Министерство образования и науки Российской Федерации

1. Санкт-Петербургский Политехнический Университет Петра Великого
2. —
3. [Институт компьютерных наук и кибербезопасности](https://dl.spbstu.ru/)

**К У Р С О В А Я Р А Б О Т А**

1. «**Моделирование автомобильного движения**»
2. по дисциплине «Структуры данных»
3. Выполнил
4. студент гр. 5151003/30002 Чуприцкий Д.В.

<*подпись*> Гопта В.Д.

1. Преподаватель
2. асс. преподавателя Панков И.Д.

<*подпись*>

1. Санкт-Петербург
2. 2024

**Содержание**

[ВВЕДЕНИЕ 3](#_Toc168333275)

[1 Описание приложения 5](#_Toc168333276)

[1.1 Описание модели 5](#_Toc168333277)

[1.2 Об интерфейсе 6](#_Toc168333279)

[2 Теоретические исследования 8](#_Toc168333280)

[2.1 Выбранные алгоритмы 8](#_Toc168333281)

[2.2 OpenGL 9](#_Toc168333282)

[3 Практическая часть 11](#_Toc168333283)

[3.1 Описание структуры карты 11](#_Toc168333284)

[3.2 Описание представления машин в пространстве карты 22](#_Toc168333288)

[3.4 Описание модельного времени 25](#_Toc168333294)

[3.5 Описание событий, происходящих в step 25](#_Toc168333294)

[3.6 Описание принципов работы алгоритмической части программы 25](#_Toc168333294)

[3.7 Описание процесса постановки пользовательской машины 25](#_Toc168333294)

[3.8 Описание аварийной ситуации 25](#_Toc168333294)

[3.9 Реализация графической части 25](#_Toc168333294)

[3.10 Реализация сохранения и выгрузки игры 28](#_Toc168333295)

[РЕЗУЛЬТАТЫ РАБОТЫ ПРОГРАММЫ 32](#_Toc168333297)

[ЗАКЛЮЧЕНИЕ 35](#_Toc168333298)

[СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ 36](#_Toc168333299)

[ПРИЛОЖЕНИЕ А 37](#_Toc168333300)

# Введение

Моделирование – исследование объектов познания на их моделях; построение и изучение моделей реально существующих объектов, процессов или явлений с целью получения объяснений этих явлений, а также для предсказания явлений, интересующих исследователей.

Целью данной работы является разработка компьютерного инструмента для моделирования автомобильного движения на языке программирования Си с использованием графической библиотеки OpenGL для отображения объектов. Программа должна поддерживать различные опции для инициализации модели: выбор карты, количество машин, количество полос на дороге и частота появления машин.

Для достижения поставленной цели были сформированы следующие задачи:

1. Выбрать модель описания движения транспортного потока
2. Выбрать структуру модели (выбрать нужные структуры данных и т.п)
3. Разработать логику обновления модели (обработка ходов, выстрелов и т.д)
4. Ознакомиться с програмным интерфейсом OpenGL
5. Привязать графическую часть к логике модели
6. Разработать сохранение и выгрузку модели

Для решения задач 1, 4 были использованы дополнительные источники информации (основная информация изложена в пункте 2 – “Теоретические исследования”). Данные пункты являются отправными точками реализации приложения. Пункты 2, 3 были решены теоретическим моделированием различных ситуаций. Таким образом были определены качества, которыми должны обладать структура и логика модели. Решение этих задач описано в 3 пункте – “Практическая часть”. Задачи 2 и 3 необходимы, чтобы упростить внедрение нового функционала по мере нарастания уже реализованного функционала. Решение задачи номер 5 описано в пункте 3.7 – “Графическая часть”. Решение задачи номер 7 описано в пункте 3.8 – “Реализация сохранения и выгрузки игры”.

1. **Описание приложения**
   1. **Описание модели**

Пользователь может инициализировать новую модель задав следующие параметры: количество полос на каждом направлении дороги (от 1 до 64), количество машин (от 1 до 1000), частоту появления машин (от 0 до 100) и карту. На выбор дается четыре карты: одна вертикальная дорога, одна горизонтальная дорога, один перекресток и четыре перекрестка. Каждая дорога двухнаправленная.

После инициализации машины случайно помещаются на начало полосы и начинают движение. Машины могут менять скорость, останавливаться, совершать обгон, перестраиваясь на соседнюю полосу, и поворачивать на другую дорогу на перекрестке.

Пользователь может поставить машину, поставить машину в состоянии аварии, просматривать информацию о текущем состоянии выбранной полосы. Также пользователь может сохранить и загрузить модель.

* 1. **Об интерфейсе**

Пользователю предлагается удобный интерфейс. В меню пользователь может перейти к инициализации новой модели, сохранению модели (если она инициализирована), загрузке модели, справке (в ней представлена информация об управлении). Так же в меню есть возможность выбрать цветовую схему и ознакомиться с информацией о создании программы.

На рисунке 1 показан вид описанного интерфейса.



Рисунок 1 – Интерфейс стартового меню

Если пользователь нажал «New Model», то открывается меню настройки параметро новой модели. В этом меню пользователь может выбрать количество машин, количество полос, частоту появления машин и тип карты. На рисунке 2 показан интерфейс настройки игры.

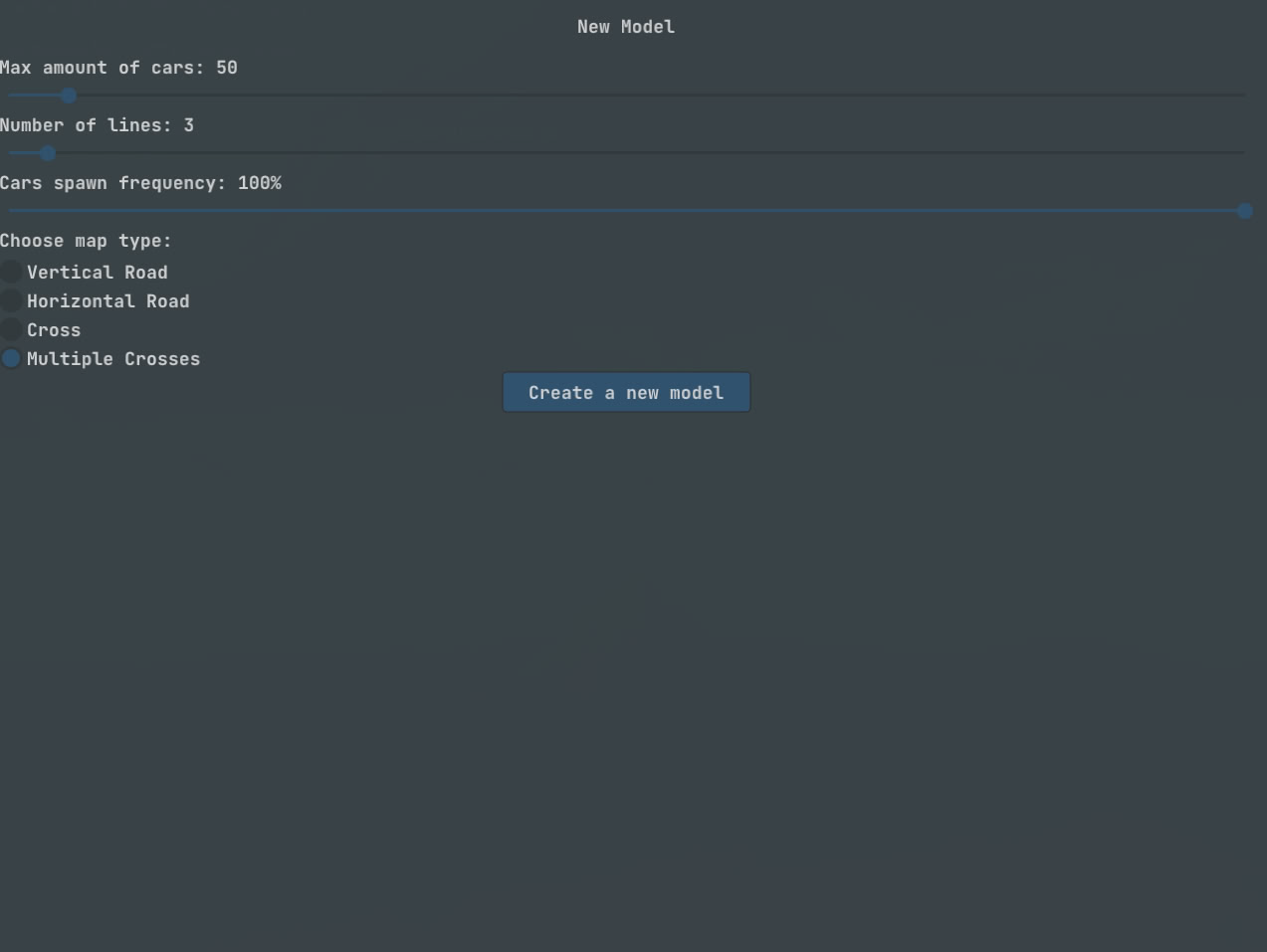


Рисунок 2 – Интерфейс «New Model»

Если пользователь нажал на «Load Model», то он перейдет к загрузке сохраненной модели. В этом меню будет представлен список всех сохранений. Пользователь может выбрать и одно из них по нажатию кнопки «Load» загрузить. На рисунке 3 показан интерфейс загрузки модели.

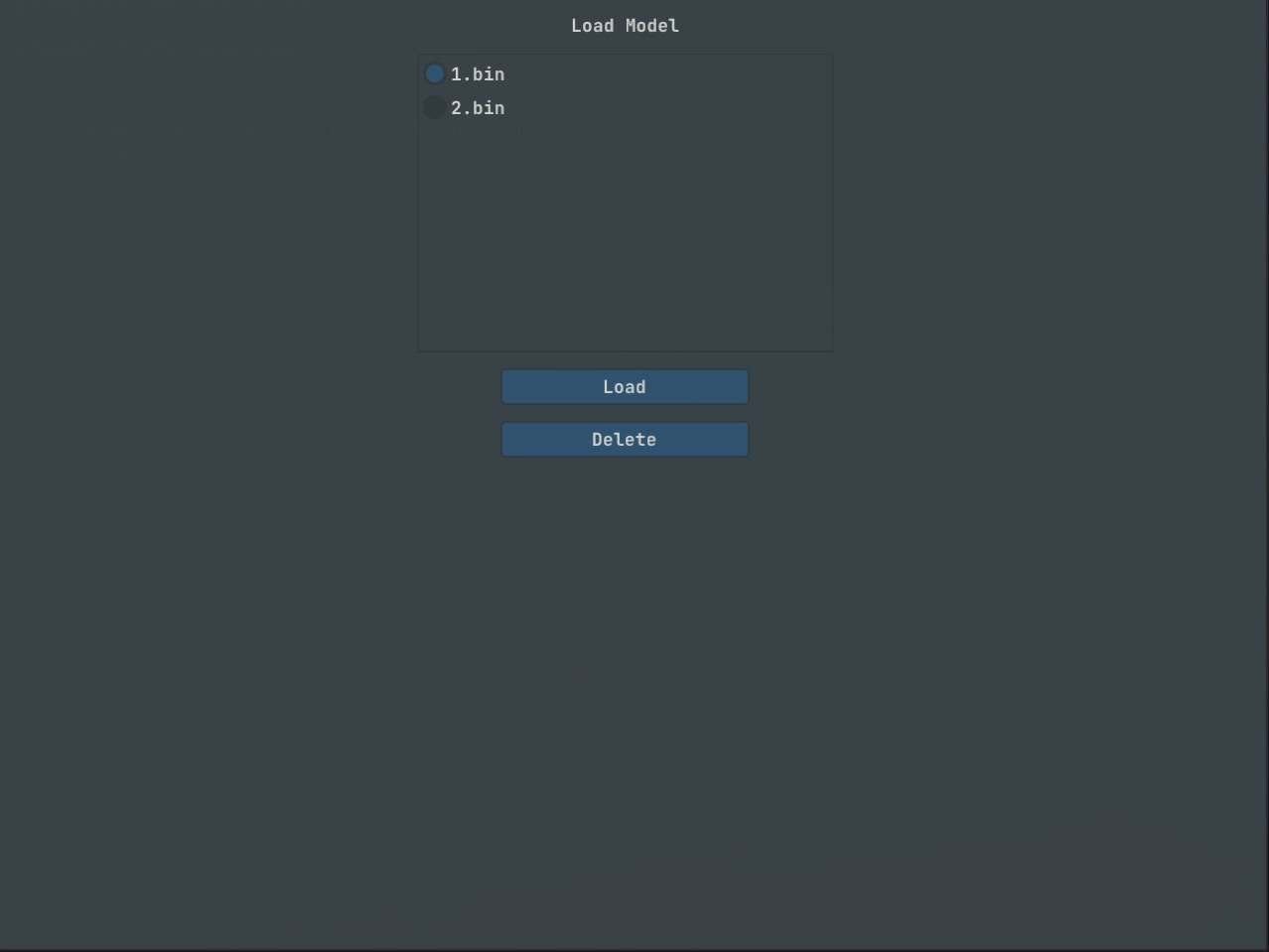


Рисунок 3 – Интерфейс «Load Model»

Если пользователь нажал на «Help», то он перейдет к справке. В справке указана информация касательно управления программой. На рисунке 3 показан интерфейс этого раздела меню.



Рисунок 4 – Интерфейс «Help»

Если модель инициализирована, но пользователю доступна кнопка «Save Model». Если пользователь нажмет на эту кнопку, он перейдет к меню сохранения модели. В нем пользователь может вписать желаемое имя для сохранения и после нажатия кнопки «Save» сохранить модель в бинарный файл. На рисунке 5 показан интерфейс этого раздела меню.

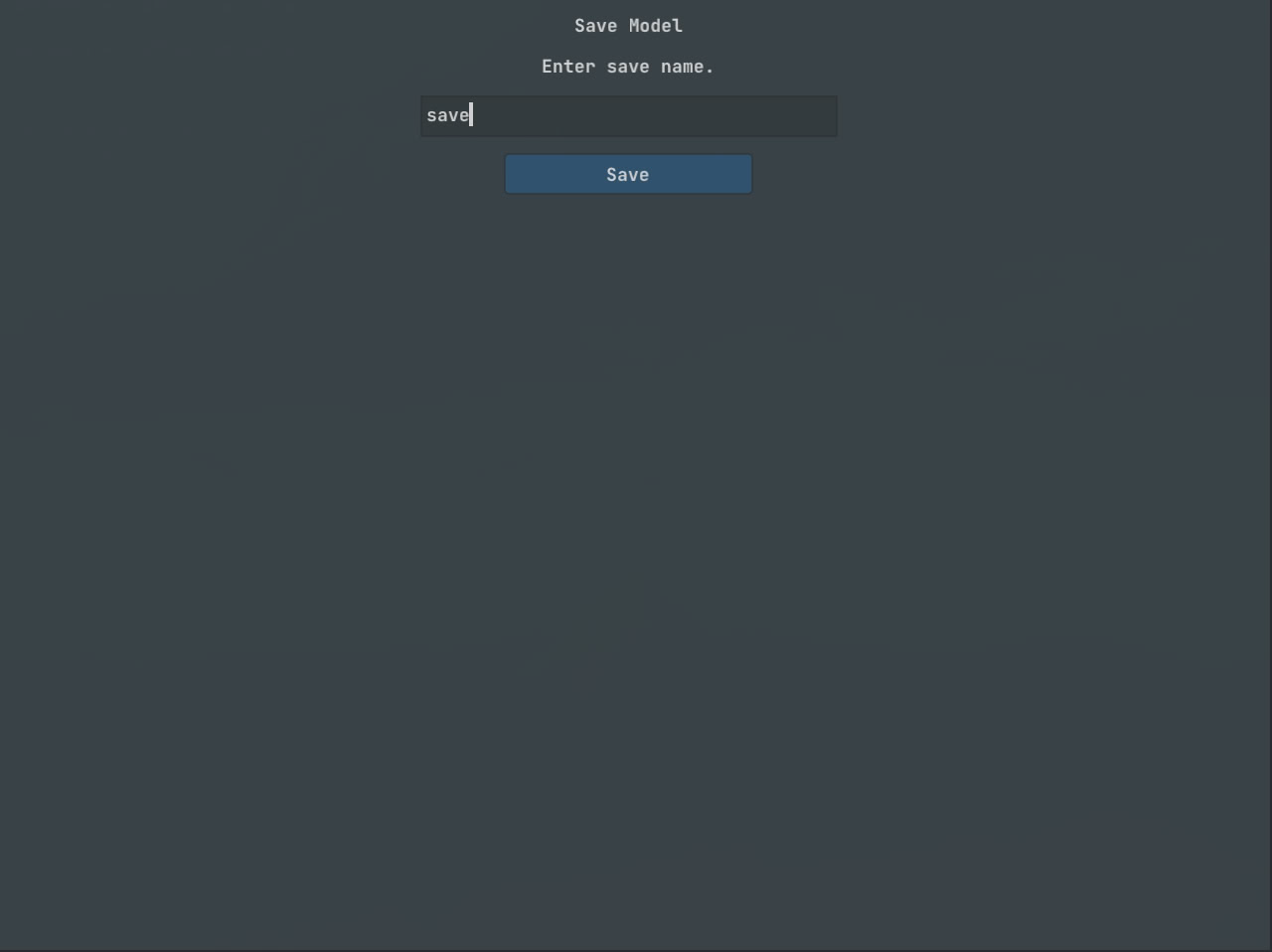


Рисунок 5 – Интерфейс «Save Model»

После инициализации модели пользователь может использовать левую кнопки мыши для того, чтобы поставить машину, и правую — для того, чтобы поставить машину в состоянии аварии. Также пользователь может нажать «Ё», чтобы открыть плавающее окно с информацией о модели. Пользователь может двигать окно. Когда оно открыто, при нажаитии левой кнопкой мыши по полосе, информация о ней будет отображаться в плавающем окне.

