TESS NG 微观交通仿真软件
Python API 使用说明书
（版本：V 2.1.0）

二 0 二二年九月

上海济达交通科技有限公司

目　　录

1. API 简介

1.1 API 研发背景

TESS NG 微观交通仿真系统融合了交通工程、软件工程、系统仿真等交叉学科领域的最新技术研发而成，主要特点为：完全自主知识产权、便捷快速的建模能力，开放的外部接口模块以及定制化的用户服务等。

TESS NG 强化设计，在软件功能扩展、项目建设过程中对大量功能的实现进行抽象，将抽象的逻辑过程在 TESS NG 内部与核心功能相融合。这些抽象的逻辑细化成接口方法，抽象接口方法的具体实现留给具体运用。在此架构设计下，开发出车路协同、在线仿真、微宏观一体化仿真等插件模块。目前这些抽象接口的设计和运用已经成熟。

TESS NG 的二次开发通过用户编写代码与 TESS NG 交互来实现能力扩张与功能定制。

TESS NG python API 与 C++API 功能完全一致。python API 是在 C++API 基础上封装的，由于 python 语法与 C++相比存在一些差别，为了使 python API 的调用方式尽可能与 C++ API 调用方式保持一致，对少数几个 C++ 方法进行了调整，最终 C++ API 与 python API 在功能上完全一致，只是 python 少数几个方法名不同（避免重载带来复杂性）。

本次 TESS NG 发布版本号为 2.1.0，API 使用手册初始版本号与 TESS NG 软件版本号保存一致，后续会丰富 API 使用手册内容，API 手册版本的小号会增加。

作者后续会通过博客详解 TESS NG C++API 和 python API 一些重要方法的使用，与用户进行交流，集中讨论在二次开发过程中遇到的问题。

作者博客：https://blog.csdn.net/nlhlx

1.2 TESS NG 的 python API 运行环境配置

1.2.1 python 版本

TESS NG python 接口封装过程是在 python3.6.6 的环境下进行的。用户进行 TESS NG 二次开发时 python 的版本推荐使用 python3.6 版本，如果在 python 3.6 以上版本下出现问题可向济达交通团队进行反馈（联系方法见文末）。

1.2.2 QT 版本

封装 TESS NG python 接口用到 QT 软件公司开发的 **pyside2** for QT5.15 工具包，及 **shiboken2** for QT5.15 工具包。用户 python 开发环境需要导入这两个工具包，一般安装 pyside2 后系统会自动安装 shiboken2。由于 pyside2 工具包已包含了 QT 动态库，用户不必另外安装 QT 软件。

1.2.3 TESS NG 动态库

已发布的 TESS NG2.1.0 是在 QT5.15.9 环境下编译的，QT 提供的 python 工具包是基于 QT5.15 版本的，所以 TESS NG for python 开发版在 QT5.15 环境下经过重新编译，重新编译的 TESS NG 动态库不可以拷贝到 TESS NG2.0 安装目录下。

1.2.4 TESS NG python 接口开发相关文件

TESS NG python 接口开发包有两个关键文件：Tessng.pyd，以及附加动态库 shiboken2.abi3.dll。另外 Tessng.pyi 是文本形式的描述文件，在开发时提供方便，但不是必须的。

开发包主要文件截图如下：

名称	修改日期	类型	大小
Tessng.pyi	2022-08-14 11:53	PYI 文件	70 KB
Tessng	2022-07-27 16:27	Python Extensio...	1,280 KB
TESS_WIN.dll	2022-07-11 15:01	应用程序扩展	1,846 KB
TESS_NET.dll	2022-07-11 15:00	应用程序扩展	750 KB
TESS_SIMU.dll	2022-07-11 15:00	应用程序扩展	496 KB
TESS_DB.dll	2022-07-11 15:00	应用程序扩展	479 KB
TessSupport.dll	2022-07-10 16:00	应用程序扩展	55 KB
TessInterfaces.dll	2022-07-10 15:38	应用程序扩展	120 KB
libPythonPanel.dll	2022-07-06 12:10	应用程序扩展	10 KB
shiboken2.abi3.dll	2021-10-11 16:29	应用程序扩展	245 KB

图 1 开发包主要文件截图

1.2.5 TESS NG Python 开发环境激活及试用期

用户在首次使用 Python 环境运行范例或开发文件时，也需要激活，具体激活界面如下：

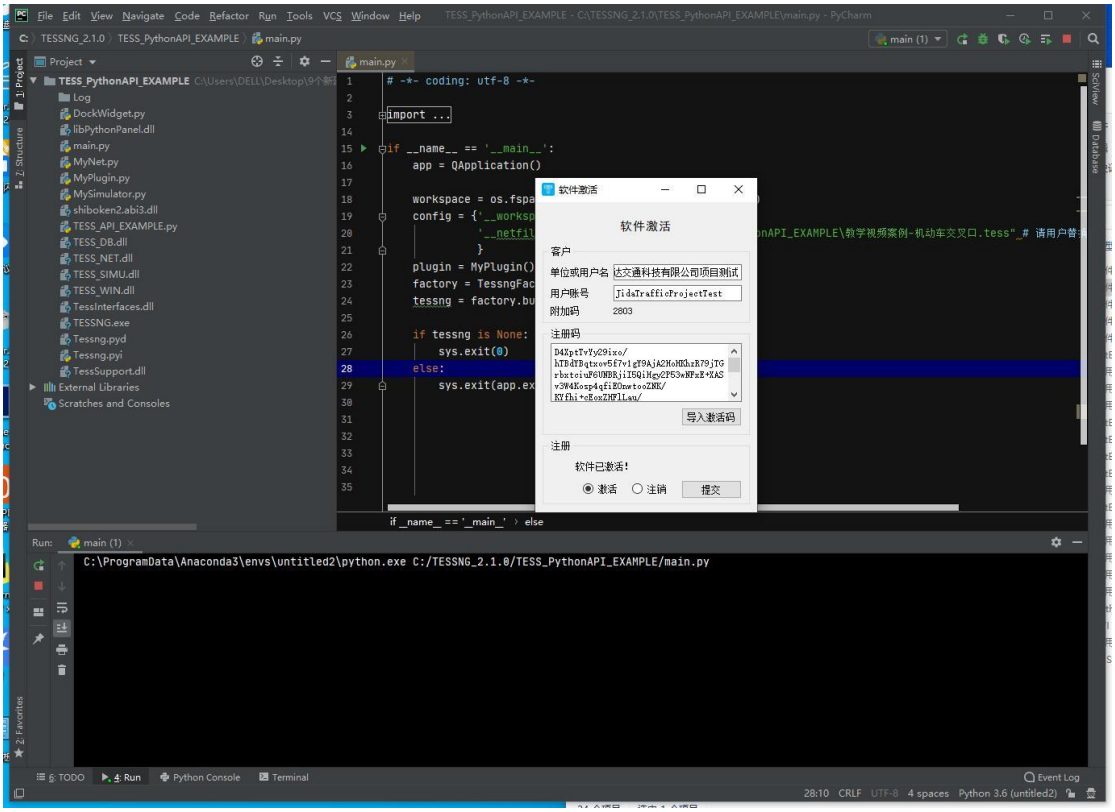


图 2 Python 开发环境激活界面

试用用户与首次激活软件流程相同，采用安装包的 Cert 文件夹下的 JidaTraffic_key 激活即可。

软件的试用期为 30 天，以前激活过软件的客户激活 V2.1 版本时重新延长 30 天试用期（识别激活电脑的物理地址）。试用期结束后将无法调用接口的二次开发功能。

购买了二次开发插件的企业版客户使用不受限制。

1.2.6 TESS NG Example 及 demo 文件运行

用户配置好 python 开发环境后，直接执行 TESS PythonAPI EXAMPLE 的 main.py 函数即可启动运行范例文件。

用户需要运行 TESS PythonAPI demo 文件时，直接将 demo 文件夹所有文件复制替换掉 EXAMPLE 文件夹中的对应文件，并核查文件路径即可运行执行 main.py 函数。**注：如果路径错误，会加载默认路网及出现其它异常情况。**

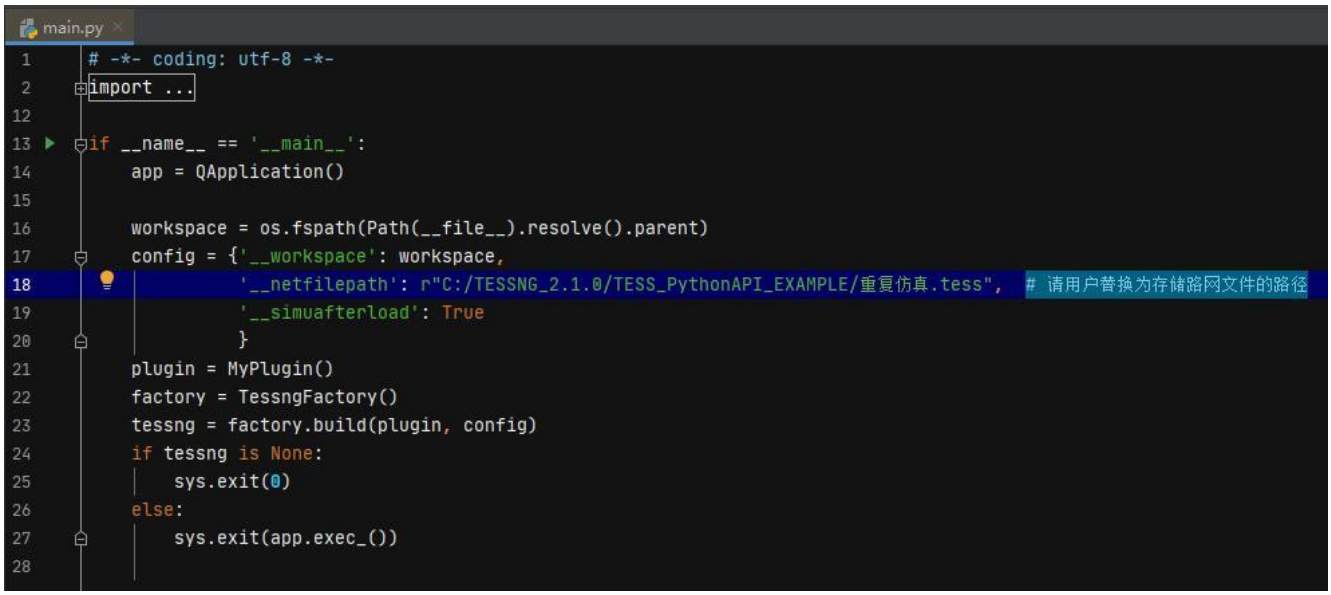


图 3 用户注意核查文件路径

注：运行过程中提示缺少其它 Python 环境模块，可直接按提示安装即可。

1.3 启动 TESS NG

1.3.1 简单启动 TESS NG

TESS NG 二次开发非常方便，在 main.py 文件中加入简单几行代码就可以启动 TESSSNG。代码如下：

```
# -*- coding: utf-8 -*-
import os
from pathlib import Path
import sys

from PySide2.QtCore import *
from PySide2.QtGui import *
from PySide2.QtWidgets import *

from Tessng import *

if __name__ == '__main__':
    app = QApplication()

    workspace = os.fspath(Path(__file__).resolve().parent)
    config = {'__workspace':workspace }
```

```
factory = TessngFactory()
tessng = factory.build(None, config)
if tessng is None:
    sys.exit(0)
else:
    sys.exit(app.exec_())
```

结果如下：

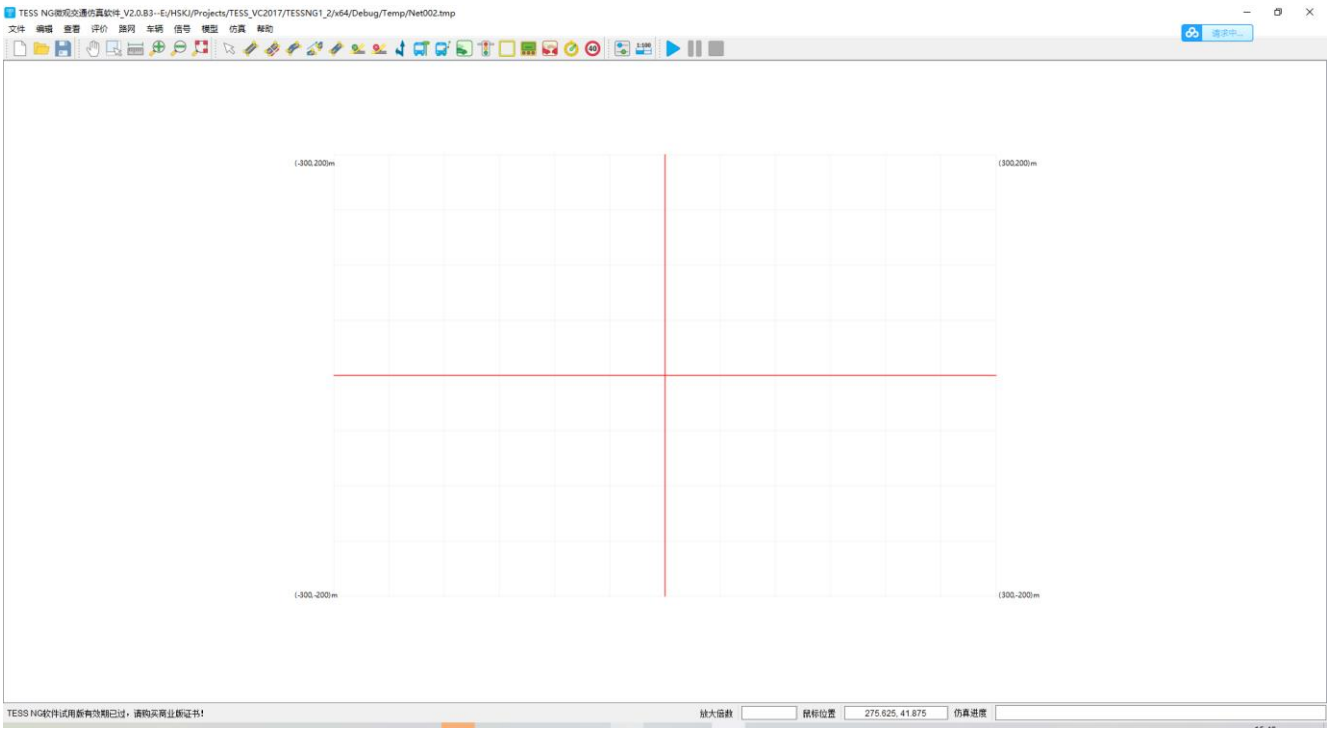


图 1 打开窗体结果

1.3.2 启动 TESS NG 时自动加载指定路网

如果打开 TESSNG 时自动加载指定路网文件，可以在配置参数里指定要加载的路网全路径名。

将

```
config = {'__workspace':workspace }
```

换成

```
config = {'__workspace':workspace,
          '__netfilepath':"C:/TESSNG_2.1.0/Example/上海虹桥枢纽.tess"
         }
```

