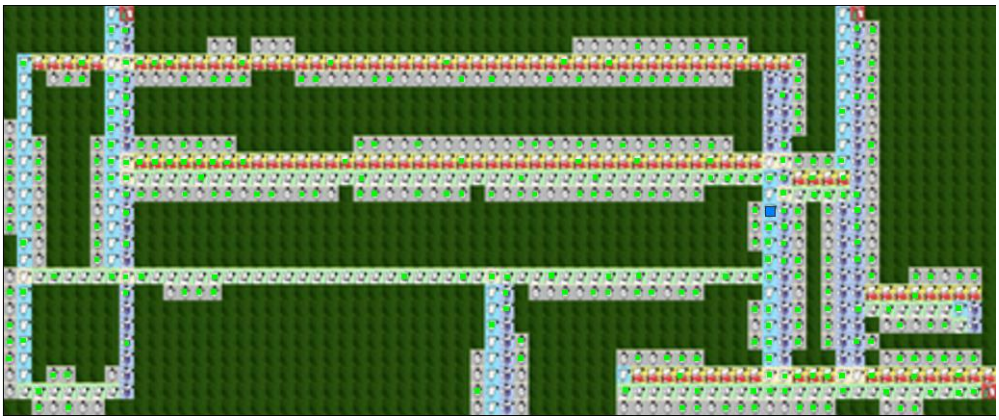


图 2-6 现有泊车系统模型分析

2.4 小结

本章节提出了一个“混乱”泊车系统，在该泊车系统中，车辆从不同进口道进入后选择行驶路线上第一个空停车位停车，当车辆驶出时，则往距离停车位最近的出口的最短路径行驶。其中，最短路径的算法使用了迪杰斯特拉算法。当发生冲突时，本“混乱”模型按照先到先通行的原则，判断前方是否有车辆通行后选择对车辆给予等待或放行的指令。

本章节模拟的“混乱”泊车系统是一个常态化下的泊车系统，相比较后文中提到的三种模型下的全自动泊车系统而言，该“混乱”模型缺乏对于进出车辆更为有效的规划原理。从

仿真结果可以看出，在“混乱”模型的情况下，车辆的进出效率极其低下，一段时间后就会在冲突点形成拥堵，而由于排队的队伍越积越长，最后过半的路径基本上都处在瘫痪状态，难以通行。

三．全自动泊车系统设计

3.1 全自动泊车系统概况

典型的自动泊车系统是由感知模块、路径规划模块、路径跟随模块以及人机接口模块共同构成的，路径规划模块会在感知模块探测出库位后根据枯萎大小和位置来规划处可行的泊车路径，随后路径将会跟随模块控制车辆沿规划路径驶入库位。其中，路径跟随模块在车速较低的情况下对车辆行驶路径进行精准控制是自动泊车系统中的关键技术。

Kanayama 等提出了一种非线性车辆运动控制器，并利用李雅普诺夫直接法证明了控制器的稳定性，但是在控制器中侧向控制和纵向控制具有耦合关系，不适用于车速信号具有强烈噪声的情况。Muller 提出了一种基于微分平坦理论的侧向运动控制器，但未涉及纵向运功控制。Chen 等设计了模糊 PID 侧向运动控制器和模糊 PI 纵向运动控制器，然后模糊逻辑的整定是极其困难的。宋金泽提出了一种参数自整定位分阶 PID 侧向运动控制器，其运算量是相对较大的。

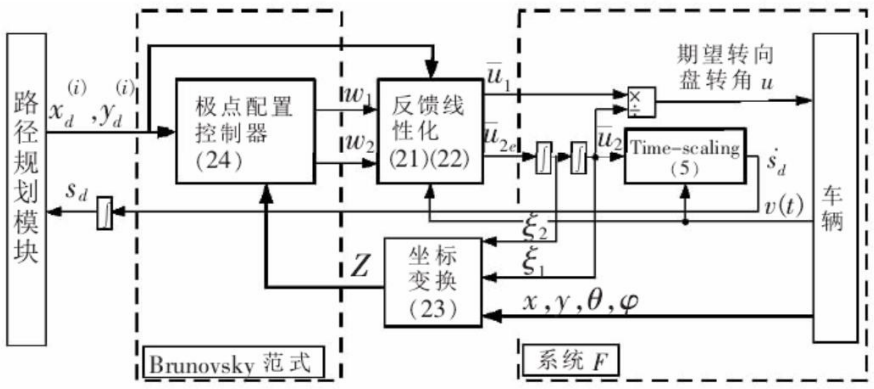


图 3-1 横向运动控制器框图

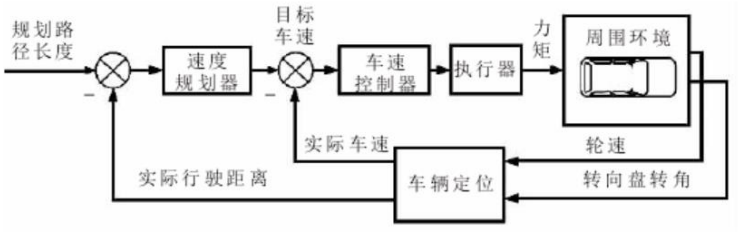


图 3-2 纵向运动控制器框图

目前各种量产车使用的泊车辅助系统大都需要有驾驶员在内操作或者控制转向及制动，大众汽车公司在德国展示的新型泊车辅助系统完全不需要驾驶员在内进行操控，而是通过控制一系列的装置使汽车能够完成泊车动作。

图 3-3 大众新型泊车辅助系统

与以上几种全自动泊车系统不同，本文所提出的面向大型全自动泊车系统的优化控制方法并非着重于关注车辆的横向及纵向运动控制，也并非全部依赖于汽车自行对外界环境进行感知从而选择路径及车位。本文所提出的泊车系统重点关注于对一个平面的停车场内的进出车辆进行一个统一的后台调控，在对停车场内的实时路况做出一个即刻的预判后给为新进入的车辆规划一条最优路径。

3.2 全自动泊车系统人机交互界面

停车作为交通系统的一个必要环节，它的秩序和效率直接影响着整个交通系统的功能发挥。智能停车场人机交互是指停车场管理中心系统通过区域停车场车位利用情况的分析、对驾驶人员进行路径诱导，使车辆能够尽快到达停车场。

本文提出的人机交互界面可以即时动态地为进入指定区域的车辆提供停车场的泊位、空

满资讯、一般路径或最优路径等与停车相关的向导资讯指示。[3]

本文以嘉亭荟地下停车场为例，建立了一个智能停车系统，其各个人机交互界面如下。首先用户需要根据其自身需求确定停车还是取车，如果该车辆有停车需求，那么在点击左侧界面后会出现一个新的界面，给出系统得到的车牌号信息以及车辆入库时间，车主确认后输入其所需的出库口以及出库时间，智能停车系统就会为其安排相应的最优车位。

如果车主有取车需求，则需点击右侧按钮，跳入取车界面，此时系统会根据车主的停车时长算得应付费用，随后界面则会出现“请刷 IC 卡的提示”，车主通过刷 IC 卡付清停车费用并显示车辆根据算法将到达的用户取车位置。[4]

