摘要：由于固定时长的红绿灯控制已经远远满足不了现代道路交通需求，因此提出一种基于不定顺序信号灯控制算法与拥堵解决算法的智能交通灯控制系统，用于提升路口通行效率。该系统通过检测车流量,灵活地调节交通灯的通行时间,实现实时控制交通流量、缓解交通拥堵的目的。本系统使用 Unity 仿真，模拟显示车辆运行环境，提供新建道路、车辆等功能，方便用户测试智能交通灯系统功能。

关键词：智能交通灯、交通控制、Unity3D

Simulation of intelligent traffic light system based on Unity

Peng Hangqi Yang Jiale
China University of Geosciences

Abstract: Since the fixed-duration traffic light control is far from meeting the needs of modern road traffic, an intelligent traffic light control system based on an indeterminate signal light control algorithm and a congestion resolution algorithm is proposed to improve the efficiency of intersection traffic. The system can flexibly adjust the passing time of traffic lights by detecting traffic flow, so as to realize the purpose of real-time control of traffic flow and alleviation of traffic congestion. This system uses Unity simulation to simulate and display the vehicle operating environment, provide functions such as new roads and vehicles, and facilitate users to test the functions of the intelligent traffic light system.

Keywords: intelligent traffic lights; traffic control; Unity3D

目录

一、概述

随着经济的快速发展，城市机动车数量迅速增加，交通拥堵问题日益严重，导致交通事故和环境污染等负面影响日益突出。城市交通问题直接制约着城市的建设和经济的发展，与人们的日常生活息息相关。如何有效缓解城市交通拥堵，合理地处理人与车、车与车的关系，在城市交通中已经是特别常见的问题。交通灯是交通系统的主要部分，在缓解交通拥堵方面起着至关重要的作用。如今，在车水马龙的道路上交通灯随处可见，但交通灯变更时间长、效率低，造成很多资源浪费。随着计算机视觉技术和网络技术的进步，以及相关理论的不断完善，智能交通系统迅速发展 ITS（Intelligence Transportation System）的研究和设计成为新的热点。