结果如下：

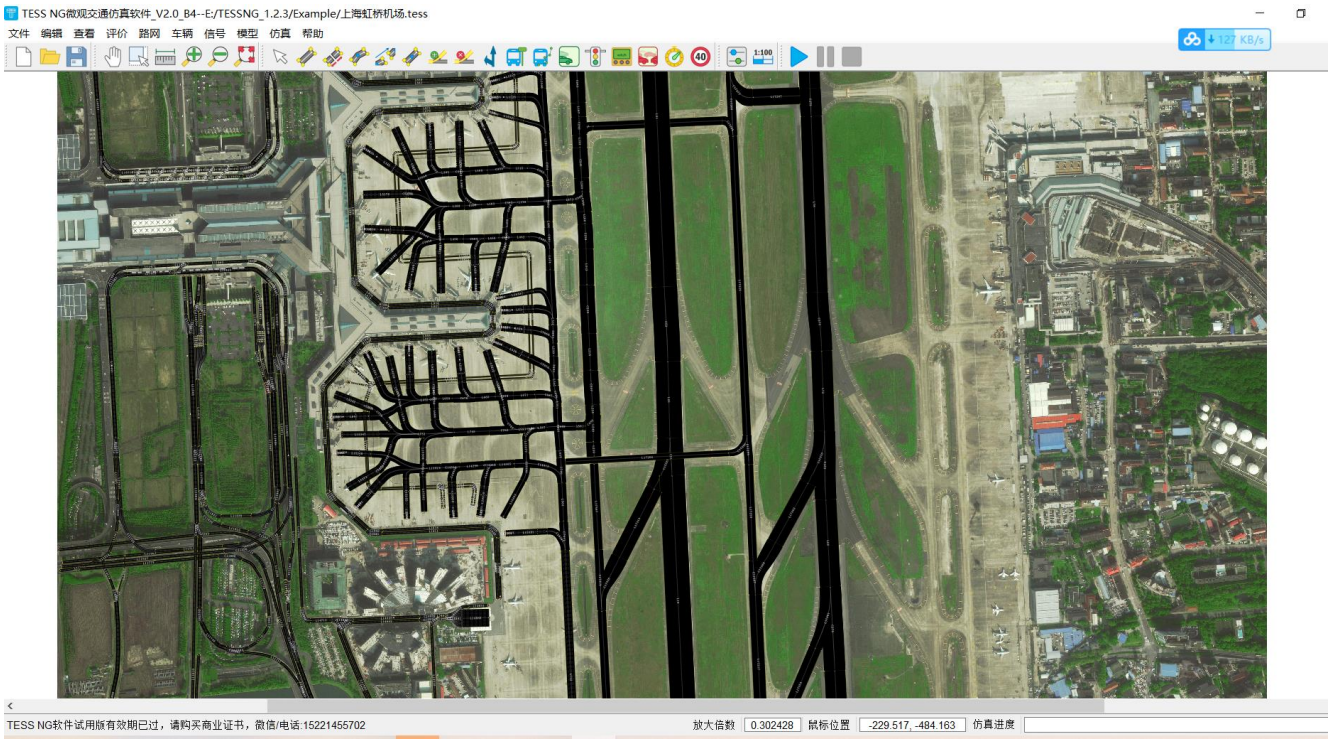


图 2 API 打开文件

1.3.3 启动 TESS NG 后自动启动仿真计算

在 config 中增加"__simuafterload"属性，如下：

```
config = {'__workspace':workspace,
          '__netfilepath':"C:/TESSNG_2.1.0/Example/上海虹桥枢纽.tess",
          '__simuafterload':True
         }
```

其中'__simuafterload'设为 True 指定 TESS NG 加载路网后自动开始仿真计算。

1.4 通过插件与 TESS NG 交互

与 TESS NG 交互、对 TESS NG 施加控制，主要是通过插件方式实现的，插件可以是符合 TESS NG 插件规范的动态连接库，由 TESS NG 启动时加载，也可以直接在内存中创建，后者是 TESS NG 二次开发的主要技术路线，二种技术路线不能同时使用。

范例通过插件在 TESS NG 主窗体增加一个 QDockWidget 对象， QDockWidget 对象包含一个用户创建的 QWidget，在这个 QWidget 上可以创建按钮、信息框等组件，用于和 TESS NG 进行交互。将 QDockWidget 对象加入 TESS NG 主窗体是通过调用 TessInterface 子接口 GuiInterface 的方法 addDockWidgetToMainWindow 来实现的：

```
def initGui(self):
    # 在 TESS NG 主界面上增加 QDockWidget 对象
    self.examleWindow = TESS_API_EXAMPLE()
```

```
iface = tessngIFace()
win = iface.guiInterface().mainWindow()

dockWidget = QDockWidget("自定义与 TESS NG 交互界面", win)
dockWidget.setObjectName("mainDockWidget")
dockWidget.setFeatures(QDockWidget.NoDockWidgetFeatures)
dockWidget.setAllowedAreas(Qt.LeftDockWidgetArea)
dockWidget.setWidget(self.examleWindow.centralWidget())
iface.guiInterface().addDockWidgetToMainWindow(Qt.DockWidgetArea(1), dockWidget)
```

范例实例化插件后作为参数传给工厂类的 build 方法得到 TESS NG 实例，代码如下：

```
plugin = MyPlugin()
factory = TessngFactory()
tessng = factory.build(plugin, config)
```

启动 TESS NG 结果如下图：

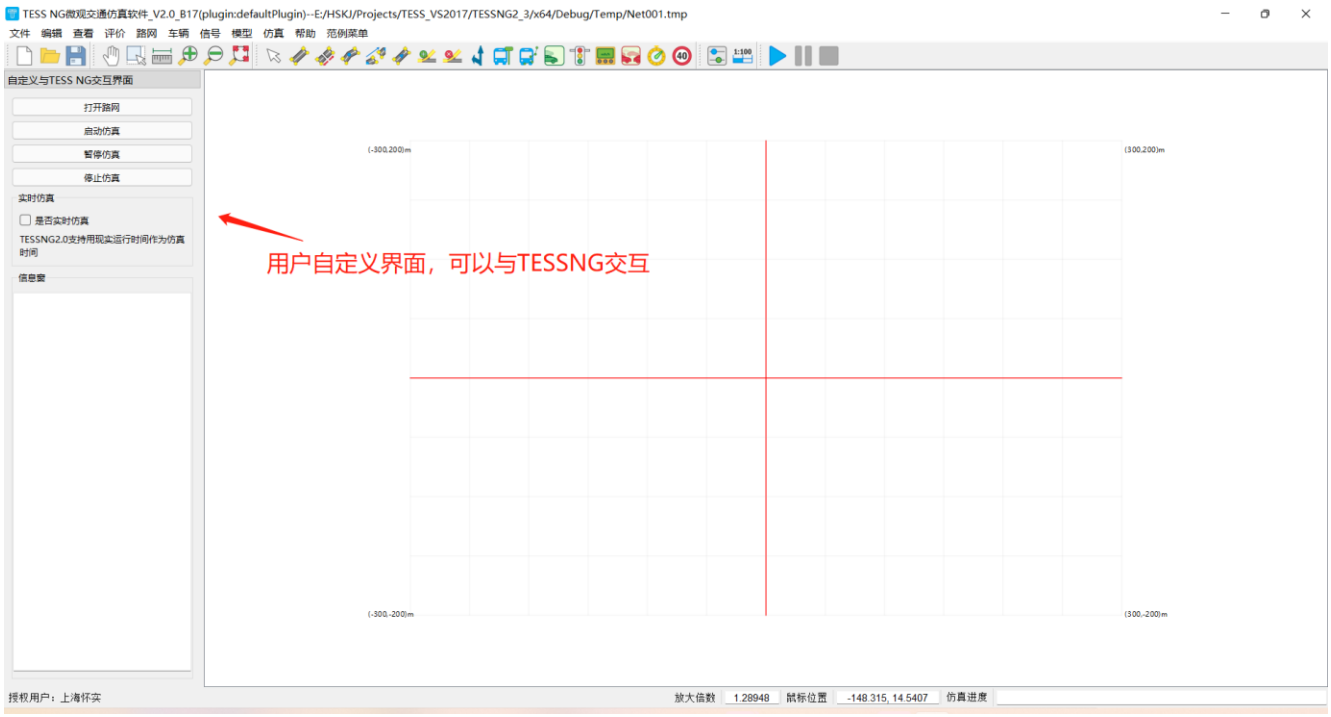


图 3 用户自定义交互界面

范例在自定义界面上按放了几个按钮，其中按钮“启动仿真”的槽函数里用下列代码启动仿真：

```
def startSimu(self):
    iface = tessngIFace()
    if not iface:
        return
    if not iface.simuInterface().isRunning() or iface.simuInterface().isPausing():
        iface.simuInterface().startSimu()
```

1.5 度量单位

TESS NG 在运行过程涉及的度量单位有多种，有基本的度量单位，如：长度单位，复合型度量单位，如：速度单位。在启动 TESS NG 后需要设置长度基本度量单位，默认基本度量单位是像素，像素与米存在转换关系，这个关系通常在画路网前设置像素比时就已确定。

在二次开发过程中需要根据说明将大部分与长度有关的单位改成像素。像素与米制的相互转换方法如下，默认情况下 1 个像素表示 1 米：

米制转换像素：def m2p(value:float) -> float:

像素转换米制：def p2m(value:float) -> float:

2. 接口架构

TESS NG 和插件相互调用，实现对 TESS NG 运行过程从精细到粗放各个层次的控制。

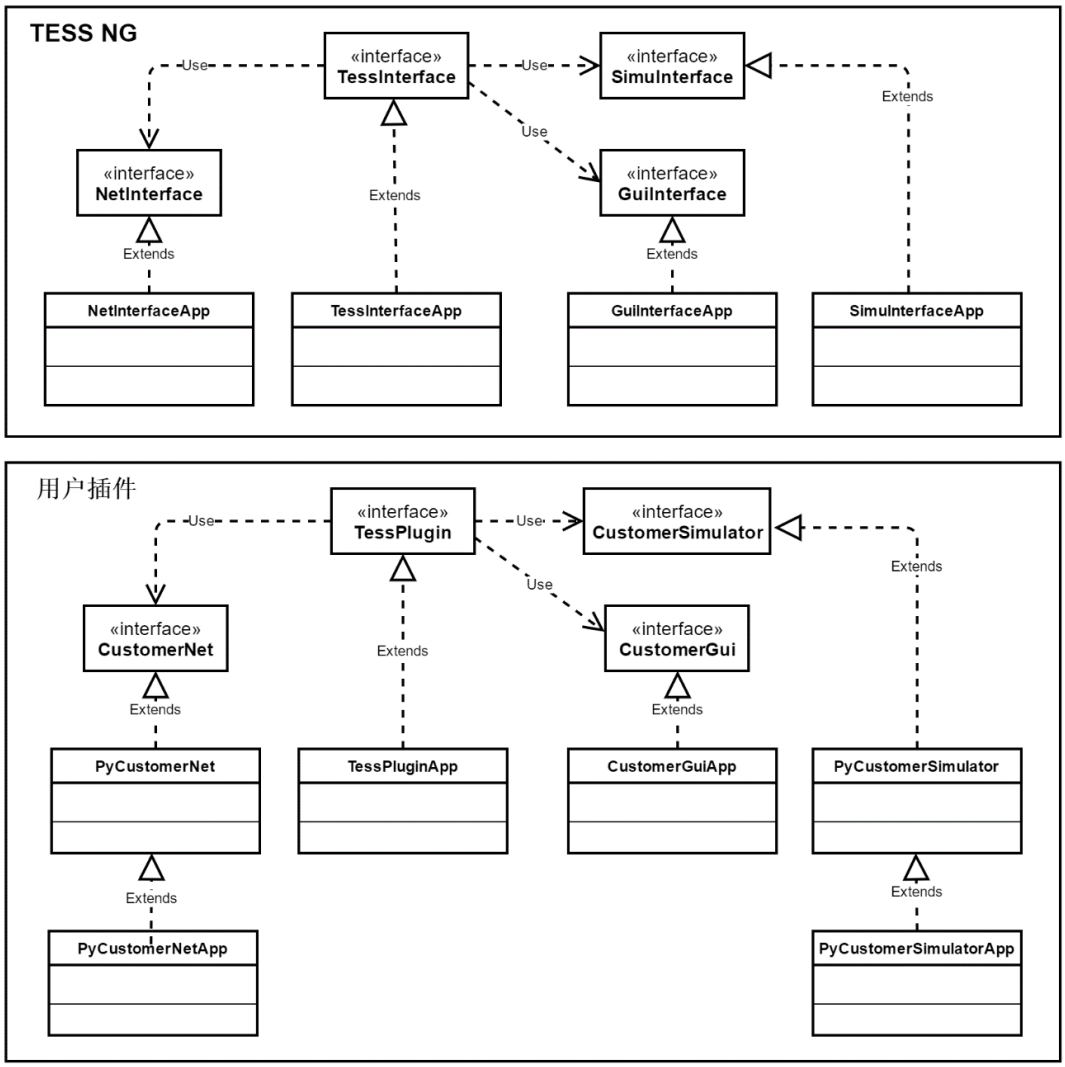


图 4 接口架构图

TESS NG 通过实现 TessInterface 及其三个子接口，将自身主要功能暴露给用户，用户启动 TESS NG 后可以通过 tessngIFace()方法获取 TESS NG 的顶层接口，再通过顶层接口获取三个子接口，调用子接口方法。TESS NG 加载插件后可以调用 python 实现的插件接口方法，用户可以在插件方法中通过 TessInterface 及其子接口控制仿真运行，及仿真过程中车辆驾驶行为、信号灯色、路径车辆分配等等。

3. 范例简介

范例“TESS NG PYAPI EXAMPLE”展示了在窗体界面、路网展示、仿真过程三个方面如何对 TESS NG 施加影响。

范例启动时从内存加载插件，企业版用户可成功加载，之后加载路网。如果成功加载了插件，插件会检测路网上的路段数，如果没有路段会创建几条路段、连接段以及发车点。之后 TESS NG 再根据 config 参数"__simuafterload"值决定是否起动仿真。

3.1 增加窗体控件

在窗体界面上的影响：范例在窗体放置一个 QDockWidget 对象，QDockWidget 对象包含了自定义界面，自定义界面上放置一些控件，通过这些控件操作 TESS NG 的路网加载、仿真运行，如下图如示：

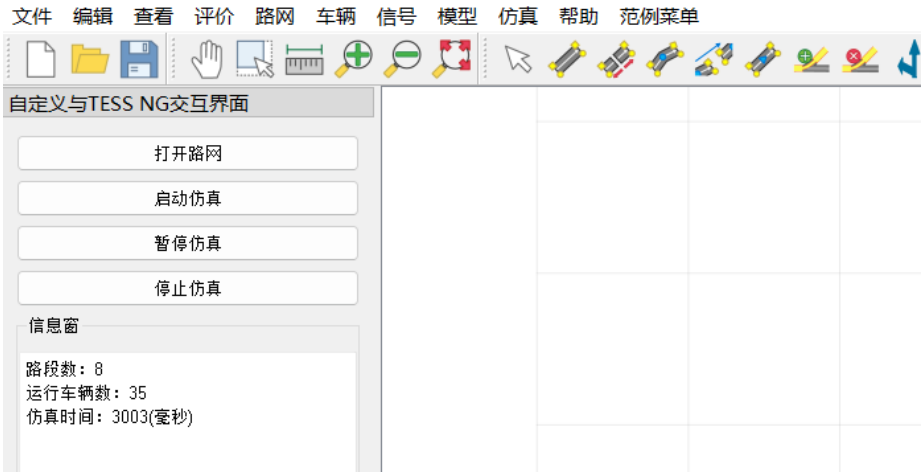


图 5 窗体控制

3.2 改变路网元素展示内容

在路网展示上的影响：范例在加载路网后判断路网上是否有路段，如果没有则创建几条路段、连接段和几个发车点。其中名称为“曹安路”的路段的标签显示路段名，其它路段标签显示 ID，连接段标签显示的都是名称，如下图所示：