图 3-4 嘉亭荟地下停车场智能系统

图 3-5 停放选取界面

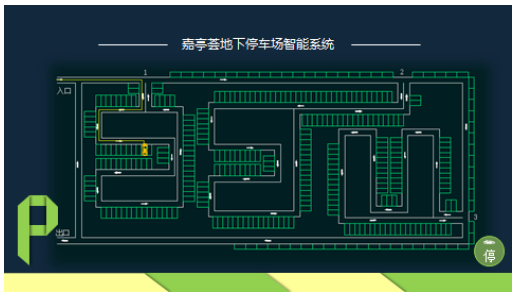


图 3-6 分配车位

图 3-7 刷卡界面

图 3-8 输入用户偏好

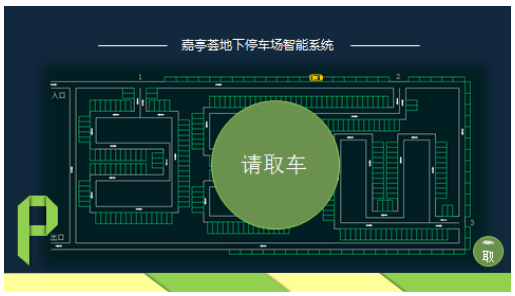


图 3-9 取车位置显示

3.3 全自动泊车系统模型建立

本文从停车场结构设计着手，在出入口数量、停车位数量相同的前提下，构建三种不同结构的全自动泊车系统：以嘉亭荟地下停车场为参考的多路泊车模型、外部设有环形取车用车位的内部多路外部环路泊车模型、外部设有环路且内部只有一条单行车道的内部单路外部环路泊车模型。

分别根据三种全自动泊车系统的内部布局，确定车辆入库停车位的选取、多辆车发生冲突时的行驶规则以及车辆出库的取车规则。

在停车场结构和车辆行驶规则确定的情况下，使用 cocos 系统开发软件结合 C++语言编程搭建全自动泊车系统仿真平台，通过定性观察与定量计算分析各个泊车系统结构的优劣，确定最优设计方案。

3.3.1 多路泊车模型

A. 泊车系统结构

多路全自动泊车系统结构参照嘉亭荟地下停车场结构，该地下泊车系统共有 3 个出口 3 个入口、282 个停车位、单行道双行道并存，泊车系统出入口、车位、道路以及行驶方向规定如下方 CAD 图所示。

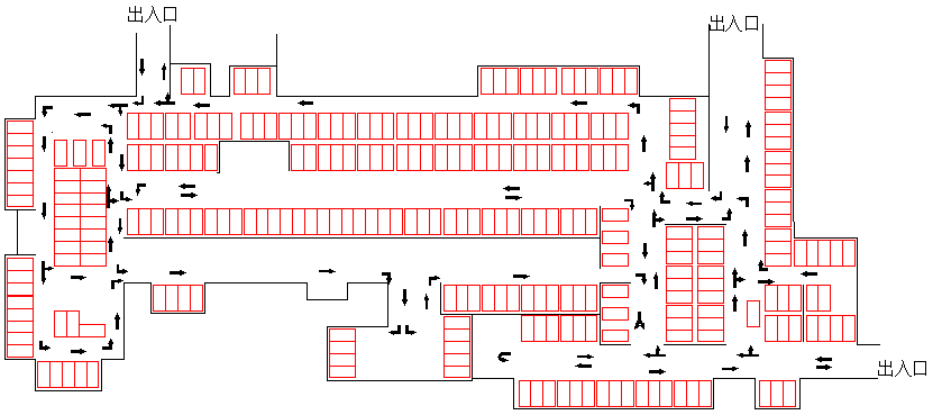


图 3-10 多路泊车结构 CAD 图

利用 Tiled 将 CAD 转化成 100*100 的网格地图——Tiled 是基于 Java/Qt 的开源区块地图编辑器，支持地图分层，并且可以为每个层次添加各种属性。除此之外，Tiled 还支持自定义对象图层，用户可以在该层上添加各种数据，这对地图的事件触发设置提供了较好的支持。

停车场构造图中使用灰色格子表示停车位，使用黄色方格表示向左的车道，使用浅绿色方格表示向右的车道，使用紫色方格表示向上的车道，使用浅蓝色方格表示向下的车道，使用红色方格表示出入口。

图 3-11 Tiled 表示的多路泊车系统结构图

B. 入库停车规则

当车辆从某一入口进入泊车系统时，驾驶员需要在系统中输入预计取车时间，系统根据“车辆依次均匀停在最靠近 3 个出口的车位”原则为车辆选定车位，车位选定后系统计算出从入口至停车位的最短路径，车辆根据系统给出的路径行驶至指定停车位。

“车辆依次均匀停在最靠近 3 个出口的车位”原则即系统默认车辆优先停在距出口最近

的停车位，为了保证三个出口附近车辆数均匀，系统安排驶入车辆依次停在 3 个出口附近距出口最近的停车位。

C．冲突规则

多辆车在路口形成冲突时，系统判断排队车辆中预计取车时间最早的一辆车所在的位置，优先让这辆车以及排在这辆车前面的所有车辆通过；下一次再判断剩余排队车辆中取车时间最早的一辆车所在的位置，优先让这辆车以及排在这辆车前面的所有车辆通过；以此类推。

D．出库取车规则

出库时车主到达停车场后按下取车按钮，车辆即开出车位出库行驶；系统通知车主车辆在几号出口出库，车主在该出口处取车通道等待取车即可。此结构由于车辆开出车库的顺序不定，多个车主在出口取车通道排队等待时需时刻注意自己的车是否已开出车库并及时取走出库车辆，否则容易在出口取车处形成拥堵和混乱。

3.3.2 内部多路外部环路泊车模型

A．泊车系统结构

内部多路外部环路全自动泊车系统结构在前一个泊车模型的基础上增加外部地上停车位，用于停放按规定取车时间开出地下车库的车辆，车辆停在地上停车位后，系统向车主发出车辆所在地上停车位的编号及所属区域，车主到该车位取车。

该地下泊车系统共有 3 个出口 3 个入口、282 个停车位、单行道双行道并存，泊车系统出入口、车位、道路以及行驶方向规定如下方 CAD 图所示。

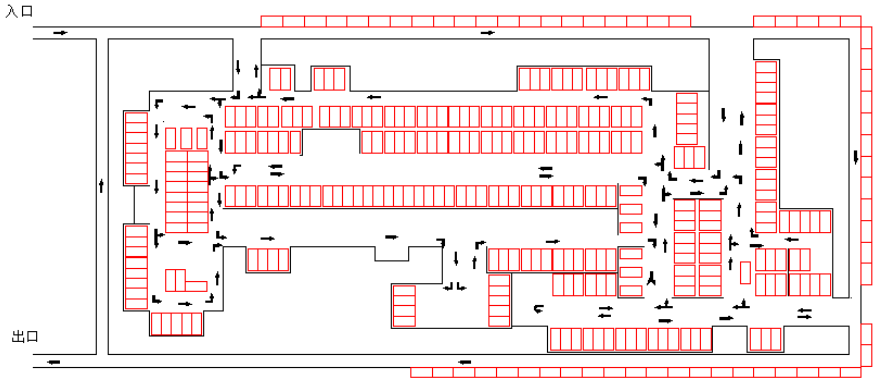


图 3-12 内部多路外部环路泊车系统

利用 Tiled 将 CAD 转化成 100*100 的网格地图，停车场构造图中使用灰色格子表示停车

位，使用黄色方格表示向左的车道，使用浅绿色方格表示向右的车道，使用紫色方格表示向上的车道，使用浅蓝色方格表示向下的车道，使用红色方格表示出入口。

图 3-13 Tiled 表示的内部多路外部环路泊车系统

B．入库停车规则

车主在地上入口存车时，驾驶员需要在系统中输入预计取车时间，系统根据内部停车情况，依据“车辆依次均匀停在最靠近 3 个出口的车位”原则判断车辆应停在哪个出口附近的哪个停车位；确定停车位位置后，系统依次计算三个地下入口到该停车位的最短路径，找出三条路径中最短的一条，以此确定车辆应从哪个入口进入；车辆在地上外围环路上绕行至该入口即可按照系统规定好的最短路径和停车位停车。

