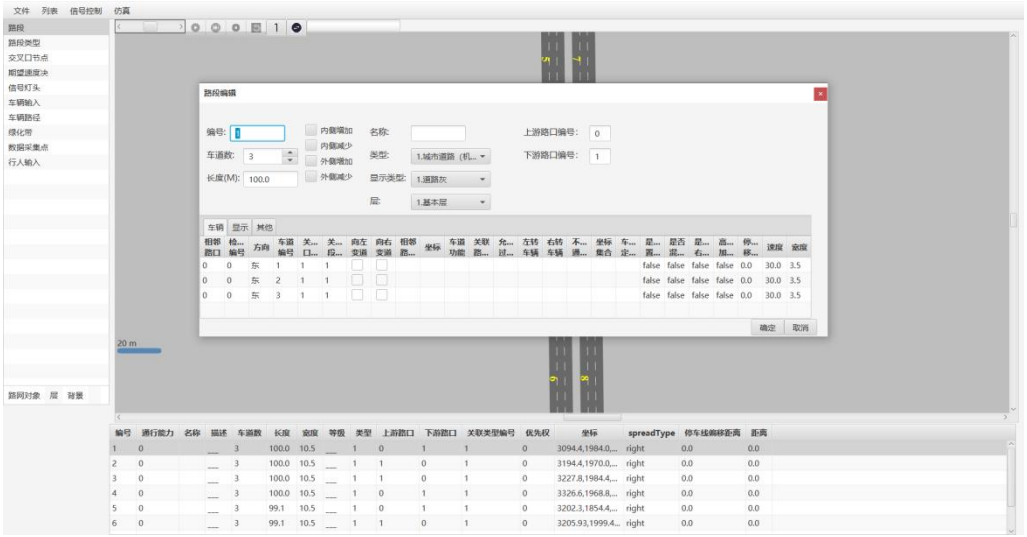
单交叉口的强化学习信号控制案例

本案例通过应用强化学习算法，优化单交叉口的信号控制策略，以提高交通流的效率。我们将以单交叉口为例，详细介绍如何通过仿真软件和强化学习算法，实现对交叉口信号灯的智能控制。案例中将特别展示 interface 模块的作用，该模块作为 Python 代码与仿真程序之间的桥梁，负责数据的交互和命令的执行。