Плавающее окно представлено на рисунке 6.



Рисунок 6 – Плавающее окно

1. **Теоретические исследования**
   1. **Выбранные алгоритмы**

Для описания движения транспорта была выбрана модель Нагеля-Шрекенберга. Для каждого автомобиля задаётся следующие параметры:   
- скорость автомобиля - Car.velocity. Скорость варьируется от 0 до величины MAX\_VELOCITY. Единица измерения Car.velocity, в реальном времени, соответствует 10 м/с. При этом, 10 метров – длина ячейки;   
- расстояние до следующей ячейки впереди идущего автомобиля - Car.Distance. При отсутствии впереди идущего автомобиля, distance = MAX\_DISTANCE.   
Правило обновления шага алгоритма Step:   
1. Ускорение текущего автомобиля: если Car.velocity < MAX\_VELOCITY, скорость текущего автомобиля увеличивается на единицу, в противном случае Car.velocity = MAX\_VELOCITY и ускорение не происходит.   
2. Торможение текущего автомобиля. Если distance <= Car.velocity, то скорость текущего автомобиля distance - 1.   
3. Стохастическое торможение текущего автомобиля. Если Car.velocity > 0, то скорость текущего автомобиля может быть снижена с некоторой вероятностью DROP\_VELOCITY\_FREQUENCY.   
4. Движение текущего автомобиля. Автомобиль совершает движение на количество ячеек сетки, соответствующее своей скорости, после выполнения предыдущих шагов алгоритма.

* 1. **OpenGL**

OpenGL (Open Graphics Library) – это программный интерфейс, который позволяет создавать компьютерную графику в реальном времени. Он используется для разработки 2D и 3D графики, а также для создания интерактивных приложений, виртуальной реальности и игр. OpenGL поддерживается на различных операционных системах и платформах, что делает его широко используемым технологическим инструментом [3].

В процессе рендеринга графики в OpenGL используются графический конвейер и шейдеры для обработки и отображения изображений [4].

Графический конвейер управляет процесом преобразования 3D-координат в 2D пиксели.

Два основных этапа графического конвейера:

1. Преобразование фигур из 3D в 2D.
2. Преобразование координат в цветные пиксели.

Шейдерами называются небольшие ядра, которые запускаются на графическом процессоре на каждом этапе конвейера.

Этапы графического конвейера:

1. Вершинный шейдер (принимает в качестве входных данных одну координату) (базовая обработка атрибутов вершин, перевод коориднат) (в процессе изучения).
2. Геометрический шейдер (необязательный этап) (принимает в качестве входных данных набор вершин, образующих примитив) (добавляет различные дополнения).
3. Примитивная сборка (принимает все вершины) (образует один или несколько примитивов и собирает их в одну форму).
4. Растеризация (принимает выходные данные из 3 этапа) (сопоставление результирующих примитивов в соответствии с пикселями на конечном экране, в результате чего получаются фрагменты).
5. Фрагментный шейдер (предназначен для вычисления конечного цвета пикселя).
6. Тестирование (альфа-тест) и блэндинг (проверка на то, где находится объект, в зависимости от чего, он может быть удалён).

Текстура – это 2D изображение (1D и 3D текстура также существуют), используемое для добавления деталей объекту [5]. Они достаточно удобны и помогают пропустить отрисовку сложных объектов. Также они минимизируют накладные ресурсы вычислительной системы.

1. **Практическая часть**

**3.1 Описание структуры карты**

Каждая карта доступная в программе собарана из составаных элементов. Это элемент дорога и элемент перекресток. Карта HIGHWAY составлена из двух таких дорог, направленных в противоположные направления. Карта CROSS составлена из элемента перекресток и четырёх примыкающих к нему двухсторонних дорог, каждая из которых составлена из двух противоположно направленных элементов дорога.

Каждый элемент разбит на иерархические уровни. Элемент дорога road состоит из линий, а каждая линия line состоит из клеток cells (границы которых можно увидеть в режиме DEBUG, задаваемом макросом DEBUG в render.h). Это представлено на рисунке 7. Элемент перекресток cross просто разбит на cells. Введена адресация. Так, адрес на дороге состоит из номера дороги, номера линии и номера клетки. Дорожный адрес описывается структурой RLC. Адрес клетки на перекрестке состоит из номера перекрестка и условных координат по x и y, что описывается структурой cross\_coord.

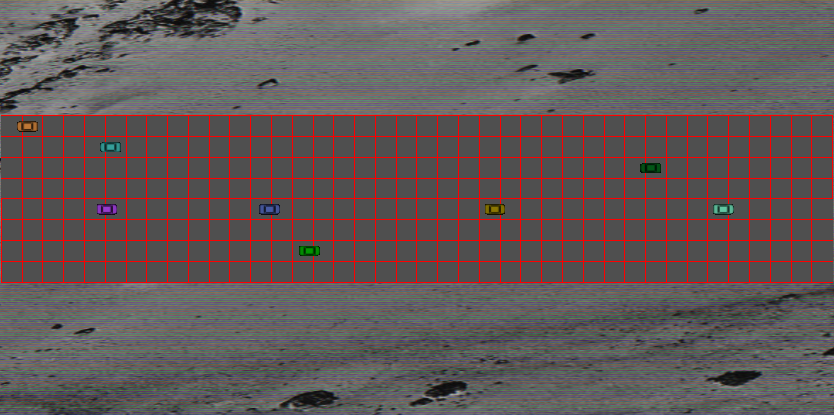


Рисунок 7 – Дорога в режиме DEBUG

**3.2 Описание представления машины в пространстве карт**

Каждая машина car представляет объект, занимающий пространство равное одной клетке. Скорость машины представлена целым числом клеток в единицу модельного времени. Машина считается существующей в клетке, если в клетке хранится указатель на эту машину. Машина имеет по две координаты для каждого элемента карты в соответствующей адресации. Одна координата означает положение, из которого началось движение, а вторая положение, в котором должно закончиться по истечении единицы модельного времени. Машина имеет свой ID и цель target, обозначающую направление дороги, на которую она должна повернуть по завершении движения по данном элементу дороги.

**3.3 Описание модельного времени**

Единицу модельного времени определяет макрос STEP\_TIME, определенный в секундах. Изменение состояния модели происходит строго через каждую единицу модельного времени. Процесс, протекающий между двумя состояниями будет называться step. Модельное время можно считать, так сказать, чистым, так как из него исключено время, которое затрачивается на отрисовку. Поэтому если визуально заметно, что отрисовка происходит заторможено, что может возникать в силу технических характеристик устройств, на которых будет запускаться разработанная программа, то это не должно вызывать никаких опасений об угрозе точности процесса моделирования.

Замечание. Нужно оговорить, что в дальнейшем всё сказанное будет относится к картам с перекрестками без упоминания карт без них, чтобы избежать повторений, так как функционирование карт с перекрестками включают в себя все те же механизмы, что и карты без них, но с дополнительными механизмами специально для них.

**3.4 Описание событий, происходящих во время step**

События, происходящие во время step, вызываются из функции update(), её блок схема представлена на рисунке 8.

Функция renewCells(), представлена на рисунке 9. Вначале происходит обновление адресов машин. Первыми обновляются адреса на перекрестках, а потом на дорогах. Нужно заметить, что если на дорогах обновление идёт в одном направлении (в направлении движения по дороге), то на перекрестке движение происходит в четырёх направлениях, поэтому появляется возможность, что следующая к обработке клетка содержит в себе уже обновлённую машину. Чтобы машина избежала двойного обновления, она заносится в массив checkedCars, который содержит все обновлённые на перекрестке машины. Существуют машины, которые попадают с перекрестка на дорогу. Существует опасность повторного обновления при обработке дорог. Поэтому такие машины заносятся в массив skipCarsFromCross, и игнорируются при встрече. Само обновление выражается в помещении нулевого указателя в начальную клетку, а указателя машины в следующую клетку, приэтом следующий адрес становится текущим. Обновление игнорируют аварийные машины и мнимые машины.

Теперь будет происходить вычисление новых параметров машин.

Нужно дать пояснение механизмам. призванным отразить в модели правила дорожного движения на перекрестке. Известно, что прежде чем попасть на перекресток машина должна пропустить все машины, что заканчивают манёвр. Очевидно, что для прибывающей машины (arrivingCar) машинами, заканчивающими манёвр (endingManeuver), будут являться машины раньше прибывшие машины. Причём этими машинами могут оказаться как те, что попали при текущем зеленом сигнале светофора, так и те, что не успели выехать с перекрестка при предыдущем зеленом сигнале. Поэтому, чтобы выполнить условие: приоритет больше у того, кто раньше прибыл, использованы структура данных очередь. А чтобы выделять среди них заканчивающих манёвр, введены две очереди. Заканчивающие манёвр находятся в очереди carsEndingManeuver, а новоприбывшие в очереди carsArriving. Каждый перекресток имеет эти две очереди, объявленные в структуре перекрестка.

Функция stepCross(), представлена на рисунке 10. При смене сигнала светофора с жёлтого на красный очередь carsArriving опустошается, копируясь в хвост очереди carsEndingManeuver. Затем происходит вычисление новых параметров в функции thoughtsOfOneCarOnCross() для carsEdningManeuver на всех перекрестках. А потом вычисление для carsArriving, но только для тех из них, которым не нужно пропускать заканчивающие манёвр машины. Остальные машины из carsArrving остаются стоять.

Функция stepRoad(), представлена на рисунке 11. Начинается с постановки пользовательских аварийных и неаварийных машин. Затем в thoughtsOfOneCar() обрабатываются все машины за исключением аварийных, мнимых и пользовательских. В конце в spawnCars() генерируются новые машинки. На появление машинки накладываются две вероятности: SPAWN\_FREQUENCY задаёт вероятность вызова функции spawnCars(), а (rand() % NUMBER\_OF\_LINES) задает количество линий потенциальных к появлению машинки.

Функция changeLightsColor() сменяет сигналы светофора.

**Замечание**

На самом деле мнимые машины являются одной машиной occupying\_car, объявленной в cars.h. Указатель на мнимую машину помещается во все следующие клетки. Это решение призвано ускорить работу алгоритмов программы, так как поиск соседних машин является достаточно затратной процедурой.

**3.5 Описание принципов работы алгоритмической части программы**

Функция thoughtsOfOneCarOnCross(). Если машина находится на клетке поворота, то поменять направление движения moveDir. Выбирается следующая клетка, такая, какой бы она была, не будь бы на перекрестке других машин, кроме рассматриваемой. Затем проверяется безопасность пути до неё. Это очень затратная процедура, требующая проверки пространства перекрестка в двух направлениях: по линии, соединяющей текущее положение и предполагаемое следующее положение, и по линии, перпендикулярной первой линии. В результате определяется максимальное безопасное расстояние, на которое можно продвинуться. Следом определяется следующая клетка и скорость направление движения на дороге.

Функция thoughtsOfOneCar(). Если машина закончила обгон (moveDir == OVERTAKE) или сдвиг(moveDir == SHIFT), или другиими словами неполноценный обгон, её напрвление движения становится тем же, что и направление дороги. Вычисляется дистанция до передней машины и указатель на неё. Если приближается к перекрестку, то сбрасывает скорость. Если стоит вплотную, то ожидает зеленый сигнал светофора и свободность выбранного адреса. В положительном итоге указатель на мнимую машин удоолжен привязаться к выбранному адресу.

Скорость машины увеличиться на единицу скорости, если расстояние до предыдущей больше её скорости. Если скорость достигла допустимого максимума, то она не изменяется. Если до машины спереди расстояние меньше, чем скорсть данной машины, то проверяется попытка обгона.

Обгон будет осуществлён, если адрес обгона свободен, если он не находится между текущим и следующиим адерсом машины, едущей сзади, а также если расстояние до впереди едущей машины на полосе обгона не меньше скорости самой машины потенциальной для обгона. Если не выполнено пооследнее условие и на полосе обгона спереди вплотную стоит машина с нулевой скоростью, то машиина просто совершает сдвиг (SHIFT).

**Замечание**

Нужно заметить, что машины с дороги, к которым назначаются новые параметры в stepRoad() с помощью thoughtsOfOneCar(), берутся с дороги с самого её конца по самое начало, так как должен сохраниться тот прицип движения реальных машин на реальной дороге, что движение каждой машины зависит действий машин спереди. Этим занимается функция getCarByRoulette(), в которой реализована, так называемая, рулетка, которая обходит все клетки дорог в нужной последовательности.

Если машина собирается попасть с одного элемента карты на другой, то в главную координату её адреса в начальном элементе кладётся макрос перехода. Таким образом, если машина совершает переход с дороги на перекресток, то в адрес nextCell в поле road будет присвоено значение NEXT\_CELL\_IS\_ON\_CROSS, и если с перекрестка на дорогу, то в адрес crossNextCell в поле crossNum будет присвоено значение NEXT\_CELL\_IS\_ON\_ROAD. Встреча данных макросов актвирует те ветки функций rebindRoadCars() и rebindCrossCars(), которые отвечают за переход в другой элемент карты.

**3.6 Описание процесса определения загруженности дорог**

Загруженность дорог хранится в массиве densityData, размерность которого равна количеству дорог. Загруженность дороги увеличивается с вызовом функции increaseDensityData() при генерации машинки на дороге в функции spawnCars(), при добавлении пользовательских неаварийных машин в processCarAddingQueue(), а аварийных в addCrushedCar(), а также при переходе машины с перекрестка на дорогу в rebindCrossCars(). Загруженность дороги уменьшается с помощью функции decreaseDensityData() при перепривязки машины с дороги на перекресток в rebindRoadCars() и при исключении машины с дороги, которая не заканчивается перекрестком, в excludeFromMap().

**3.7 Описание процесса постановки пользовательской машины**

Пользователь может поставить машинку только на дорогу. Когда пользователь кликает по по дороге, программа получает координаты клика, а затем с помощью инструмента поиска getRLCbyDot(), реализованного в search\_tools.c, переводит координату пространства OpenGL в координату пространства дорог, представленную в структуре RLC. Будет игнорироваться клик, произведённый не по дороге, и клик, попавший на границу клеток. Затем адрес помещается в массивы carAddingQueue и carsAddingQueue\_CRUSH для неаварийных и аварийных машин. Эти массивы будут обработаны в stepRoad() с помощью функций processCarAddingQueue() и processCarAddingQueue\_CRUSH(). Эти функции проверяют занятость клетки и в положительном случае подбирает свободный индекс для машины из массива машин cars. После массивы carAddingQueue и carAddingQueue\_CRUSH очищаются.

**3.8 Описание аварийной ситуации**

Пользователю предоставлена возможность ставить аварийные машины, собирая из них нужные ему конфигурации аварийных карт. Аварийные машины имеют активное поле isCrushed и нулевую скорость. Эти машины никогда не обновляют значения своих параметров за всё время моделирования и воспринимаются кака препятствия. Но нужно заметить, что если машинка всё таки встала впритык с аварийной, то создаётся особая ситуация, которую не получается обработать обычными условиями определения поведения. Поэтому в такие моменты в коде присутствует ветка с условием на это исключительное положение.