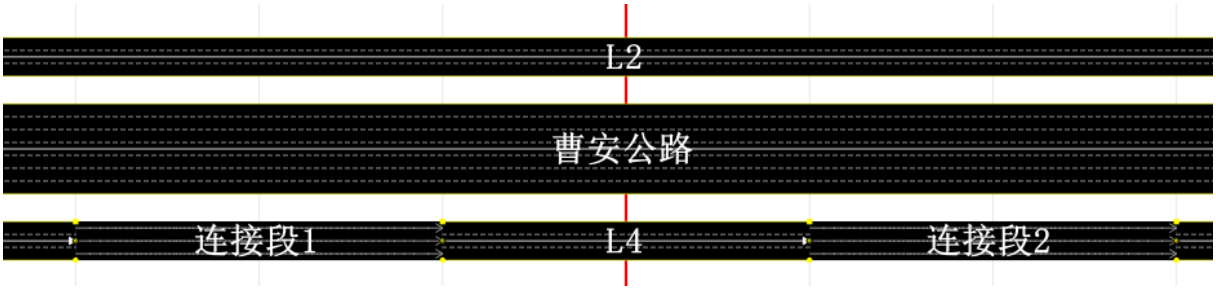
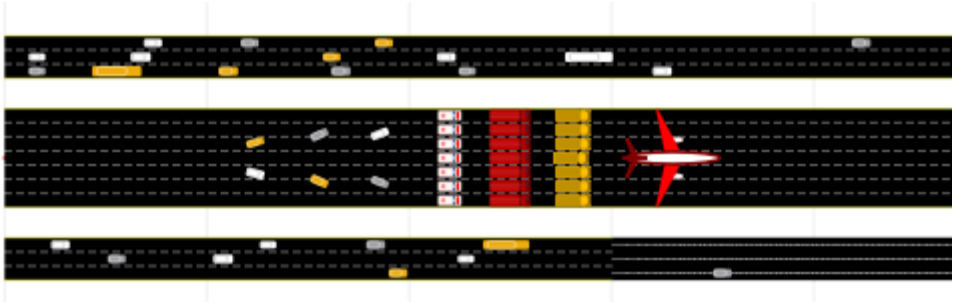


图 6 改变路网元素展示内容

3.3 控制车辆驾驶行为

在仿真过程上的影响：范例在仿真过程的影响包括初始化车辆车道、位置、速度，从几个方面改变车辆速度，以及控制自由变道，如下图所示：



通过车辆对象设置车辆长度函数 setLength(self, len:float, bRestWidth:bool)中，如果 bRestWidth 为“True”，则车身宽度会随长度等比例变化，如果设为 False，则车身宽度不会改变。

图 7 控制车辆驾驶行为

3.4 在路段和连接段中间任意位置发车

在仿真过程上的影响：范例在仿真过程动态从路段及连接段上距起点一定距离的位置发车，无需发车点发车，如下图所示：

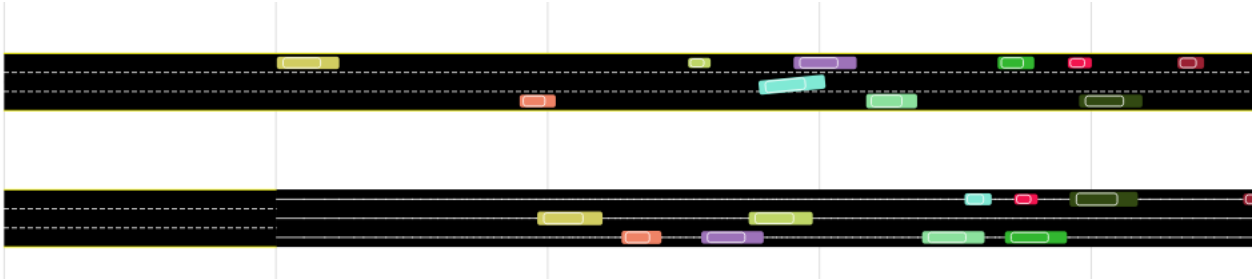


图 8 在任意位置发车

4. 接口详解

4.1 配置字典 config 及插件方法调用频次

4.1.1 配置字典 config 属性详解

在创建 TESS NG 工厂类实例前创建字典 config，config 配置了一些重要信息，说明如下：

```
{
    "__workspace"xxxxxx",
    "__netfilepath":"xxx.tess",
    "__simuafterload":False,
    "__timebycpu":False,
    "__custsimubysteps":False
}
```

"__workspace"：指定“当前工作路径”，TESS NG 会在“当前工作路径”的 Cert 子文件夹下读取认证文件，在”SimuResult”子文件夹下保存仿真结果，等。

"__netfilepath"：指定 TESSNG 启动后加载的路网文件全路径名；

"__simuafterload"：指定 TESSNG 加载路网文件（指定的路网文件或临时空白路网文件）后是否启动仿真；

"__timebycpu"：指定每个仿真周期时间计算依据，是 cpu 时钟确定的时长（现实时长），还是由仿真精度确定的时长。在线仿真且算力吃紧时可以尝试设置此属性为 True；

"__custsimubysteps"：设置 TESSNG 对插件方法调用频次的依据，设为 False 表示每个计算周期都会调用一次插件实现的方法，即不依据插件端设置的调用频次；设为 True 时 TESSNG 依据插件设置的调用频次对插件实现的 PyCustomerSimulator 方法进行调用。

python 二次开发环境下，如果运行车辆不多，可以将"__custsimubysteps"设为 False。如果运行车辆较多，可以将"__custsimubysteps"设为 True，再设定实现的方法调用频次，使对仿真效率的负面影响最小化。

4.1.2 插件方法调用频次及是否允许插件对车辆进行重绘

TESSNG 调用插件方法的频次是指对插件实现的 PyCustomerSimulator 接口方法调用频次。

可以在 PyCustomerSimulator 的 initVehicle(self, pIVehicle:Tessng.IVehicle)方法里通过 pIVehicle 设置 TESSNG 对 PyCustomerSimulator 不同方法调用频次及是否允许插件重绘车辆。

是否允许对车辆重绘方法的调用：默认为 False，如果允许，可以传入 True，如：pIVehicle.setIsPermitForVehicleDraw(True)。可以通过 pIVehicle 得到该车辆类型及 ID 等信息来确定是否允许对该车辆重绘。

当"__custsimubysteps"设置为 True 时，**默认调用频次比较低，很多低到毫无意义，只为减少调用次数，不至于影响仿真运行效率**。如果某方法被实现，需要对该方法调用频次进行调整。可参见范例。

假设仿真精度是 steps，即每秒计算 steps 次，各方法默认调用频次如下：

1）、车辆相关方法调用频次

计算下一位置前处理方法 beforeNextPoint 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次；

具体车辆一个步长计算完成后的处理方法 afterStep 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次；

确定是否停止车辆运行并移出路网方法 isStopDriving 调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次；

2）、驾驶行为相关方法调用频次

重新设置期望速度方法 reCalcdesirSpeed 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次，如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

计算最大限速方法 calcMaxLimitedSpeed 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

计算限制车道方法 calcLimitedLaneNumber 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

计算车道限速方法 calcSpeedLimitByLane 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

计算安全变道方法 calcChangeLaneSafeDist 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

重新计算是否可以左强制变道方法 reCalcToLeftLane 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

重新计算是否可以右强制变道方法 reCalcToRightLane 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

重新计算是否可以左自由变道方法 reCalcToLeftFreely 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

重新计算是否可以右自由变道方法 reCalcToRightFreely 被调用频次：每 steps 个仿真周期调用一次，即每秒调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

计算跟驰类型后处理方法 afterCalcTracingType 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 20 个计算周期调用 1 次或更小。

连接段上汇入到车道前处理方法 beforeMergingToLane 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

重新计算跟驰状态参数方法 reSetFollowingType 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

计算加速度方法 calcAcce 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

重新计算加速度方法 reSetAcce 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

重置车速方法 reSetSpeed 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

重新计算角度方法 reCalcAngle 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

计算后续道路前处理方法 beforeNextRoad 被调用频次：每 steps * 300 个仿真周期调用一次，即 5 分钟调用一次。如果该方法被实现，建议将该方法调用频次设为 1 个计算周期调用 1 次或更大。

4.2 路网基本元素

4.2.1 IRoadNet

路网基本信息接口，设计此接口的目的是为了 TESS NG 在导入外源路网时能够保存这些路

网的属性，如路网中心点坐标、空间参考等。

接口方法：

- **def id(self) -> int: ...**

  路网 ID

- **def netName(self) -> str: ...**

  路网名称

- **def url(self) -> str: ..**

  源数据路径，可以是本地文件，可以是网络地址

- **def type(self) -> str: ...**

  来源分类："TESSNG"表示 TESSNG 自建；"OpenDrive"表示由 OpenDrive 数据导入；"GeoJson"表示由 geojson 数据导入

- **def bkgUrl(self) -> str: ...**

  背景路径

- **def otherAttrs(self) -> typing.Dict: ...**

  其它属性字曲数据

4.2.2 ILink

路段接口，方法如下：

- **def id(self) -> int: ...**

  获取路段 ID

- **def length(self) -> float: ...**

  获取路段长度，默认单位：像素

- **def width(self) -> float: ...**

  获取路段宽度

- **def name(self) -> str: ...**

  获取路段名称

- **def laneCount(self) -> int: ...**

  获取车道数

- **def limitSpeed(self) -> float: ...**

  获取路段最高限速，单位：千米/小时

- **def setLimitSpeed(self, speed:float) -> None: ...**

  设置最高限速

  参数：

  [in] speed：最高限速，单位：千米/小时

- **def minSpeed(self) -> float: ...**

  最低限速，单位：千米/小时

- **def lanes(self) -> typing.List: ...**

  车道接口列表

- **def centerBreakPoints(self) -> typing.List: ...**

  路段中心线断点集

- **def leftBreakPoints(self) -> typing.List: ...**

  路段左侧线断点集

- **def rightBreakPoints(self) -> typing.List: ...**

  路段右侧线断点集

- **def centerBreakPoint3Ds(self) -> typing.List: ...**

  路段中心线断点(三维)集

- **def leftBreakPoint3Ds(self) -> typing.List: ...**

  路段左侧线断点(三维)集

- **def rightBreakPoint3Ds(self) -> typing.List: ...**

  路段右侧线断点(三维)集

4.2.3 ILane

车道接口，方法如下：

- **def id(self) -> int: ...**

  获取车道 ID

- **def link(self) -> Tessng.ILink: ...**

  获取车道所在路段

- **def length(self) -> float: ...**

  获取车道长度，单位：像素

- **def number(self) -> int: ...**

获取车道序号，从 0 开始，自外侧往内侧

- **def actionType(self) -> str: ...**

  获取车道的行为类型

- **def centerBreakPoints(self) -> typing.List: ...**

  获取车道中心点断点集，断点坐标用像素表示

- **def leftBreakPoints(self) -> typing.List: ...**

  车道左侧线断点集

- **def rightBreakPoints(self) -> typing.List: ...**

  车道右侧线断点集

- **def centerBreakPoint3Ds(self) -> typing.List: ...**

  车道中心线断点(三维)集

- **def leftBreakPoint3Ds(self) -> typing.List: ...**

  车道左侧线断点(三维)集

- **def rightBreakPoint3Ds(self) -> typing.List: ...**

  车道右侧线断点(三维)集

- **def setLaneType(self, type:str) -> None: ...**

  设置车道类型

  参数：

  [in] type：车道类型，选下列几种类型其中一种："机动车道"、"机非共享"、"非机动车道"、"公交专用道"

4.2.4 IConnector

连接段接口，方法如下：

- **def id(self) -> int: ...**

  获取连接段 ID

- **def fromLink(self) -> Tessng.ILink: …**

  获取起始路段

- **def toLink(self) -> Tessng.ILink: ...**

  获取目标路段

- **def limitSpeed(self) -> float: ...**

获取最高限速，以起始路段的最高限速作为连接段的最高限速

- **def minSpeed(self) -> float: ...**

  获取最低限速，以起始路段的最低限速作为连接段的最低限速

- **def laneConnectors(self) -> typing.List: ...**

  获取“车道连接”列表

4.2.5 ILaneConnector

“车道连接”接口，方法如下：

- **def fromLane(self) -> Tessng.ILane: ...**

  上游车道

- **def toLane(self) -> Tessng.ILane: ...**

  下游车道

- **def length(self) -> float: ...**

  “车道连接”长度，单位：像素

- **def centerBreakPoints(self) -> typing.List: ...**

  获取“车道连接”中心线断点集，断点坐标用像素表示

- **def leftBreakPoints(self) -> typing.List: ...**

  “车道连接”左侧线断点集

- **def rightBreakPoints(self) -> typing.List: ...**

  “车道连接”右侧线断点集

- **def centerBreakPoint3Ds(self) -> typing.List: ...**

  “车道连接”中心线断点(三维)集

- **def leftBreakPoint3Ds(self) -> typing.List: ...**

  “车道连接”左侧线断点(三维)集

- **def rightBreakPoint3Ds(self) -> typing.List: ...**

  “车道连接”右侧线断点(三维)集

4.2.6 IConnectorArea

面域接口，方法如下：

- **def id(self) -> int: ...**

  面域 ID

- **def allConnector(self) -> typing.List: ...**

  面域相关所有连接段

4.2.7 IDispatchPoint

发车点接口，方法如下：

- **def id(self) -> int: ...**

  获取发车点 ID

- **def name(self) -> str: ...**

  获取发车名称

- **def link(self) -> Tessng.ILink: ...**

  获取发车点所在路段

- **def addDispatchInterval(self, vehiCompId:int, interval:int, vehiCount:int) -> int: ...**

  为发车点增加发点间隔

  参数：

  vehicCompId：车型组成 ID

  interval：时间段，单位：秒

  vehiCount：发车数

  返回值：

  返回发车间隔 ID

  举例：

  为新建的发车点增加一个发车间隔：车型组成 ID 为 1，3600 秒发送 3600 辆车。

```
dp = netiface.createDispatchPoint(link1)
if dp != None :
    # 设置发车间隔，含车型组成、时间间隔、发车数
    dp.addDispatchInterval(1, 3600, 3600)
```

4.2.8 IDecisionPoint

决策点接口，接口方法：

- **def id(self) -> int: ...**

  决策点 ID

- **def name(self) -> str: ...**

  决策点名称

- **def link(self) -> Tessng.ILink: ...**

  决策点所在路段

4.2.9 IRouting

路径接口，接口方法：

- **def id(self) -> int: ...**

  路径 ID

- **def calcuLength(self) -> float: ...**

  计算路径长度

- **def getLinks(self) -> typing.List: ...**

  获取路段序列

4.2.10 ISignalLamp

信号灯接口

- **def id(self) -> int: ...**

  获取信号灯 ID

- **def setLampColor(self, colorStr:str) -> None: ...**

  设置信号灯颜色

  参数：

  [in] colorStr：字符串表达的颜色，有四种可选，分别是"红"、"绿"、"黄"、"灰"，或者是"R"、"G"、"Y"、"grey"。

- **def phaseId(self) -> int: ...**

  获取相位 ID

- **def signalGroupId(self) -> int: ...**

  获取信号灯组 ID

4.2.11 IBusLine

公交线路接口，接口方法：

- **def id(self) -> int: ...**

  获取公交线路 ID

- **def name(self) -> str: ...**

  线路名称

- **def length(self) -> float: ...**

  长度,单位：像素

- **def dispatchFreq(self) -> int: ...**

  发车间隔(秒)