本文使用不定顺序信号灯控制算法与禁行策略，结合 Unity 仿真来设计交通信号灯控制系统。

二、相关原理与技术

2.1 Unity3D

2.1.1 脚本运行逻辑

Unity 允许使用脚本来自行创建组件。使用脚本可以触发游戏事件，随时修改组件属性，并以所需的任何方式响应用户的输入。

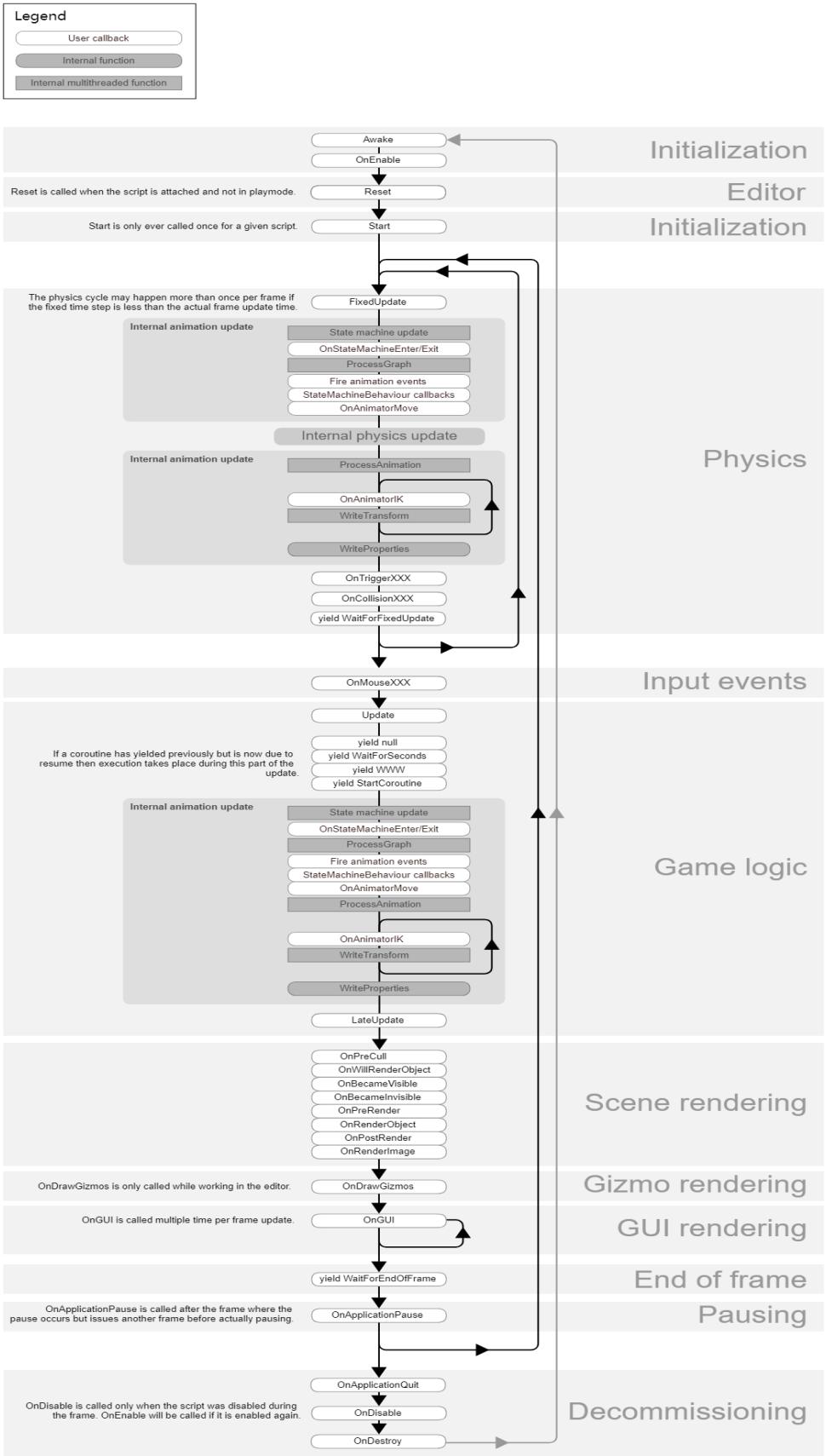

图 1 脚本生命周期流程图

2.1.2 用户界面逻辑

Unity3D 中，画布 (Canvas) 组件表示进行 UI 布局和渲染的抽象空间。所有 UI 元素都必须是附加了画布组件的游戏对象的子对象。从菜单 (GameObject > Create UI) 创建 UI 元素对象时,如果场景中没有画布 (Canvas) 对象，则会自动创建该对象。

Canvas 下 Render Mode 属性可以决定其 UI 在屏幕上或作为 3D 空间对象进行渲染的方式：Screen Space - Overlay 模式下画布会进行缩放来适应屏幕，然后直接渲染而不参考场景或摄像机（即使场景中根本没有摄像机，也会渲染 UI)。如果更改屏幕的大小或分辨率，则 UI 将自动重新缩放进行适应。UI 将绘制在所有其他图形(例如摄像机视图)上,此模式适用于用户主界面。World Space 模式下，画布作为三维物体展示，在本系统中适用于显示红绿灯倒计时。

本系统中主要使用了可视组件——文本(Text)、遮罩(Mask)与交互组件——按钮(Button)、输入字段(Input Field)。其中按钮可以响应点击事件，实现相应功能。

三、需求分析

功能性需求

1. **道路场景搭建**:我们需要完成道路场景的搭建,并且为了追求场景的真实性，我们将建立包括但不限于以下模型：道路、十字路口、交通灯、道路边的装饰并设计仿真地形。道路需要设计成左转、直行、右转三车道的规格。
2. **车辆建模**：需要完成对车辆模型的搭建。
3. **车辆状态管理**：我们需要车辆的数量和车速等动态且仿真地增加或减少。
4. **车辆仿真行驶**： 包括但不限于以下功能：（1）车辆直线行驶过程仿真；（2）车辆转弯过程仿真；（3）车速变化过程（包括车辆起步和停止）仿真；（4）车辆沿道路行驶不会跑出道路；（5）车辆遵守“红灯停，绿灯行”的交通规