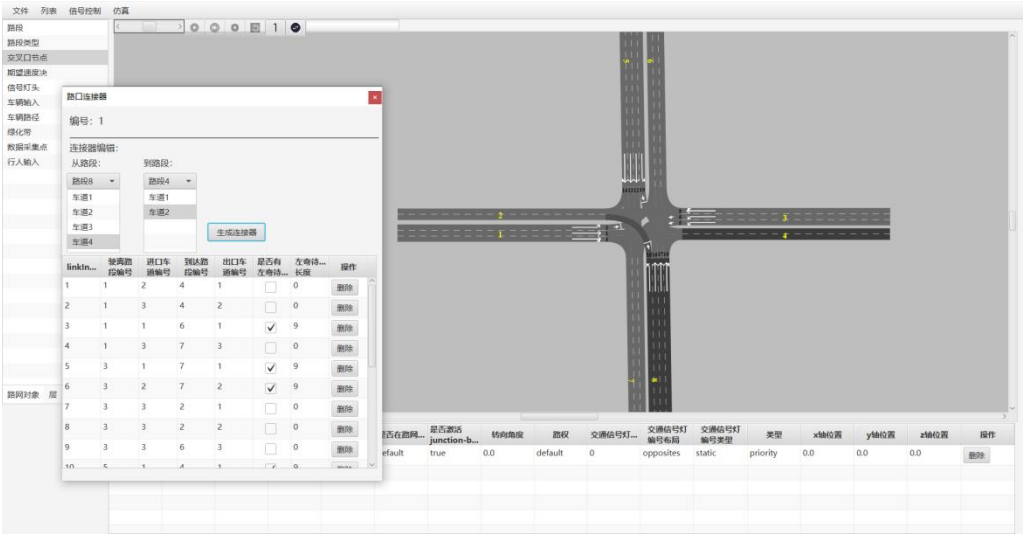
1. 环境搭建与仿真准备

1.1. 建立路网

点击左侧列表“路段“属性，右键选择添加路段，指定路段长度和车道数量。可在路段上右键点击直接生成反向路段。

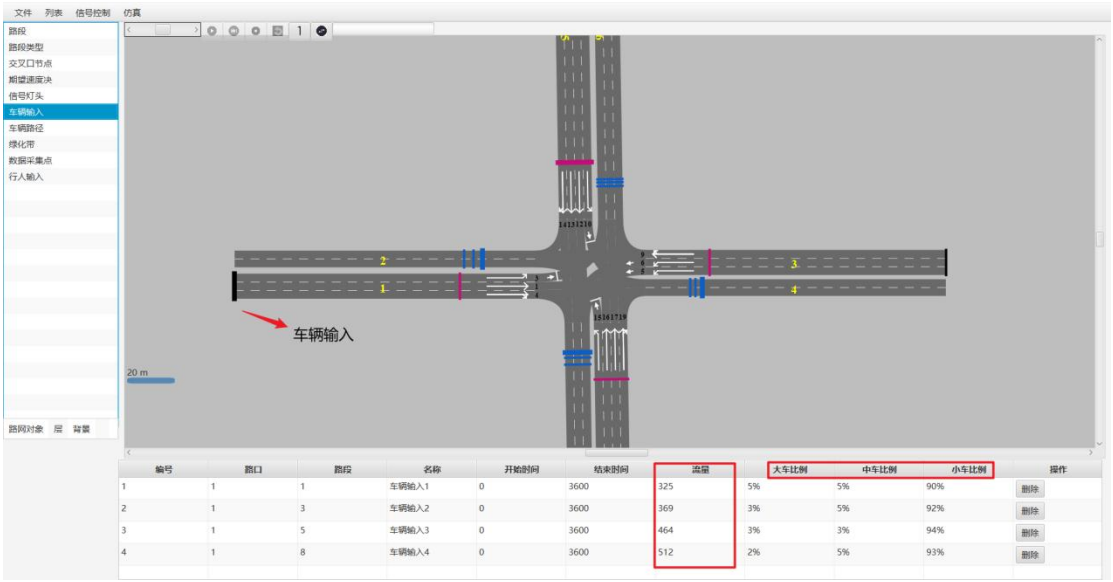


点击左侧列表“交叉口节点“属性，框选目前的交叉口所有路段，右键选择路口连接器，选择对应的路段上对应的车道生成连接器。可在路口连接器界面设置待转区域和待转区长度。



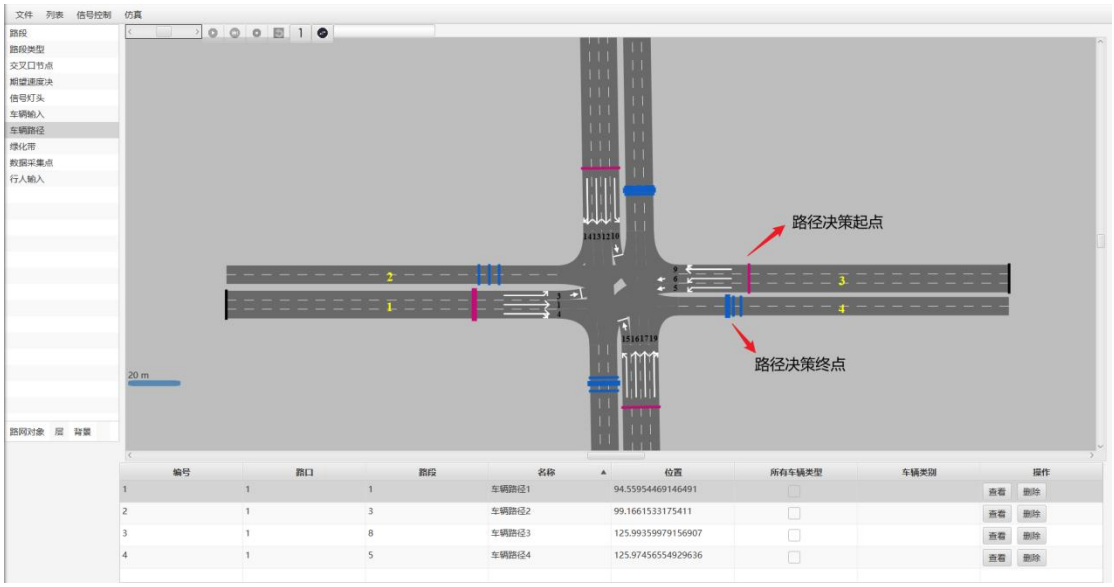