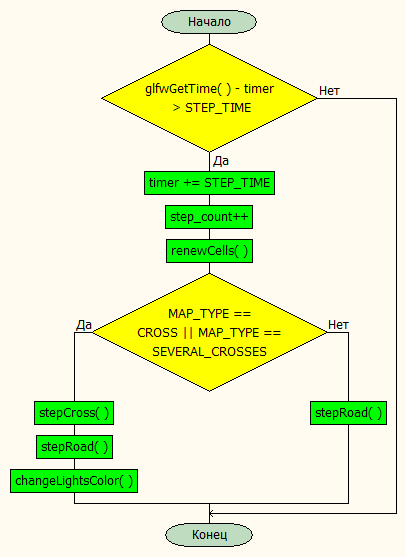


Рисунок 8 – Блок схема update()

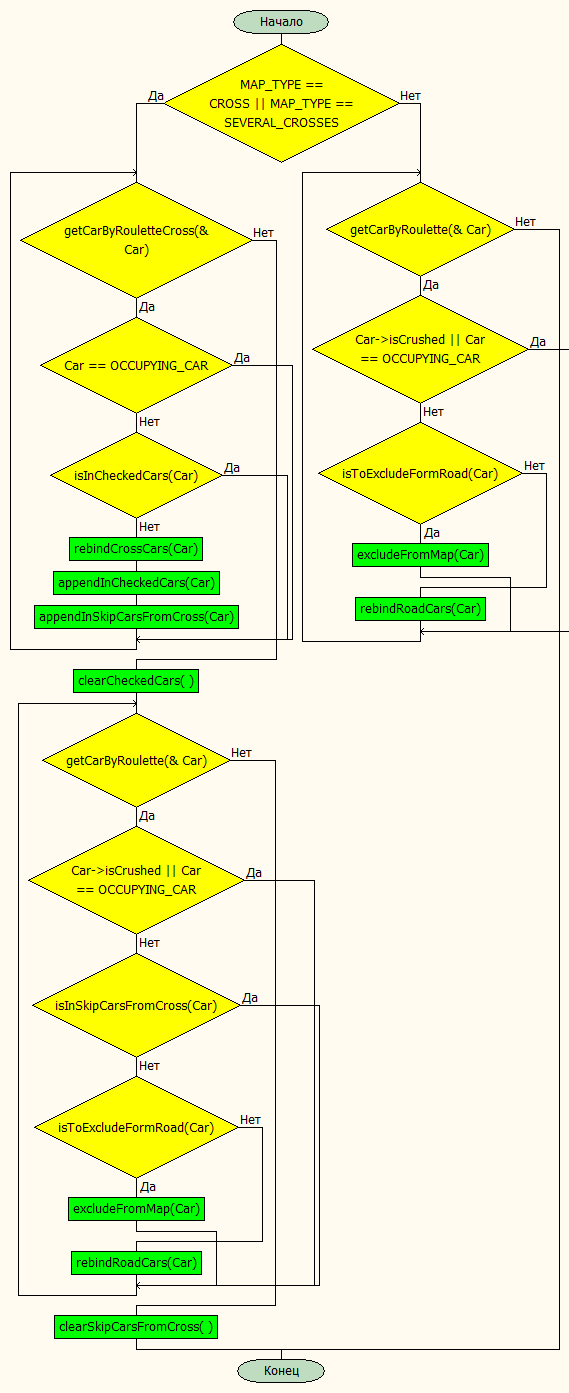


Рисунок 9 – Блок схема renewCells()

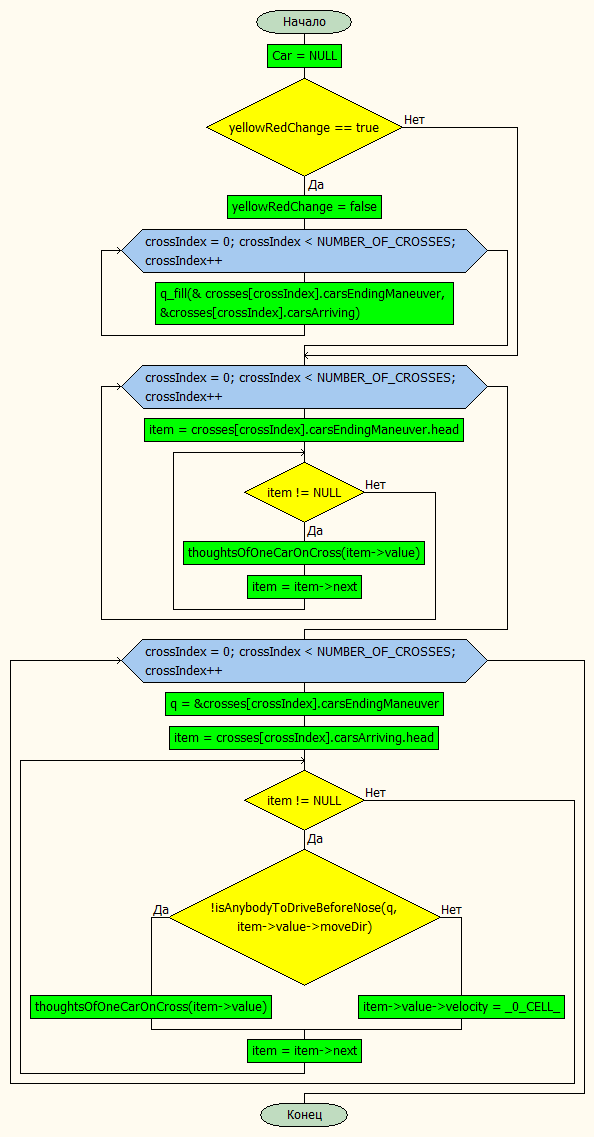


Рисунок 10 – Блок схема stepCross()

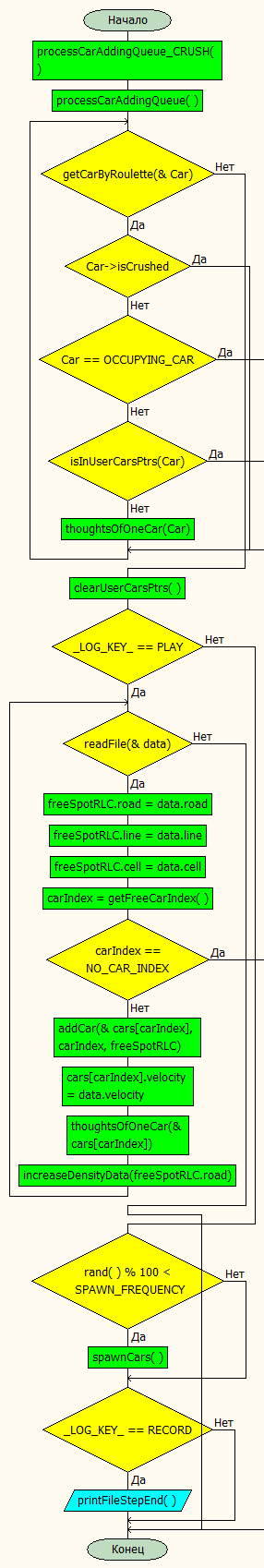


Рисунок 11 – Блок схема stepRoad()

**3.9** **Реализация графической части**

Для отображения интерфейса было решено использовать single-header библиотку Nuklear. Она предоставляет удобные готовые решения для рендера текста таких элементов интерфейса как кнопка, слайдер, радио и т. п.

Для отображения машин, светофоров и заднего фона использовались текстуры. Дороги и полосы имеют константный вектор цвета.

Разные объекты рендерятся с помощью разных шейдеров, так как у каждого объекта разные требования к рендеру.

Для отображения машин используется instanced rendering: создается один VAO, VBO и EBO для всех машин, где хранятся неизменяемые данные машины: вершины, индексы. Для того чтобы машины двигались, создан VBO carInstances, в котором хранится матрица трансформации для каждой машины. Каждый цикл рендера матрица трансформации обновляется в зависимости от состояния модели (может изменяться y координата машины — движение вперед, x координата — движение при обгоне, также машина может повернуться). Машины рендерятся с помощью всего одного вызова glDrawElementsInstanced. Такой подход сильно повышает производительность программы и уменьшает расходы памяти. Текстура у машин общая, но в фрагментном шейдере текстуре придается случайный цвет, а сидом для рандомной функции выступает instanceID.

**3.10** **Реализация сохранения и выгрузки игры**

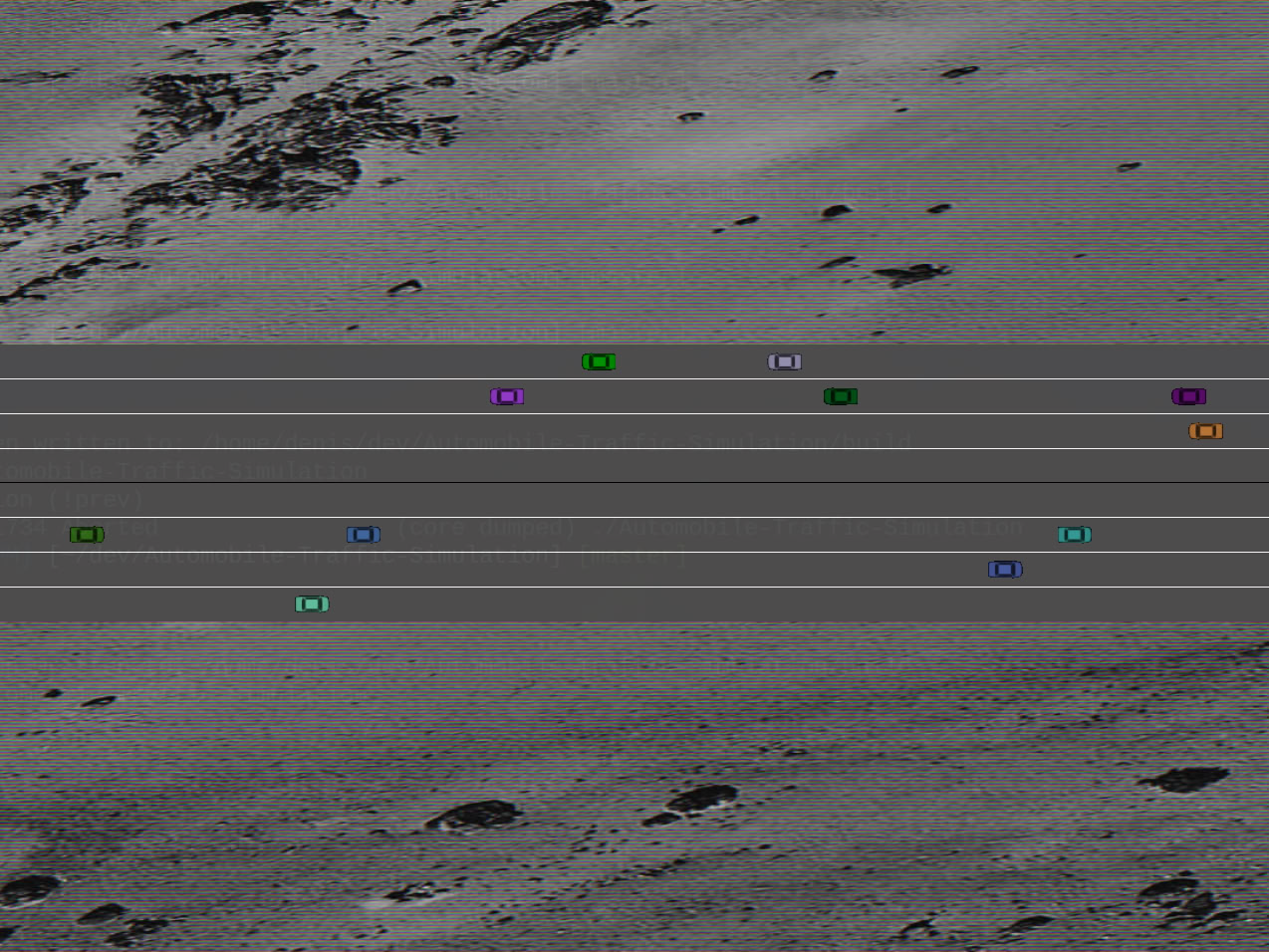
Сохранение осуществляется за счет выгрузки ключевых структур в файл в байтовом виде. Для всех типов карт выгружаются: конфигурация карты, массив машин и количество активных машин. Для CROSS и SEVERAL\_СROSSES также выгружается массив перекрестков и светофоров.

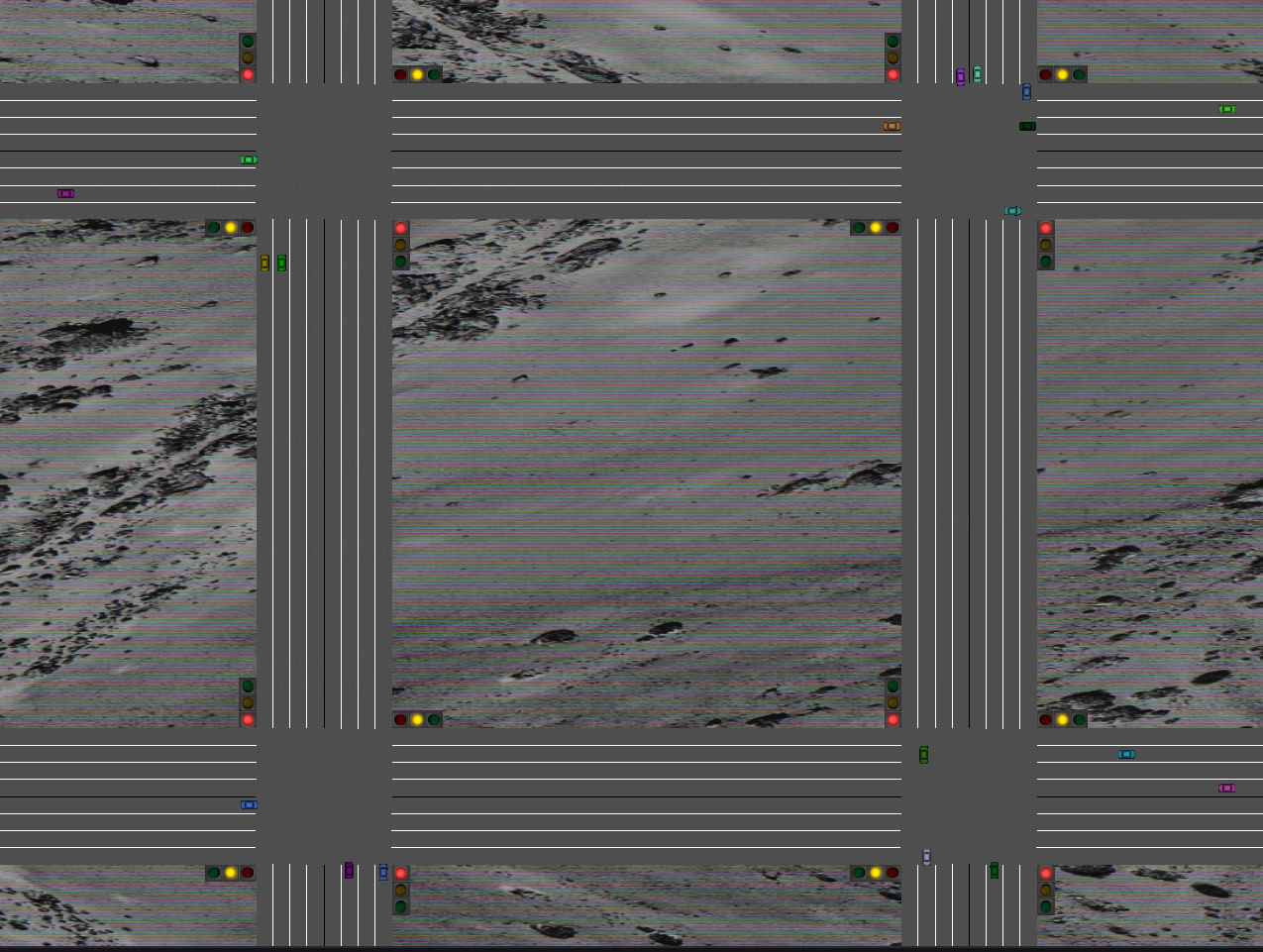
При загрузке модели из файла происхдит проверка: если модель ранее была инициализирована, но освобождается вся динамическая память. Затем все переменные заново инициализируется в зависимости от конфигурации модели. После чего из файла сохранения последовательно считываются данные ключевых структур.

# Результаты работы программы

Рисунок 12 — Работа программы на CROSS.

Рисунок 12 — Работа программы на VERTICAL.

Рисунок 12 — Работа программы на HORIZONTAL.

Рисунок 12 — Работа программы на SEVERAL CROSSES.

# ЗАКЛЮЧЕНИЕ

В ходе работы были получны навыки работы с OpenGL. Были закреплены навыки работы с различными типами и структурами данных. Был получен навык создания многофайлового проекта с использованием Cmakе для кросс-платформенности. Был разработан инструмент для моделирования автомобильного движения на языке программирования Си. Данная программа предоставляет пользователю интуитивный графический интерфейс, в котором он может настроить модель.

# СПИСОК ИСПОЛЬЗУЕМЫХ ИСТОЧНИКОВ

1. Верма, Р.Д. Введение в OpenGL [Текст] / Р.Д. Верма. – Москва: Горячая Линия – Телеком, 2017. – 304 с.

2. Вольф Д. OpenGL 4. Язык шейдеров. Книга рецептов [Текст] / Д. Вольф. – Москва: ДМК Пресс, 2015. – 368 с.

3. Гинсбург Д. OpenGL ES 3.0. Руководство разработчика [Текст] / Д. Гинсбург. – Москва: ДМК Пресс, 2015. – 439 с.