- **def dispatchStartTime(self) -> int: ...**

  发车开始时间(秒)

- **def dispatchEndTime(self) -> int: ...**

  发车结束时间(秒)

- **def desirSpeed(self) -> float: ...**

  期望速度(km/h)

- **def passCountAtStartTime(self) -> int: ...**

  起始载客人数

- **def links(self) -> typing.List: ...**

  公交线路经过的路段

- **def stations(self) -> typing.List: ...**

  线路所有站点

- **def stationLines(self) -> typing.List: ...**

  公交站点线路，当前线路相关站点的上下客等参数

4.2.12 IBusStation

公交站点接口，接口方法：

- **def id(self) -> int: ...**

  获取公交站点 ID

- **def name(self) -> str: ...**

  线路名称

- **def laneNumber(self) -> int: ...**

  公交站点所在车道序号

- **def x(self) -> float: ...**

  位置 X

- **def y(self) -> float: ...**

  位置 Y

- **def length(self) -> float: ...**

  长度，单位：像素

- **def stationType(self) -> int: ...**

  站点类型 1：路边式、2：港湾式

- **def link(self) -> Tessng.ILink: ...**

  公交站点所在路段

- **def lane(self) -> Tessng.ILane: ...**

  公交站点所在车道

4.2.13 IBusStationLine

公交站点-线路接口，通过此接口可以获取指定线路某站点运行参数，如靠站时间、下客百分比等，还可以设置这些参数。

接口方法：

- **def id(self) -> int: ...**

  获取公交“站点-线路”ID

- **def stationId(self) -> int: ...**

  公交站点 ID

- **def lineId(self) -> int: ...**

  公交线路 ID

- **def busParkingTime(self) -> int: ...**

公交车辆停靠时间(秒)

- **def getOutPercent(self) -> float: ...**

  下客百分比

- **def getOnTimePerPerson(self) -> float: ...**

  平均每位乘客上车时间，单位：秒

- **def getOutTimePerPerson(self) -> float: ...**

  平均每位乘客下车时间，单位：秒

- **def setBusParkingTime(self, time:int) -> None: ...**

  设置车辆停靠时间(秒)

- **def setGetOutPercent(self, percent:float) -> None: ..**

  设置下客百分比

- **def setGetOnTimePerPerson(self, time:float) -> None: ...**

  设置平均每位乘客上车时间

- **def setGetOutTimePerPerson(self, time:float) -> None: ...**

  设置平均每位乘客下车时间

4.3 车辆及驾驶行为

4.3.1 IVehicle

车辆接口，用于访问、控制车辆。通过此接口可以读取车辆属性，初始化时设置车辆部分属性，仿真过程读取当前道路情况、车辆前后左右相邻车辆及与它们的距离，可以在车辆未驰出路网时停止车辆运行等。

接口方法：

- def id(self) -> int: ...

车辆 ID，车辆 ID 的组成方式为 x * 100000 + y，每个发车点的 x 值不一样，从 1 开始递增，y 是每个发车点所发车辆序号，从 1 开始递增。第一个发车点所发车辆 ID 从 100001 开始递增，第二个发车点所发车辆 ID 从 200001 开始递增。

- **def startLink(self) -> Tessng.ILink: ...**

  车辆进入路网时起始路段

- **def startSimuTime(self) -> int: ...**

车辆进入路网时起始时间

- **def roadId(self) -> int: ...**

  车辆所在路段或连接段 ID

- **def roadIsLink(self) -> bool: ...**

  车辆所在道路是否路段

- **def roadName(self) -> str: ...**

  道路名

- **def initSpeed(self, speed:float=...) -> float: ...**

  初始化车速

  参数：

  [in] speed：车速，如果大于 0，车辆以指定的速度从发车点出发，单位：像素/秒

  返回：初始化车速，单位：像素/秒

- **def initLane(self, laneNumber:int, dist:float=..., speed:float=...) -> None: ...**

  初始化车辆, laneNumber:车道序号，从 0 开始；dist，距起点距离，单位像素；speed：车速，像素/秒

  参数：

  [in] laneNumber：车道序号，从 0 开始

  [in] dist：距离路段起点距离，单位：像素

  [in] speed：起动时的速度，单位：像素/秒

- **def initLaneConnector(self, laneNumber:int, toLaneNumber:int, dist:float=..., speed:float=...) -> None: ...**

  初始化车辆, laneNumber: “车道连接”起始车道在所在路段的序号，从 0 开始自右往左；toLaneNumber:“车道连接”目标车道在所在路段的序号，从 0 开始自右往左， dist，距起点距离，单位像素；speed：车速，像素/秒

  参数：

  [in] laneNumber：车道序号，从 0 开始自右侧至左侧

  [in] toLaneNumber：车道序号，从 0 开始自右侧至左侧

  [in] dist：距离路段起点距离，单位：像素

  [in] speed：起动时的速度，单位：像素/秒

- **def setVehiType(self, code:int) -> None: ...**

设置车辆类型，车辆被创建时已确定了类型，通过此方法可以改变车辆类型

参数：

[in] code：车辆类型编码

- **def length(self) -> float: ...**

路段或连接段长度，单位：像素

- **def setLength(self, len:float, bRestWidth:bool=...) -> None: ...**

设置车辆长度

参数：

[in] len：车辆长度，单位：像素

[in] bRestWidth：是否同比例约束宽度，默认为 false

- **def laneId(self) -> int: ...**

如果 toLaneId() 小于等于 0，那么 laneId()获取的是当前所在车道 ID，如果 toLaneId()大于 0，则车辆在“车道连接”上，laneId()获取的是上游车道 ID

- **def toLaneId(self) -> int: ...**

下游车道 ID。如果小于等于 0，车辆在路段的车道上，否则车辆在连接段的“车道连接”上

- **def lane(self) -> Tessng.ILane: ...**

获取当前车道，如果车辆在“车道连接”上，获取的是“车道连接”的上游车道

- **def toLane(self) -> Tessng.ILane: ...**

如果车辆在“车道连接”上，返回“车道连接”的下游车道，如果当前不在“车道连接”上，返回对象为空

- **def currBatchNumber(self) -> int: ...**

当前仿真计算批次

- **def roadType(self) -> int: ...**

车辆所在道路类型。包 NetItemType 中定义了一批常量，每一个数值代表路网上一种元素类型。如：GLinkType 代表路段、GConnectorType 代表连接段。

- **def limitMaxSpeed(self) -> float: ...**

车辆所在路段或连接段最大限速，兼顾到车辆的期望速度，单位：像素/秒

- **def limitMinSpeed(self) -> float: ...**

车辆所在路段或连接段最小限速，兼顾到车辆的期望速度，单位：像素/秒

- **def vehicleTypeCode(self) -> int: ...**

车辆类型编码。打开 TESSNG，通过菜单“车辆”->“车辆类型”打开车辆类型编辑窗体，可以看到不同类型车辆的编码

- **def vehicleTypeName(self) -> str: ...**

  获取车辆类型名，如“小客车”

- **def vehicleDriving(self) -> Tessng.IVehicleDriving: ...**

  获取车辆驾驶行为接口

- **def driving(self) -> None: ...**

  驱动车辆。在每个运算周期，每个在运行的车辆被调用一次该方法

- **def pos(self) -> PySide2.QtCore.QPointF: ...**

  当前位置，横纵坐标单位：像素

- **def zValue(self) -> float: ...**

  当前高程，单位：像素

- **def acce(self) -> float: ...**

  当前加速度，单位：像素/秒^2

- **def currSpeed(self) -> float: ...**

  当前速度，单位：像素/秒

- **def angle(self) -> float: ...**

  当前角度，北向 0 度顺时针

- **def isStarted(self) -> bool: ...**

  是否在运行，如果返回 false，表明车辆已驰出路网或尚未上路

- **def vehicleFront(self) -> Tessng.IVehicle: ...**

  前车

- **def vehicleRear(self) -> Tessng.IVehicle: ...**

  后车

- **def vehicleLFront(self) -> Tessng.IVehicle: ...**

  左前车

- **def vehicleLRear(self) -> Tessng.IVehicle: ...**

  左后车

- **def vehicleRFront(self) -> Tessng.IVehicle: ...**

  右前车

- **def vehicleRRear(self) -> Tessng.IVehicle: ...**

  右后车

- **def vehiDistFront(self) -> float: ...**

  前车间距，单位：像素

- **def vehiSpeedFront(self) -> float: ...**

  前车速度，单位：像素/秒

- **def vehiDistRear(self) -> float: ...**

  后车间距，单位：像素

- **def vehiSpeedRear(self) -> float: ...**

  后车速度，单位：像素/秒

- **def vehiDistLLaneFront(self) -> float: ...**

  相邻左车道前车间距，单位：像素

- **def vehiSpeedLLaneFront(self) -> float: ...**

  相邻左车道前车速度，单位：像素/秒

- **def vehiDistLLaneRear(self) -> float: ...**

  相邻左车道后车间距，单位：像素

- **def vehiSpeedLLaneRear(self) -> float: ...**

  相邻左车道后车速度，单位：像素/秒

- **def vehiDistRLaneFront(self) -> float: ...**

  相邻右车道前车间距，单位：像素

- **def vehiSpeedRLaneFront(self) -> float: ...**

  相邻右车道前车速度，单位：像素/秒

- **def vehiDistRLaneRear(self) -> float: ...**

  相邻右车道后车间距，单位：像素

- **def vehiSpeedRLaneRear(self) -> float: ...**

  相邻右车道后车速度，单位：像素/秒

- **def setIsPermitForVehicleDraw(self, bDraw:bool) -> None: ...**

  设置是否允许插件绘制车辆

以下方法设置 TESS NG 调用与车辆及驾驶行为相关方法时的调用频次

- **def setSteps_afterCalcTracingType(self, steps:int) -> None: ...**

设置计算跟驰类型后处理方法 afterCalcTracing 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_afterStep(self, steps:int) -> None: ...**

设置车辆一个计算周期后的处理方法 afterStep 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_beforeMergingToLane(self, steps:int) -> None: ...**

设置车辆在连接段汇入前处理方法 beforeMergingToLane 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_beforeNextRoad(self, steps:int) -> None: ...**

设置计算后续道路前处理方法 beforeNextRoad 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcAcce(self, steps:int) -> None: ...**

设置计算加速度方法 calcAcce 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcChangeLaneSafeDist(self, steps:int) -> None: ...**

设置计算安全变道距离方法 calcChangeLaneSafeDist 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcDistToEventObj(self, steps:int) -> None: ...**

设置计算到事件对象距离方法 calcDistToEventObj 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcLimitedLaneNumber(self, steps:int) -> None: ...**

设置计算限行车道方法 calcLinitedLaneNumber 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcMaxLimitedSpeed(self, steps:int) -> None: ...**

设置计算最大限速方法 calcMaxLinitedSpeed 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_calcSpeedLimitByLane(self, steps:int) -> None: ...**

设置计算车道限速方法 calcSpeedLimitByLane 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_isStopDriving(self, steps:int) -> None: ...**

设置是否停止运行方法 isStopDriving 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcAngle(self, steps:int) -> None: ...**

设置重新计算角度方法 reCalcAngle 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcToLeftFreely(self, steps:int) -> None: ...**

设置计算左自由变道方法 reCalcToLeftFreely 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcToLeftLane(self, steps:int) -> None: ...**

设置计算左强制变道方法 reCalcToLeftLane 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcToRightFreely(self, steps:int) -> None: ...**

  设置计算右自由变道方法 reCalcToRightFreely 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcToRightLane(self, steps:int) -> None: ...**

  设置计算右强制变道方法 reCalcToRightLane 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reCalcdesirSpeed(self, steps:int) -> None: ...**

  设置重新计算期望速度方法 reCalcdesirSpeed 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reSetAcce(self, steps:int) -> None: ...**

  设置重新计算加速度方法 reSetAcce 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reSetFollowingType(self, steps:int) -> None: ...**

  设置重新计算跟驰类型方法 reSetFollowingType 被调用频次，即 steps 个计算周期调用 1 次

- **def setSteps_reSetSpeed(self, steps:int) -> None: ...**

  设置重新计算车速方法 reSetSpeed 被调用频次，即 steps 个计算周期调用 1 次

4.3.2 IVehicleDriving

驾驶行为接口，通过此接口可以控制车辆的左右变道、设置车辆角度，对车辆速度、坐标位置等进行控制，可以在路网中间停止车辆运行，将车辆移出路网，等等。

接口方法：

- **def isOnRouting(self) -> bool: ...**

  当前是否在路径上

- **def stopVehicle(self) -> None: ...**

  停止运行，车辆移出路网

- **def angle(self) -> float: ...**

  旋转角，北向 0 度顺时针

- **def setAngle(self, angle:float) -> None: ...**

  设置车辆旋转角

  参数：

  [in] angle：旋转角，一周 360 度

- **def desirSpeed(self) -> float: ...**

  当前期望速度，与车辆自身期望速度和道路限速有关，不大于道路限速，单位：像素/秒

- **def differToTargetLaneNumber(self) -> int: ...**

与目标车道序号的差值，不等于 0 表示有强制变道意图，大于 0 有左变道意图，小于 0 有右变道意图，绝对值大于 0 表示需要强制变道次数

- **def toLeftLane(self) -> None: ...**

  左变道

- **def toRightLane(self) -> None: ...**

  右变道

- **def laneNumber(self) -> int: ...**

  当前车道序号，最右侧序号为 0

- **def setLaneNumber(self, number:int) -> None: ...**

  设置当前车道序号

  参数：

  [in] number：车道序号

- **def currDistance(self) -> float: ...**

  当前计算周期移动距离，单位：像素

- **def currDistanceInRoad(self) -> float: ...**

  当前路段或连接上已行驶距离，单位：像素

- **def setCurrDistanceInRoad(self, dist:float) -> None: ...**

  设置当前路段已行驶距离

  参数：

  [in] dist：距离，单位：像素

- **def setVehiDrivDistance(self, dist:float) -> None: ...**

  设置当前已行驶总里程

  参数：

  [in] dist：总里程，单位：像素

- **def getVehiDrivDistance(self) -> float: ...**

  已行驶总里程

- **def setRouting(self, pRouting:Tessng.IRouting) -> bool: ...**

  设置路径，外界设置的路径不一定有决策点，可能是临时创建的，如果车辆不在此路径上则设置不成功并返回 false

  [in] pRouting：路径

- **def setSegmentIndex(self, index:int) -> None: ...**

  设置分段序号

  [in] index：分段序号

- **def currDistanceInSegment(self) -> float: ...**

  当前在分段上已行驶距离

- **def setCurrDistanceInSegment(self, dist:float) -> None: ...**

  设置在分段上已行驶距离

- **def setX(self, posX:float) -> None: ...**

  设置横坐标

  参数：

  [in] posX：横坐标：单位：像素

- **def setY(self, posY:float) -> None: ...**

  设置纵坐标

  参数：

  [in] posY：纵坐标：单位：像素

- **def distToEndpoint(self, fromVehiHead:bool=...) -> float: ...**

  在车道或“车道连接”上车辆到终端距离

  参数：

  [in] fromVehiHead：是否从车头计算，如果为 false，从车辆中心点计算，默认值为 false

- **def changingTrace(self) -> typing.List: ...**

  变轨点集，如变道轨迹、公交车进入港湾式站点轨迹。

- **def changingTraceLength(self) -> float: ...**

  变轨长度，如变道轨迹长度、公交车进入港湾式站点轨迹长度，单位：像素。

- **def calcTraceLength(self) -> None: ...**

  计算变轨长度，如计算变道轨迹长度等。

- **def setTrace(self, lPoint:typing.Sequence) -> None: ...**

  设置变轨轨迹

- **def setTracingType(self, type:int) -> None: ...**

  设置轨迹类型

  [in] type：轨迹类型 0：跟驰，1：左变道，2：右变道，3：左虚拟变道, 4：右虚拟变道，5：

左转待转，6：右转待转, 7：入湾，8：出湾

4.4 TessInterface

TessInterface 是 TESSN 对外暴露的顶级接口，下面有三个子接口：NetInterface、SimuInterface、GuiInterface，分别用于访问或控制路网、仿真过程和用户交互界面。