1.2. 车辆输入

点击左侧列表“车辆输入”属性，在车辆驶入路段右键生成车辆输入，并在下方属性面板输入开始时间、结束时间、流量和车辆比例。



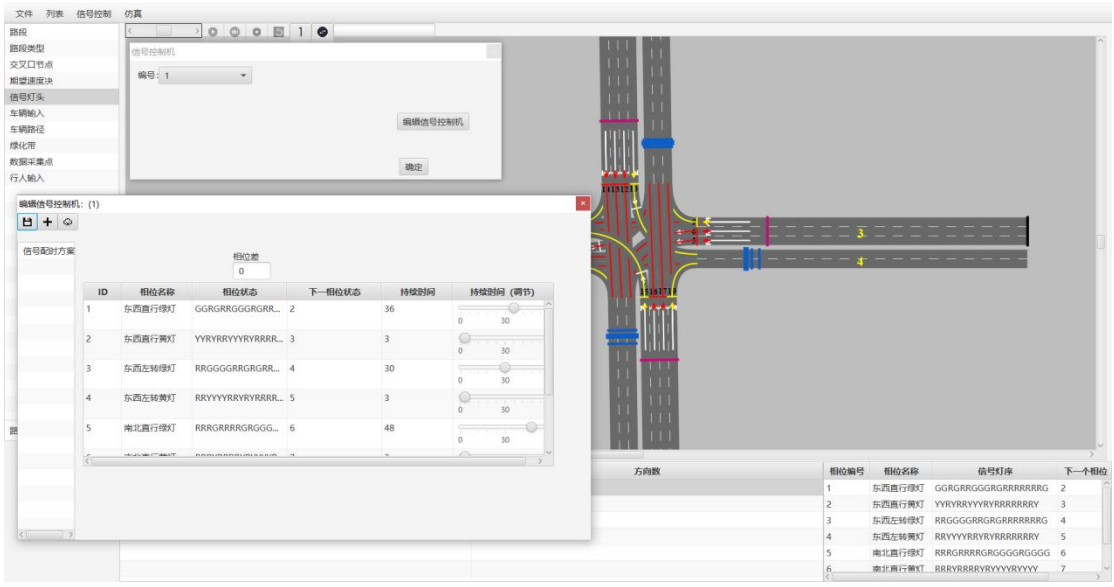
1.3. 车辆路径

点击左侧列表“车辆路径”属性，在车辆驶入路段的进口道处右键设置路径决策起点（红色），并在对应的路段右键设置路径决策终点（蓝色）。点击查看按钮，可设置左转、右转、直行方向的相对车流量。设置结束后左键双击空白处，结束该路段的车辆路径设置并继续完成后续路段的车辆路径设置。