C．冲突规则

多辆车在路口形成冲突时，系统判断排队车辆中预计取车时间最早的一辆车所在的位置，优先让这辆车以及排在这辆车前面的所有车辆通过；下一次再判断剩余排队车辆中取车时间最早的一辆车所在的位置，优先让这辆车以及排在这辆车前面的所有车辆通过；以此类推。

D．出库取车规则

车主到达停车场后按下取车按钮，这时车主的车辆将会从地下车库中驶入地上外围的停车位；车辆到达外围停车位后向车主发出信号通知车主到指定地上停车位取车。

若有车主未按时取车，车辆在地上停车位停留时间超过 15min 后，车辆将会被系统判定为取车违规车辆，违规车辆自动驶出至外部环路，重新作为驶入车辆再次进入泊车系统。

3.3.3 内部单路外部环路泊车模型

A．泊车系统结构

内部单路外部环路全自动泊车系统结构中根据 3 个出口 3 个入口分为三个片区，每个片区中结构相同且只有一条单向车道供侧向停车车辆行驶，单向车道的设置保证泊车系统中不会出现路口冲突，但内部车辆的停驶次数相比前两个模型大很多。外部环路的设计形上一个内部多路外部环路泊车系统模型。

该地下泊车系统共有 3 个出口 3 个入口 3 个片区、282 个停车位，每个片区结构相同、各包含 1 个出口 1 个入口、分别有 88、100、94 个停车位、内部只有一条单向行驶道路，泊车系统出入口、车位、道路以及行驶方向规定如下方 CAD 图所示。

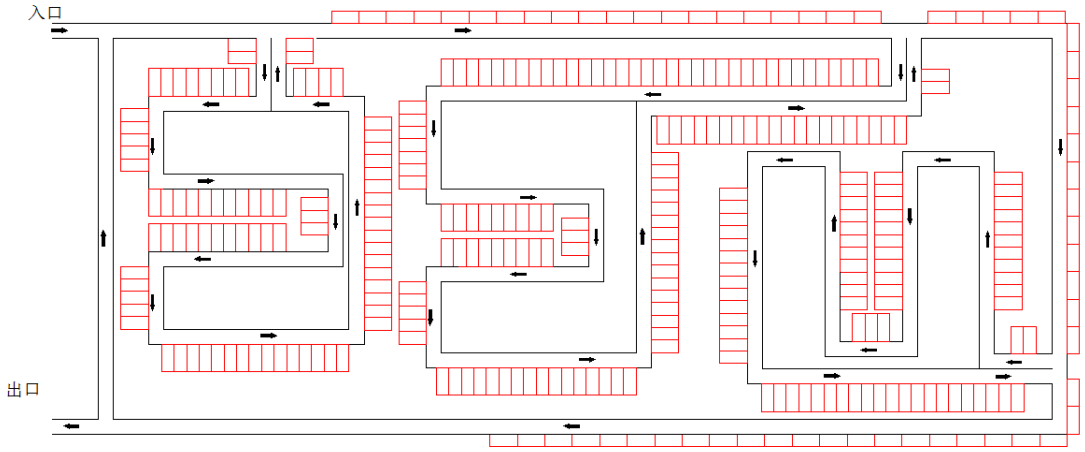


图 3-14 内部单路外部环路泊车模型

利用 Tiled 将 CAD 转化成 100*100 的网格地图，停车场构造图中使用灰色格子表示停车位，使用黄色方格表示向左的车道，使用浅绿色方格表示向右的车道，使用紫色方格表示向上的车道，使用浅蓝色方格表示向下的车道，使用红色方格表示出入口。

图 3-15 Tiled 表述的内部单路外部环路泊车模型

B. 入库停车规则

当车辆决定停放后，系统会得到一个相应的停车时间，而这在系统中也正是该车辆的优

先级，在车辆驶入停车位的途中，由于系统的是三圈单路，在行驶的过程中，后方车辆需要排队等待至自己的停车位进行停车。

C. 出库取车规则

当车辆决定驶离时，系统同样会根据其驶离时间得到一个相应的优先级，车辆前方的路径上会有着排队等待驶离的车辆,此时需要对这一队车辆以及待驶离车辆的优先级进行一次比较，当发现待驶离车辆的优先级在前方两排队车辆中间时，排队车辆需要从该处进行一次打断，后方的车辆停驻，前方的车辆自队头开始每辆车向前行驶一个车身长度，直到在打断出留出一个车位，使得待驶离车辆可以驶入从而跟随排队队伍驶离停车场为止。

四．全自动泊车系统模型算法及仿真

4.1 元胞自动机

元胞自动机是一时间和空间都离散的动力系统。散布在规则格网 (Lattice Grid)中的每一元胞(Cell)取有限的离散状态,遵循同样的作用规则,依据确定的局部规则作同步更新。大量元胞通过简单的相互作用而构成精态系统的演化。本文引入元胞自动机建立模型，深入分析当一定车流进入停车场后各个车辆的行为，构造不同的仿真演化规则，以更真实地反映不同构造不同结构的停车场的运行特征。最后通过计算机模拟，对三种模型的停车场中车辆运行特性进行分析，以总结出适应嘉亭荟地下停车场的模型。

4.2 Cocos 系统

Cocos 是由触控科技推出的游戏开发一站式解决方案,将 Cocos 家族中的所有开发框架、工具和编辑器整合到了一起，包含了从新建立项、游戏制作、到打包上线的全套流程。开发者可以通过 Cocos 快速生成代码、编辑资源和动画，最终输出适合于多个平台的游戏产品。

本文利用 Cocos 平台为停车构造图上不同类型的方格赋不同的 ID，给不同行驶方向的道路赋不同的 ID，并以此为基础进行仿真模拟。

4.3 优先队列(priority queue)

普通的队列是一种先进先出的数据结构，元素在队列尾追加，而从队列头删除。在优先队列中，元素被赋予优先级。当访问元素时，具有最高优先级的元素最先删除。优先队列具有最高级先出 (largest-in，first-out)的行为特征。

本文的内部单路外部环路模型依照优先队列，按照车辆预计离开停车场的先后顺序排列，预计越早离开停车场的车辆停的位置离出口越近，当一辆车进入某一分区之后，系统将对比该车辆和该分区内现有车辆的预计取车时间，比该车辆早离开的车辆将向出口移动，该车辆将停在腾出来的车位里。[5]

4.4 实现代码

4.4.1 入库规则设置

A 选定车位

```
int park_in = car[i]->park_in;
for (int j = 0; j < district[park_in].size(); ++j)
    if (district[park_in][j].flag) {
        car_end = j;
        break;
    }
```

图 4-1 入库规则算法实现

B 入库路径

1）计算从指定入口到停车位的最短路径（多路泊车模型）

```
    while (!t.empty()) {
        S m = t.front();
        M mm = M(m.x, m.y);
        district[k].push_back(mm);
        t.pop();
        if (cardSprite[m.x][m.y]->type=='p')cardSprite[m.x][m.y]->route_out = t;
    }
}
```

图 4-2 入库最短路径选取

2）对比三个入口到该停车位的路径中的最短路径（内部多路外部环路模型）

```
(car[district[park_in][j].CarNum]->time_out < car[i]->time_out) {
 int x = district[park_in][j].x;
 int y = district[park_in][j].y;
 car[i]->route_in = cardSprite[x][y]->route_in;
 car[i]->route_out = cardSprite[x][y]->route_out;
 district[park_in][j].CarNum = car[i]->CarNum;
 park = j;//
```