# приложЕНИЕ a

Листинг программы «Automobile-Traffic-Simulation»

cmake\_minimum\_required(VERSION 3.6.0)  
project(Automobile-Traffic-Simulation VERSION 0.1.0 LANGUAGES C)  
  
add\_executable(Automobile-Traffic-Simulation   
external/stb/stb\_image.c  
external/glad/glad.c  
source/main.c   
source/algorithms.c  
source/cars.c  
source/roads.c  
source/map.c  
source/render.c  
source/camera.c  
source/shader.c  
source/gui.c  
source/cross.c  
source/search\_tools.c  
source/traffic\_density.c  
source/traffic\_light.c  
source/texture.c  
source/log.c  
source/dbg.c  
)  
  
if (WIN32)  
target\_link\_libraries(Automobile-Traffic-Simulation   
${CMAKE\_SOURCE\_DIR}/external/GLFW/glfw3.lib)  
else()  
target\_link\_libraries(Automobile-Traffic-Simulation   
glfw   
m   
)  
endif()  
  
target\_include\_directories(  
Automobile-Traffic-Simulation  
PRIVATE  
${CMAKE\_SOURCE\_DIR}/external   
${CMAKE\_SOURCE\_DIR}/include   
${CMAKE\_SOURCE\_DIR}/source  
)

#ifndef ALGO\_H

#define ALGO\_H

#include <macros.h>

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

///Standart

#include <stdbool.h>

// Custom modules

#include <rlc.h>

#include <cars.h>

#include <road.h>

#include <direction.h>

#define MIN\_SPAWN\_VELOCITY \_4\_VELOCITY

#define MAX\_VELOCITY (\_9\_VELOCITY)

#define NUMBER\_OF\_VELOCITY\_TYPES (9)

#define VELOCITY\_MULTIPLIER (CELL\_LENGTH)

#define ERROR\_LIMIT 0.0000001f

#define EMPTY -1

#define STEP\_TIME 1.0f //seconds

#define MAX\_CELL\_DIGIT (NUMBER\_OF\_CELLS - 1)

#define NEXT\_CELL\_IS\_ON\_CROSS -45737

typedef enum MOVING\_TYPE MOVING\_TYPE;

typedef enum VELOCITY VELOCITY;

typedef struct car car;

enum VELOCITY {

\_0\_CELL\_ = 0,

\_1\_CELL\_ = \_1\_VELOCITY,

\_2\_CELL\_ = \_2\_VELOCITY,

\_3\_CELL\_ = \_3\_VELOCITY,

\_4\_CELL\_ = \_4\_VELOCITY,

\_5\_CELL\_ = \_5\_VELOCITY,

\_6\_CELL\_ = \_6\_VELOCITY,

\_7\_CELL\_ = \_7\_VELOCITY,

\_NO\_CAR\_ = 666

};

extern int MAX\_LINE\_DIGIT;

extern int MAX\_ROAD\_DIGIT;

extern bool alghorithmsInit;

extern RLC rouletteRLC;

extern car \*\* userCarsPtrs;

extern GLint innerUserCarsPtrsIndex;

extern RLC \* carAddingQueue;

extern GLint innerCarAddingQueueIndex;

extern RLC \* carAddingQueue\_CRUSH;

extern GLint innerCarAddingQueueIndex\_CRUSH;

extern car\*\* skipCarsFromCross;

extern GLint innerSkipCarsFromCrossIndex;

extern car\*\* checkedCars;

extern GLint innerCheckedCarsIndex;

extern int step\_count;

GLvoid initRoadCell(RLC \*rlc, car\* Car);

bool isToExcludeFormRoad(car\* Car);

GLint getVelocityByRLC(RLC rlc);

GLvoid excludeFromMap(car\* Car);

GLvoid thoughtsOfOneCar(car\* Car);

GLint distanceToForthCar(RLC rlc, car\*\* forthCar);

GLint distanceToBackCar(RLC rlc, car\*\* backCar);

MOVING\_TYPE checkChangeLineAbility(car\* Car, RLC\* rlc);

MOVING\_TYPE isSafetyForthAndBack(car\* Car, RLC rlc);

GLvoid update();

GLvoid renewCells();

GLvoid stepRoad();

GLvoid spawnCars();

GLvoid appendInUserCarsPtrs(car\* Car);

GLvoid clearUserCarsPtrs();

bool isInUserCarsPtrs(car\* Car);

GLvoid appendInSkipCarsFromCross(car\* Car);

GLvoid clearSkipCarsFromCross();

bool isInSkipCarsFromCross(car\* Car);

GLvoid appendInCheckedCars(car\* Car);

GLvoid clearCheckedCars();

bool isInCheckedCars(car\* Car);

GLvoid rebindRoadCars(car\* Car);

car\* getCarPtr(RLC\* rlc);

bool getCarByRoulette(car\*\* Car);

bool rollRouletteRLC();

bool isFurtherThanEndCell(car\* Car);

bool compareRLCs(RLC\* rlc1, RLC\* rlc2);

GLvoid processCarAddingQueue();

GLvoid processCarAddingQueue\_CRUSH();

GLvoid appendRLCinCarAddingQueue\_CRUSH(RLC rlc);

// bool isInCarAddingQueue(RLC rlc);

GLvoid clearCarAddingQueue\_CRUSH();

GLvoid appendRLCinCarAddingQueue(RLC rlc);

bool isInCarAddingQueue(RLC rlc);

GLvoid clearCarAddingQueue();

bool isRLCsuitableForSettingCar(RLC rlc);

GLvoid printRLC(RLC rlc, char\* string);

GLvoid printCrossCell(cross\_cell c);

GLint isRLCbad(RLC rlc);

GLvoid printCarProperties(RLC rlc);

GLvoid printDir(DIRECTION dir);

GLvoid printSkipCars();

GLvoid printCheckedCars();

GLvoid printCar(car\* Car);

#endif

#ifndef CAMERA\_H

#define CAMERA\_H

// External

#include <cglm/cglm.h>

#include <glad/glad.h>

// Custom modules

#include <direction.h>

#include <stdbool.h>

#define MAX\_CAMERA\_DISTANCE 1.0f

extern bool cameraInit;

extern vec3 cameraPos;

extern vec3 cameraFront;

extern vec3 cameraUp;

extern GLfloat cameraFOV;

typedef enum camDir camDir;

enum camDir {

UP, DOWN, RIGHT, LEFT

};

GLvoid setProjection(int shader);

GLvoid setView(int shader);

GLvoid moveCamera(camDir dir);

GLint isPossibleToMoveCam(camDir dir, GLfloat offset);

GLvoid getTestVertex(GLint roadIndex, camDir dir, vec4 dest);

GLint isVertexOnCamBorders(vec4 vertex, camDir dir, GLfloat offset, DIRECTION roadDir);

GLvoid setTestVertex(vec4 vertex, vec3 newCamPos);

#endif

#ifndef CARS\_H

#define CARS\_H

#include <macros.h>

// Standard

#include <stdbool.h>

// External

#include <glad/glad.h>

#include <cglm/cglm.h>

// Custom modules

#include <direction.h>

#include <rlc.h>

#include <cross.h>

// #define CAR\_WIDTH (ROAD\_WIDTH \* 2.0f / ((GLfloat)NUMBER\_OF\_LINES + 1.0f) / 3.0f)

#define CAR\_WIDTH (CELL\_LENGTH / 1)

#define CAR\_LENGTH (CAR\_WIDTH)

// #define CAR\_LENGHT (CAR\_WIDTH \* 1.7f)

#define NO\_INNER\_INDEX 0

#define NO\_CAR\_INDEX -1

// typedef struct RLC RLC;

typedef enum DIRECTION DIRECTION;

typedef enum MOVING\_TYPE MOVING\_TYPE;

typedef enum VELOCITY VELOCITY;

typedef struct cross\_cell cross\_cell;

typedef struct car car;

enum MOVING\_TYPE {

FORWARD, SHIFT, OVERTAKE

};

struct car {

cross\_cell crossCurrCell;

cross\_cell crossNextCell;

RLC currCell;

RLC nextCell;

GLint velocity;

GLint roadDirMultiplier;

GLint ID;

DIRECTION target;

MOVING\_TYPE move;

DIRECTION moveDir;

GLfloat realPos;

bool isActive;

bool isCrushed;

bool checked;

cross\_cell curvingCell;//the pole stays uninitializing after GLvoid clearCarProperties(car\* Car)

};

extern mat3 carTransformMatrixes[1000];

extern GLuint carVAO, carVBO, carEBO, carInstanceVBO;

extern GLfloat carVertices[4 \* 4];

extern GLint carIndices[6];

extern GLint freeCars;

extern car occupying\_car;

extern car\* OCCUPYING\_CAR;

GLvoid addCar(car\* Car, GLint carIndex, RLC rlc);

GLvoid addCrushedCar(RLC rlc);

GLvoid setBornCar(car\* Car, GLint carIndex, RLC address);

GLvoid setCarByRLC(car\* Car, GLint carIndex, RLC rlc);

GLvoid getFreeSpotAddress(RLC\* rlc);

GLint getFreeCarIndex();

GLvoid setBornCarProperties(car\* Car, GLint carIndex, RLC rlc);

GLvoid setCrushedCarProperties(car\* Car, GLint carIndex, RLC rlc);

GLvoid setCarsToDefault();

GLvoid clearCarProperties(car\* Car);

GLvoid bindCellAndCar(RLC\* rlc, car\* Car);

GLvoid setOccupyingCarProperties();

DIRECTION getOppositeDir(DIRECTION dir);

GLvoid setTarget(car\* Car);

void setCarOnCross(cross\_cell cell, car \* Car, int carIndex);

#endif

#ifndef CROSS\_H

#define CROSS\_H

#include <macros.h>

#include <map.h>

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

// Custom modules

#include <traffic\_light.h>

#include <road.h>

#define NUMBER\_OF\_CROSS\_ROADS 4

extern int MAX\_CROSS\_DIGIT;

extern int MAX\_CROSS\_CELL\_DIGIT;

#define NEXT\_CELL\_IS\_ON\_ROAD -55555

#define CROSS\_VELOCITY (\_1\_VELOCITY)

#define MAX\_CROSS\_VELOCITY (\_6\_VELOCITY)

#define NOTHING -1

typedef struct car car;

typedef struct road road;

typedef enum DIRECTION DIRECTION;

typedef struct cross\_cell cross\_cell;

typedef struct cross\_roulette cross\_roulette;

typedef struct cross cross;

typedef struct q\_item q\_item;

typedef struct queue queue;

typedef enum cross\_flood cross\_flood;

struct q\_item {

car\* value;

q\_item\* next;

};

struct queue {

q\_item\* head;

q\_item\* tail;

GLint qauntity;

};

struct cross\_cell {

GLint crossNum;

GLint x;

GLint y;

};

struct cross\_roulette {

GLint cellNum;

GLint crossNum;

};

struct cross {

GLint enterRoadsIndexes[NUMBER\_OF\_CROSS\_ROADS];

GLint exitRoadsIndexes[NUMBER\_OF\_CROSS\_ROADS];

road\* enterRoadsPtrs[NUMBER\_OF\_CROSS\_ROADS]; //DIRECTION type values can be indexes of roads of corresponding directions

road\* exitRoadsPtrs[NUMBER\_OF\_CROSS\_ROADS];

car\*\* cells;

queue carsEndingManeuver;

queue carsArriving;

};

extern int CROSS\_SIDE;

extern int HALF\_CROSS\_SIDE;

extern float CROSS\_WIDTH;

extern int NUMBER\_OF\_CROSS\_CELLS;

extern int MAX\_CELL\_NUM;

extern cross\_roulette rouletteCross;

#define TWO 2

extern DIRECTION crossQuaters[NUMBER\_OF\_DIRECTIONS \* TWO];

enum cross\_flood {

ENTER, EXIT

};

GLvoid q\_append(car\* Car, queue\* q);

car\* q\_pop(queue\* q);

GLvoid q\_delete(queue\* q);

GLvoid q\_del\_item(queue\* q, car\* Car);

GLvoid q\_fill(queue\* dest, queue\* src);

GLint getCrossEnter(GLint cell, DIRECTION dir);

GLint getCrossExit(GLint cell, DIRECTION dir);

GLvoid getCurvingCell(cross\_cell\* c, car\* Car, cross\_cell firstCellOnRoad);

GLint getQuaterNum(cross\_cell c);

cross\_flood getCodirectional(DIRECTION carDir, GLint quaterNum);

car\* getCarByCrossCell(cross\_cell\* c);

GLvoid initCrossCell(cross\_cell\* c, car\* Car);

bool isEndedWithCross(RLC\* rlc);

GLvoid stepCross();

bool getCarByRouletteCross(car\*\* Car);

bool rollRouletteCross();

GLvoid rebindCrossCars(car\* Car);

bool isAnybodyToDriveBeforeNose(queue\* q, DIRECTION ourCarDir);

GLvoid thoughtsOfOneCarOnCross(car\* Car);

bool isItCurvingCell(cross\_cell\* crossCurrCell, cross\_cell\* curvingCell);

GLvoid getNextCrossCell(car\* Car, cross\_cell\* c);

bool isInCrossBoards(cross\_cell\* c);

GLvoid transformCrossCellIntoRLC(RLC\* rlc, car\* Car);

GLvoid transformRLCIntoCrossCell(cross\_cell\* c, car\* Car);

GLint getLineOfAppearingOnRoadFromCross(car\* Car);

GLvoid addCross(GLint crossIndex, GLfloat start\_x, GLfloat start\_y, GLint\* enterRoadIndexes, GLint\* exitRoadIndexes);

GLvoid setCrossProperties(GLint crossIndex, GLint\* enterRoadIndexes, GLint\* exitRoadIndexes);

bool checkCrossCellSafety(cross\_cell\* c, DIRECTION dir);

GLvoid countCarVelocityAndNextCell(car\* Car, cross\_cell\* c);

bool isCrossCellsEqual(cross\_cell\* c, cross\_cell\* d);

// debug tools

GLvoid printCrossRoulette();

GLvoid printCarCharacter(car\* Car);

GLvoid q\_print(queue\* q);

#endif

#ifndef DBG\_H

#define DBG\_H

// External

#include <glad/glad.h>

#include <cglm/cglm.h>

GLvoid printGrid(GLint crossIndex);

#endif#ifndef DIRECTION\_H

#define DIRECTION\_H

#define NUMBER\_OF\_DIRECTIONS 4

typedef enum DIRECTION DIRECTION;

enum DIRECTION {

NORTH, SOUTH, EAST, WEST, NONE

};

#endif#ifndef GUI\_H

#define GUI\_H

#include <stdio.h>

#include <rlc.h>

#define MAX\_BUFFER\_SIZE 256

#define MAX\_SAVES 64

typedef struct config config;

struct config {

int max\_cars;

int lines;

int spawn\_frequency;

int map\_type;

};

void initGUI();

void initFont();

void showSaveMenu();

void showLoadMenu();

void showHelpMenu();

void save();

void load(char\* fileName);

void init(FILE\* saveFile);

extern char userSaveName[MAX\_BUFFER\_SIZE];

extern int activeFileIndex;

extern config initConfig;

#endif

#ifndef LOG\_H

#define LOG\_H

// External

#include <glad/glad.h>

#include <cglm/cglm.h>

#include <stdbool.h>

#define SEP " "

typedef struct log\_data log\_data;

struct log\_data{

GLint road, line, cell, velocity;

};

typedef enum log\_key log\_key;

enum log\_key{

NO, PLAY, RECORD

};

extern FILE\* f;

extern log\_key \_LOG\_KEY\_;

extern char fileName[100];

extern GLint fileSize;

bool readFile(log\_data\* data);

GLvoid openFile();

GLvoid closeFile();

GLvoid writeFile(log\_data\* data);

GLvoid printFileEmptyStep();

GLvoid printFileStepEnd();

#endif#ifndef MACROS\_H

#define MACROS\_H

#define TURN\_LEFT\_FREQUENCY 70

#define DROP\_VELOCITY\_FREQUENCY 0

#define \_1\_VELOCITY (1)

#define \_2\_VELOCITY (\_1\_VELOCITY \* 2)

#define \_3\_VELOCITY (\_1\_VELOCITY \* 3)

#define \_4\_VELOCITY (\_1\_VELOCITY \* 4)

#define \_5\_VELOCITY (\_1\_VELOCITY \* 5)

#define \_6\_VELOCITY (\_1\_VELOCITY \* 6)

#define \_7\_VELOCITY (\_1\_VELOCITY \* 7)

#define \_8\_VELOCITY (\_1\_VELOCITY \* 8)

#define \_9\_VELOCITY (\_1\_VELOCITY \* 9)

#endif

#ifndef MAP\_H

#define MAP\_H

#include <glad/glad.h>

#define VERTICAL 1

#define HORIZONTAL 2

#define CROSS 3

#define SEVERAL\_CROSSES 4

#define CELL\_LENGTH 0.05f

#define CELL\_WIDTH (CELL\_LENGTH)

typedef struct cross cross;

typedef struct traffic\_light traffic\_light;

typedef struct road road;

typedef struct car car;

extern int NUMBER\_OF\_TRAFFIC\_LIGHTS;

extern int NUMBER\_OF\_CROSSES;

extern int NUMBER\_OF\_ROADS;

extern int NUMBER\_OF\_LINES;

extern int NUMBER\_OF\_CELLS;

extern int MAP\_TYPE;

extern int MAX\_CARS;

extern int SPAWN\_FREQUENCY;

extern float DEFAULT\_FOV;

extern float ROAD\_WIDTH;

extern float HALF\_ROAD\_WIDTH;

extern traffic\_light\* lights;

extern cross\* crosses;

extern road\* roads;

extern unsigned int roadVAO, roadVBO, roadEBO;

extern float\* roadVertices;

extern int\* roadIndices;

extern unsigned int lineVAO, lineVBO;

extern float\* lineVertices;

extern car\* cars;

extern int \* densityData;

GLvoid setMap(int map\_type, int lines, int max\_cars, int spawn\_frequency);

GLvoid setRoadsAroundCross(GLfloat start\_x, GLfloat start\_y);

#endif

#ifndef RENDER\_H

#define RENDER\_H

// Standard

#include <stdbool.h>

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <cglm/cglm.h>

#include <rlc.h>

#define FPS 60.0f

//#define DEBUG

#define WINDOWS 0

#define UNIX 1

#if defined(\_WIN32) || defined(WIN32)

#define OS WINDOWS

#else

#define OS UNIX

#endif

typedef struct car car;

extern GLuint WINDOW\_WIDTH;

extern GLuint WINDOW\_HEIGHT;

extern GLchar WINDOW\_NAME[];

extern mat4 view;

extern mat4 projection;

extern bool paused;

extern bool isInitMenuActive;

extern bool isSaveMenuActive;

extern bool isLoadMenuActive;

extern bool isHelpMenuActive;

extern bool isShowInfo;

extern bool isInit;

extern GLFWwindow\* window;

extern struct nk\_glfw glfw;

extern struct nk\_context\* context;

extern GLuint isCarLoc, projectionLoc, modelLoc;

extern GLdouble limitFPS;

extern GLdouble lastTime;

extern GLdouble deltaTime, currTime, endPauseTime;

extern GLdouble timer;

extern float mousePosX;

extern float mousePosY;

extern unsigned int backgroundVBO, backgroundVAO, backgroundEBO;

extern bool isLinePicked;

extern RLC pickedRLC;