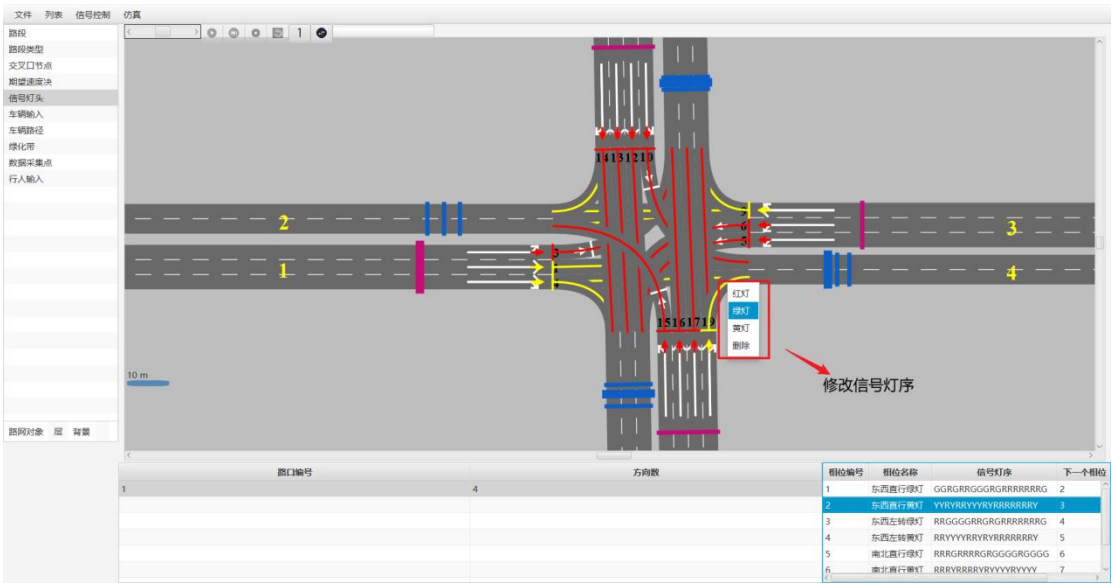


1.4. 信号配时

点击上方“信号控制”，选择“信号控制机”，点击“编辑信号控制机”，点击左上角加号按钮，编辑各相位名称、相位状态、下一相位状态和持续时间。



点击左侧列表“信号灯头”，可以在右下方查看每个相位对应的信号灯状态，并检验是否设置正确。选中右下方相位编号，在连接器上右键点击，可直接修改信号灯序。



以上步骤在提供路网时可以略过。

2. 强化学习模型训练

2.1. 模型初始化

初始化强化学习模型，包括状态空间、动作空间、奖励函数等；初始化记忆池；初始化仿真，初始化训练过程。

python 参考代码：

```
class DRL_Model():
    def __init__(self, num_layers, width, batch_size, learning_rate, input_dim, output_dim, tau, num_atoms):
class Memory:
    def __init__(self, capacity, size_min):
class Simulation:
    def __init__(self, Model, Memory, gamma, max_steps, green_duration, yellow_duration, num_states, num_actions):
class Training:
    def __init__(self, Model, Memory, training_epochs, update_epochs, batch_size):
```