获取顶层接口的方法是：tessngIFace()。

下面是几个接口方法的说明：

- **def netInterface(self) -> Tessng.NetInterface: ...**

  返回用于访问控制路网的接口 NetInterface

- **def simuInterface(self) -> Tessng.SimuInterface: ...**

  返回用于控制仿真过程的接口 SimuInterface

- **def guiInterface(self) -> Tessng.GuiInterface: ...**

  返回用于访问控制用户介面的接口 GuiInterface

- **def loadPluginFromMem(self, pPlugin:Tessng.TessPlugin) -> bool: ...**

  从内存加载插件，此方法便于用户基于 API 进行二次开发。

下面对三个子接口进行详解：

4.4.1 NetInterface

NetInterface 是 TessInterface 的子接口，用于访问、控制路网的接口，通过这个接口可以从文件加载路网、创建路段、连接段、发车点等。

下面对 NetInterface 接口方法作详细解释。

- **def openNetFle(self, filePath:str) -> None: ...**

  打开保存在文件中的路网

  参数：

  [in] filePath：路网文件全路径名

  举例：

  ```
  openNetFile("C:/TESSNG/Example/杭州武林门区域路网公交优先方案.tess")
  ```

- **def saveRoadNet(self) -> None: ...**

  保存路网

- **def netFilePath(self) -> str: ...**

获取路网文件全路径名

- **def netAttrs (self) -> Tessng.IRoadNet: ...**

获取路网对象，如果路网是从opendrive导入的，此路网对象可能保存了路网中心点所在的经纬度坐标，以及大地坐标等信息

- **def setNetAttrs(self, name:str, sourceType:str=..., centerPoint:PySide2.QtCore.QPointF=..., backgroundUrl:str=..., otherAttrsJson:typing.Dict=...) -> Tessng.IRoadNet: ...**

设置路网基本信息

参数：

[in] name:路网名称

[in] centerPoint:中心点坐标所在路网，默认为(0,0)

[in] sourceType:数据来源分类，默认为 “TESSNG”，表示路网由 TESSNG 软件直接创建。取值“OPENDRIVE”，表示路网是经过 opendrive 路网导入而来

[in] backgroundUrl：底图路径

[in] otherAttrsJson:保存在 json 对象中的其它属性，如大地坐标等信息。

- **def graphicsScene(self) -> PySide2.QtWidgets.QGraphicsScene: ...**

获取场景对象

- **def graphicsView(self) -> PySide2.QtWidgets.QGraphicsView: ...**

获取视图对象

- **def sceneScale(self) -> float: ...**

场景中的像素比，单位：米/像素

- **def setSceneSize(self, w:float, h:float) -> None: ...**

设置场景大小，参数w及h分别是场景宽度和高度，单位：米

- **def sceneWidth(self) -> float: ...**

场景宽度，单位：米

- **def sceneHeight(self) -> float: ...**

场景高度，单位：米

- **def linkIds(self) -> typing.List: ...**

路段 ID 集

- **def linkCount(self) -> int: ...**

路段数

- **def links(self) -> typing.List: ...**

路段集

- **def findLink(self, id:int) -> Tessng.ILink: ...**

  根据路段 ID 查找路段

- **def connectorIds(self) -> typing.List: ...**

  连接段 ID 集

- **def connectorCount(self) -> int: ...**

  连接段数

- **def connectors(self) -> typing.List: ...**

  连接段集

- **def findConnector(self, id:int) -> Tessng.IConnector: ...**

  根据连接段 ID 查找连接段

- **def dispatchPoints(self) -> typing.List: ...**

  发车点集。

- **def findDispatchPoint(self, id:int) -> Tessng.IDispatchPoint: ...**

  根据发车点 ID 查找发车点

  参数：

  [in] id：发车点 ID

- **def buslines(self) -> typing.List: ...**

  公交线路集

- **def findBusline(self, buslineId:int) -> Tessng.IBusLine: ...**

  根据公交线路 ID 查找公交线路

  参数：

  [in] buslineId：公交线路 ID

- **def findBuslineByFirstLinkId(self, linkId:int) -> Tessng.IBusLine: ...**

  根据公交线路起始路段 ID 查找公交线路

  参数：

  [in] linkId：公交线路起始段 ID

- **def busStations(self) -> typing.List: ...**

  公交站点集

- **def findBusStation(self, stationId:int) -> Tessng.IBusStation: ...**

  根据公交站点 ID 查询公交站点

- **def allConnectorArea(self) -> typing.List: ...**

  面域集

- **def laneCenterPoints(self, laneId:int) -> typing.List: ...**

  指定车道中心线断点集

  参数：

  [in]laneId：指定车道 ID

- **def linkCenterPoints(self, linkId:int) -> typing.List: ...**

  指定路段中心线断点集

  参数：

  [in]linkId：指定路段 ID

- **def judgeLinkToCross(self, linkId:int) -> bool: ...**

  判断路段去向是否进入交叉口， 以面域是否存在多连接段以及当前路段与后续路段之间的角度为依据

- **def getIDByItemName(self, name:str) -> int: ...**

  根据路网元素名获取自增 ID

  参数：

  [in] name：路网元素名。路网元素名的定义在文件 plugin/_netitem.h 中定义

- **def createLink(self, lCenterPoint:typing.Sequence, laneCount:int, linkName:str=..., bAddToScene:bool=...) -> Tessng.ILink: ...**

  创建路段

  参数：

  [in] lCenterPoint：路段中心线断点集

  [in] laneCount：车道数

  [in] linkName：路段名，默认为空，将以路段 ID 作为路段名

  [in] bAddToScene：创建后是否放入路网场景，默认为 True

  举例：

  ```
  startPoint = QPointF(m2p(-300), 0)
  endPoint = QPointF(m2p(300), 0)
  lPoint = [startPoint, endPoint]
  link1 = netiface.createLink(lPoint, 7, "曹安公路")
  ```

  返回：路段对象。

- **def createLink3D(self, lCenterV3:typing.Sequence, laneCount:int, linkName:str=..., bAddToScene:bool=...) -> Tessng.ILink: ...**

  创建路段

  参数：

  [in] lCenterV3：路段中心线断点序列，每个断点都是三维空间的点

  [in] laneCount：车道数

  [in] linkName：路段名

  返回：路段对象。

- **def createLinkWithLaneWidth(self, lCenterPoint:typing.Sequence, lLaneWidth:typing.Sequence, linkName:str=..., bAddToScene:bool=...) -> Tessng.ILink: ...**

  创建路段

  参数：

  [in] lCenterPoint：路段中心线断点序列

  [in] lLaneWidth：车道宽度列表

  [in] linkName：路段名

  [in] bAddToScene：是否加入场景，默认为 True

  返回：路段对象。

- **def createLink3DWithLaneWidth(self, lCenterV3:typing.Sequence, lLaneWidth:typing.Sequence, linkName:str=..., bAddToScene:bool=...) -> Tessng.ILink: ...**

  创建路段

  参数：

  [in] lCenterV3：路段中心线断点序列，每个断点都是三维空间的点

  [in] lLaneWidth：车道宽度列表

  [in] linkName：路段名

  [in] bAddToScene：是否加入场景，默认为 True

  返回：路段对象。

- **def createLink3DWithLanePoints(self, lCenterLineV3:typing.Sequence, lanesWithPoints:typing.Sequence, linkName:str=..., bAddToScene:bool=...) -> Tessng.ILink: ...**

  创建路段

参数：

[in] lCenterLineV3：路段中心点集(对应 TESSNG 路段中心点)，每个[in\点都是三维空间的

[in] lanesWithPoints：车道数据集合，每个成员是 QMap<QString, QList<QVector3D>>类型数据，有三个 key，分别是“left”、“center”、“right”、分别表示一条车道左、中、右侧断点序列。

[in] linkName：路段名，默认为路段 ID

[in] bAddToScene：是否加入路网，默认 True 表示加入

返回：路段对象

- **def createConnector(self, fromLinkId:int, toLinkId:int, lFromLaneNumber:typing.Sequence, lToLaneNumber:typing.Sequence, connName:str=..., bAddToScene:bool=...) -> Tessng.IConnector: ...**

  创建连接段

  参数：

  [in] fromLinkId：起始路段 ID

  [in] toLinkId：目标路段 ID

  [in] lFromLaneNumber：连接段起始车道序号集

  [in] LToLaneNumber：连接段目标车道序号集

  [in] connName：连接段名，默认为空，以两条路段的 ID 连接起来作为名称

  [in] bAddToScene：创建后是否放入路网场景，默认为 True

  举例：

起始段到目标路段创建两条车道连接，第一条为起始路段 1 号车道连接到目标路段 1 号车道，第二条为起始路段 1 号车道连接到目标路段 2 号车道

```
#创建第三条连接段
if link7 is not None and link8 is not None:
    lFromLaneNumber = [1, 2, 3]
    lToLaneNumber = [1, 2, 3]
    conn1 = netiface.createConnector(link7.id(), link8.id(), lFromLaneNumber, lToLaneNumber, "动态发车连接段", True)
```

- **def createConnector3DWithPoints(self, fromLinkId:int, toLinkId:int, lFromLaneNumber:typing.Sequence, lToLaneNumber:typing.Sequence, laneConnectorWithPoints:typing.Sequence, connName:str=..., bAddToScene:bool=...) -> Tessng.IConnector: ...**

创建连接段，创建连接段后将“车道连接”中自动计算的断点集用参数 laneConnectorWithPoints 断点替换

参数：

[in] fromLinkId：起始路段 ID

[in] toLinkId：目标路段 ID

[in] lFromLaneNumber：起始路段参于连接的车道序号

[in] lToLaneNumber：目标路段参于连接的车道序号

[in] laneConnectorWithPoints：“车道连接”数据列表，成员是 QMap<QString, QList<QVector3D>>类型数据，有三种 key，分别是“left”、“center”、“right”，表示一条“车道连接”左、中、右侧断点序列

[in] connName：连接段名，默认将起始路段 ID 和目标路段 ID 用“_”连接表示连接段名，如“100_101”。

[in] bAddToScene：是否加入到场景，默认为 True

返回：连接段对象

- **def createDispatchPoint(self, pLink:Tessng.ILink, dpName:str=..., bAddToScene:bool=...) -> Tessng.IDispatchPoint: ...**

  创建发车点

  参数：

  [in] pLink：路段，在其上创建发车点

  [in] dpName：发车点名称，默认为空，将以发车点 ID 作为名称

  [in] bAddToScene：创建后是否放入路网场景，默认为 True

- **def createVehicleComposition(self, name:str, lVehiComp:typing.Sequence) -> int: ...**

  创建车型组成，如果车型组成名已存在或相关车型编码不存在或相关车型占比小于 0 则返回-1，否则新建车型组成，并返回车型组成编码

  参数：

  [in] name：车型组成名

  [in] lVehiComp：不同车型占比列表

```
# 在路段 link1 上创建发车点
dp = netiface.createDispatchPoint(link1)
if dp != None :
    # 设置发车间隔，含车型组成、时间间隔、发车数
    dp.addDispatchInterval(1, 2, 28)
```

- **def shortestRouting(self, pFromLink:Tessng.ILink, pToLink:Tessng.ILink) -> Tessng.IRouting: ...**

  计算最短路径

  参数：

  [in] pFromLink：起始路段

  [in] pToLink：目标路段

  返回：最短路径对象，包含经过的路段对象序列

- **IRouting *createRouting(QList<ILink*> lILink)**

  用连续通达的路段序列创建路径

  参数：

  [in] lILink：路段对象序列

  返回：路径对象

- **def decisionPoints(self) -> typing.List: ...**

  决策点列表

- **def findDecisionPoint(self, id:int) -> Tessng.IDecisionPoint: ...**

  根据 ID 查找决策点

  [in] id：决策点 ID

  返回：决策点对象

4.4.2 SimuInterface

SimuInterface 是 TessInterface 的子接口， 通过此接口可以启动、暂停、停止仿真，可以设置仿真精度，获取仿真过程车辆对象、车辆状态（包括位置信息），获取几种检测器检测的样本数据和集计数据，等等。

下面对 SimuInterface 接口方法作详细解释。

- **def byCpuTime(self) -> bool: ...**

  仿真时间是否由现实时间确定。

一个计算周期存在两种时间，一种是现实经历的时间，另一种是由仿真精度决定的仿真时间，如果仿真精度为每秒 20 次，仿真一次相当于仿真了 50 毫秒。默认情况下，一个计算周期的仿真时间是由仿真精度决定的。在线仿真时如果算力不够，按仿真精度确定的仿真时间会与现实