图 4-3 对比三个路口最短路径

3）计算车辆在环路上绕行至指定停车位对应入口到达停车位的路径（内部单路外部环路模型）

```
S s = car[CarNum]->route_out.front();
car[CarNum]->route_out.pop();
car[CarNum]->x = s.x;
car[CarNum]->y = s.y;
car[CarNum]->setPosition(Vec2(79 + 11 * s.x, 157 + 11 * s.y));
```

图 4-4 环路绕行至停车位路径

4.4.2 出库规则设置

1. 发出指令后车辆驶出（多路泊车模型和内部多路外部环路模型）

```
if (car[i]->OnRuning&&car[i]->type) {
   car[i]->removeFromParent();
   S next = car[i]->route_out.front();
   car[i]->x = next.x;
   car[i]->y = next.y;
   car[i]->route_out.pop();
   car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
```

图 4-5 发出指令后车辆驶出

2. 内部单路外部环路模型中出库路径为最短路径，使用的算法和上文计算入库最短路径相同，不再赘述。

4.4.3 冲突规则设置（多路泊车模型和内部多路外部环路模型）

1. 判断冲突车辆中预计取车时间最早的一辆

```
if (time1 > car[i]->time_in&&car[i]->OnRuning==1&&!car[i]->type)
```

图 4-6 判断冲突车辆

2. 使该车辆优先通过冲突点

```
car[i]->removeFromParent();
S next = car[i]->route_in.front();
car[i]->x = next.x;
car[i]->y = next.y;
car[i]->route_in.pop();
car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
```

图 4-7 车辆优先通过冲突点

4.4.4 环路停车等待机制

1.根据等待时间判别车辆违规（内部单路外部环路模型和内部多路外部环路模型）

```
if (time1 > 15&&cacar[i]->OnRuning==1&&!car[i]->type)
```

图 4-8 环路停车等待

2.违规车辆再次驶入停车场按照该模型的入库路径选择规则（选定车位离入口处最近），不再赘述。

4.5 运行结果

4.5.1 多路泊车模型运行结果

多路泊车模型最终得出的单位化用户取车时间为 124.311，停车场效率系数为 0.432，车辆能耗为 21.026 次。该模型下的车辆单位能耗最低，然而其停车场效率也相对最低，单位化用户取车时间最长。

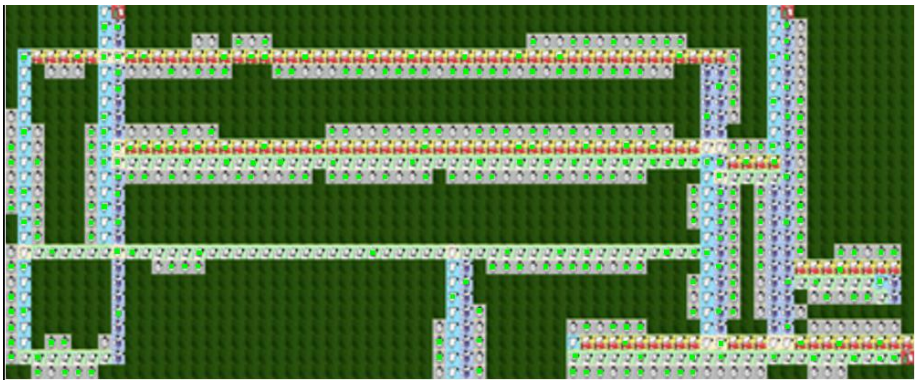


图 4-9 多路泊车模型运行结果

4.5.2 内部多路外部环路泊车模型运行结果

内部多路外部环路的泊车模型得到的单位化用户取车时间为 90.963，停车场效率系数为 0.582，车辆能耗为 14.745 次。三种模型中，该模型的单位化用户取车时间，停车场效率系数以及车辆能耗都处于中位数。

图 4-10 内部多路外部环路泊车模型运行结果

4.5.3 内部单路外部环路泊车模型运行结果

内部单路外部环路泊车模型得到的单位化用户取车时间为 32.578，停车场效率系数为 0.890，车辆能耗为 49.612 次.在三个模型中,该模型的车辆能耗最高，然而其停车场效率最高，单位化用户取车时间最短。

图 4-11 内部单路外部环路泊车模型运行结果

五．多结构全自动泊车系统综合评价

5.1 基于层次分析法的自动泊车系统评价模型

5.1.1 模型分析

对自动泊车系统进行评价，既要考虑到用户的停取车便捷性和取车等待时间，又要考虑停车场运营的效率，同时要兼顾车辆能耗情况。通过资料查找与分析，各个层面的运营状况可以用若干指标的数值综合表示，由于各个指标间相互影响，又共同作用于最终的目标，我们选用层次分析法，建立自动泊车系统评价指标体系，分别进行各种结构停车场指标的综合比较。

层次分析法（AHP）是一种定性与定量相结合的多目标、多准则的决策分析方法。对各种类型问题，尤其是复杂问题的决策分析，具有较广泛的实用性，是目前国内外确定指标体系最常用的方法。鉴于在自动泊车系统的指标体系中，许多指标相互关联，甚至相互包含，因而在评判时所起的作用不一。故本模型采用了层次分析法对各指标进行综合评价。我们从系统最优的目标出发，将其衍生到存取车用户、停车场与自动行驶车辆 3 个层面分别进行分析，进一步划分了用户平均等待时间、停车场效率系数、车辆能耗 3 个具体指标，对上述指标进行了赋值，对 3 种停车场结构模型与行驶规则比较与评价。

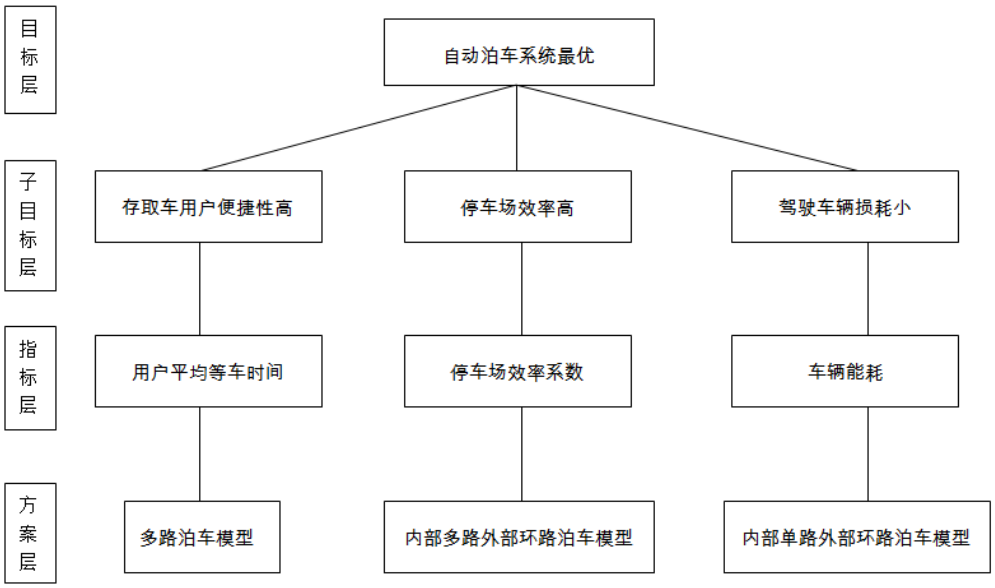


图 5-1 层次分析法结构建立

5.1.2 数据处理

1. 数据预处理

研究三种不同结构的自动停泊系统，每类模型进行多指标综合评价。不同类型停车场系统指标统计值归纳如下表：

	多路泊车模型	内部多路外部环路泊车模型	内部单路外部环路泊车模型
用户平均取车时间（单位化时间）	124.311	90.963	32.578
停车场效率系数	0.432	0.582	0.890
车辆能耗（次）	21.026	14.745	49.612