则；（6）车辆在交通灯前能根据行程需求选择走左转、直行、右转车道。

5. **交通灯智能管理：**首先交通灯应该能控制道路上车辆的行驶。同时交通灯的转换频率能够根据车流量智能的变化而不是采用预先设定的固定交通控制方案。为了能够有效的调节整个交通系统的通行，不同的交通灯之间还应能进行联调联控，共同缓解道路的拥堵。
6. **交互管理系统:**用户应该可以自行设置整个交通系统的参数,包括但不限于：最高车速、车辆数量，交通灯数量（即十字路口数量）。
7. **视角移动功能：**允许用户使用方向键移动视角。
8. **用户图形界面：**我们需要制作一个友好的美观的用户图形界面供用户使用。

非功能性需求

1. **观感需求：**为了提高用户的使用体验：我们对界面设计做出以下规定：（1）局内设计风格保持一致；（2）用户确认以及用户提示，方式保持统一；（3）界面布局方便用户操作；（4）场景设计美观仿真；
2. **易用性：**系统应保证操作方便，符合日常操作习惯，按钮位置便于操作；提供容错处理，不能出现系统崩溃卡死等情况，错误应有直观的提示信息。
3. **性能需求：**为保证系统运行的流畅度并且降低对硬件设施的要求，应尽量减少计算资源的使用及内存的占用大小。如优化碰撞检测等。
4. **可维护性：**为保证系统的可维护性，代码应具有完备的注释且编码应遵循规范。并且应具有详细的用户手册。

四、概要设计

1.车辆行驶逻辑

要求车辆在非十字路口处可以直线行驶，同时若与前方车辆间隔低于一定距离后支持停车，并在与前方车辆拉开距离后启动。车辆在交通信号灯处支持转弯进入其他车道。

2.交通信号灯控制

支持按照不定顺序控制算法在红黄绿间转换，同时将信号传递给进入路口的车辆。

3.车辆遵守交通信号灯规则

车辆抵达路口处要求按照从交通信号灯处得到的信息行驶（红灯停，绿灯和黄灯继续行驶），在红灯转为绿灯后启动车辆。在拥堵情况下，还应能禁止右转的车辆通行。

4.动态调整车辆数量

支持用户通过点击道路选择新生成车辆初始点，并生成用户输入的车辆数；同时支持用户通过点击车辆来降低车辆数量。

5.动态新建道路

支持用户新建横/纵向车道，并在新十字路口新建交通信号灯。

6.调整车辆最大速度

支持用户将车辆最大速度调整为输入值。

五、详细设计

1.车辆行驶逻辑

车辆在正常行驶过程中，是无法感知到周围环境的，只会按当前速度匀速直线行驶。只有当与预先设置好的标志方块（或其他车辆）发生接触时（调用接触检测函数），才会感知到当前所处状况。例如，只有当车辆与标志左转的方块发

生接触时，车辆才会“感知”到下一步将会进行左转操作；与其他车辆碰撞也是同理。

由于车辆在正常行驶状态下只会保持匀速直线运动，因此转弯操作的实现也需要引入接触检测才能完成：场景中会预先设置一系列指引方块用来标记转弯路径。当车辆发生转弯时，需要选择转弯路径，然后依据在指引方块的指引完成转弯操作。

基于上述逻辑，本程序将道路建模为双向六车道。单向道路分为左转、右转、直行三车道，在交通信号灯路口，将默认车辆在对应车道上只能进行对应的操作。例如，小车在直行车道进入路口时只能选择直行而不能选择进行左转或右转。同时，在满足该规定的情况下，为了使车辆行驶的状况逼近现实，车辆进入路口后可以随机选择进入下一段道路时行驶的车道。也就是说车辆会在当前路口随机决定下个路口将要进行的操作。