GLdouble getPauseTime();

GLvoid scrollCallback(GLFWwindow\* window, double xoffset, double yoffset);

GLvoid framebufferSizeCallback(GLFWwindow\* window, int width, int height);

GLvoid keyCallback(GLFWwindow\* window, int key, int scancode, int action, int mods);

GLvoid cursorPositionCallback(GLFWwindow\* window, double xpos, double ypos);

void mouseButtonCallback(GLFWwindow\* window, int button, int action, int mods);

GLvoid initRoads();

GLvoid initLines();

GLvoid initCars();

GLvoid render();

GLvoid quit();

GLvoid initGL();

GLvoid moveCarOnScreen(GLint carIndex);

GLvoid processKeyboardInput();

void showInitMenu();

void showInfo();

void initBackground();

void errorCallback(int e, const char \*d);

GLfloat getScreenVelocity(car\* Car);

GLfloat getScreenVelocityShift(car\* Car);

#ifdef DEBUG

GLvoid dbgInitCells();

GLvoid dbgRenderCells();

extern GLuint cellsVBO, cellsVAO;

extern GLfloat \* cellsVertices;

extern bool dbgIsCellsInit;

extern bool dbgVerticesInit;

#endif

#endif

#ifndef RLC\_H

#define RLC\_H

#include <glad/glad.h>

typedef struct

{

GLint road;

GLint line;

GLint cell;

} RLC;

#endif﻿#ifndef ROAD\_H

#define ROAD\_H

// Standard

#include <stdbool.h>

// External

#include <glad/glad.h>

// Custom modules

#include <rlc.h>

#include <direction.h>

#define NO\_ROAD\_INDEX -1

#define NO\_LINE\_INDEX -1

#define NO\_CELL\_INDEX -1

extern int midLinesCounter;

typedef enum DIRECTION DIRECTION;

typedef struct cross cross;

typedef struct line line;

typedef struct road road;

typedef struct car car;

typedef struct traffic\_light traffic\_light;

struct line {

car\*\* cells;

float carSpawnCoord;

};

struct road {

bool isEdge;

bool isBeginCross;

bool isEndCross;

GLint endCrossNum;

cross\* beginCross;

cross\* endCross;

GLfloat stem;

GLfloat startLineCoord;

GLfloat endLineCoord;

DIRECTION dir;

line\* lines;

GLint traffic\_light\_index;

traffic\_light\* traffic\_light\_ptr;//for the celerity of inquiring

};

GLvoid addRoad(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, DIRECTION dir);

GLvoid setRoad(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, GLfloat lenght, DIRECTION dir);

GLvoid setLines(GLint roadIndex);

GLint getDirMultiplier(DIRECTION dir);

DIRECTION getRoadDir(car\* Car);

DIRECTION getLeftMoveDir(DIRECTION dir);

DIRECTION getRightMoveDir(DIRECTION dir);

car\*\* getFirstCellPtr(RLC rlc);

GLvoid setRoadBoards(GLint roadIndex, GLfloat start\_x, GLfloat start\_y);

GLvoid setEdgeState(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, DIRECTION dir);

GLint isFurhterThanEndLine(car\* Car, road\* Road);

GLint isOutOfScreenSpace(GLfloat realPos);

int getCarsNumOnLine(int roadIndex, int lineIndex);

GLvoid printRoadProperties(GLint i);

#endif

#ifndef SEARCH\_TOOLS\_H

#define SEARCH\_TOOLS\_H

// Standard

#include <stdbool.h>

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

// Custom modules

#include <rlc.h>

#define NO\_COORD 666.0f

typedef struct dot\_coord dot\_coord;

struct dot\_coord {

GLfloat x;

GLfloat y;

};

extern dot\_coord \_DOT\_;

GLfloat absFloat(GLfloat a);

bool getRLCbyDot(RLC\* rlc, dot\_coord\* dot);

GLint getRoadIndex(dot\_coord\* dot);

bool isInRoadRange(GLint roadIndex, dot\_coord\* dot);

GLint getLineIndex(dot\_coord\* dot, GLint roadIndex);

GLfloat getRoadLeftCheek(GLint roadIndex);

GLint getCellIndex(dot\_coord\* dot, GLint roadIndex, GLint lineIndex);

GLvoid clearDot(dot\_coord\* dot);

#endif#ifndef SHADER\_H

#define SHADER\_H

// External

#include <glad/glad.h>

extern int shaderProgram, carShader, backgroundShader;

void genShader(int \* shader, char \* vertexShaderName, char \* fragmentShaderName);

char\* getShaderContent(const GLchar\* fileName);

#endif

extern unsigned int carTexture,

backgroundTexture,

redLight,

greenLight,

yellowLight;

void initTextures();

void loadTexture(char \* image\_name, unsigned int \* texture);

#ifndef TRAFFIC\_DENSITY

#define TRAFFIC\_DENSITY

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <map.h>

#include <stdbool.h>

extern bool densityDataInit;

GLvoid increaseDensityData(GLint roadIndex);

GLvoid decreaseDensityData(GLint roadIndex);

GLint getDensityData(GLint roadIndex);

GLvoid printDensity();

#endif

#ifndef TRAFFIC\_LIGHT\_H

#define TRAFFIC\_LIGHT\_H

#include <macros.h>

#include <map.h>

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

///Standart

#include <stdbool.h>

#define TRAFFIC\_LIGHTS\_TIME\_DIGIT (4)

extern const int RED\_TIME;

extern const int YELLOW\_TIME;

extern const int GREEN\_TIME;

typedef struct car car;

typedef enum COLOR COLOR;

enum COLOR {

GREEN, YELLOW, RED

};

typedef struct traffic\_light traffic\_light;

struct traffic\_light {

COLOR color;

GLint changeTimer;

unsigned int VAO, VBO, EBO;

};

extern GLint timer\_digit;

extern GLint traffic\_light\_adding\_index;

extern bool yellowRedChange;

GLvoid changeLightsColor();

COLOR getLightColor(GLint lightIndex);

COLOR getLightColorByCar(car\* Car);

GLint getFreeTrafficLightIndex();

GLvoid setDefaultTrafficLightProperties();

GLvoid printLights();

#endif

#include "map.h"

#include <algorithms.h>

#include <macros.h>

#include <stdlib.h>

#include <traffic\_density.h>

#include <cars.h>

#include <road.h>

#include <cross.h>

#include <render.h>

#include <log.h>

#include <dbg.h>

#include <cglm/cglm.h>

bool alghorithmsInit = false;

int MAX\_LINE\_DIGIT;

int MAX\_ROAD\_DIGIT;

RLC rouletteRLC;

GLint innerUserCarsPtrsIndex = NO\_INNER\_INDEX;

car \*\* userCarsPtrs;

RLC \* carAddingQueue;

GLint innerCarAddingQueueIndex = NO\_INNER\_INDEX;

RLC \* carAddingQueue\_CRUSH;

GLint innerCarAddingQueueIndex\_CRUSH = NO\_INNER\_INDEX;

car\*\* skipCarsFromCross;

GLint innerSkipCarsFromCrossIndex = NO\_INNER\_INDEX;

car\*\* checkedCars;

GLint innerCheckedCarsIndex = NO\_INNER\_INDEX;

int step\_count;

bool compareRLCs(RLC\* rlc1, RLC\* rlc2)

{

return (rlc1->road == rlc2->road && rlc1->line == rlc2->line && rlc1->cell == rlc2->cell);

}

GLvoid printCrossCell(cross\_cell c){

printf("(%d,%d,%d) ", c.crossNum, c.x, c.y);

}

car\* getCarPtr(RLC\* rlc)

{

return roads[rlc->road].lines[rlc->line].cells[rlc->cell];

}

GLvoid printDir(DIRECTION dir)

{

if (dir == NORTH) printf("NORTH\n");

else if (dir == SOUTH) printf("SOUTH\n");

else if (dir == WEST) printf("WEST\n");

else if (dir == EAST) printf("EAST\n");

else if (dir == NONE) printf("NONE\n");

}

GLvoid update()

{

if (!alghorithmsInit) {

MAX\_LINE\_DIGIT = NUMBER\_OF\_LINES;

MAX\_ROAD\_DIGIT = NUMBER\_OF\_ROADS - 1;

rouletteRLC.road = MAX\_ROAD\_DIGIT;

rouletteRLC.line = -1;

rouletteRLC.cell = MAX\_CELL\_DIGIT;

userCarsPtrs = (car \*\*)malloc(sizeof(car \*) \* MAX\_CARS);

innerUserCarsPtrsIndex = NO\_INNER\_INDEX;

innerCarAddingQueueIndex = NO\_INNER\_INDEX;

if (userCarsPtrs == NULL) {

printf("malloc failed on userCarsPtrs");

exit(1);

}

carAddingQueue = (RLC \*)malloc(sizeof(RLC) \* MAX\_CARS);

if (carAddingQueue == NULL) {

printf("malloc failed on carAddingQueue");

exit(1);

}

carAddingQueue\_CRUSH = (RLC \*)malloc(sizeof(RLC) \* MAX\_CARS);

if (carAddingQueue\_CRUSH == NULL) {

printf("malloc failed on carAddingQueue\_CRUSH");

exit(1);

}

skipCarsFromCross = (car \*\*)malloc(sizeof(car\*) \* MAX\_CARS);

if (skipCarsFromCross == NULL){

printf("malloc failed on skipCars");

exit(1);

}

checkedCars = (car \*\*)malloc(sizeof(car \*) \* MAX\_CARS);

if (checkedCars == NULL){

printf("malloc failed on checkedCars");

exit(1);

}

alghorithmsInit = true;

}

if (glfwGetTime() - timer > STEP\_TIME)

{

// printf("Step: %lf\n", glfwGetTime());

timer += STEP\_TIME;

step\_count++;

renewCells();

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES){

stepCross();

stepRoad();

changeLightsColor();

}

else{

stepRoad();

}

// printf("=======================================\n");

}

}

GLvoid printRLC(RLC rlc, char\* string)

{

printf("%s:(%d, %d, %d)\n", string, rlc.road, rlc.line, rlc.cell);

}

GLvoid stepRoad()

{

car\* Car;

processCarAddingQueue\_CRUSH();

processCarAddingQueue();

while (getCarByRoulette(&Car)){

if (Car->isCrushed || !Car->isActive){

continue;

}

if (Car == OCCUPYING\_CAR){

continue;

}

if (isInUserCarsPtrs(Car)){

continue;

}

thoughtsOfOneCar(Car);

}

clearUserCarsPtrs();

if (\_LOG\_KEY\_ == PLAY){

log\_data data;

while (readFile(&data)){

RLC freeSpotRLC;

freeSpotRLC.road = data.road;

freeSpotRLC.line = data.line;

freeSpotRLC.cell = data.cell;

GLint carIndex = getFreeCarIndex();

if (carIndex == NO\_CAR\_INDEX){

return;

}

addCar(&cars[carIndex], carIndex, freeSpotRLC);

cars[carIndex].velocity = data.velocity;

thoughtsOfOneCar(&cars[carIndex]);

--freeCars;

increaseDensityData(freeSpotRLC.road);

}

return;

}

if (rand() % 100 < SPAWN\_FREQUENCY){

spawnCars();

}

if (\_LOG\_KEY\_ == RECORD){

printFileStepEnd();

}

}

bool getCarByRoulette(car\*\* Car)

{

car\* tmpCar;

while (rollRouletteRLC()){

tmpCar = getCarPtr(&rouletteRLC);

if (tmpCar != NULL){

\*Car = tmpCar;

return true;

}

}

return false;

}

bool rollRouletteRLC()

{

if (rouletteRLC.line < MAX\_LINE\_DIGIT){

rouletteRLC.line += 1;

}else{

rouletteRLC.line = 0;

if (rouletteRLC.cell > 0){

rouletteRLC.cell -= 1;

}else{

rouletteRLC.cell = MAX\_CELL\_DIGIT;

if (rouletteRLC.road > 0){

rouletteRLC.road -= 1;

}else{

rouletteRLC.road = MAX\_ROAD\_DIGIT;

rouletteRLC.line = -1;

return false;

}

}

}

return true;

}

GLvoid spawnCars()

{

GLint carsToSpawn = rand() % NUMBER\_OF\_LINES;

GLint spawnedCars = 0;

if (freeCars > 0){

int counter = freeCars;

for (int i = 0; i < counter; i++){

if (spawnedCars > carsToSpawn){

return;

}

GLint carIndex = getFreeCarIndex();

if (carIndex == NO\_CAR\_INDEX){

return;

}

RLC freeSpotRLC = { EMPTY, EMPTY, EMPTY };

getFreeSpotAddress(&freeSpotRLC);

// if (roads[freeSpotRLC.road].dir != NORTH){

// return;

// }

if (freeSpotRLC.road != EMPTY && carIndex != NO\_CAR\_INDEX){

addCar(&cars[carIndex], carIndex, freeSpotRLC);

if (\_LOG\_KEY\_ == RECORD){

log\_data data = {

.road = freeSpotRLC.road,

.line = freeSpotRLC.line,

.cell = freeSpotRLC.cell,

.velocity = cars[carIndex].velocity};

writeFile(&data);

}

thoughtsOfOneCar(&cars[carIndex]);

--freeCars;

++spawnedCars;

increaseDensityData(freeSpotRLC.road);

}

}

}

}

GLvoid appendInUserCarsPtrs(car\* Car)

{

if (innerUserCarsPtrsIndex == MAX\_CARS){

printf("extra car is trying to appned in userCarsPtrs\n");

exit(-1);

}

userCarsPtrs[innerUserCarsPtrsIndex] = Car;

++innerUserCarsPtrsIndex;

}

bool isInUserCarsPtrs(car\* Car)

{

for (int i = 0; i < innerUserCarsPtrsIndex; i++){

if (Car == userCarsPtrs[i]){

return true;

}

}

return false;

}

GLvoid clearUserCarsPtrs()

{

innerUserCarsPtrsIndex = NO\_INNER\_INDEX;

}

GLvoid initRoadCell(RLC \*rlc, car\* Car)

{

roads[rlc->road].lines[rlc->line].cells[rlc->cell] = Car;

}

//................................................................................................

GLvoid renewCells(){

car\* Car;

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES){

while (getCarByRouletteCross(&Car))

{

if (Car == OCCUPYING\_CAR){

continue;

}

if (isInCheckedCars(Car)){

continue;

}

rebindCrossCars(Car);

appendInCheckedCars(Car);

appendInSkipCarsFromCross(Car);

}

clearCheckedCars();

while (getCarByRoulette(&Car)){

if (Car->isCrushed || Car == OCCUPYING\_CAR || !Car->isActive){

continue;

}

if (isInSkipCarsFromCross(Car)){

continue;

}

if (isToExcludeFormRoad(Car)){

excludeFromMap(Car);

continue;

}

rebindRoadCars(Car);

}

clearSkipCarsFromCross();

}else{

while (getCarByRoulette(&Car)){

if (Car->isCrushed || Car == OCCUPYING\_CAR || !Car->isActive){

continue;

}

if (isToExcludeFormRoad(Car)){

excludeFromMap(Car);

continue;

}

rebindRoadCars(Car);

}

}

}

GLvoid appendInSkipCarsFromCross(car\* Car){

if (innerSkipCarsFromCrossIndex == MAX\_CARS){

printf("extra car is trying to appned in skipCarsFromCross\n");

exit(-1);

}

skipCarsFromCross[innerSkipCarsFromCrossIndex] = Car;

++innerSkipCarsFromCrossIndex;

}

GLvoid printSkipCars(){

printf("============skip============\n");

for (int i = 0; i < innerSkipCarsFromCrossIndex; i++){

printCarCharacter(skipCarsFromCross[i]);

printf("`\n");

}

printf("============================\n");

}

GLvoid clearSkipCarsFromCross(){

innerSkipCarsFromCrossIndex = NO\_INNER\_INDEX;

}

bool isInSkipCarsFromCross(car\* Car)

{

for (int i = 0; i < innerSkipCarsFromCrossIndex; i++){

if (Car == skipCarsFromCross[i]){

return true;

}

}

return false;

}

GLvoid appendInCheckedCars(car\* Car){

if (innerCheckedCarsIndex == MAX\_CARS){

printf("extra car is trying to appned in checkedCars\n");

exit(-1);

}

checkedCars[innerCheckedCarsIndex] = Car;

++innerCheckedCarsIndex;

}

GLvoid printCheckedCars(){

printf("============check===========\n");

for (int i = 0; i < innerCheckedCarsIndex; i++){

printCarCharacter(checkedCars[i]);

printf("`\n");

}

printf("============================\n");

}

GLvoid clearCheckedCars(){

innerCheckedCarsIndex = NO\_INNER\_INDEX;

}

bool isInCheckedCars(car\* Car)

{

for (int i = 0; i < innerCheckedCarsIndex; i++){

if (Car == checkedCars[i]){

return true;

}

}

return false;

}

//...............................................................................................