表 5-1 不同系统指标统计值

2. 数据标准化处理

在进行道路交通评价之前，为尽可能反映实际情况，排除由于各项指标的单位不同以及数量级间的悬殊差别所带来的影响，需要对指标进行无量钢化处理。由于评价体系中指标均为定量指标，并且评分的比较在两两间进行，所以这里提出用比例法进行数据标准化处理。当该指标与系统总目标正相关时，按该方案指标值所占比例作为打分；当该指标与系统总目标负相关时，用总数 1 减去该方案指标值所占比例作为打分结果。最终无量钢化结果符合越大越优原则：

$$r_i = \begin{cases} \dfrac{x_i}{x_i + y_i}, & \text{第 i 项指标} U_i \text{与目标层成正相关} \\ 1 - \dfrac{y_i}{x_i + y_i}, & \text{第 i 项指标} U_i \text{与目标层成负相关} \end{cases}$$

式 5-1 无量纲化结果

式中，x_i，y_i分别为评价指标U_i的两个值。

5.1.3 模型建立

运用上述各指标数据与统一定义的数据标准化规则，计算得到不同指标的无量纲值标定，整理如表 所示。依据各个层级指标对目标指标的影响系数查阅相关文献后，我们采用构建判断矩阵的方法，对某层次中属于统一体系的各个因素进行成对比较，采用指定数值，将评价结果归一化处理作为各个指标所占权重。两两比较的结果值a_{ij}一般取 1/9、1/8、…、1/2、1、2…8、9 以此表示从不重要到重要的过渡，得到用户平均等待时间、停车场效率系数、车辆能耗的权重如表 所示，最终综合得到各项指标权重如下表所示：

特征指标	多路泊车模型	内部多路外部环路泊车模型	内部单路外部环路泊车模型
用户平均等车时间（s）	0.498	0.633	0.869
停车场效率系数	0.227	0.306	0.467
车辆能耗（次）	0.754	0.827	0.419

表 5-2 三种模型各项特征指标值

准则	用户平均等车时间	停车场效率系数	车辆能耗	求和（V）	归一化（W）
用户平均等车时间（s）	1	5	5/2	8.5	0.6250
停车场效率系数	1/5	1	1/2	1.7	0.1250
车辆能耗（次）	2/5	2	1	3.4	0.2500
求和				13.6	1.0000

表 5-3 各项指标值求和及归一化结果

5.1.4 模型运用

计算三种自动泊车系统评价得分。运用以上 3 种停车场模型，结合建模求得的三种类型停车系统的指标的数值，通过定义评分规则以及应用判断矩阵定义指标权重，用 Matlab 计算得出三种类型泊车系统的评价得分如下表所示。

泊车类型	多路泊车模型				内部多路外部环路泊车模型				内部单路外部环路泊车模型			
目标层	用户	停车场	车辆	综合	用户	停车场	车辆	综合	用户	停车场	车辆	综合
	0.311	0.023	0.188	0.522	0.396	0.038	0.207	0.641	0.543	0.058	0.105	0.706

表 5-4 三种系统得分情况

5.2 最优泊车模型推选与未来展望

从用户取车等待时间来看，内部单路外部环路泊车模型大大优于另外两个结构的停车场，其原因在于车辆已根据取车时间在车库内部自动排好顺序，出库时按车位顺序依次前行不会产生冲突和延误；从停车场利用效率来看，仍然是内部单路外部环路泊车模型较优，因为这种结构的停车场内车辆除了入库的过程在道路上行驶其余状态都要占用停车位，在停车位间实现移动，因此车库内部车位使用率高于另外两个停车位仅用于停车的泊车系统；从车辆能耗来看，内部多路外部环路泊车模型最优，因为外围环路有效地减少了取车时产生的冲突并且方便自动驾驶车辆选择最优的入口方案进入车库，进而减少了因停驶造成的能耗损失；而内部单路外部环路泊车模型在车辆能耗方面表现最差，因为车辆在车库内部时除了在出库入库行驶，实时变换停车位、在停车位间的移动大大增加了车辆行驶的起停次数，从而增加了车辆的能耗。

综合而言，从上述层次分析法评价结果中可以看出，综合考虑用户、停车场和车辆三方面因素，内部单路外部环路泊车模型的运行效果最优。虽然此泊车系统有较高的车辆能耗，但它在减少用户取车等待时间和增加停车场利用效率方面远大于另外两个模型，因此优先推选内部单路外部环路泊车模型。

在不久的将来，如果新能源汽车特别是太阳能汽车的使用能得到普及和推广，那么车辆能耗的问题将会得到很好的解决，这时内部单路外部环路泊车模型的弊端将会大大缩小，其优势会在各种泊车系统中更加凸显。若内部单路外部环路泊车模型能够得到推广和应用，用户存取车体验会更佳、停车场效率将会高，为自动驾驶泊车系统提供了很好的设计方案。

结论

随着我国国民经济的发展以及生活水平的提高，停车难已经成为了当今社会一个亟待解决的问题。由于城市汽车保有量节节攀升的同时，停车场内的车位越来越紧张，再加之国内现今停车场内部的停车效率相对较低，车辆在停车场内花费了大量的时间在让行以及寻找车位上，因此本文认为，一个有着系统统一调控的全自动泊车系统是有强烈的市场需求的。

本文提出了一种面向大型全自动泊车系统的优化控制方法，希望能通过本方法的实现缓解现今城市面临的停车效率地下问题。本文的研究成果主要有以下几个方面。

（1）“混乱”自动停车系统的问题分析。本文拟建了一个“混乱”自动停车系统，在该系统中，车辆的停放与驶离都由系统调控，但并未对其设立优化规则。停入的车辆选取离自己距离最近的空车位进行停放，驶离的车辆选取离自己最近的出口进行驶离。由于没有系统对路况进行一个统一的处理及预判，最后仿真结果显示该泊车系统内部出现了大面积的堵塞。

（2）分析优化系统的指标。根据“混乱”模型的构建，本文将优化的重点放在设置入库停车规则，冲突规则以及出库取车规则上，通过对上述三种规则进行探究完成整个自动控制系统的优化。

（3）人机交互界面的设计。为了使得该控制系统的运行理念能够更好地服务人群，本文设计了地下停车场内的人机交互界面，对每一个界面都进行了详尽的描述及设计，突出系统优化效果的同时保障其可行性。

（4）三种泊车模型的建立。为了使得优化规则具有普适性，本文构建了三种多路泊车模型，分别为多路泊车模型，内部多路外部环路泊车模型以及内部单路外部环路泊车模型，并分别对其三种规则进行了约束。在建立规则时，本文使用了优先队列的思想，将元素赋予相应的优先级，使其具有最高级先出的行为特征。

（5）建立基于 Cocos 的模型仿真环境。本文利用 Cosos 平台为停车构造图上不同类型的方格赋不同的 ID，给不同行驶方向的道路赋不同的 ID，并以此为基础进行仿真模拟。

（6）基于层次分析法的自动泊车系统评价模型。在对仿真软件运行结果进行评价时，要综合考虑到用户取停车的便捷性以及停车场的运营效率，本文在查找相关资料之后将各个层面的运营指标用若干指标的数值综合表示，得出内部单路外部环路模型在三种模型中相对较优的结论。

本文主要是对嘉亭荟地下停车场这一既有停车场进行了研究，并且由于本文的研究内容仅限于一个平面，在对面向大型全自动泊车系统的优化控制方法这一问题的研究还可以再未来进行相应的拓展，在设计了人机交互界面之余，配置在相应终端的 APP 也应应运而生，因此，本文认为，在本文的研究成果基础之上，可以开展后续的工作以便对全自动地下停车控制系统进行进一步的完善：

（1）将成果应用于更多规格以及所处位置的停车场，对其建立相应的仿真环境探究本文的研究结果是否对于停车场有着横向的普适意义，随后对算法进行完善和补充。

（2）将现有的平面停车场模型拓展为多层停车场模型，在研究同一平面内车辆行驶之余还应考虑到将结论应用于多层停车楼时需要纳入考虑范围的新问题，以及需要增加的算法及约束。

（3）对于相应的 APP 进行研发，驾驶者可以通过手机实现与泊车系统的交互，输入系统取停车时间、获取车位信息、了解停车场结构和实时车况，使得该优化后的全自动停车方法可以更加便捷地服务大众，造福大众。

参考文献

[1]骆泽雨,杨雪,桑海伟. 智能寻车和停车场车位引导系统[J]. 物联网技术,2017,(01):100-101.