2. 智能交通灯控制

根据对智能交通信号灯的理解，我们在《基于车流量检测的智能交通灯不定顺序控制算法研究》中提出的算法的基础上做出改进，设计出如下算法：

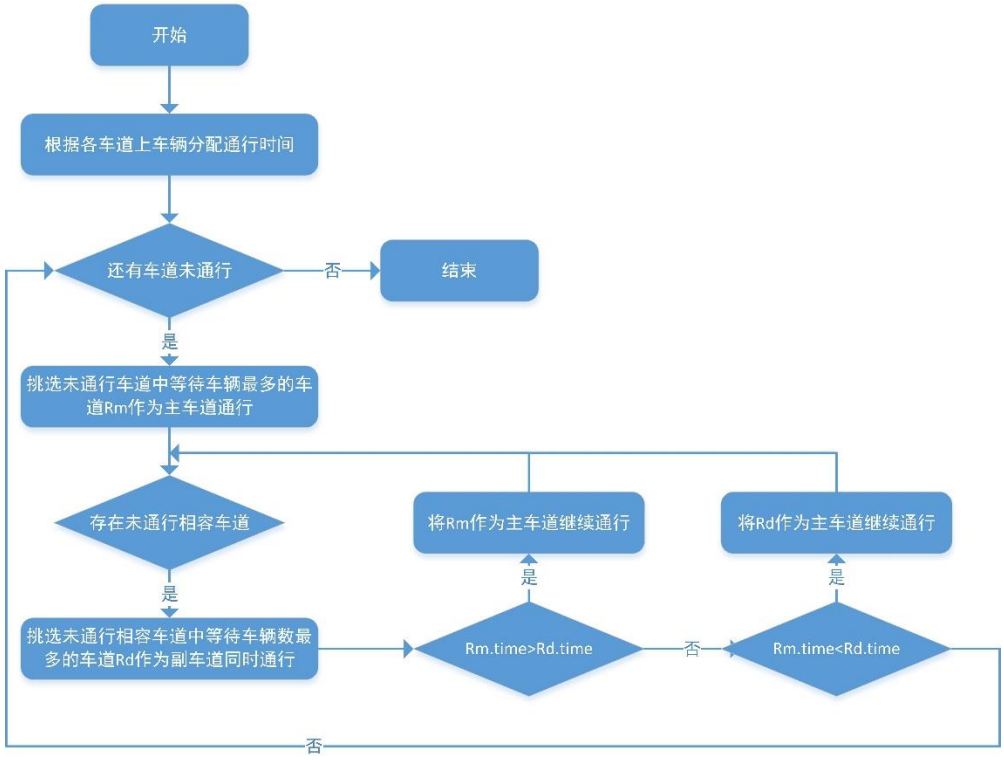

图 2 不定顺序控制算法流程图

我们改进点在于：

1）通行时间的分配

原算法只粗糙的设置了若干个通行时间的固定值，判定车辆数落在某个区间时，为其分配该区间对应的通行时间。而本算法则认为通行时间的分配可以更精细，通行时间T的计算公式如下：

$$T = mp + np \tag{1}$$

其中n为车道上的车辆数。m为固定参数$(m > 1)$，用于保证车辆数为0时，该车道仍能得到一定通行时间。p为原子时间，其计算公式如下：

$$p = \lambda \frac{s}{v_{max}} \tag{2}$$

其中s为车辆通过路口时经过的路程，v_{max}为车辆通过路口时的行驶速度，λ为固定参数$(\lambda > 1)$，用于延长通行时间以抵消车辆在非路口路段的行驶时间。

显然，该分配方案不仅精细的为本车道每辆车都分配通行时间，还考虑到了车辆通过路口时的速度。

2）通行顺序的确定

原算法中选择让车道上车辆数多的优先行驶，而本算法则考虑到这样一种情况：当该车道上大部分车辆未进入路口时，若让该车道先通行会浪费掉大量通行时间而其他车道上正在等待的车辆得不到通行。

因此，本算法为减少车辆平均等待时间，选择让路口等待车辆更多的车道先通行，这样就在一定程度上避开了上述情况的出现。

同时，智能交通灯算法应该能够处理拥堵。经过阅读文献我们了解到，当拥堵发生时，禁止车辆进入拥堵区域是一个能够有效处理拥堵的策略。因此，该算法将在拥堵发生时，对能够进入该路段的3车道（分别是从拥堵发生路段开始逆时针旋转的左转、直行、右转车道）实行禁行。此外，为了使拥堵尽快消散以防止拥堵传播，还应该让发生拥堵的车道优先通行。下文中，我们将优先通行的车道称为优先车道。