GLvoid rebindRoadCars(car\* Car)

{

if (Car->currCell.cell < NUMBER\_OF\_CELLS){

initRoadCell(&Car->currCell, NULL);

}

if (Car->nextCell.road != NEXT\_CELL\_IS\_ON\_CROSS){

if (Car->nextCell.cell < NUMBER\_OF\_CELLS){

initRoadCell(&Car->nextCell, Car);

}

Car->currCell = Car->nextCell;

}else{

decreaseDensityData(Car->currCell.road);

initCrossCell(&Car->crossNextCell, Car);

Car->crossCurrCell = Car->crossNextCell;

q\_append(Car, &crosses[Car->crossNextCell.crossNum].carsArriving);

// printf(">> NEXT ON CROSS << ID: %d\n", Car->ID);

}

//else // what would be ohterwise ?

/\*

a car won't belong to the road, but will be still drawning

then a single way to delete from screen is done by using cars massive.

\*/

}

GLint isRLCbad(RLC rlc)

{

if (rlc.road >= NUMBER\_OF\_ROADS)

return 1;

if (rlc.line >= NUMBER\_OF\_LINES + 1)

return 1;

if (rlc.cell >= NUMBER\_OF\_CELLS)

return 1;

return 0;

}

GLvoid printCarProperties(RLC rlc)

{

if (isRLCbad(rlc))

return;

car\* Car = getCarPtr(&rlc);

if (Car == NULL)

{

printf("NULL\n\n\n\n\n\n");

}

else

{

printf("ID: %d\n", Car->ID);

printf( "currCell (%d, %d, %d)\nnextCell (%d, %d, %d)\nvelocity %d\nisActive %d\nisCrushed %d\n",

Car->currCell.road, Car->currCell.line, Car->currCell.cell,

Car->nextCell.road, Car->nextCell.line, Car->nextCell.cell,

Car->velocity,

Car->isActive,

Car->isCrushed);

if (Car->moveDir == NORTH){

printf("moveDir: NORTH\n");

}

else if (Car->moveDir == SOUTH){

printf("moveDir: SOUTH\n");

}

else if (Car->moveDir == EAST){

printf("moveDir: EAST\n");

}

else if (Car->moveDir == WEST){

printf("moveDir: WEST\n");

}

if (Car->target == NORTH){

printf("target: NORTH\n");

}

else if (Car->target == SOUTH){

printf("target: SOUTH\n");

}

else if (Car->target == EAST){

printf("target: EAST\n");

}

else if (Car->target == WEST){

printf("target: WEST\n");

}

}

}

GLvoid printCar(car\* Car){

if (Car == NULL)

{

printf("NULL\n\n\n\n\n\n");

}

else

{

printf("ID: %d\n", Car->ID);

printf( "currCell (%d, %d, %d)\nnextCell (%d, %d, %d)\nvelocity %d\nisActive %d\nisCrushed %d\n",

Car->currCell.road, Car->currCell.line, Car->currCell.cell,

Car->nextCell.road, Car->nextCell.line, Car->nextCell.cell,

Car->velocity,

Car->isActive,

Car->isCrushed);

if (Car->moveDir == NORTH){

printf("moveDir: NORTH\n");

}

else if (Car->moveDir == SOUTH){

printf("moveDir: SOUTH\n");

}

else if (Car->moveDir == EAST){

printf("moveDir: EAST\n");

}

else if (Car->moveDir == WEST){

printf("moveDir: WEST\n");

}

if (Car->target == NORTH){

printf("target: NORTH\n");

}

else if (Car->target == SOUTH){

printf("target: SOUTH\n");

}

else if (Car->target == EAST){

printf("target: EAST\n");

}

else if (Car->target == WEST){

printf("target: WEST\n");

}

}

}

GLvoid thoughtsOfOneCar(car\* Car)

{

bool isAllowedToOvertake = true;

if (Car->move == OVERTAKE){ //as end fo line changing which is just done

Car->move = FORWARD;

Car->moveDir = getRoadDir(Car);

}

else if (Car->move == SHIFT){

Car->move = FORWARD;

Car->moveDir = getRoadDir(Car);

}

RLC overtakeRLC;

car\* forthCar = NULL;

GLint distance = distanceToForthCar(Car->currCell, &forthCar);//the foo takes in cognisance that road can have endCross

// printf("dist: %d, isEnd: %d\n", distance, isEndedWithCross(&Car->currCell));

if (distance == \_NO\_CAR\_ && isEndedWithCross(&Car->currCell)){

distance = NUMBER\_OF\_CELLS - Car->currCell.cell;

isAllowedToOvertake = false;

if (distance == 1) {

cross\_cell c;

transformRLCIntoCrossCell(&c, Car);

// printRLC(Car->currCell, "TO CROSS");

// printCrossCell(c);

if (getLightColorByCar(Car) == GREEN && checkCrossCellSafety(&c, Car->moveDir)){

// printf("ID: %d, jump:", Car->ID); printCrossCell(c);printf("\n");

Car->crossNextCell = c;

Car->nextCell.road = NEXT\_CELL\_IS\_ON\_CROSS;

Car->velocity = CROSS\_VELOCITY;

initCrossCell(&c, OCCUPYING\_CAR);

getCurvingCell(&Car->curvingCell, Car, c);

}

else{

Car->velocity = \_0\_CELL\_;

}

return;

}

}

// condition "distance - 1 > Car->velocity" will stay strickt because there considered is the faculty of mounting the car velocity a one more

if (Car->velocity != \_0\_CELL\_){

if (distance - 1 > Car->velocity || distance == \_NO\_CAR\_){

/\* car gets velocity if this car is newBorn or if this car doesn't stop because of the crushed car a cell before. \*/

if (rand() % 100 < DROP\_VELOCITY\_FREQUENCY){

Car->velocity = rand() % Car->velocity;

}

else{

if (Car->velocity < MAX\_VELOCITY){

Car->velocity += \_1\_CELL\_;

}

}

Car->nextCell.cell += Car->velocity;

return;

}

}else{

if (distance > 1){

Car->velocity = \_1\_CELL\_;

Car->nextCell.cell += Car->velocity;

return;

}

}

/\*

condition "forthCarVelocity < Car->velocity" will stay strickt

because the conceding the unstrickt condition means additional condition

that the driver is sure that forth-car-on-the-near-line velocity is greater than tha driver's one.

this additional condition is not done.

moreover, including the additional condition deprives the driver of ability to escape from the case when there is an row of less-velocity cars before him.

\*/

if (isAllowedToOvertake &&

(

Car->velocity != \_0\_CELL\_ && forthCar->velocity < Car->velocity /\*&& forthCar != OCCUPYING\_CAR\*/

|| Car->velocity == \_0\_CELL\_ && distance == 1

)

)

{

RLC rlc = {NO\_ROAD\_INDEX, NO\_LINE\_INDEX, NO\_CELL\_INDEX};

MOVING\_TYPE move = checkChangeLineAbility(Car, &rlc);

if (move != FORWARD){

Car->move = move;

Car->nextCell = rlc;

bindCellAndCar(&rlc, OCCUPYING\_CAR);

if (Car->move == OVERTAKE && Car->velocity == \_0\_CELL\_){

Car->velocity = \_1\_CELL\_;

}

return;

}

}

//according to the so-called all-excluded condition

Car->velocity = distance - 1;

Car->nextCell.cell += Car->velocity;

}

bool isToExcludeFormRoad(car\* Car)

{

return (Car->nextCell.cell >= NUMBER\_OF\_CELLS);

}

bool isFurtherThanEndCell(car\* Car)

{

return (Car->nextCell.cell >= NUMBER\_OF\_CELLS);

}

// road ends up to the cross then Car is exluded only from the road, otherwise from road and MAP

GLvoid excludeFromMap(car\* Car)

{

if (Car->currCell.cell < NUMBER\_OF\_CELLS){

initRoadCell(&Car->currCell, NULL);

}

decreaseDensityData(Car->currCell.road);

if (!isEndedWithCross(&Car->currCell)){

clearCarProperties(Car);

++freeCars;

}

// as cars' amount on the road is decreasing

}

GLint getVelocityByRLC(RLC rlc)

{

car\*\* ptrCell = ((roads + rlc.road)->lines + rlc.line)->cells;

if (ptrCell[rlc.cell] == NULL){

return \_NO\_CAR\_;

}else{

return ptrCell[rlc.cell]->velocity;

}

}

MOVING\_TYPE checkChangeLineAbility(car\* Car, RLC\* rlc)

{

GLint newLine;

switch (rand() % 100 < TURN\_LEFT\_FREQUENCY){

case 0:{

newLine = Car->currCell.line - 1;

if (newLine >= 0 && newLine < NUMBER\_OF\_LINES + 1)//the utter right line is excluded

{

rlc->road = Car->currCell.road;

rlc->line = newLine;

rlc->cell = Car->currCell.cell;

// printRLC(\*rlc, "safeRLC left");

if(getCarPtr(rlc) == NULL)

{

MOVING\_TYPE res = isSafetyForthAndBack(Car, \*rlc);

if (res != FORWARD)

{

Car->moveDir = getLeftMoveDir(getRoadDir(Car));

return res;

}

}

}

newLine = Car->currCell.line + 1;

if (newLine >= 0 && newLine < NUMBER\_OF\_LINES + 1)//the utter right line is excluded

{

rlc->road = Car->currCell.road;

rlc->line = newLine;

rlc->cell = Car->currCell.cell;

// printRLC(\*rlc, "safeRLC right");

if(getCarPtr(rlc) == NULL)

{

MOVING\_TYPE res = isSafetyForthAndBack(Car, \*rlc);

if (res != FORWARD)

{

Car->moveDir = getRightMoveDir(getRoadDir(Car));

return res;

}

}

}

break;

}case 1:{

newLine = Car->currCell.line + 1;

if (newLine >= 0 && newLine < NUMBER\_OF\_LINES + 1)//the utter right line is excluded

{

rlc->road = Car->currCell.road;

rlc->line = newLine;

rlc->cell = Car->currCell.cell;

// printRLC(\*rlc, "safeRLC right");

if(getCarPtr(rlc) == NULL)

{

MOVING\_TYPE res = isSafetyForthAndBack(Car, \*rlc);

if (res != FORWARD)

{

Car->moveDir = getRightMoveDir(getRoadDir(Car));

return res;

}

}

}

newLine = Car->currCell.line - 1;

if (newLine >= 0 && newLine < NUMBER\_OF\_LINES + 1)//the utter left line is excluded

{

rlc->road = Car->currCell.road;

rlc->line = newLine;

rlc->cell = Car->currCell.cell;

// printRLC(\*rlc, "safeRLC left");

if(getCarPtr(rlc) == NULL)

{

MOVING\_TYPE res = isSafetyForthAndBack(Car, \*rlc);

if (res != FORWARD)

{

Car->moveDir = getLeftMoveDir(getRoadDir(Car));

return res;

}

}

}

break;

}

}

return FORWARD;

}

MOVING\_TYPE isSafetyForthAndBack(car\* Car, RLC rlc)

{

car\* forthCar = NULL;

GLint forth = distanceToForthCar(rlc, &forthCar);

car\* backCar = NULL;

GLint back = distanceToBackCar(rlc, &backCar);

if (back ==\_NO\_CAR\_ || (backCar && back - 1 >= backCar->velocity)){

if (Car->velocity != \_0\_CELL\_){

if (forth - 1 > Car->velocity){

return OVERTAKE;

}

}else{

if (forth - 1 > \_1\_CELL\_){

return OVERTAKE;

}

if (forth != 0){ //this is that on the neighbour cell there is no cacr

return SHIFT;

}

}

}

return FORWARD;

}

GLint distanceToForthCar(RLC rlc, car\*\* Car)

{

car\*\* ptrCell = getFirstCellPtr(rlc);

for (int i = 1 + rlc.cell; i < NUMBER\_OF\_CELLS; ++i){

if (ptrCell[i] == NULL){

continue;

}

else if (ptrCell[i]->isActive == true){

\*Car = ptrCell[i];

return (i - rlc.cell);

}

}

return \_NO\_CAR\_;

}

GLint distanceToBackCar(RLC rlc, car\*\* Car)

{

car\*\* ptrCell = getFirstCellPtr(rlc);

for (int i = -1 + rlc.cell; i >= 0; --i){

if (ptrCell[i] == NULL){

continue;

}

if (ptrCell[i]->isActive == true){

\*Car = ptrCell[i];

return (rlc.cell - i);

}

}

return \_NO\_CAR\_;

}

bool isRLCsuitableForSettingCar(RLC rlc)

{

if (getCarPtr(&rlc) != NULL){

return false;

}

car\* forthCar = NULL;

GLint forth = distanceToForthCar(rlc, &forthCar);

car\* backCar = NULL;

GLint back = distanceToBackCar(rlc, &backCar);

if (forth > 1 && back > 1){

return true;

}

return false;

}

GLvoid appendRLCinCarAddingQueue(RLC rlc)

{

if (innerCarAddingQueueIndex >= 1){

return;

}

carAddingQueue[innerCarAddingQueueIndex] = rlc;

innerCarAddingQueueIndex += 1;

}

GLvoid clearCarAddingQueue()

{

innerCarAddingQueueIndex = NO\_INNER\_INDEX;

}

GLvoid appendRLCinCarAddingQueue\_CRUSH(RLC rlc)

{

if (innerCarAddingQueueIndex\_CRUSH == 1){

return;

}

carAddingQueue\_CRUSH[innerCarAddingQueueIndex\_CRUSH] = rlc;

++innerCarAddingQueueIndex\_CRUSH;

}

GLvoid clearCarAddingQueue\_CRUSH()

{

innerCarAddingQueueIndex\_CRUSH = NO\_INNER\_INDEX;

}

GLvoid processCarAddingQueue()

{

for (GLint i = 0; i < innerCarAddingQueueIndex; i++){

if (isRLCsuitableForSettingCar(carAddingQueue[i])){

GLint carIndex = getFreeCarIndex();

if (carIndex == NO\_CAR\_INDEX){

return;

}

car\* Car = &cars[carIndex];

addCar(Car, carIndex, carAddingQueue[i]);

thoughtsOfOneCar(Car);

--freeCars;

increaseDensityData(carAddingQueue[i].road);

appendInUserCarsPtrs(Car);

}

}

clearCarAddingQueue();

}

GLvoid processCarAddingQueue\_CRUSH()

{

for (int i = 0; i < innerCarAddingQueueIndex\_CRUSH; i++){

addCrushedCar(carAddingQueue\_CRUSH[i]);

}

clearCarAddingQueue\_CRUSH();

}

bool isInCarAddingQueue(RLC rlc)

{

for (int i = 0; i < innerCarAddingQueueIndex; i++){

if (compareRLCs(&rlc, &carAddingQueue[i])){

return true;

}

}

return false;

}

// Custom modules

#include <camera.h>

#include <map.h>

#include <shader.h>

#include <render.h>

#include <road.h>

vec3 cameraPos = { 0.0f, 0.0f, 3.0f };

vec3 cameraFront = { 0.0f, 0.0f, -1.0f };

vec3 cameraUp = { 0.0f, 1.0f, 0.0f };

mat4 projection;

mat4 view;

GLfloat cameraFOV;

bool cameraInit = false;

GLvoid setProjection(int shader)

{

if (!cameraInit) {

cameraFOV = DEFAULT\_FOV;

cameraInit = true;

}

GLuint projLoc = glGetUniformLocation(shader , "projection");

glm\_perspective(glm\_rad(cameraFOV), (float)WINDOW\_WIDTH / (float)WINDOW\_HEIGHT, 1.0f, 100.0f, projection);

glUniformMatrix4fv(projLoc, 1, GL\_FALSE, (float \*)projection);

}

GLvoid setView(int shader)

{

vec3 temp;

glm\_vec3\_add(cameraPos, cameraFront, temp);

glm\_lookat(cameraPos, temp, cameraUp, view);

GLuint viewLoc = glGetUniformLocation(shader, "view");

glUniformMatrix4fv(viewLoc, 1, GL\_FALSE, (float \*)view);

}

GLvoid moveCamera(camDir dir)

{

float cameraSpeed = 0.01;

if (dir == UP && isPossibleToMoveCam(dir, cameraSpeed))

{

cameraPos[1] += cameraSpeed;

}

if (dir == DOWN && isPossibleToMoveCam(dir, cameraSpeed))

{

cameraPos[1] -= cameraSpeed;

}

if (dir == RIGHT && isPossibleToMoveCam(dir, cameraSpeed))

{

cameraPos[0] += cameraSpeed;

}

if (dir == LEFT && isPossibleToMoveCam(dir, cameraSpeed))

{

cameraPos[0] -= cameraSpeed;

}

}

GLint isVertexOnCamBorders(vec4 vertex, camDir dir, GLfloat offset, DIRECTION roadDir)

{

vec3 newCameraPos;

glm\_vec3\_copy(cameraPos, newCameraPos);

if (dir == UP)

{

if (roadDir == WEST || roadDir == EAST)

{

return 1;

}

newCameraPos[1] += offset;

setTestVertex(vertex, newCameraPos);

if (vertex[1] >= 2.0)

{

return 1;

}

}

if (dir == DOWN)

{

if (roadDir == WEST || roadDir == EAST)

{

return 1;

}

newCameraPos[1] -= offset;

setTestVertex(vertex, newCameraPos);

if (vertex[1] <= -2.0)

{

return 1;

}

}

if (dir == RIGHT)

{

if (roadDir == NORTH || roadDir == SOUTH)

{

return 1;

}

newCameraPos[0] += offset;

setTestVertex(vertex, newCameraPos);

if (vertex[0] >= 2.0)

{

return 1;

}

}

if (dir == LEFT)