时间存在时差。

- **def setByCpuTime(self, bByCpuTime:bool) -> bool: ...**

设置是否由现实时间确定仿真时间，如果设为 True，每个仿真周期现实经历的时间作为仿真时间，这样仿真时间与现实时间相吻合。

参数：

[in] bByCpuTime：是否由现实时间确定仿真时间

- **def startSimu(self) -> bool: ...**

启动仿真

- **def pauseSimu(self) -> bool: ...**

暂停仿真

- **def stopSimu(self) -> bool: ...**

停止仿真运行

- **def pauseSimuOrNot(self) -> None: ...**

暂停或恢复仿真。如果当前处于仿真运行状态，此方法暂停仿真，如果当前处于暂停状态，此方法继续仿真

- **def isRunning(self) -> bool: ...**

仿真是否在进行

- **def isPausing(self) -> bool: ...**

仿真是否处于暂停状态

- **def simuAccuracy(self) -> int: ...**

获取仿真精度

- **def setSimuAccuracy(self, accuracy:int) -> None: ...**

设置仿真精度，即每秒计算次数

参数：

[in] accuracy：每秒计算次数

- **def acceMultiples(self) -> int: ...**

获取加速倍数

- **def setAcceMultiples(self, multiples:int) -> None: ...**

设置加速倍数

参数：

[in] multiples 加速位数

- **def batchNumber(self) -> int: ...**

  当前批次

- **def startMSecsSinceEpoch(self) -> int: ...**

  获取仿真开始的现实时间

- **def stopMSecsSinceEpoch(self) -> int: ...**

  仿真结束的现实时间

- **def simuTimeIntervalWithAcceMutiples(self) -> int: ...**

  获取当前已仿真时间

- **def delayTimeOnBatchNumber(self, batchNumber:int) -> int: ...**

  仿真到指定批次时总延误，单位：毫秒；

  在算力不足的情况下，存在仿真计算每一个周期所需时间大于设置周期时间的情况，造成延误。

  参数：

  [in] batchNumber：仿真批次

  返回值：仿真到 batchNumber 批次时的总延误

- **def vehiCountTotal(self) -> int: ...**

  车辆总数，包括已创建尚未进入路网的车辆、正在运行的车辆、已驶出路网的车辆

- **def vehiCountRunning(self) -> int: ...**

  正在运行车辆数

- **def getVehicle(self, vehiId:int) -> Tessng.IVehicle: ...**

  根据车辆 ID 获取车辆对象

- **def allVehiStarted(self) -> typing.List: ...**

  所有正在运行车辆

- **def allVehicle(self) -> typing.List: ...**

  所有车辆，包括已创建尚未进入路网的车辆、正在运行的车辆、已驶出路网的车辆

- **def getVehisStatus(self) -> typing.List: ...**

  获取所有正在运行的车辆状态，包括轨迹

  返回：车辆状态（包括轨迹）Online.VehicleStatus 列表

  举例：

```
# TESSNG 顶层接口
iface = tessngIFace()
# TESSNG 仿真子接口
simuiface = iface.simuInterface()
# 当前正在运行车辆列表
vehis = simuiface.allVehiStarted()
```

- **def getVehiTrace(self, vehiId:int) -> typing.List: ...**

  获取指定车辆运行轨迹

  参数：

  [in] vehiId：车辆 ID

  返回：车辆运行轨迹，即 Online.VehiclePosition 列表

- **def getSignalPhasesColor(self) -> typing.List: ...**

  获取当前所有信号灯组相位颜色

  返回:当前相位颜色 Online.SignalPhaseColo 列表，包括各相位当前颜色设置的时间和已持续时间。

- **def getVehisInfoCollected(self) -> typing.List: ...**

  获取当前完成穿越车辆数据采集器的所有车辆信息

  返回：采集的车辆信息列表。数据结构 Online::VehiInfoCollected 在文件 Plugin/_datastruct.h 中定义。

  举例：

```
# TESSNG 顶层接口
iface = tessngIFace()
# TESSNG 仿真子接口
simuiface = iface.simuInterface()
# 获取当前仿真时间完成穿越采集器的所有车辆信息
lVehiInfo = simuiface.getVehisInfoCollected()
```

- **def getVehisInfoAggregated(self) -> typing.List: ...**

  获取最近集计时间段内采集器采集的所有车辆集计信息

  返回：采集器集计数据 Online.VehiInfoAggregated 列表

- **def getVehisQueueCounted(self) -> typing.List: ...**

  获取当前排队计数器计数的车辆排队信息

  返回：车辆排队信息 Online.VehiQueueCounted 列表

- **def getVehisQueueAggregated(self) -> typing.List: ...**

  获取最近集计时间段内排队计数器集计数据

  返回：排队计数器集计数据 Online.VehiQueueAggregated 列表

- **def getVehisTravelDetected(self) -> typing.List: ...**

  获取当前行程时间检测器完成的行程时间检测信息

  返回：行程时间检测器数据 Online.VehiTravelDetected 列表

- **def getVehisTravelAggregated(self) -> typing.List: ...**

  获取最近集计时间段内行程时间检测器集计数据

  返回：行程时间集计数据 Online.VehiTravelAggregated 列表

- **def createGVehicle(self, dynaVehi:Tessng.Online.DynaVehiParam) -> Tessng.IVehicle: ...**

  动态创建车辆

  参数:

  [in]: dynaVehi：动态车辆信息

  举例：

```
# TESSNG 顶层接口
iface = tessngIFace()
# TESSNG 仿真子接口
simuiface = iface.simuInterface()
dvp = Online.DynaVehiParam()
dvp.vehiTypeCode = random.randint(0, 4) + 1
dvp.roadId = 6
dvp.laneNumber = random.randint(0, 3)
dvp.dist = 50
dvp.speed = 20
dvp.color = color
vehi1 = simuiface.createGVehicle(dvp)
```

  以上是范例中的代码。

- **def stopVehicleDriving(self, pVehicle:Tessng.IVehicle) -> None: ...**

  停止指定车辆的仿真运行，车辆被移出路网

  参数：

  [in] pVehicle：车辆对象

- **def vehisInLink(self, linkId:int) -> typing.List: ...**

  指定 ID 路段上的车辆

  参数

  [in] linkId：路段 ID

  返回：车辆列表

  举例：

```
# TESSNG 顶层接口
iface = tessngIFace()
# TESSNG 仿真子接口
simuiface = iface.simuInterface()
# ID 等于 1 路段上车辆
vehis = iface.simuInterface().vehisInLink(1)
```

- **def vehisInLane(self, laneId:int) -> typing.List: ...**

  指定 ID 车道上的车辆

  参数：

  [in] laneId：车道 ID

  返回：车辆列表

- **def vehisInConnector(self, connectorId:int) -> typing.List: ...**

  指定 ID 连接段上的车辆

  参数：

  [in] connectorId：连接段 ID

  返回：车辆列表

- **def vehisInLaneConnector(self, connectorId:int, fromLaneId:int, toLaneId:int) -> typing.List: ...**

  指定连接段 ID 及上游车道 ID 和下游车道 ID 相关“车道连接”上的车辆

  参数：

  [in] connectorId: 连接段 ID

  [in] fromLaneId：上游车道 ID

  [in] toLaneId：下游车道 ID

4.4.3 GuiInterface

GuiInterface 是 TessInterface 的子接口， 通过此接口可以访问控制 TESSNG 主窗体，在主窗体上创建菜单、自定义窗体等。

- **def mainWindow(self) -> PySide2.QtWidgets.QMainWindow: ...**

  获取 TESS NG 主窗体

4.5 TessPlugin

TessPlugin 是用户开发的插件顶级接口，下面有三个子接口：PyCustomerNet、PyCustomerSimulator、CustomerGui。TESS NG 通过这三个子接口分别在路网、仿真过程、窗体这三个方面与用户插件进行交互。

获取插件顶层接口的方法：tessngPlugin()。

虽然用户可以通过接口 TessInterface 下的三个子接口访问控制 TESS NG 的路网、仿真过程及窗体，但用户只能调用 TESS NG 接口方法，不能深入接口方法内部改变运行逻辑。通过实现接口 TessPlugin 子接口的方法，用户可以在 TESS NG 的方法内部施加影响，改变运行逻辑。

TessPlugin 下的子接口 PyCustomerNet、PyCustomerSimulator 可以让用户较多地参于加载路网及仿真过程，改变 TESSNG 内部运行逻辑。比如，通过实现 PyCustomerNet、PyCustomerSimulator 接口方法可以让用户加载路网后进行必要的处理，点击仿真按钮后根据需要确定是否继续仿真或者放弃，还可以在仿真过程对部分或全部车辆的速度施加影响，主动干预车辆的自由变道，等等。

插件的三个子接口 PyCustomerNet、PyCustomerSimulator、CustomerGui 的所有方法都有默认实现，用户可以根据需要实现其中部分方法或全部方法，这些方法都由 TESSNG 在加载并初始化插件、打开路网前后、仿真前、仿真过程中、仿真结束后进行调用，正是通过 TESS NG 对这些接口方法的调用达到控制或影响 TESS NG 运行的目的。

由于插件接口方法调用的场景、目的都不一样，为了尽可尽可能统一对插件接口方法理解，很多方法采用如下结构形式：

```
def method(self, outParam:type) -> bool
```

TESS NG 在调用这些方法时作以下理解：如果返加值为 False，视为用户没有反应，忽略。如果返回值为 True，表明用户有反应，这时再视参数 outParam 值进行处理。举范例中的一个例子，曹安路上的车辆排成方正，飞机后的车辆速度重新设置，保持与飞机相同的速度。PyCustomerSimulator 的子类 MySimulator 实现了 reSetSpeed 方法如下：

```
def ref_reSetSpeed(self, vehi, ref_inOutSpeed):
    tmpId = vehi.id() % 100000
    roadName = vehi.roadName()
    if roadName == "曹安公路":
        if tmpId == 1:
            self.mrSpeedOfPlane = vehi.currSpeed()
        elif tmpId >= 2 and tmpId <= self.mrSquareVehiCount:
            ref_inOutSpeed.value = self.mrSpeedOfPlane
        return True
```

```
return False
```

TESS NG 在计算车辆的速度后会调用插件的 reSetSpeed 方法，如果该方法返回 True，视插件对此方法作出响应，这时再用 outSpeed 值取代原先计算的车速。

下面对 PyCustomerNet、PyCustomerSimulator 两个子接口进行说明

4.5.1 PyCustomerNet

PyCustomerNet 是 TessPlugin 子接口，用户实现这个接口，TESSNG 在加载路网前后会调用用户实现的接口方法。范例在加载临时路网后创建路段、连接段和发车点。TESSNG 在绘制部分路网元素时也会调用 PyCustomerNet 实现类相关方法。范例通过实现方法 labelNameAndFont 让部分路段和连接段用路段名（默认为 ID）绘制标签。

下面对 PyCustomerNet 接口方法作详细解释。

- **def beforeLoadNet(self) -> None: ...**

  打开路网前调用，用户可以通过此方法在加载路网前作必要的初始化准备工作

- **def afterLoadNet(self) -> None: ...**

  加载路网后调用。

  举例：

范例加载路网后读路段数，如果路段数为 0 创建路段、连接段和发车点，创建完成后根据参数’__simuafterload’值决定是否启动仿真：

```
def afterLoadNet(self):
    # 代表 TESS NG 的接口
    iface = tessngIFace()
    # 代表 TESS NG 的路网子接口
    netiface = iface.netInterface()
    # 获取路段数
    count = netiface.linkCount()
    if(count == 0):
        self.createNet()
```

- **def isPermitForCustDraw(self) -> bool: ...**

在绘制路网过程中是否允许调用客户绘制逻辑，默认为 False。本方法的目的是在 python 环境减少不必要的对 python 代码调用，消除对运行效率的负面影响。可参数范例。

- **def paint(self, itemType:int, itemId:int, painter:PySide2.QtGui.QPainter) -> bool: ...**

  绘制路网元素

  参数：

  [in] itemType：路网元素类型，在包 NetItemType 中定义

[in] itemId：路网元素 ID

[in] painter：QPainter 对象

返回：如果返回 True，TESS NG 认为插件已绘制，TESS NG 不再绘制，否则 TESS NG 进行绘制。

- **def linkBrushAndPen(self, linkId:int, brush:PySide2.QtGui.QBrush, pen:PySide2.QtGui.QPen) -> bool: ...**

  根据指定 ID 设置绘制路段的笔刷。

  参数：

  [in] linkId：路段 ID

  [out] brush：绘刷

  [out] pen：绘笔

  返回：False 忽略，True 用 brush 及 pen 参数绘制 ID 等于 linkId 的路段。

- **def laneBrushAndPen(self, laneId:int, brush:PySide2.QtGui.QBrush, pen:PySide2.QtGui.QPen) -> bool: ...**

  根据指定车道 ID 设置绘制车道的笔刷。

  [in] laneId：车道 ID

  [out] brush：绘刷

  [out] pen：绘笔

  返回：False 忽略，True 用 brush 及 pen 参数绘制 ID 等于 laneId 的车道

- **def connectorAreaBrushAndPen(self, connAreaId:int, brush:PySide2.QtGui.QBrush, pen:PySide2.QtGui.QPen) -> bool: ...**

  根据指定面域 ID 设置绘制面域的笔刷。

  [in] connAreaId：面域 ID

  [out] brush：绘刷

  [out] pen：绘笔

  返回：False 忽略，True 用 brush 及 pen 参数绘制 ID 等于 connAreaId 的面域

- **def ref_labelNameAndFont(self, itemType:int, itemId:int, ref_outPropName:Tessng.objint, ref_outFontSize:Tessng.objreal) -> None: ...**

  根据路网元素类型及 ID 确定用标签用 ID 或名称作为绘制内容。

  参数：

[in] itemType：路段元素类型，类型常量在文件 Plugin/_netitemtype.h 中定义；

[in] itemId：路网元素 ID；

[out] outPropName：枚举值，在文件 Plugin/_netitem.h 中定义，如果赋值 GraphicsItemPropName::Id，则用 ID 作为绘制内容，如果赋值 GraphicsItemPropName::Name，则用路网元素名作为绘制内容；

[out] outFontSize：字体大小，单位：米。假设车道宽度是 3 米，如果赋给 outFontSize 的值是 6，绘出的文字将占用两个车道的宽度。

返回：False 忽略，True 则根据设定的 outPropName 值确定用 ID 或名称绘制标签，并且用指定大小绘制。

举例：

范例中的路段和连接段的标签内容部分是名称，部分是 ID。

```
def ref_labelNameAndFont(self, itemType, itemId, ref_outPropName, ref_outFontSize):
    # 代表 TESS NG 的接口
    iface = tessngIFace()
    # 代表 TESS NG 仿真子接口
    simuiface = iface.simuInterface()
    # 如果仿真正在进行，设置 ref_outPropName.value 等于 GraphicsItemPropName.None_，路段和车道都不绘制标签
    if simuiface.isRunning():
        ref_outPropName.value = GraphicsItemPropName.None_
        return
    # 默认绘制 ID
    ref_outPropName.value = GraphicsItemPropName.Id
    # 标签大小为 6 米
    ref_outFontSize.value = 6
    # 如果是连接段一律绘制名称
    if itemType == NetItemType.GConnectorType:
        ref_outPropName.value = GraphicsItemPropName.Name
    elif itemType == NetItemType.GLinkType:
        if itemId == 1 or itemId == 5 or itemId == 6:
            ref_outPropName.value = GraphicsItemPropName.Name
```

- **def isDrawLinkCenterLine(self, linkId:int) -> bool: ...**

是否绘制路段中心线

参数：

[in] linkId：路段 ID；

返回值：True 绘制，False 不绘制。

- **def isDrawLinkCorner(self, linkId:int) -> bool: ...**

是否绘制路段四个拐角的圆形和正方型。

参数：

[in] linkId：路段 ID；

返回值：True 绘制，False 不绘制。

- **def isDrawLaneCenterLine(self, laneId:int) -> bool: ...**

  是否绘制车道中心线。

  参数：

  [in] laneId：车道 ID；

  返回值：True 绘制，False 不绘制。

- **def afterViewKeyReleaseEvent(self, event:PySide2.QtGui.QKeyEvent) -> None: ...**

  QGraphicsView 的 keyReleaseEvent 事件后行为，用户可以根据自己的需要接入键盘事件，实现自身业务逻辑。

- **def afterViewMouseDoubleClickEvent(self, event:PySide2.QtGui.QMouseEvent) -> None: ...**

  QGraphicsView 的 mouseDoubleClickEvent 事件后的行为，用户可以根据自己的需要编写鼠标双击事件响应代码。

- **def afterViewMouseMoveEvent(self, event:PySide2.QtGui.QMouseEvent) -> None: ...**