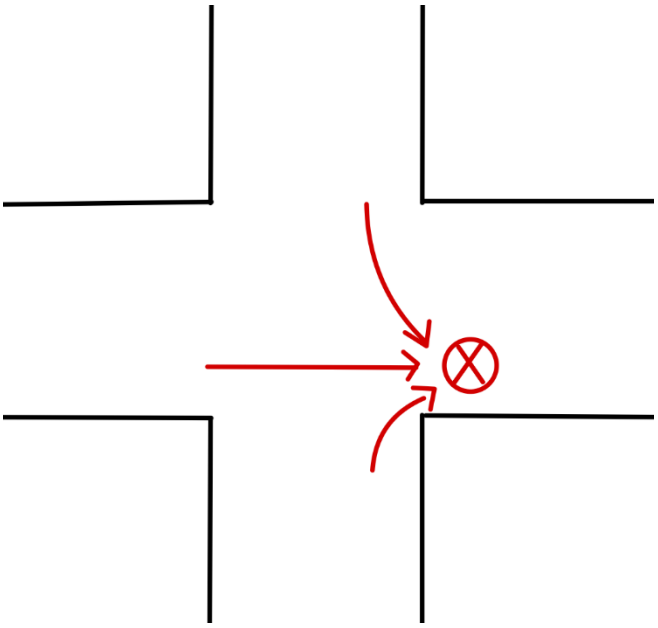

图 3 禁行车道

我们注意到，不定顺序控制算法是可以兼容上述策略从而达到处理拥堵的目的的，综上我们就得到了拥堵处理算法：应用禁行策略的不定顺序控制算法。

3. 车辆遵守交通信号灯规则

在完成上述设计后，显然还需要实现车辆在行驶过程中能够遵守交通信号灯规则的功能。根据小车行驶逻辑的设计，我们将当前车道是否可以通行的信息设置在小方块上（由于禁行策略的存在，标志右转的小方块也需要设置能否通行的信息）这样，只需在车辆与路口小方块发生接触时，即可判断应该停车还是继续通行，即车辆“感知”到了信号灯的状态。

4. 动态调整车辆数量

选择车道生产车辆：在用户选择此功能后，计算出用户点击位置在实际场景中的坐标，然后判断此坐标是不是仅在一条道路上，如是则计算出车辆的旋转方向与修正后的坐标。然后每个 2 秒调用 Instantiate 函数在目标位置实例化车辆即可实现该功能。

动态销毁车辆：用户选择此功能后，计算出用户点击位置在实际场景中的坐标，然后遍历小车数组，判断是否有车辆与点击位置处于同一车道且距离在±10

以内，有则删除该车辆。

5.动态新建道路

新建横向车道：在用户选择该功能后，计算出用户点击位置在实际场景中的坐标，在此坐标处调用 Instantiate 函数复制横向车道。然后计算出此横向车道与纵向车道的交叉点生成十字路口。

新增纵向车道：与新建横向车道类似。

6.调节车辆最大速度

在用户选择此功能后，遍历存放有所有车辆 Transform 的数组，逐个将 Car 下面脚本中的最大速度值修改为用户输入值。

六、系统实现

1.智能交通灯控制

不定顺序控制算法实行的前提在于需要得知各车道的总车辆数以及等待车辆数，基于车辆行驶逻辑，我们无法判定当前车辆所处的路段以及具体车道。因此我们需要借助车道首端与末端的小物块识别是否车辆驶入与驶出当前车道以计算车道上车辆数，每个车道维护这样一对小物块并将车道上车辆信息汇总到交通信号灯控制逻辑，这样交通灯控制逻辑就得知了各车道上的总车辆数。而等待车辆数则要根据小物块维护的等待车辆队列来得知，该队列具体实现细节将在车辆遵守交通信号灯规则这一小节中详细阐述。

除此以外，拥堵处理算法也需要交给交通灯来完成。首先需要判断拥堵，判断逻辑是每帧检测车道末端小物块上一次接触的车辆速度是否为 0，若车辆速度为 0 时与小物块距离小于一定值，则将判断该车道发生拥堵。

拥堵发生后，拥堵路口交通灯与拥堵路口上游路口交通灯的控制逻辑将被停用，交通灯交给主控逻辑控制来进行全局的调控，下文中我们将采用禁行和优先

通行的路口称为异常路口。为了方便维护异常路口的运转，主控逻辑为每个异常路口都维护了一个结构体，里面存放了应用禁行策略的不定顺序控制算法需要的所有数据，每一帧都对该结构体进行更新。下面我们将阐述如何将禁行策略与不定顺序控制算法进行兼容：