[2]卢锦川,蔡万雄,苏莉萍. 基于RFID和移动平台的智能停车场管理服务系统[J]. 信息通信,2017,(01):123-124.

[3]熊璐. 央视智能停车场人机交互设计研究[J]. 装饰,2015,(01):120-121.

[4]马广韬,张玫珊. 停车场人机交互引导系统的设计与研究[J]. 机电产品开发与创新,2014,(06):62-64.

[5]程昆朋,陈慧. 全自动泊车系统的路径跟随[J]. 汽车技术,2013,(10):26-29+34.

致谢

本文是在杜豫川老师的耐心指导下完成的，杜豫川老师丰富的教学经验是本文得以顺利完成的关键。在撰写论文的过程中，杜老师给予了本研究小组许多方向性的指导以及帮助，当本组在研究的过程中遇到问题时，杜老师也对我们进行了及时的指正。衷心地感谢老师对本组的帮助与指点。

附录

HelloWorldScene.h(可视化界面属性)

```
#ifndef __HELLOWORLD_SCENE_H__
#define __HELLOWORLD_SCENE_H__
#include "cocostudio/CocoStudio.h"
#include "ui/CocosGUI.h"
//#include"CardSprite.h"
#include "cocos2d.h"

USING_NS_CC;


class HelloWorld : public cocos2d::Layer
{
public:
    // there's no 'id' in cpp, so we recommend returning the class instance pointer
    static cocos2d::Scene* createScene();

    // Here's a difference. Method 'init' in cocos2d-x returns bool, instead of returning
'id' in cocos2d-iphone
    virtual bool init();
    //void CreateCars(Ref * pSender, ui::Widget::TouchEventType type);
    //void update(float dt);
    //bool onTouchBegan(Touch * touch, Event * event);
    void initBoard();
    void initRoute();
    void initCar();
    void update(float dt);
    // implement the "static create()" method manually
    CREATE_FUNC(HelloWorld);
};

struct S
{
    int x, y;
    S() :x(-1), y(-1) {}
    S(int xx, int yy) :x(xx), y(yy) {}
    bool operator<(const S &s) const
    {
        if (s.x>x || (s.x == x&&s.y>y))return true;
```

```
        else return false;
    }
};

struct M
{
    int x, y;
    int CarNum;
    bool flag;
    M() :x(-1), y(-1), flag(false),CarNum(-1) {}
    M(int xx,int yy) :x(xx), y(yy),  flag(false),CarNum(-1) {}
};
#endif // __HELLOWORLD_SCENE_H__
```