{

if (roadDir == NORTH || roadDir == SOUTH)

{

return 1;

}

newCameraPos[0] -= offset;

setTestVertex(vertex, newCameraPos);

if (vertex[0] <= -2.0)

{

return 1;

}

}

return 0;

}

GLvoid setTestVertex(vec4 vertex, vec3 newCamPos)

{

mat4 projection;

glm\_perspective(glm\_rad(cameraFOV), (float)WINDOW\_WIDTH / (float)WINDOW\_HEIGHT, 0.01f, 100.0f, projection);

mat4 newView;

vec3 temp;

glm\_vec3\_add(newCamPos, cameraFront, temp);

glm\_lookat(newCamPos, temp, cameraUp, newView);

glm\_mat4\_mulv(newView, vertex, vertex);

glm\_mat4\_mulv(projection, vertex, vertex);

}

GLint isPossibleToMoveCam(camDir dir, GLfloat offset)

{

int roadsOnEdge = 0, roadsOnCamBorder = 0;

if (MAP\_TYPE == CROSS) {

if (dir == UP) {

vec4 upNorthRoadEdge = { roadVertices[4 \* 5 \* 4 + 10], roadVertices[4 \* 5 \* 4 + 11], 1.0f, 1.0f };

return isVertexOnCamBorders(upNorthRoadEdge, dir, offset, NORTH);

}

if (dir == DOWN) {

vec4 downNorthRoadEdge = { roadVertices[0 \* 5 \* 4 + 0], roadVertices[0 \* 5 \* 4 + 1], 1.0f, 1.0f };

return isVertexOnCamBorders(downNorthRoadEdge, dir, offset, NORTH);

}

if (dir == RIGHT) {

vec4 rightWestRoadEdge = { roadVertices[3 \* 5 \* 4 + 0], roadVertices[3 \* 5 \* 4 + 1], 1.0f, 1.0f };

return isVertexOnCamBorders(rightWestRoadEdge, dir, offset, WEST);

}

if (dir == LEFT) {

vec4 leftWestRoadEdge = { roadVertices[7 \* 5 \* 4 + 10], roadVertices[7 \* 5 \* 4 + 11], 1.0f, 1.0f };

return isVertexOnCamBorders(leftWestRoadEdge, dir, offset, WEST);

}

}

if (MAP\_TYPE == SEVERAL\_CROSSES)

return true;

for (int i = 0; i < NUMBER\_OF\_ROADS; i++)

{

if (roads[i].isEdge)

{

roadsOnEdge++;

vec4 testVertex = { 0.0, 0.0, 1.0, 1.0 };

getTestVertex(i, dir, testVertex);

if (isVertexOnCamBorders(testVertex, dir, offset, roads[i].dir))

{

roadsOnCamBorder++;

}

}

}

if (roadsOnCamBorder == roadsOnEdge)

{

return 1;

}

return 0;

}

GLvoid getTestVertex(GLint roadIndex, camDir dir, vec4 dest)

{

if (dir == UP)

{

if (roads[roadIndex].dir == NORTH)

{

dest[0] = roadVertices[10 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[11 + roadIndex \* 4 \* 5];

}

if (roads[roadIndex].dir == SOUTH)

{

dest[0] = roadVertices[0 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[1 + roadIndex \* 4 \* 5];

}

}

if (dir == DOWN)

{

if (roads[roadIndex].dir == NORTH)

{

dest[0] = roadVertices[0 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[1 + roadIndex \* 4 \* 5];

}

if (roads[roadIndex].dir == SOUTH)

{

dest[0] = roadVertices[10 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[11 + roadIndex \* 4 \* 5];

}

}

if (dir == RIGHT)

{

if (roads[roadIndex].dir == WEST)

{

dest[0] = roadVertices[0 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[1 + roadIndex \* 4 \* 5];

}

if (roads[roadIndex].dir == EAST)

{

dest[0] = roadVertices[10 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[11 + roadIndex \* 4 \* 5];

}

}

if (dir == LEFT)

{

if (roads[roadIndex].dir == WEST)

{

dest[0] = roadVertices[10 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[11 + roadIndex \* 4 \* 5];

}

if (roads[roadIndex].dir == EAST)

{

dest[0] = roadVertices[0 + roadIndex \* 4 \* 5];

dest[1] = roadVertices[1 + roadIndex \* 4 \* 5];

}

}

}

#include <road.h>

#include <macros.h>

#include <stdlib.h>

#include <string.h>

// Custom modules

#include <traffic\_density.h>

#include <algorithms.h>

#include <direction.h>

#include <map.h>

#include <cars.h>

#include <rlc.h>

car occupying\_car = {.isActive = true, .velocity = \_0\_CELL\_};

car\* OCCUPYING\_CAR = &occupying\_car;

mat3 carTransformMatrixes[1000];

GLuint carVAO, carVBO, carEBO, carInstanceVBO;

GLfloat carVertices[4 \* 4] =

{

// VERTICES COORDS // TEXTURE COORDS

0.0f, 0.0f, 0.0f, 0.0f,

CAR\_WIDTH, 0.0f, 1.0f, 0.0f,

0.0f, CAR\_LENGTH, 0.0f, 1.0f,

CAR\_WIDTH, CAR\_LENGTH, 1.0f, 1.0f

};

GLint carIndices[6] =

{

0, 2, 3,

0, 1, 3

};

GLint freeCars;

GLvoid setOccupyingCarProperties()

{

occupying\_car.velocity = \_0\_CELL\_;

}

GLvoid bindCellAndCar(RLC\* rlc, car\* Car)

{

roads[rlc->road].lines[rlc->line].cells[rlc->cell] = Car;

}

GLvoid addCar(car\* Car, GLint carIndex, RLC rlc)

{

setBornCarProperties(&cars[carIndex], carIndex, rlc);

setBornCar(&cars[carIndex], carIndex, rlc);

}

GLvoid setBornCarProperties(car\* Car, GLint carIndex, RLC rlc)

{

Car->move = FORWARD;

Car->moveDir = roads[rlc.road].dir;

Car->ID = carIndex;

setTarget(Car);//NONE can't be as it out of range of number of directions

Car->velocity = MIN\_SPAWN\_VELOCITY + rand() % (NUMBER\_OF\_VELOCITY\_TYPES - MIN\_SPAWN\_VELOCITY + 1);

Car->isActive = true;

Car->roadDirMultiplier = getDirMultiplier(Car->moveDir);

memcpy(&Car->currCell, &rlc, sizeof(RLC));

memcpy(&Car->nextCell, &rlc, sizeof(RLC));

bindCellAndCar(&rlc, Car);

}

GLvoid setTarget(car\* Car){

Car->target = rand() % NUMBER\_OF\_DIRECTIONS;//NONE can't be as it out of range of number of directions

if (Car->target == getOppositeDir(Car->moveDir)){

Car->target = Car->moveDir;

}

}

DIRECTION getOppositeDir(DIRECTION dir){

if (dir == NORTH) return SOUTH;

if (dir == SOUTH) return NORTH;

if (dir == EAST) return WEST;

if (dir == WEST) return EAST;

}

GLvoid setCrushedCarProperties(car\* Car, GLint carIndex, RLC rlc)

{

Car->isCrushed = true;

Car->ID = carIndex;

Car->velocity = \_0\_CELL\_;

Car->isActive = true;

memcpy(&Car->currCell, &rlc, sizeof(RLC));//what's it for ?

memcpy(&Car->nextCell, &rlc, sizeof(RLC));

bindCellAndCar(&rlc, Car);

}

GLvoid setBornCar(car\* Car, GLint carIndex, RLC rlc)

{

GLfloat margin = CELL\_WIDTH / 4;

DIRECTION carDir = roads[rlc.road].dir;

GLfloat x1;

GLfloat x2;

GLfloat y1 = 0, y2 = 0;

glm\_mat3\_identity(carTransformMatrixes[carIndex]);

switch (carDir)

{

case NORTH:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord - margin \* 2.0f;

x2 = x1 - margin \* 2.0f;

y1 = roads[rlc.road].startLineCoord + rlc.cell \* CELL\_LENGTH + 0 \* margin;

y2 = y1 + rlc.cell \* CELL\_LENGTH + CAR\_LENGTH + 0 \* margin;

Car->realPos = y1;

vec2 carTranslationVector = { x2, y1 };

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

case SOUTH:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord;

x2 = x1;

y1 = roads[rlc.road].startLineCoord - CAR\_LENGTH - rlc.cell \* CELL\_LENGTH - 0 \* margin;

y2 = 1.0f;//y1 - CAR\_LENGTH - rlc.cell \* CELL\_LENGTH - 0 \* margin;

Car->realPos = y1;

vec2 carTranslationVector = { x2 + CAR\_WIDTH, y1 + CAR\_LENGTH};

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(180.0f));

break;

}

case EAST:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord;

x2 = x1;

y1 = x1;

y2 = x2;

x1 = roads[rlc.road].startLineCoord + rlc.cell \* CELL\_LENGTH;

x2 = roads[rlc.road].startLineCoord + rlc.cell \* CELL\_LENGTH;

Car->realPos = x1;

vec2 carTranslationVector = { x1, y2 + CAR\_WIDTH };

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(-90.0f));

break;

}

case WEST:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord - margin \* 2.0f;

x2 = x1 - margin \* 2.0f;

y1 = x1;

y2 = x2;

x1 = -roads[rlc.road].startLineCoord + rlc.cell \* CELL\_LENGTH;

x2 = -roads[rlc.road].startLineCoord + rlc.cell \* CELL\_LENGTH;

Car->realPos = x2;

vec2 carTranslationVector = { y2, x2 };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

}

}

void setCarOnCross(cross\_cell cell, car \* Car, int carIndex) {

int roadIndex;

for (int i = 0; i < 4; i++) {

roadIndex = crosses[cell.crossNum].enterRoadsIndexes[i];

if (roads[roadIndex].dir == WEST)

break;

}

float start\_x = roadVertices[roadIndex \* 4 \* 5 + 10];

float start\_y = roadVertices[roadIndex \* 4 \* 5 + 11];

float x = start\_x - CELL\_LENGTH \* cell.x;

float y = start\_y - CELL\_LENGTH \* cell.y - CELL\_LENGTH;

glm\_mat3\_identity(carTransformMatrixes[carIndex]);

switch (Car->moveDir) {

case NORTH: {

vec2 carTranslationVector = { y + CELL\_LENGTH, x };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(270.0f));

break;

}

case SOUTH: {

vec2 carTranslationVector = { y, x - CELL\_LENGTH };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

break;

}

case EAST: {

vec2 carTranslationVector = { y + CELL\_LENGTH, x };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(180.0f));

break;

}

case WEST: {

vec2 carTranslationVector = { y, x };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

}

}

GLvoid addCrushedCar(RLC rlc)

{

if (roads[rlc.road].lines[rlc.line].cells[rlc.cell] != NULL){

printf("The crushed car is tried to be set on the occupied cell rlc(%d, %d, %d).\n", rlc.road, rlc.line, rlc.cell);

return;

}

GLint carIndex = getFreeCarIndex();

if (carIndex == NO\_CAR\_INDEX){

// printf("MAX\_CARS: %d", MAX\_CARS);

// for (GLint i = 0; i < MAX\_CARS; i++){

// printCar(&cars[i]);

// }

printf("No free carIndex. It is unable to set a crushed car.\n");

return;

}

increaseDensityData(rlc.road);

--freeCars;

setCrushedCarProperties(&cars[carIndex], carIndex, rlc);

setBornCar(&cars[carIndex], carIndex, rlc);

}

GLfloat getCellWall(RLC rlc)

{

switch (roads[rlc.road].dir)

{

case NORTH:

{

return (roads[rlc.road].startLineCoord + CELL\_LENGTH \* rlc.cell);

}

case SOUTH:

{

return (roads[rlc.road].startLineCoord - CELL\_LENGTH \* rlc.cell);

}

case WEST:

{

return (roads[rlc.road].startLineCoord - CELL\_LENGTH \* rlc.cell);

}

case EAST:

{

return (roads[rlc.road].startLineCoord + CELL\_LENGTH \* rlc.cell);

}

}

}

GLvoid setCarByRLC(car\* Car, GLint carIndex, RLC rlc)

{

GLfloat margin = CELL\_WIDTH / 4;

DIRECTION carDir = roads[rlc.road].dir;

GLfloat x1;

GLfloat x2;

GLfloat y1 = 0, y2 = 0;

glm\_mat3\_identity(carTransformMatrixes[carIndex]);

GLfloat cellWall = getCellWall(rlc);

switch (carDir)

{

case NORTH:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord - (CAR\_WIDTH + 0 \* margin);

x2 = x1 - 0 \* margin;

y1 = cellWall + 0 \* margin;

y2 = y1 + CAR\_LENGTH + 0 \* margin;

Car->realPos = y1;

vec2 carTranslationVector = { x2, y1 };

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

case SOUTH:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord - 0 \* margin;

x2 = x1;

y1 = cellWall - CAR\_LENGTH - 0 \* margin;

y2 = y1 - CAR\_LENGTH - 0 \* margin;

Car->realPos = y1;

vec2 carTranslationVector = { x2, y1 };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(180.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

case EAST:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord + margin \* 2.0f;

x2 = x1 + margin \* 2.0f;

y1 = x1;

y2 = x2 - CAR\_WIDTH;

x1 = cellWall;

x2 = cellWall;

Car->realPos = x1;

vec2 carTranslationVector = { x1, y2 + CAR\_WIDTH };

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(-90.0f));

break;

}

case WEST:

{

x1 = roads[rlc.road].lines[rlc.line].carSpawnCoord - margin \* 2.0f;

x2 = x1 - margin \* 2.0f;

y1 = x1;

y2 = x2;

x1 = -cellWall;

x2 = -cellWall;

Car->realPos = x2;

vec2 carTranslationVector = { y2, x2 };

glm\_rotate2d(carTransformMatrixes[carIndex], glm\_rad(90.0f));

glm\_translate2d(carTransformMatrixes[carIndex], carTranslationVector);

break;

}

}

}

GLvoid getFreeSpotAddress(RLC\* rlc)

{

RLC \* freeSpots = (RLC \*)malloc(sizeof(RLC) \* NUMBER\_OF\_CELLS \* (NUMBER\_OF\_LINES + 1));

if (freeSpots == NULL) {

printf("malloc failed on freeSpots");

exit(1);

}

int freeSpotsCounter = 0;

int randFreeSpotIndex = 0;

for (int i = 0; i < NUMBER\_OF\_ROADS; i++)

{

if (/\*roads[i].isEdge ||\*/ !roads[i].isBeginCross)

{

for (int j = 0; j < NUMBER\_OF\_LINES + 1; j++)

{

if (roads[i].lines[j].cells[0] == NULL)

{

freeSpots[freeSpotsCounter].road = i;

freeSpots[freeSpotsCounter].line = j;

freeSpotsCounter++;

}

}

}

}

if (freeSpotsCounter)

{

randFreeSpotIndex = rand() % freeSpotsCounter;

rlc->road = freeSpots[randFreeSpotIndex].road;

rlc->line = freeSpots[randFreeSpotIndex].line;

rlc->cell = 0;

}

free(freeSpots);

}

GLint getFreeCarIndex()

{

for (int i = 0; i < MAX\_CARS; i++){

if (!cars[i].isActive)

return i;

}

return NO\_CAR\_INDEX;

}

GLvoid setCarsToDefault()

{

freeCars = MAX\_CARS;

for (int i = 0; i < MAX\_CARS; i++)

{

clearCarProperties(&cars[i]);

}

}

GLvoid clearCarProperties(car\* Car)

{

Car->currCell.road = EMPTY;

Car->currCell.line = EMPTY;

Car->currCell.cell = EMPTY;

Car->nextCell.road = EMPTY;

Car->nextCell.line = EMPTY;

Car->nextCell.cell = EMPTY;

Car->crossCurrCell.crossNum = EMPTY;

Car->crossCurrCell.x = EMPTY;

Car->crossCurrCell.y = EMPTY;

Car->crossNextCell.crossNum = EMPTY;

Car->crossNextCell.x = EMPTY;

Car->crossNextCell.y = EMPTY;

Car->roadDirMultiplier = 0;

Car->ID = EMPTY;

Car->realPos = EMPTY;

Car->target = NONE;

Car->velocity = 0;

Car->isActive = false;

Car->isCrushed = false;

Car->move = FORWARD;

Car->moveDir = NONE;

Car->checked = false;// used only on the cross

}

#include <cross.h>

#include <macros.h>

#include <map.h>

#include <direction.h>

#include <map.h>

#include <road.h>

#include <cars.h>

#include <string.h>

#include <traffic\_light.h>

#include <traffic\_density.h>

#include <algorithms.h>

#include <render.h>

#include <stdlib.h>

#include <stdbool.h>

int CROSS\_SIDE;

int HALF\_CROSS\_SIDE;

float CROSS\_WIDTH;

int MAX\_CROSS\_DIGIT;

int MAX\_CROSS\_CELL\_DIGIT;

cross\_roulette rouletteCross = {.crossNum=0, .cellNum=-1};

DIRECTION crossQuaters[8] = {

NORTH, EAST,

SOUTH, WEST,

WEST, NORTH,

EAST, SOUTH

};

//...................................................................................................................