经过分析，我们可分为以下两种情况进行处理：①当异常路口不存在优先车道时（即只存在禁行车道），我们只需在原算法的基础上将禁行车道的通行时间设置为 0 即可；②当异常路口中存在一个或多个优先车道时，我们可随机选择一个优先车道将其通行时间设为无穷大，并且只为该车道的相容车道分配通行时间（禁行车道不分配时间），当相容车道通行时间均耗尽时将进入新周期重新分配通行时间。注意，优先通行的车道可能因为下游路口的拥堵而被禁行，由于禁行策略优先级高于优先策略，因此这样的车道我们认为是禁行车道而非优先车道。

当优先车道内车辆数小于 5 时，则判定该路段拥堵解除，取消优先策略以及上游车道禁行策略。若该路口无其他车道异常，则路口异常状态解除，重新将交通灯控制交还给本地控制逻辑。

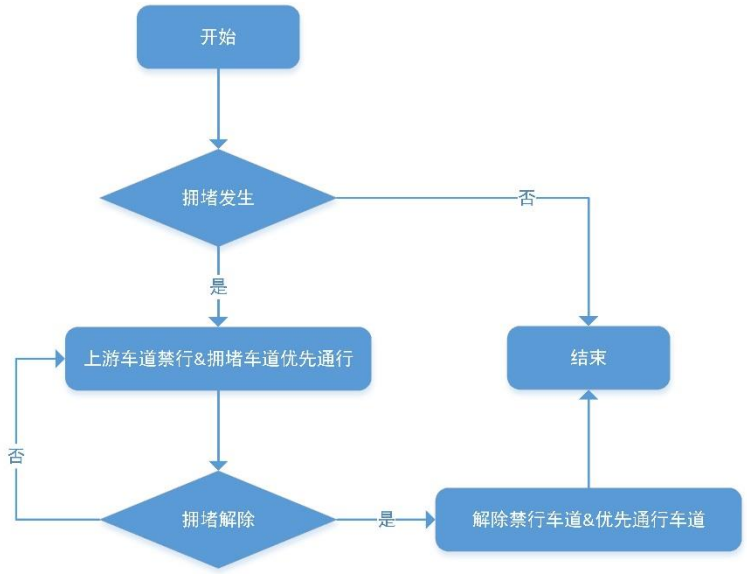

图 4 拥堵处理流程图

2. 车辆遵守交通信号灯规则

实现细节：基于车辆行驶逻辑的设计，若当前车辆因为红灯或前方等红灯车辆停车，其后将无法“感受”到信号灯的变化，即无法通过自身来完成车辆的启动操作。因此需要让其他物体（小物块）来对当前车辆进行“唤醒操作”，为了

实现上述功能，小物块需要维护一个等待车辆队列，当通行信息从禁止变为通行时，启动等待车辆队列中所有车辆，使其继续行驶。

综上，车辆遵守交通信号灯规则的完整逻辑如下：

当车辆行驶到路口时，若小物块允许通行，车辆正常通过路口；若小物块禁止通行，车辆把自身添加到小物块的等待车辆队列并保存小物块。若后方车辆检测到前方车辆停止，将获取前方车辆保存的小物块并进行同样的操作。当小物块允许通行后，将等待车辆队列全部弹出并调用该车辆启动函数，这样就完成了信号灯对车辆的“唤醒操作”。

3.动态调整车辆数量

UI 设计：在 UI 界面中设置了“调整车辆数量”按钮，点击后进入第二层 UI，其中有“点击选择车道新增车辆”与“销毁点击车辆”两个按钮。

数据支持：在 Canvas 下的脚本中维护了两个一维数组，分别存放横纵车道的中心点坐 x/z 坐标，数组的值即为横/纵车道的数量。另维护了一个存放车辆 Transform 的一维数组。

在按下“点击选择车道新增车辆”按钮时，程序开始监听用户鼠标右键。当用户按下并松开右键后，将摄像机与用户点击位置的射线与地面的交点作为目标位置。再判断目标位置是否仅在一条道路内，如是则对目标位置进行修正，使其位于车道中间，同时确定车辆的旋转方向，使其可以正常行驶，再创建一个输入框。用户可以在输入框中输入希望新增车辆数，按下回车后脚本记录该数据，并关闭输入框。最后，间隔 2 秒调用 Instantiate 函数在目标位置实例化车辆即可实现该功能。