HelloWorldScene.cpp（可视化界面功能）

```
#include "HelloWorldScene.h"
#include "cocostudio/CocoStudio.h"
#include "ui/CocosGUI.h"
#include "ui/UIButton.h"
#include"CardSprite.h"
#include"Car.h"
#include<iostream>
#include<fstream>
#include<vector>
#include<map>
//using namespace std;

USING_NS_CC;

using namespace cocostudio::timeline;

//std::vector<std::vector<CardSprite*>> cardSprite(30);
CardSprite* cardSprite[30][73];
TMXTiledMap* recMap;
std::vector<Car*>car;
std::vector<M>district[4];
int time1 = 0;


//更新车辆停车情况

void HelloWorld::update(float dt)
{
```

```
    int car_start = -1, car_end = -1, park;
    ++time1;
    for (int i = 1; i < car.size(); ++i) {

        if (time1 == car[i]->time_in) {//车辆开始入库

            park = -1;
            car[i]->createCarSprite(car[i]->CarNum, 1, 0);
            addChild(car[i]);
            int park_in = car[i]->park_in;

            for (int j = 0; j < district[park_in].size(); ++j)//找到车队的队尾

                if (district[park_in][j].flag) {
                    car_end = j;
                    break;
                }
            if (car_end == -1)car_end = 0;

            for (int j = car_end; j <district[park_in].size();++j)//找到车队的队头

                if (!district[park_in][j].flag) {
                    car_start = j-1;
                    break;
                }
            if (car_start == -1)car_start = 0;


            for (int j = car_end; j <=car_start; ++j) {

                if (district[park_in][j].flag) {//停有车辆

                    if (car[district[park_in][j].CarNum]->time_out < car[i]->time_out)
{
                        int x = district[park_in][j].x;
                        int y = district[park_in][j].y;
                        car[i]->route_in        =        cardSprite[x][y]->route_in;
car[i]->p_route_in = 0;
                        car[i]->route_out       =       cardSprite[x][y]->route_out;
car[i]->park_out = 0;

                        park = j;//找到停入车位

                        for (int k = car_start; k >= park; --k) {//车辆前移

                            int x = district[park_in][k].x;
                            int y = district[park_in][k].y;
                            //if (cardSprite[x][y]->type == 'p') {
                                int CarNum = district[park_in][k].CarNum;
```

```
                    if (car[CarNum]->x == x&&car[CarNum]->y == y) {//车辆
已停进库
                        //更新车辆位置
                        car[CarNum]->removeFromParent();
                        S s = car[CarNum]->route_out.front();
                        car[CarNum]->route_out.pop();
                        car[CarNum]->x = s.x;
                        car[CarNum]->y = s.y;
                        car[CarNum]->setPosition(Vec2(79 + 11 * s.x, 157
+ 11 * s.y));
                        addChild(car[CarNum]);
                        if          (cardSprite[s.x][s.y]->type          !=
'p')car[CarNum]->OnRuning = 2;
                        /*district[park_in][jj + 1].flag = true;
                        district[park_in][jj      +      1].CarNum      =
district[park_in][jj].CarNum;*/
                        //更新车位状态
                        district[park_in][k      +      1].CarNum      =
district[park_in][k].CarNum;
                        district[park_in][k + 1].flag = true;
                    }
                    else {//车辆还在入库途中
                        //更新车辆出入库路径
                        while
(cardSprite[car[CarNum]->route_out.front().x][car[CarNum]->route_out.front().y]->type  !
= 'p'&&!car[CarNum]->route_out.empty())car[CarNum]->route_out.pop();
                        S s = car[CarNum]->route_out.front();
                        int xx = car[CarNum]->x;
                        int yy = car[CarNum]->y;
                        car[CarNum]->route_in                          =
cardSprite[s.x][s.y]->route_in;
                        car[CarNum]->route_out                         =
cardSprite[s.x][s.y]->route_out;
                        while (car[CarNum]->route_in.front().x != xx ||
car[CarNum]->route_in.front().y                                        !=
yy&&!car[CarNum]->route_in.empty())car[CarNum]->route_in.pop();
                        car[CarNum]->route_in.pop();
```

```
                                    /*//向后走一步

    car[CarNum]->x=car[CarNum]->route_in[car[CarNum]->p_route_in].x;
                                    car[CarNum]->y                                  =
car[CarNum]->route_in[car[CarNum]->p_route_in].y;
                                    car[CarNum]->setPosition(Vec2(79 + 11 * s.x, 157
+ 11 * s.y));
                                    addChild(car[CarNum]);
                                    ++car[CarNum]->p_route_in;*/

                                    //更新车位状态

                                    district[park_in][k      +      1].CarNum      =
district[park_in][k].CarNum;
                                    district[park_in][k + 1].flag = true;
                                }
                            //}
                        }
                        district[park_in][park].CarNum = car[i]->CarNum;
                        break;
                    }
                }
                else continue;
            }

            if (park == -1) {//车位位置为队头

                for (int j = car_start; j < district[park_in].size();++j)
                    if (!district[park_in][j].flag) {
                        park = j;
                        break;
                    }
                district[park_in][park].flag = true;
                district[park_in][park].CarNum = car[i]->CarNum;
                car[i]->route_in                                                    =
cardSprite[district[park_in][park].x][district[park_in][park].y]->route_in;
                car[i]->route_out                                                   =
cardSprite[district[park_in][park].x][district[park_in][park].y]->route_out;
            }
            //district[park_in][park].CarNum = car[i]->CarNum;
        }

        else if (time1 > car[i]->time_in&&car[i]->OnRuning==1&&!car[i]->type) {//车辆正
在入库
```

```
            car[i]->removeFromParent();
            S next = car[i]->route_in.front();
            car[i]->x = next.x;
            car[i]->y = next.y;
            car[i]->route_in.pop();
            car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
            addChild(car[i]);

            if (car[i]->route_in.empty()) {//车辆已停入车位

                car[i]->OnRuning = 0;
                car[i]->type = 1;
            }
        }

        else if (time1 + 1 == car[i]->time_out) {//车辆准备出库

            car[i]->OnRuning = 1;
            for (int j = 0; j < district[car[i]->park_in].size(); ++j)
                if                (district[car[i]->park_in][j].x                ==
car[i]->x&&district[car[i]->park_in][j].y == car[i]->y) {
                    district[car[i]->park_in][j].flag = false;
                    break;
                }
        }

        else if (car[i]->OnRuning==1&&car[i]->type) {//车辆正在出库

            car[i]->removeFromParent();
            S next = car[i]->route_out.front();
            car[i]->x = next.x;
            car[i]->y = next.y;
            car[i]->route_out.pop();
            car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
            addChild(car[i]);

            if (car[i]->route_out.empty()) {//车辆已出库

                car[i]->OnRuning = 0;
            }
        }

        else if (car[i]->OnRuning == 2) {//车辆正在调整车位

            car[i]->removeFromParent();
            S next = car[i]->route_out.front();
            car[i]->x = next.x;
            car[i]->y = next.y;
            car[i]->route_out.pop();
```

```
            car[i]->setPosition(Vec2(79 + 11 * car[i]->x, 157 + 11 * car[i]->y));
            addChild(car[i]);
            if (cardSprite[next.x][next.y]->type == 'p')car[i]->OnRuning = 0;
        }
    }
}

Scene* HelloWorld::createScene()
{
    // 'scene' is an autorelease object
    auto scene = Scene::create();

    // 'layer' is an autorelease object
    auto layer = HelloWorld::create();

    // add layer as a child to scene
    scene->addChild(layer);

    // return the scene
    return scene;
}

// on "init" you need to initialize your instance
bool HelloWorld::init()
{
    /**  you can create scene with following comment code instead of using csb file.
    // 1. super init first
    if ( !Layer::init() )
    {
        return false;
    }

    Size visibleSize = Director::getInstance()->getVisibleSize();
    Vec2 origin = Director::getInstance()->getVisibleOrigin();

    /////////////////////////////
    // 2. add a menu item with "X" image, which is clicked to quit the program
    //    you may modify it.

    // add a "close" icon to exit the progress. it's an autorelease object
    auto closeItem = MenuItemImage::create(
                                           "CloseNormal.png",
                                           "CloseSelected.png",
                                           CC_CALLBACK_1(HelloWorld::menuCloseCallback,
```

```
this));

    closeItem->setPosition(Vec2(origin.x + visibleSize.width - closeItem->getContentSize().width/2 ,
                                origin.y + closeItem->getContentSize().height/2));

    // create menu, it's an autorelease object
    auto menu = Menu::create(closeItem, NULL);
    menu->setPosition(Vec2::ZERO);
    this->addChild(menu, 1);

    /////////////////////////////
    // 3. add your codes below...

    // add a label shows "Hello World"
    // create and initialize a label

    auto label = Label::createWithTTF("Hello World", "fonts/Marker Felt.ttf", 24);

    // position the label on the center of the screen
    label->setPosition(Vec2(origin.x + visibleSize.width/2,
                            origin.y + visibleSize.height - label->getContentSize().height));

    // add the label as a child to this layer
    this->addChild(label, 1);

    // add "HelloWorld" splash screen"
    auto sprite = Sprite::create("HelloWorld.png");

    // position the sprite on the center of the screen
    sprite->setPosition(Vec2(visibleSize.width/2 + origin.x, visibleSize.height/2 + origin.y));

    // add the sprite as a child to this layer
    this->addChild(sprite, 0);
    **/

    /////////////////////////////
    // 1. super init first
    if ( !Layer::init() )
    {
        return false;
    }
```

```
    auto rootNode = CSLoader::createNode("MainScene.csb");
    addChild(rootNode);


    //将背景图片导入场景中

    auto* background = Sprite::create("background33.jpg");
    background->setAnchorPoint(Vec2(0, 0));
    background->setPosition(0, 0);
    addChild(background);


    //将瓦片地图加入到场景中

    recMap = TMXTiledMap::create("map5.tmx");
    recMap->setAnchorPoint(Vec2(0, 0));
    recMap->setPosition(79, 157);
    TMXLayer* recLayer;

    recLayer = recMap->layerNamed("Layer1");//获取地图中的层Layer1

    addChild(recMap);
    //for (int i = 0; i < cardSprite.size(); ++i)cardSprite[i].resize(73);


    initBoard();
    initRoute();
    initCar();
    scheduleUpdate();


    return true;
}

//导入车库地图

void HelloWorld::initBoard()
{
    int i, j, id;
    auto* recMap = TMXTiledMap::create("map5.tmx");
    TMXLayer* recLayer = recMap->layerNamed("Layer1");
    for (i = 0; i < 73; ++i)
        for (j = 0; j < 30; ++j) {
            cardSprite[j][i] = new CardSprite();

            int id= recLayer->tileGIDAt(Vec2(i, j));//获取地图上某个图块的id值

            cardSprite[j][i] = CardSprite::createCardSprite(id, i, j);
            addChild(cardSprite[j][i]);
        }
```

```
    cardSprite[1][13]->direction[1] = 1; cardSprite[1][13]->direction[3] = 1;
    cardSprite[1][61]->direction[1] = 1; cardSprite[1][61]->direction[3] = 1;
    cardSprite[23][71]->direction[1] = 1; cardSprite[23][71]->direction[2] = 1;
    cardSprite[28][1]->direction[0] = 1; cardSprite[28][1]->direction[2] = 1;
}

//导入出入库最短路径

void HelloWorld::initRoute()
{
    std::string str;
    std::queue<S>t;
    std::ifstream fin;
    std::string fullPath;
    int k = 0;

    //读入"park_route_out.txt"中的路径

    fullPath = FileUtils::getInstance()->fullPathForFilename("park_route_out.txt");
    fin.open(fullPath);
    while (getline(fin, str)) {
        int x, y, sx, sy;
        int i = 1;
        while (!t.empty())t.pop();
        std::string s = "";
        ++k;
        while (i < str.size()) {
            s = "";
            while (str[i] >= '0'&&str[i] <= '9') {
                s = s + str[i];
                ++i;
            }
            x = atoi(s.c_str());
            ++i;
            s = "";
            while (str[i] >= '0'&&str[i] <= '9') {
                s = s + str[i];
                ++i;
            }
            y = atoi(s.c_str());
            t.push(S(x, y));
            i = i + 4;
        }
        while (!t.empty()) {
            S m = t.front();
            M mm = M(m.x, m.y);
```

```
            if (cardSprite[m.x][m.y]->type=='p')district[k].push_back(mm);
            t.pop();
            if (cardSprite[m.x][m.y]->type == 'p')cardSprite[m.x][m.y]->route_out = t;
        }
    }
    fin.close();

    //读入"route_1_0.txt"中的路径

    fullPath = FileUtils::getInstance()->fullPathForFilename("route_1_0.txt");
    fin.open(fullPath);
    while (getline(fin, str)) {
        int x, y, sx, sy;
        int i = 1;
        while (!t.empty())t.pop();
        std::string s = "";
        while (i < str.size()) {
            s = "";
            while (str[i] >= '0'&&str[i] <= '9') {
                s = s + str[i];
                ++i;
            }
            x = atoi(s.c_str());
            ++i;
            s = "";
            while (str[i] >= '0'&&str[i] <= '9') {
                s = s + str[i];
                ++i;
            }
            y = atoi(s.c_str());
            t.push(S(x, y));
            i = i + 4;
        }
        t.pop();
        cardSprite[x][y]->route_in = t;
    }
    fin.close();
}

//导入车辆信息

void HelloWorld::initCar()
{
    std::string str;
    std::vector<S>t;
    std::string fullPath = FileUtils::getInstance()->fullPathForFilename("parking.txt");
```

```
std::ifstream fin;
fin.open(fullPath);
Car* x = new Car();
car.push_back(x);
while (getline(fin, str)) {
    Car* t=new Car();
    if (str.size() < 1)continue;
    int i = 0;
    std::string s = "";
    while (str[i] >= '0'&&str[i] <= '9') {
        s = s + str[i];
        ++i;
    }
    t->CarNum = atoi(s.c_str());
    ++i;
    s = "";

    while (str[i] >= '0'&&str[i] <= '9') {
        s = s + str[i];
        ++i;
    }
    t->time_in = atoi(s.c_str());
    ++i;
    s = "";

    while (str[i] >= '0'&&str[i] <= '9') {
        s = s + str[i];
        ++i;
    }
    t->park_in = atoi(s.c_str());
    ++i;
    s = "";

    while (str[i] >= '0'&&str[i] <= '9') {
        s = s + str[i];
        ++i;
    }
    t->time_out = atoi(s.c_str());
    ++i;
    s = "";

    while (str[i] >= '0'&&str[i] <= '9') {
        s = s + str[i];
        ++i;
```

```
            }
            t->park_out = atoi(s.c_str());
            ++i;
            s = "";
            car.push_back(t);
        }
        fin.close();
}
```