  QGraphicsView 的 mouseMoveEvent 事件后的行为，用户可以根据自己的需要编写鼠标移动事件响应代码。

- **def afterViewMousePressEvent(self, event:PySide2.QtGui.QMouseEvent) -> None: ...**

  QGraphicsView 的 mousePressEvent 事件后的行为，用户可以根据自己的需要编写鼠标点击事件响应代码。

- **def afterViewMouseReleaseEvent(self, event:PySide2.QtGui.QMouseEvent) -> None: ...**

  QGraphicsView 的 mouseReleaseEvent 事件后的行为，用户可以根据自己的需要编写鼠标释放事件响应代码。

- **def afterViewResizeEvent(self, event:PySide2.QtGui.QResizeEvent) -> None: ...**

  QGraphicsView 的 resizeEvent 事件后的行为，用户可以根据自己的需要编写屏幕缩放事件响应代码。

- **def afterViewWheelEvent(self, event:PySide2.QtGui.QWheelEvent) -> None: ...**

  QGraphicsView 的鼠标滚动事件后的行为，用户可以根据自己的需要编写鼠标滚动事件后响应代码。

- **def afterViewScrollContentsBy(self, dx:int, dy:int) -> None: ...**

  QGraphicsView 滚动条移动事件后的行为，用户可以根据自己的需要实现视窗滚动条移动后响应代码。

4.5.2 PyCustomerSimulator

PyCustomerSimulator 是 TessPlugin 子接口，用户实现这个接口。TESS NG 在仿真前后以及仿真过程中调用这个接口实现的方法，达到与插件交互的目的，用户可以通过这个接口的实现在仿真前后以及仿真运算过程中对 TESS NG 的仿真进行干预，大到可以控制仿真是否进行，小到干预某一车辆的驾驶行为。

用户对车辆驾驶行为的干预主要通过车速和变道来实现。对车速的干预主要有以下几个方法：

1）重新计算车速；

2）修改路段限速；

3）重新计算加速度；

4）修改跟驰安全距离和安全时距、重新设置前车距

以上几个方法的优先级依次降低。在没有插件干预的情况下，车辆行驶的最高速度受到道路的最高速度限制；在有插件的干预下，如果直接修改了车速，则不受道路最高限速的限制。

下面对 PyCustomerSimulator 接口方法作详细解释。

- **def ref_beforeStart(self, ref_keepOn:Tessng.objbool) -> None: ...**

  仿真前的准备。如果需要，用户可通过设置 keepOn 为 false 来放弃仿真。

  参数：

  [out] ref_keepOn：是否继续，默认为 True；

- **def afterStart(self) -> None: ...**

  启动仿真后的操作。这个方法的处理时间尽量短，否则影响仿真时长的计算，因为调用这个方法的过程仿真已经计时。仿真前的操作尽可能放到 beforeStart 方法中处理。

- **def afterStop(self) -> None: ...**

  仿真结束后的操作，如果需要，用户可以在此方法释放资源。

- **def calcDynaDispatchParameters(self) -> typing.List: ...**

  计算动态发车信息，用来修改发车点相关参数，此方法可以用来实现实时动态仿真。

  返回：动态发车信息 Online.DispatchInterval 列表。

- **def calcDynaFlowRatioParameters(self) -> typing.List: ...**

一个或一次数据来源里保存的所有决策点在一个时间间隔的路径流量分配信息，此方法可以用来实现实时动态仿真。

返回：决策点流量分配信息 Online.DecipointFlowRatioByInterval 列表。

- **def calcDynaSignalContralParameters(self) -> typing.List: ...**

一个或一次数据来源里保存的所有信号灯组的信号控制信息。

返回：信号灯组控制参数 Online.SignalContralParam 列表。

- **def initVehicle(self, pIVehicle:Tessng.IVehicle) -> None: ...**

初始化车辆，此方法在车辆起动加入路网时被调用，用户可以在这个方法里调用 IVehicle 的 setVehiType 方法重新设置类型，调用 initLane 或 initLaneConnector 方法对车辆的车道序号、起始位置、车辆大小进行初始化。

参数：

[in] pIVehicle：车辆对象

举例：

```
def initVehicle(self, vehi):
    tmpId = vehi.id() % 100000
    #  车辆所在路段名或连接段名
    roadName = vehi.roadName()
    #  车辆所在路段 ID 或连接段 ID
    roadId = vehi.roadId()
    if roadName == '曹安公路':
        #飞机
        if tmpId == 1:
            vehi.setVehiType(12)
            vehi.initLane(3, m2p(105), 0)
        #工程车
        elif tmpId >=2 and tmpId <=8:
            vehi.setVehiType(8)
            vehi.initLane((tmpId - 2) % 7, m2p(80), 0)
        #消防车
        elif tmpId >=9 and tmpId <=15:
            vehi.setVehiType(9)
            vehi.initLane((tmpId - 2) % 7, m2p(65), 0)
        #消防车
        elif tmpId >=16 and tmpId <=22:
            vehi.setVehiType(10)
            vehi.initLane((tmpId - 2) % 7, m2p(50), 0)
        #最后两队列小车
        elif tmpId == 23:
            vehi.setVehiType(1)
```

```
            vehi.initLane(1, m2p(35), 0)
        elif tmpId == 24:
            vehi.setVehiType(1)
            vehi.initLane(5, m2p(35), 0)
        elif tmpId == 25:
            vehi.setVehiType(1)
            vehi.initLane(1, m2p(20), 0)
        elif tmpId == 26:
            vehi.setVehiType(1)
            vehi.initLane(5, m2p(20), 0)
        elif tmpId == 27:
            vehi.setVehiType(1)
            vehi.initLane(1, m2p(5), 0)
        elif tmpId == 28:
            vehi.setVehiType(1)
            vehi.initLane(5, m2p(5), 0)
        # 最后两列小车的长度设为一样长，这个很重要，如果车长不一样长，导致的前车距就不一样，会使它们变道轨迹长度不一样，就会步调不一致。
        if tmpId >= 23 and tmpId <= 28:
            vehi.setLength(m2p(4.5), True)
    return True
```

此处宽度设置为 True，表示车身宽度也等比例变化，如果为 False，则车身宽度不变

- **def shape(self, pIVehicle:Tessng.IVehicle, outShape:PySide2.QtGui.QPainterPath) -> bool: ...**

  车辆外型，用户可以用此方法改变车辆外观

  参数：

  [in] pIVehicle：车辆对象

  [in、out] outShape：车辆外形

  返回：如果返回 False，则忽略

- **def ref_beforeCalcLampColor(self, ref_keepOn:Tessng.objbool) -> bool: ...**

  计算信号灯色前的预处理。

  参数：

  [in、out] 是否断续计算

  返回：如果返回 True，且 keepOn 等于 False，TESS NG 不再计算信号灯色。

- **def calcLampColor(self, pSignalLamp:Tessng.ISignalLamp) -> bool: ...**

  计算信号灯的灯色。ISignalLamp 有设置信号灯颜色方法。

  参数：

  [in] pSignalLamp：信号灯对象；

返回值：

如果返回 True，表明用户已修改了信号灯颜色，TESS NG 不再计算灯色。

- **def reCalcToLeftLane(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

计算是否要左强制变道，TESS NG 在移动车辆时计算强制左变道的条件，当条件不足时让插件计算，如果返回值为 True，强制左变道。

参数：

[in] pIVehicle：车辆对象。

- **def reCalcToRightLane(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

计算是否要右强制变道，TESS NG 在先移动车辆时计算强制右变道的条件，当条件不足时让插件计算，如果返回值为 True，强制右变道。

参数：

[in] pIVehicle：车辆对象

- **def reCalcToLeftFreely(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

计算是否要自由左变道。TESS NG 在移动车辆时计算自由左变道条件，当条件不足时让插件计算，如果返回值为 True，自由左变道。

- **def reCalcToRightFreely(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

计算是否要自由右变道。TESS NG 在移动车辆时计算自由右变道条件，当条件不足时让插件计算，如果返回值为 True，自由右变道。

参数：

[in] pIVehicle：车辆对象

- **def reCalcDismissChangeLane(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

重新计算是否撤销变道，通过 pIVehicle 获取到自身条件数据及当前周边环境条件数据，判断是否要撤销正在进行的变道。

参数：

[in] pIVehicle：车辆

返回：True 如果当前变道完成度不超过三分之一，则撤销当前变道行为；false 忽略。

- **def ref_reCalcdesirSpeed(self, pIVehicle:Tessng.IVehicle, ref_desirSpeed:Tessng.objreal) -> bool: ...**

重新计算期望速度，TESS NG 调用此方法时将车辆当前期望速度赋给 inOutDesirSpeed，如果需要，用户可在此方法重新计算期望速度，并赋给 inOutDesirSpeed。

参数：

[in] pIVehicle：车辆对象；

[in、out] inOutDesirSpeed：重新设置前后的车辆期望速度，单位：像素/秒；

举例：

范例中飞机的期望速度被重新计算，仿真前 5 秒期望速度为 0，飞机处于静止状态，5 至 10 秒飞机的期望速度是 20km/h，10 秒后期望速度是 40km/h。这段代码展示如何通过重新设置期望速度来控制车速。代码如下：

```
bool MySimulator::reCalcdesirSpeed(IVehicle *pIVehicle, qreal &inOutDesirSpeed) {
    long tmpId = pIVehicle->id() % 100000;
    QString roadName = pIVehicle->roadName();
    if (roadName == QString::fromLocal8Bit("曹安路")) {
        if (tmpId <= mrSquareVehiCount) {
            //当前已仿真时间
            long simuTime =
gpTessInterface->simuInterface()->simuTimeIntervalWithAcceMutiples();
            if (simuTime < 5 * 1000) {
                inOutDesirSpeed =  0;
            }
            else if (simuTime < 10 * 1000) {
                inOutDesirSpeed = m2p(20 / 3.6);
            }
            else {
                inOutDesirSpeed = m2p(40 / 3.6);
            }
            return true;
        }
    }
    return false;
}
```

- **def ref_reSetFollowingParam(self, pIVehicle:Tessng.IVehicle, ref_inOutSafeInterval:Tessng.objreal, ref_inOutSafeDistance:Tessng.objreal) -> bool: ...**

重新设置跟驰模型的安全间距和安全时距。

参数：

[in] pIVehicle：车辆对象；

[in、out] inOutSafeInterval：安全时距，单位：秒；

[in、out] inOutSafeDistance：安全间距：单位：像素；

举例：

范例将第二条连接段上的车辆跟车安全间距设为 30 米。代码如下：

```
def ref_reSetFollowingParam(self, vehi, ref_inOutSi, ref_inOutSd):
    roadName = vehi.roadName()
    if roadName == "连接段 2":
        ref_inOutSd.value = m2p(30);
        return True
    return False
```

- **def reSetFollowingParams(self) -> typing.List: ...**

重新设置跟驰模型参数，影响所有车辆。此方法被 TESS NG 调用，用返回的跟驰模型取代当前仿真正在采用的跟驰模型。

返回：跟驰参数列表，可对机动车和非机车的跟驰参数重新设置，设置以后会被采用，直到被新的参数所代替。

- **def ref_reSetSpeed(self, pIVehicle:Tessng.IVehicle, ref_inOutSpeed:Tessng.objreal) -> bool: ...**

重新设置车速。TESS NG 调用此方法时将当前计算所得车速赋给 **ref_inOutSpeed.value**，如果需要，用户可以在此方法重新计算车速并赋给 ref_inOutSpeed.value。

参数：

[in] pIVehicle：车辆对象；

[in、out] inOutSpeed：重新计算前后的车速，单位：像素/秒。

举例：

范例中跟在飞机后的车辆速度都被重新设置，设置的速度与飞机的速度相等。代码如下：

```
def ref_reSetSpeed(self, vehi, ref_inOutSpeed):
    tmpId = vehi.id() % 100000
    roadName = vehi.roadName()
    if roadName == "曹安公路":
        if tmpId == 1:
            self.mrSpeedOfPlane = vehi.currSpeed()
        elif tmpId >= 2 and tmpId <= self.mrSquareVehiCount:
            ref_inOutSpeed.value = self.mrSpeedOfPlane
        return True
    return False
```

- **def ref_beforeMergingToLane(self, pIVehicle:Tessng.IVehicle, ref_keepOn:Tessng.objbool) -> None: ...**

在“车道连接”上汇入车道前的计算，可以让 TESS NG 放弃汇入计算，以便于用户实现自己

的汇入逻辑。

参数：

[in] pIVehicle：车辆对象；

[out] **ref_keepOn**：是否放弃，默认为 True。赋值 **ref_keepOn.value** 为 False，TESSNG 则放弃汇入。

- **def afterOneStep(self) -> None: ...**

一个计算批次后的计算，这个时候所有车辆都完成同一个批次的计算。通常在这个方法中获取所有车辆轨迹、检测器数据、进行必要的小计等。在这个方法中进行的计算基本不影响仿真结果的一致性，但效率不高，如果计算量大对仿真效率会有影响。

举例：

范例中在这个方法中获取车辆对象和轨迹等信息。代码如下：

```
def afterOneStep(self):
    #= == == == == == =以下是获取一些仿真过程数据的方法 == == == == == ==
    # TESSNG 顶层接口
    iface = tessngIFace()
    # TESSNG 仿真子接口
    simuiface = iface.simuInterface()
    # TESSNG 路网子接口
    netiface = iface.netInterface()
    # 当前仿真计算批次
    batchNum = simuiface.batchNumber()
    # 当前已仿真时间，单位：毫秒
    simuTime = simuiface.simuTimeIntervalWithAcceMutiples()
    # 开始仿真的现实时间
    startRealtime = simuiface.startMSecsSinceEpoch()
    # 当前正在运行车辆列表
    vehis = simuiface.allVehiStarted()
```

- **def duringOneStep(self) -> None: ...**

该方法在各个线程进行同一批次的计算过程中调用，这时存在部分车辆计算完成，部分车辆仍在计算过程中。这个方法中的计算不够安全，但效率较高。

- **def candidateLaneConnectors(self, pIVehicle:Tessng.IVehicle, lInLC:typing.Sequence) -> typing.List: ...**

计算当车辆离开路段时后续可经过的“车道连接”，lInLC 是已计算出的当前车道可达的所有“车道连接”，用户可以从中筛选或重新计算。

参数：

[in] pIVehicle 当前车辆

[in] lInLC：TESS NG 计算出的后续可达“车道连接”列表

返回：用户确定的后续可达“车道连接”列表

- **def ref_beforeNextPoint(self, pIVehicle:Tessng.IVehicle, ref_keepOn:Tessng.objbool) -> None: ...**

计算车辆移动到下一点前的操作，用户可以通过此方法让 TESSNG 放弃对指定车辆到下一点的计算。

参数：

[in] pIVehicle：车辆对象；

[out] keepOn：是否继续，默认为 True，如果 keepOn 赋值为 false，TESSNG 放弃移动到下一点的计算，但不移出路网。

- **def calcLimitedLaneNumber(self, pIVehicle:Tessng.IVehicle) -> typing.List: ...**

计算限制车道序号：如管制、危险等，最右侧编号为 0。

参数：

[in] pVehicle：车辆对象；

返回：车道序号集，保存车辆不可以驰入的车道序号。

- **def ref_calcSpeedLimitByLane(self, pILink:Tessng.ILink, laneNumber:int, ref_outSpeed:Tessng.objreal) -> bool: ...**