在按下“销毁点击车辆”按钮时，程序也通过上述方法获取目标位置，然后遍历保存有车辆 Transform 的数组，查询是否有车辆与点击位置处于同一车道且距离在± 10以内，如有则在数组删除该车辆 Transform 并销毁车辆。

4.动态新建道路

UI 界面：在 UI 界面内设置了“新建道路”按钮，点击后进入第二层 UI，其中有“纵向新建道路”与“横向新建道路”两个按钮。

数据支持：在 Canvas 下 x 的脚本中维护了两个一维数组，分别存放横纵车道的中心点坐 x/z 坐标，数组的值即为横/纵车道的数量。

当用户点击 “新建横向车道”后，脚本开始监听用户鼠标和键盘。当用户鼠标左键点击后，计算摄像机和鼠标左键点击位置所在直线与地面的交点（记为 X 点），将此点的 z 坐标修改为 0，再使用 Instantiate 方法在此点生成一个横向车道的副本。在新建横向车道后，查看纵向车道数量（对应数组的长度），遍历数组取 X 点的 x，y 坐标与纵向数组中存放的数据（z 轴坐标）组成红绿灯的坐标，并新建红绿灯。

当用户点击 “新建纵向车道”后，与上述流程类似。

5.调节车辆最大速度

UI 界面：在 UI 界面内设置了“调整车速”按钮。

数据支持：在 Canvas 下的脚本中维护了一个存放车辆 Transform 的一维数组；在车辆 Car 下的数组中维护有车辆的最大速度。

在用户点击“调整车速”按钮后，创建一个输入框，用户可在输入框内输入希望的车速最大值，输入完成后按下回车，程序开始遍历存放车辆 Transform 的数组，逐个修改车辆下脚本内的最大速度。

七、测试

完成系统设计后，对整个系统进行测试。通过车辆延误时间这一指标对算法进行评价。

在道路负载较轻时（车辆数=50），平均车辆延误时间为 2.4s，未观测到拥堵发生；在道路负载适中时（车辆数=250），平均车辆延误时间为 10.8s，同样未观测到拥堵发生；而在道路负载较大时（车辆数=500）.平均车辆延误时间为 25.9s，观测到拥堵发生，拥堵处理算法能够有效运行防止拥堵传播加快拥堵消散。

通过结果得出结论：该算法能够在负载适中的情况下，在一定程度上阻止拥堵的发生。而当负载较大时，仍然能够发挥作用，阻止拥堵传播并加快拥堵消散。

在测试过程中还发现了以下问题：

1）在车辆行驶逻辑的设计过程中，我们取消了车辆的碰撞体积，导致车辆通过路口时会出现重合，一定程度上干扰了测试结果，使拥堵能够更快的消散。

2）车辆的启动和停止过程，过于生硬与现实生活不符，同样干扰了测试结果。

由于本次测试未能与使用传统交通灯控制算法的交通系统做出对比，缺乏对照，无法得出本算法相对于传统算法的优秀情况。但是经过大量测试发现，本算法能够确实地有效的阻止拥堵传播并加快拥堵消散。

八、总结

本次任务要求使用 Unity 进行开发，在项目开发的初期阶段，我们从零开始学习了 Unity 相关知识，让我们对 Unity 的运行逻辑有了一定的认识。同时，通过大量阅读 Unity 官方文档，使我们能顺利地解决相关问题，实现对应需求。除此以外，在脚本逻辑编写过程中，也帮助我们复习了 C#相关知识，使我们能够更熟练的运用 C#开发应用。

在构思智能交通灯算法的过程中，也查阅了大量的文献。从《基于车流量检测的智能交通灯不定顺序控制算法研究》中了解到相容车道的概念，并在原不定顺序控制算法的基础上与前期制作的交通仿真系统完成适配并做出改进。并从《城市道路交通拥堵传播规律及消散控制策略研究》中认识到禁行策略是一种能够有效阻止拥堵传播，加快拥堵消散的方法，并与改进的不定顺序控制算法相结合，最终设计出能够有效解决问题的智能交通灯控制算法。

九、参考文献

[1] 基于车流量检测的智能交通灯不定顺序控制算法研究 徐成

[2] 城市道路交通拥堵传播规律及消散控制策略研究 龙建成

[3] 突发事故下交通拥堵控制策略设计 张敖木翰