CardSprite.h（图块属性）

```
#pragma once
#include "cocos2d.h"
#include "HelloWorldScene.h"

class CardSprite :public cocos2d::Layer
{
    //friend class HelloWorld;
public:

    int direction[4];//记录道路方向属性

    char type;//'r'为路，'w'为墙，'p'为停车位

    cocos2d::Sprite* card;
    std::queue<S>route_in, route_out;
    CardSprite();

    static CardSprite* createCardSprite(int type, float x, float y);//创建图块

    void cardInit(int type, float x, float y);//图块初始化

    virtual bool init();
    CREATE_FUNC(CardSprite);
};
```

CardSprite.cpp（图块属性定义）

```
#include"CardSprite.h"
#include"HelloWorldScene.h"

CardSprite::CardSprite() {
    type = '\0';
    for (int i = 0; i < 4; ++i)direction[i] = 0;
    card = new cocos2d::Sprite();
```

```
}

//创建图块
CardSprite* CardSprite::createCardSprite(int type, float x, float y)
{
    CardSprite* card = new CardSprite();
    if (card&&card->init()) {
        card->autorelease();
        card->cardInit(type, x, y);
        return card;
    }
    CC_SAFE_DELETE(card);
    return NULL;
}

bool CardSprite::init()
{
    if (!Layer::init())return false;
    return true;
}

//图块初始化
void CardSprite::cardInit(int type, float x, float y)
{
    switch (type) {
    case 1:this->type = 'w'; break;
    case 2: {
        this->type = 'r';
        this->direction[0] = 1;
        break;
    }
    case 3: {
        this->type = 'r';
        this->direction[1] = 1;
        break;
    }
    case 4: {
        this->type = 'r';
        this->direction[2] = 1;
        break;
    }
    case 5: {
```

```
            this->type = 'r';
            this->direction[3] = 1;
            break;
        }
    case 6:this->type = 'r'; break;
    case 7:this->type = 'p'; break;
    case 8:this->type = 'e'; break;
    }
    card->setAnchorPoint(Vec2(0, 0));
    if (this->type == 'r') {
        card = cocos2d::Sprite::create("road.jpg");
        card->setPosition(84 + 11 * x, 157 + 11 * y);
        addChild(card);
    }
    else if (this->type == 'p') {
        card = cocos2d::Sprite::create("parking.jpg");
        card->setPosition(84 + 11 * x, 157 + 11 * y);
        addChild(card);
    }
    else if (this->type == 'w') {
        card = cocos2d::Sprite::create("wall.jpg");
        card->setPosition(84 + 11 * x, 157 + 11 * y);
        addChild(card);
    }
}
```

Car.h（车辆属性）

```
#pragma once
#include "cocos2d.h"
#include "HelloWorldScene.h"

class Car :public cocos2d::Layer
{
    //friend class HelloWorld;

public:
    int CarNum;

    int OnRuning;//0为停止状态，1为运动状态，2为调整车位状态

    int x, y, time_in, time_out, park_in, park_out;

    int type;//0为入库车，1为出库车

    std::queue<S>route_in, route_out;
```

```
    cocos2d::Sprite* car;
    Car();
    Car(int carnum, int onruning, int xx, int yy, int tt);

    static Car* createCarSprite(int num,int x,int y);//创建图块


    void carInit(int num, int x, int y);//图块初始化

    virtual bool init();
    CREATE_FUNC(Car);
};
```

Car.cpp（车辆属性定义）

```
#include"Car.h"
#include"HelloWorldScene.h"

Car::Car()
{
    OnRuning = 1;
    CarNum = -1;
    x = 1;
    y = 0;
    type = 0;
    car = new cocos2d::Sprite();
}

Car::Car(int carnum,int onruning,int xx,int yy,int tt) {
    OnRuning = onruning;
    CarNum = carnum;
    x = xx;
    y = yy;
    type = tt;
    car = new cocos2d::Sprite();
}

Car* Car::createCarSprite(int num,int x,int y)
{
    Car* car = new Car();
    if (car&&car->init()) {
        car->autorelease();
        car->carInit(num, x, y);
        return car;
    }
    CC_SAFE_DELETE(car);
```

```
    return NULL;
}

bool Car::init()
{
    if (!Layer::init())return false;
    return true;
}

void Car::carInit(int num,int x,int y)
{
    this->x = x;
    this->y = y;
    this->CarNum = num;
    car = cocos2d::Sprite::create("car.png");
    car->setPosition(82 + 11 * 68, 601 - 11 * 22);
    addChild(car);
}

/* operator<(const Car &s1, const Car &s2)const
{
    if  (s1.time_out>s2.time_out  ||  (s1.time_out  ==  s2.time_out&&s1.time_in  >
s2.time_in))return true;
    else return false;
}*/
```