2.2. 仿真与训练

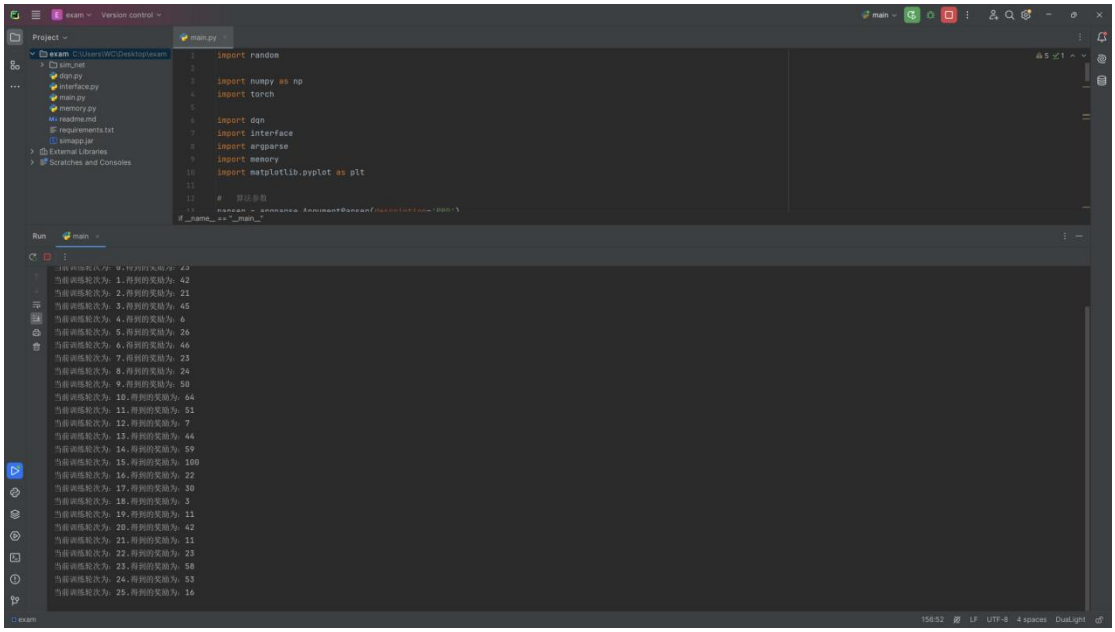
导入接口包中的 interface，按照接口包中的需求配置环境。

配置之前画的路网文件的路径和配时文件(若无配时文件，可以为空字符串)，jarpath 和 jvmpath 在提供的接口包中，配置好路径即可。若提供的 jar 包是有界面版本，则 set_gui_model 设置为 true。切在开始训练时需要先打开仿真软件。

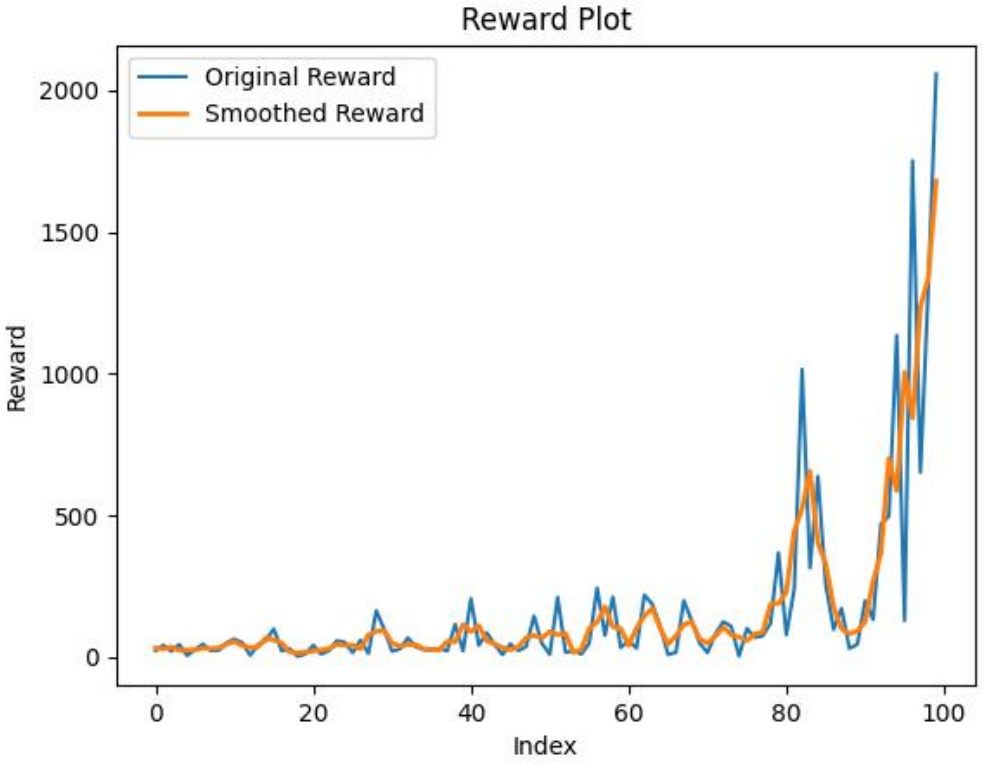
```
if __name__ == "__main__":
    #    连接仿真软件，注意顺序，路径。
    interface.set_gui_mode(True)
    jarPath = "C:/Users/Administrator/Desktop/LiikeSim-research/simapp-1.0.2.jar"
    jvmPath = "C:/Users/Administrator/Desktop/LiikeSim-research/jre/bin/server/jvm.dll"
    interface.connect(jvmPath, jarPath)
    net_file = "C:/Users/Administrator/Desktop/code/single intersection exam/sim_net/example_net.xml"
    time_file = "C:/Users/Administrator/Desktop/code/single intersection exam/sim_net/example_timing.xml"
    interface.start_simulation(net_file, time_file)
```