由车道确定的限制车速（最高速度，公里/小时）,laneNumber:车道序号，最右侧编号为 0。

- **def ref_calcMaxLimitedSpeed(self, pIVehicle:Tessng.IVehicle, ref_inOutLimitedSpeed:Tessng.objreal) -> bool: ...**

重新计算车辆当前最大限速，不受道路限速的影响。在没有插件干预的情况下，车辆速度大于道路限度时按道路最大限速行驶，在此方法的干预下，可以提高限速，让车辆大于道路限速行驶。

TESS NG 调用此方法时将当前最高限速赋给 inOutLimitedSpeed，如果需要，用户可以在方法里重新设置 inOutLimitedSpeed 值。

参数：

[in] pIVehicle：车辆对象；

[in、out] inOutLimitedSpeed：计算前后的最大限速，单位：像素/秒。

返回结果：

如果返回 false 则忽略，否则取 inOutLimitedSpeed 为当前道路最大限速：

- **def ref_calcChangeLaneSafeDist(self, pIVehicle:Tessng.IVehicle, ref_dist:Tessng.objreal) -> bool: ...**

计算安全变道距离。

参数：

[in] pIVehicle：车辆，计算该车辆安全变道距离。

[in、out] dist：安全变道距离，dist.value 保存了 TESSNG 已算得的安全变道距离，用户可以在此方法重新计算。

返回：False 忽略，True TESS NG 取 dist.value 作为安全变道距离

- **def afterStep(self, pIVehicle:Tessng.IVehicle) -> None: ...**

完成车辆 pIVehicle“一个批次计算”后的处理。可以在此获取车辆当前信息，如当前道路、位置、方向角、速度、期望速度、前后左右车辆等。

参数：

[in] pIVehicle：车辆对象；

- **def ref_calcAcce(self, pIVehicle:Tessng.IVehicle, ref_acce:Tessng.objreal) -> bool: ...**

计算加速度

[in] pIVehicle：待计算加速度的车辆

[out] ref_acce：计算结果，单位：像素/秒^2

返回：False 忽略，True 则 TES NG 用调用此方法后所得 ref_acce.value 作为当前车辆的加速度。

- **def ref_reSetAcce(self, pIVehicle:Tessng.IVehicle, ref_inOutAcce:Tessng.objreal) -> bool: ...**

重新计算加速度。TESS NG 调用此方法时将当前计算所得加速度赋给 inOutAcce，如果需要，用户可以在此方法中重新计算加速度并赋给 ref_inOutAcce.value。

参数：

[in] pIVehicle：车辆对象

[in、out] ref_inOutAcce：重新计算前及计算后的加速度，单位：像素/秒^2

返回结果：

如果返回 False 则忽略，如果返回 True，则将 inOutAcce 作为当前加速度。

举例：

范例中车辆在第一条连接段上的加速度被重新计算。代码如下：

```
def ref_reSetAcce(self, vehi, inOutAcce):
    roadName = vehi.roadName()
    if roadName == "连接段 1":
        if vehi.currSpeed() > m2p(20 / 3.6):
            inOutAcce.value = m2p(-5)
            return True
        elif vehi.currSpeed() > m2p(20 / 3.6):
            inOutAcce.value = m2p(-1)
            return True
    return False
```

- **def ref_paintVehicleWithRotation(self, pIVehicle:Tessng.IVehicle, painter:PySide2.QtGui.QPainter, ref_inOutRotation:Tessng.objreal) -> bool: ...**

  以设定的角度绘制车辆

  参数：

  [in] pIVehicle，车辆对象

  [in] painter：QT 的 QPainter 对象

  [in、out] ref_inOutRotation：角度，TESS NG 在调用此方法时传入车辆的旋转角，这个方法内部可以修改这个角度，改变 TESS NG 计算结果

  返回：如果 True，TESS NG 不再绘制，否则 TESS NG 按原有规则绘制车辆。

- **def paintVehicle(self, pIVehicle:Tessng.IVehicle, painter:PySide2.QtGui.QPainter) -> bool: ...**

  绘制车辆

  参数：

  [in] pIVehicle，要重绘制的车辆

  [in] painter，QPainter 对象

  返回：如果返回 True，TESS NG 不再绘制，否则 TESS NG 按原有规则绘制车辆。

- **def rePaintVehicle(self, pIVehicle:Tessng.IVehicle, painter:PySide2.QtGui.QPainter) -> None: ...**

  绘制车辆后的再绘制，客户可在此方法增加绘制内容

  参数：

  [in] pIVehicle，要重绘制的车辆

  [in] painter，QPainter 对象

- **def ref_reCalcAngle(self, pIVehicle:Tessng.IVehicle, ref_outAngle:Tessng.objreal) -> bool: ...**

  重新计算角度。TESS NG 调用此方法时将当前算得的角度赋给 ref_outAngle.value，如果需

要，用户可在此方法中重新计算车辆角度，并将算得的角度赋给 ref_outAngle.value。

参数：

[in] pIVehicle：车辆对象；

[in、out] ref_outAngle：重新计算前后角度，北向 0 度顺时针，一周 360 度

- **def isStopDriving(self, pIVehicle:Tessng.IVehicle) -> bool: ...**

是否停车运行，TESS NG 在计算下一点位置后调用，判断是否要停止车辆 pIVehicle 的运行。

参数：

[in] pIVehicle：车辆对象；

返回结果：

如果返回 True，则停止该车辆运行，移出路网。

- **def beforeStopVehicle(self, pIVehicle:Tessng.IVehicle) -> None: ...**

车辆停止运行前的处理。

参数：

[in] pIVehicle：车辆对象；

- **def afterStopVehicle(self, pIVehicle:Tessng.IVehicle) -> None: ...**

车辆停止运行后的处理

参数：

[in] pIVehicle：车辆对象。

- **def vehiRunInfo(self, pIVehicle:Tessng.IVehicle) -> str: ...**

车辆运行信息。在仿真过程中如果某辆车被单选，按 ctrl+i 会弹出被单选车辆运行状态，文本框中的“其它信息”就是当前方法返回的字符串，开发者可以借此对实现的业务逻辑进行了解，用户可以了解仿真过程中具体车辆的一些特殊信息。

图 9 车辆运行过程属性窗口

5. API 运用示范

5.1 TessInterface 及其子接口运用示范

5.1.1 增加菜单及菜单项

可以在插件 init()方法中创建菜单及菜单项，代码如下:

```
iface = tessngIFace()
menuBar = iface.guiInterface().menuBar()
menu = QMenu(menuBar)
menu.setObjectName("menuExample")
menuBar.addAction(menu.menuAction())
menu.setTitle("范例菜单")
actionOk = menu.addAction("范例菜单项")
actionOk.setCheckable(True)
actionOk.triggered.connect(self.examleWindow.isOk)
```

结果如下：

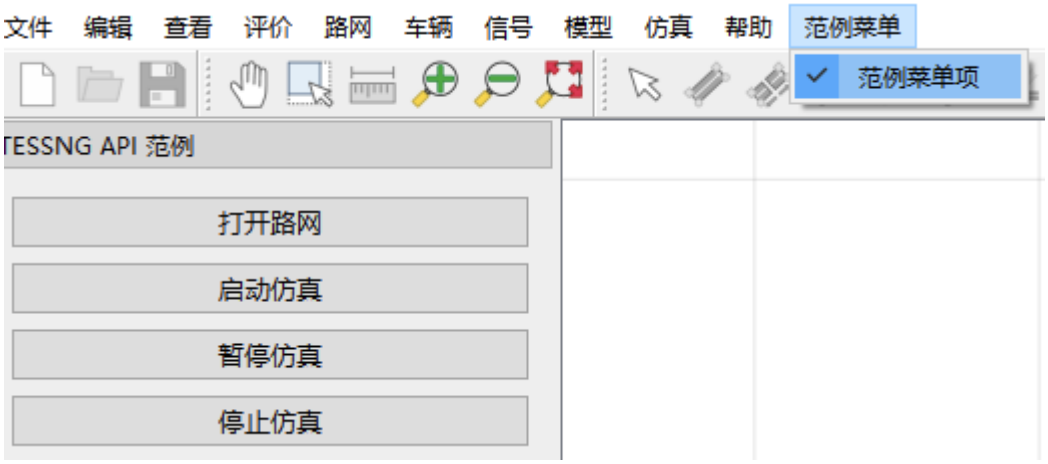


图 10 创建范例菜单

点击菜单项“范例菜单项”弹出对话框：

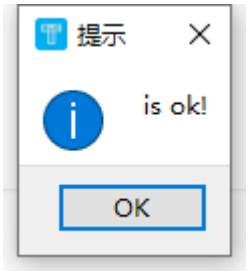


图 11 范例菜单点击效果

5.1.2 获取车辆在路网二维平面上所有可移动的轨迹

路网二维平面主要由路段连接段组成，路段由路段自身和车道组成，连接段由连接段自身和“车道连接”组成，路段、车道、“车道连接”都存在中心线断点序列，车辆都是在这些点构成的线上移动的。取得这些断点序列的步骤如下：

1）获取所有路段；

2）获取路段的所有车道；

3）获取车道的中心线断点序列

4）获取所有连接段

5）获取连接段的所有“车道连接”

6）获取“车道连接”中心线断点序列

5.2 TessPlugin 及其子接口运用示范

5.2.1 增加车辆显示内容

下图展示自动驾驶车辆，雷达探测范围 100 米。

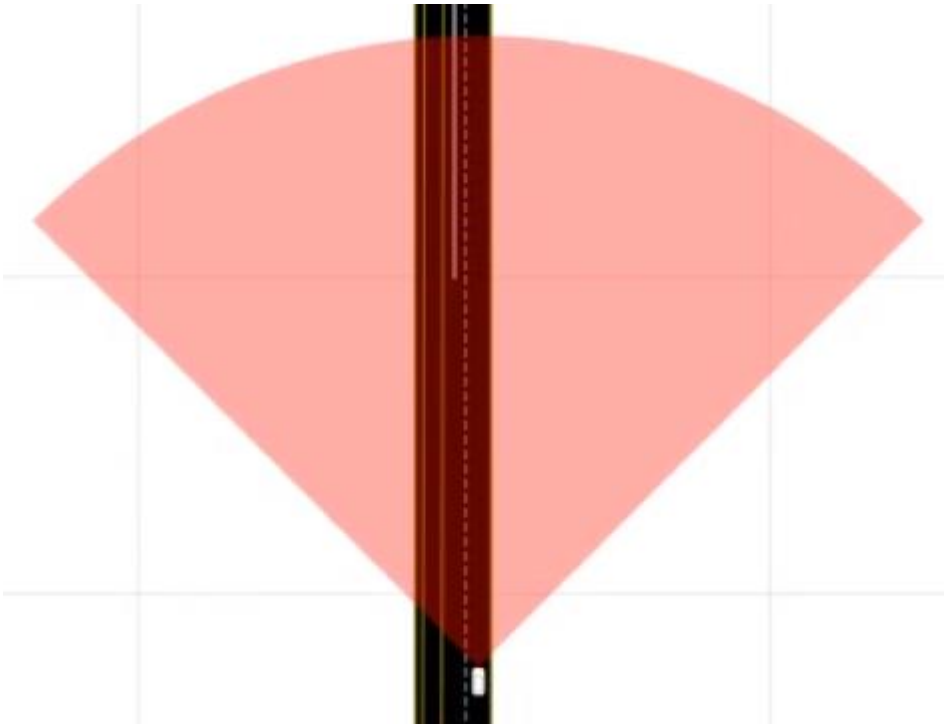


图 12 增加车辆显示内容

为了增添车辆展示的内容，可以实现接口 PyCustomerSimulator 的如下方法

```
def rePaintVehicle(self, pIVehicle:Tessng.IVehicle, painter:PySide2.QtGui.QPainter) -> None: ...
```

5.2.2 倒车入库

在实现自动泊车的仿真及调度时需要展示车辆倒车入库的过程，这个过程可让车辆顺向前移，

将车辆旋转 180 度，展现的便是倒着行驶状态。

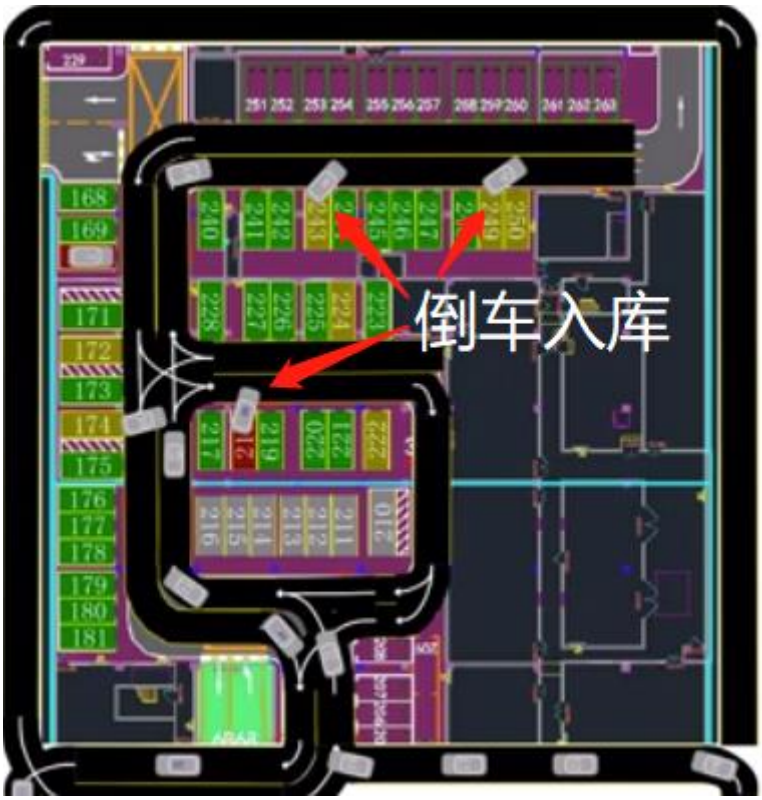


图 13 车辆倒车入库场景

为了实现倒着行驶效果，可以实现 PyCustomerSimulator 的如下方法

```
def ref_reCalcAngle(self, pIVehicle:Tessng.IVehicle, ref_outAngle:Tessng.objreal) -> bool: ...
```

5.3 循环仿真

5.3.1 基本原理

对加载的路网启动仿真后，不需要人工干预的情况下重复仿真指定次数，且每次仿真前自动进行参数设置，仿真后输出仿真结果到文件。

1）、仿真前判断当前仿真路网与上次仿真路网是否相同，如果不同设置仿真次数为 1；

2）、仿真结束后判断仿真次数是否小于指定次数，如果小于指定次数则发送要求仿真的消息给 TESSNG 主窗体，TESSNG 主窗体启动仿真。

版权声明：

软件全称：TESS NG 微观交通仿真软件

软件简称：TESS NG

版本号：V2.1.0

公司地址：上海市嘉定区曹安公路 4801 号 B 区南楼 604-605

使用的相关问题可以通过官网 www.jidatraffic.com 加入用户交流群讨论。

技术交流：胡立新 博客：https://blog.csdn.net/nlhlx

联系人/使用反馈：刘启远 电话/微信:15221455702

联系人/使用反馈：章洺浩 电话/微信:13122212617

软件销售：杨瑶 电话/微信：+86-13889919022

邮箱：jidatraffic@126.com

软件官网：https://www.jidatraffic.com