开始仿真时注意配时文件和路网文件都需要导入。开始训练。

带界面仿真时如果出现问题，可以改为绝对路径。



在训练过程中，控制台会不断打印信息。在训练结束后，出自动绘制一个曲线。如果曲线总体呈现上升趋势，训练成功。




```
while episode < total_episodes:
    print('\n----- Episode', str(episode + 1), 'of', str(total_episodes))
    epsilon = max(0.0, (1.0 - ((
                                episode + 1) / total_episodes)))  # set the epsilon for this episode according to epsilon-greedy policy
    # epsilon = 0
    simulation_time = Simulation.run(episode, epsilon)  # run the simulation
    training_time = Training.run()  # train the model
    print('Simulation time:', simulation_time, 's - Training time:', training_time, 's - Total:',
          round(simulation_time + training_time, 1), 's')
    episode += 1
```

2.3. interface 模块的应用

在于环境的交互中，使用 interface 提供的接口与仿真进行交互，使用到的接口包括:

1)set_gui_mode(gui_mode)

参数:

gui_mode (bool): 如果为 True，则启用 GUI 模式，否则启用无界面模式。

作用: 设置应用程序的 GUI 模式。

返回值: 无 (None)。

2)connect(jvm_path, jar_path)（有界面模式其实无需调用该函数连接）

参数:

jvm_path (str): JVM 的路径。

jar_path (str): Java 类库的路径。

作用: 连接到 Java 虚拟机并初始化所需的 Java 类。

返回值: 无 (None)。

3)start_simulation(net_xml, timing_xml)

参数:
net_xml (str): 路网 XML 文件路径。
timing_xml (str): 配时 XML 文件路径。
作用: 启动交通模拟。
返回值: 无 (None)。

4)simulation_step()
作用: 执行模拟的单步操作，并根据模拟速度进行延时（有界面）。
返回值: 无 (None)。

5)simulation_restart()
作用: 重启模拟过程。
返回值: 无 (None)。

6)get_vehicle_id_list()
作用: 获取当前模拟中所有车辆的 ID 列表。
返回值: 车辆 ID 列表 (list)。

7)get_vehicle_edge_id(vehicle_id)
参数:
vehicle_id (str): 车辆的唯一标识符。
作用: 根据车辆 ID 获取车辆所在的路段 ID。
返回值: 车辆所在的路段 ID (int)。

8)get_vehicle_lane_id(vehicle_id)
参数:
vehicle_id (str): 车辆的唯一标识符。
作用: 根据车辆 ID 获取车辆所在车道的 ID。
返回值: 车辆所在车道的 ID (int)。

9)get_vehicle_max_decel(vehicle_id)
参数:
vehicle_id (str): 车辆的唯一标识符。
作用: 获取车辆的最大减速度。
返回值: 车辆的最大减速度 (float)。

10)get_vehicle_accumulated_waiting_time(vehicle_id)
参数:
vehicle_id (str): 车辆的唯一标识符。
作用: 获取车辆累积的等待时间。
返回值: 车辆的累积等待时间 (int)。

11)get_edge_halting_number(edge_id)

参数:

edge_id (int): 路段的唯一标识符。

作用: 获取指定路段上停止的车辆数量。

返回值: 停止在路段上的车辆数量 (int)。