GLvoid printCrossRoulette(){

printf("roulette: cross:%d, cell:%d\n", rouletteCross.crossNum, rouletteCross.cellNum);

}

GLvoid printCarCharacter(car\* Car){

if (Car == NULL)

{

printf("Car == NULL\n");

}

else

{

// printf( "currCell:(%d, %d, %d)\n",

// Car->currCell.road, Car->currCell.line, Car->currCell.cell

// );

printf("ID: %d\n", Car->ID);

printf(

"crossCurrCell:(%d, %d, %d)\ncrossNextCell:(%d, %d, %d)\nvelocity:%d\ncurvingCell: (%d, %d, %d)\n",

Car->crossCurrCell.crossNum, Car->crossCurrCell.x, Car->crossCurrCell.y,

Car->crossNextCell.crossNum, Car->crossNextCell.x, Car->crossNextCell.y,

Car->velocity,

Car->curvingCell.crossNum, Car->curvingCell.x, Car->curvingCell.y

);

if (Car->moveDir == NORTH){

printf("moveDir: NORTH\n");

}

else if (Car->moveDir == SOUTH){

printf("moveDir: SOUTH\n");

}

else if (Car->moveDir == EAST){

printf("moveDir: EAST\n");

}

else if (Car->moveDir == WEST){

printf("moveDir: WEST\n");

}

if (Car->target == NORTH){

printf("target: NORTH\n");

}

else if (Car->target == SOUTH){

printf("target: SOUTH\n");

}

else if (Car->target == EAST){

printf("target: EAST\n");

}

else if (Car->target == WEST){

printf("target: WEST\n");

}

}

}

GLvoid q\_print(queue\* q){

q\_item\* tmp = q->head;

// printf(".........queue.........: %d\n", q->qauntity);

GLint count = q->qauntity;

while (tmp != NULL){

printf("ID: %d, ", tmp->value->ID);

// printf("v: %d, ", tmp->value->velocity);

// printf("act: %d, ", tmp->value->isActive);

// printf("is: %d, ", getCarByCrossCell(&tmp->value->crossCurrCell) != NULL);

// printCrossCell(tmp->value->crossCurrCell);

// printf("!");

// printCrossCell(tmp->value->crossNextCell);

printf("\n");

// printCarCharacter(tmp->value);

// printf("`\n");

tmp = tmp->next;

--count;

}

// printf(".........q.............\n");

}

//...................................................................................................................

car\* getCarByCrossCell(cross\_cell\* c)

{

return crosses[c->crossNum].cells[c->x + c->y \* CROSS\_SIDE];

}

GLvoid initCrossCell(cross\_cell\* c, car\* Car)

{

crosses[c->crossNum].cells[c->x + c->y \* CROSS\_SIDE] = Car;

}

bool isEndedWithCross(RLC\* rlc)

{

return (roads[rlc->road].isEndCross);

}

//....................................................................................................................

GLint getCrossEnter(GLint cell, DIRECTION dir)

{

if (dir == NORTH) return (cell + HALF\_CROSS\_SIDE);

if (dir == SOUTH) return abs(HALF\_CROSS\_SIDE - cell) - 1;

if (dir == EAST) return (cell + HALF\_CROSS\_SIDE);

if (dir == WEST) return abs(HALF\_CROSS\_SIDE - cell) - 1;

}

GLint getCrossExit(GLint cell, DIRECTION dir)

{

if (dir == NORTH) return abs(HALF\_CROSS\_SIDE - cell);

if (dir == SOUTH) return abs(HALF\_CROSS\_SIDE - cell) - 1;

if (dir == EAST) return abs(HALF\_CROSS\_SIDE - cell);

if (dir == WEST) return abs(HALF\_CROSS\_SIDE - cell) - 1;

}

GLvoid getCurvingCell(cross\_cell\* c, car\* Car, cross\_cell firstCellOnRoad)

{

c->crossNum = firstCellOnRoad.crossNum;

DIRECTION dir = Car->moveDir;

DIRECTION target = Car->target;

if (dir == NORTH)

{

c->x = firstCellOnRoad.x;

if (target == EAST) {

c->y = firstCellOnRoad.x;

}

else if (target == WEST) {

c->y = CROSS\_SIDE - firstCellOnRoad.x - 1;

}

}

else if (dir == SOUTH)

{

c->x = firstCellOnRoad.x;

if (target == WEST) {

c->y = firstCellOnRoad.x;

}

else if (target == EAST) {

c->y = CROSS\_SIDE - firstCellOnRoad.x - 1;

}

}

else if (dir == EAST)

{

c->y = firstCellOnRoad.y;

if (target == SOUTH) {

c->x = CROSS\_SIDE - firstCellOnRoad.y - 1;

}

else if (target == NORTH) {

c->x = firstCellOnRoad.y;

}

}

else if (dir == WEST)

{

c->y = firstCellOnRoad.y;

if (target == NORTH) {

c->x = CROSS\_SIDE - firstCellOnRoad.y - 1;

}

else if (target == SOUTH) {

c->x = firstCellOnRoad.y;

}

}

}

//....................................................................................................................

bool getCarByRouletteCross(car\*\* Car)

{

car\* tmpCar;

while (rollRouletteCross())

{

tmpCar = crosses[rouletteCross.crossNum].cells[rouletteCross.cellNum];

if (tmpCar != NULL)

{

\*Car = tmpCar;

return true;

}

}

return false;

}

bool rollRouletteCross()

{

if (rouletteCross.cellNum < MAX\_CROSS\_CELL\_DIGIT){

rouletteCross.cellNum += 1;

}else{

rouletteCross.cellNum = 0;

if (rouletteCross.crossNum < MAX\_CROSS\_DIGIT){

rouletteCross.crossNum += 1;

}

else{

rouletteCross.crossNum = 0;

rouletteCross.cellNum = -1;

return false;

}

}

return true;

}

GLvoid stepCross()

{

car\* Car = NULL;

if (yellowRedChange == true)

{

yellowRedChange = false;

for (GLint crossIndex = 0; crossIndex < NUMBER\_OF\_CROSSES; crossIndex++)

{

q\_fill(&crosses[crossIndex].carsEndingManeuver, &crosses[crossIndex].carsArriving);

}

}

for (GLint crossIndex = 0; crossIndex < NUMBER\_OF\_CROSSES; crossIndex++)

{

q\_item\* item = crosses[crossIndex].carsEndingManeuver.head;

while (item != NULL)

{

thoughtsOfOneCarOnCross(item->value);

item = item->next;

}

}

for (GLint crossIndex = 0; crossIndex < NUMBER\_OF\_CROSSES; crossIndex++)

{

queue\* q = &crosses[crossIndex].carsEndingManeuver;

q\_item\* item = crosses[crossIndex].carsArriving.head;

while (item != NULL)

{

if (!isAnybodyToDriveBeforeNose(q, item->value->moveDir)) {

thoughtsOfOneCarOnCross(item->value);

}else{

item->value->velocity = \_0\_CELL\_;

}

item = item->next;

}

}

}

bool isAnybodyToDriveBeforeNose(queue\* q, DIRECTION ourCarDir)

{

DIRECTION beforeNoseDir = getRightMoveDir(ourCarDir);

q\_item\* item = q->head;

while (item != NULL) {

if (item->value->moveDir == beforeNoseDir) {

return true;

}

item = item->next;

}

return false;

}

//....................................................................................................................

GLvoid rebindCrossCars(car\* Car)

{

initCrossCell(&Car->crossCurrCell, NULL);

if (Car->crossNextCell.crossNum != NEXT\_CELL\_IS\_ON\_ROAD){

initCrossCell(&Car->crossNextCell, Car);

Car->crossCurrCell = Car->crossNextCell;

// printf(">> INIT CROSS << ID: %d\n", Car->ID);

}

else{

increaseDensityData(Car->nextCell.road);

initRoadCell(&Car->nextCell, Car);

Car->currCell = Car->nextCell;

q\_del\_item(&crosses[Car->crossCurrCell.crossNum].carsEndingManeuver, Car);

q\_del\_item(&crosses[Car->crossCurrCell.crossNum].carsArriving, Car);

// printf(">> NEXT ON ROAD << ID: %d\n", Car->ID);

}

}

//....................................................................................................................

GLvoid q\_append(car\* Car, queue\* q)

{

// printf("APPEND ID: %d\n", Car->ID); //printCarCharacter(Car);

q\_item\* item = (q\_item \*)malloc(sizeof(q\_item));

if (item == NULL) exit(-1);

item->value = Car;

item->next = NULL;

// printf("\*\*\*\n");

// q\_print(q);

// printf("\*\*\*\n");

// if (q->head) printf("head ID: %d\n", q->head->value->ID);

// if (q->tail) printf("tail ID: %d\n", q->tail->value->ID);

if (q->head && q->tail) {

q->tail->next = item;

q->tail = item;

}else{

q->head = item;

q->head->next = NULL;

q->tail = item;

}

++q->qauntity;

// printf("======= arrive queue =======\n");

// q\_print(q);

// printf("============================\n");

}

car\* q\_pop(queue\* q)

{

if (q->head == NULL){

exit(-2);

}

car\* Car = q->head->value;

if (q->qauntity > 1) {

q\_item\* tmp = q->head;

q->head = q->head->next;

free(tmp);

}

else {

free(q->head);

q->head = NULL;

q->tail = NULL;

}

--q->qauntity;

return Car;

}

GLvoid q\_fill(queue\* dest, queue\* src)

{

if (src->qauntity == 0){

return;

}

if (dest->qauntity == 0){

dest->head = src->head;

}

else{

dest->tail->next = src->head;

}

dest->tail = src->tail;

dest->qauntity += src->qauntity;

src->head = NULL;

src->tail = NULL;

src->qauntity = 0;

}

GLvoid q\_delete(queue\* q)

{

if (q->head == NULL){

return;

}

q\_item\* tmp;

GLint quantity = q->qauntity;

while (--quantity >= 0)

{

tmp = q->head;

q->head = q->head->next;

free(tmp);

}

q->head = NULL;

q->tail = NULL;

q->qauntity = 0;

}

GLvoid q\_del\_item(queue\* q, car\* Car){

if (q->head == NULL){

return;

}

if (q->head->value == Car){

q\_item\* del\_item = q->head;

if (q->head == q->tail){

q->head = NULL;

q->tail = NULL;

}

else{

q->head = q->head->next;

}

free(del\_item);

--q->qauntity;

return;

}

q\_item\* prev = q->head;

q\_item\* tmp = prev->next;

while (tmp != NULL){

if (tmp->value == Car){

prev->next = tmp->next;

if (q->tail == tmp){

q->tail = prev;

}

free(tmp);

--q->qauntity;

return;

}

prev = tmp;

tmp = tmp->next;

}

}

//....................................................................................................................

GLvoid thoughtsOfOneCarOnCross(car\* Car)

{

if (isItCurvingCell(&Car->crossCurrCell, &Car->curvingCell)){

if (Car->moveDir == NORTH) {

if (Car->target == WEST) {

glm\_translate2d\_x(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(90.0f));

}

if (Car->target == EAST) {

glm\_translate2d\_y(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(-90.0f));

}

}

if (Car->moveDir == SOUTH) {

if (Car->target == WEST) {

glm\_translate2d\_y(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(-90.0f));

}

if (Car->target == EAST) {

glm\_translate2d\_x(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(90.0f));

}

}

if (Car->moveDir == WEST) {

if (Car->target == NORTH) {

glm\_translate2d\_y(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(-90.0f));

}

if (Car->target == SOUTH) {

glm\_translate2d\_x(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(90.0f));

}

}

if (Car->moveDir == EAST) {

if (Car->target == NORTH) {

glm\_translate2d\_x(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(90.0f));

}

if (Car->target == SOUTH) {

glm\_translate2d\_y(carTransformMatrixes[Car->ID], CELL\_LENGTH);

glm\_rotate2d(carTransformMatrixes[Car->ID], glm\_rad(-90.0f));

}

}

Car->moveDir = Car->target;

Car->roadDirMultiplier = getDirMultiplier(Car->target);

}

cross\_cell c;

getNextCrossCell(Car, &c);

if (isInCrossBoards(&c)){

countCarVelocityAndNextCell(Car, &c);

if (Car->velocity != \_0\_CELL\_/\*!isCrossCellsEqual(&Car->crossCurrCell, &Car->crossNextCell)\*/){

initCrossCell(&Car->crossNextCell, OCCUPYING\_CAR);

// printf("occupy is set\n");

}

}else{

RLC rlc;

transformCrossCellIntoRLC(&rlc, Car);

if (getCarPtr(&rlc) == NULL){

Car->nextCell = rlc;

Car->crossNextCell.crossNum = NEXT\_CELL\_IS\_ON\_ROAD;

Car->velocity = \_1\_CELL\_;

setTarget(Car);

initRoadCell(&rlc, OCCUPYING\_CAR);

}else{

Car->velocity = \_0\_CELL\_;

}

}

}

bool isItCurvingCell(cross\_cell\* crossCurrCell, cross\_cell\* curvingCell)

{

// printf("curr\n");printCrossCell(\*crossCurrCell);

// printf("curving\n");printCrossCell(\*curvingCell);

return (crossCurrCell->crossNum == curvingCell->crossNum

&& crossCurrCell->x == curvingCell->x

&& crossCurrCell->y == curvingCell->y);

}

GLvoid getNextCrossCell(car\* Car, cross\_cell\* c)

{

// printf("==========[ID: %d]=========\n", Car->ID);

\*c = Car->crossCurrCell;

cross\_cell cell = \*c;

if (Car->moveDir == NORTH){

if (c->y - 1 < 0){

c->y -= 1;//in order to be processed in foo isInCrossBoards()

return;

}

for (cell.y -= 1; cell.y >= 0; cell.y--){

if (!checkCrossCellSafety(&cell, Car->moveDir)){

break;

}

if (isItCurvingCell(&cell, &Car->curvingCell)){

\*c = cell;

return;

}

}

c->y = cell.y + 1;

}

else if (Car->moveDir == SOUTH){

if (c->y + 1 >= CROSS\_SIDE){

c->y += 1;

return;

}

for (cell.y += 1; cell.y < CROSS\_SIDE; cell.y++){

if (!checkCrossCellSafety(&cell, Car->moveDir)){

break;

}

if (isItCurvingCell(&cell, &Car->curvingCell)){

\*c = cell;

return;

}

}

c->y = cell.y - 1;

}

else if (Car->moveDir == EAST){

if (c->x + 1 >= CROSS\_SIDE){

c->x += 1;

return;

}

for (cell.x += 1; cell.x < CROSS\_SIDE; cell.x++){

if (!checkCrossCellSafety(&cell, Car->moveDir)){

break;

}

if (isItCurvingCell(&cell, &Car->curvingCell)){

\*c = cell;

return;

}

}

c->x = cell.x - 1;

}

else if (Car->moveDir == WEST){

if (c->x - 1 < 0){

c->x -= 1;

return;

}

for (cell.x -= 1; cell.x >= 0; cell.x--){

if (!checkCrossCellSafety(&cell, Car->moveDir)){

break;

}

if (isItCurvingCell(&cell, &Car->curvingCell)){

\*c = cell;

return;

}

}

c->x = cell.x + 1;

}

}

bool checkCrossCellSafety(cross\_cell\* c, DIRECTION dir){

// printf(" f: ");

// printCrossCell(\*c);

car\* Car = getCarByCrossCell(c);

if (Car != NULL){

// printf("problem ID: %d: ", Car->ID);

// printf("!=null\n");

return false;

}

bool RESULT\_1 = true, RESULT\_2 = true;

// printf("\n");

cross\_cell tmp = \*c;

if (dir == NORTH || dir == SOUTH){

// printf("--x:\n");

for (tmp.x -= 1; tmp.x >= 0; tmp.x--){

Car = getCarByCrossCell(&tmp);

if (Car == NULL/\* || Car == OCCUPYING\_CAR\*/){

// printf("null car\n");

continue;

}

if (Car == OCCUPYING\_CAR){

// printf("occupy car\n");

continue;

}

if (Car->moveDir == getLeftMoveDir(dir) || Car->moveDir == getRightMoveDir(dir)){

// printf("problem ID: %d: ", Car->ID); printCrossCell(tmp);

if (c->x - Car->crossNextCell.x > 0){

// printf("true\n");

RESULT\_1 = true;

break;

}else{

// printf("false\n");

RESULT\_1 = false;

break;

}

}

}

tmp = \*c;

// printf("++x:\n");

for (tmp.x += 1; tmp.x < CROSS\_SIDE; tmp.x++){

Car = getCarByCrossCell(&tmp);

if (Car == NULL/\* || Car == OCCUPYING\_CAR\*/){

// printf("null car\n");

continue;

}

if (Car == OCCUPYING\_CAR){

// printf("occupy car\n");

continue;

}

if (Car->moveDir == getLeftMoveDir(dir) || Car->moveDir == getRightMoveDir(dir)){

// printf("problem ID: %d: ", Car->ID); printCrossCell(tmp);

if (Car->crossNextCell.x - c->x > 0){

// printf("true\n");

RESULT\_2 = true;

break;

}else{

// printf("false\n");

RESULT\_2 = false;

break;

}

}

}

}else{

// printf("--y:\n");

for (tmp.y -= 1; tmp.y >= 0; tmp.y--){

Car = getCarByCrossCell(&tmp);

if (Car == NULL/\* || Car == OCCUPYING\_CAR\*/){

// printf("null car\n");

continue;

}

if (Car == OCCUPYING\_CAR){

// printf("occupy car\n");

continue;

}

if (Car->moveDir == getLeftMoveDir(dir) || Car->moveDir == getRightMoveDir(dir)){

// printf("problem ID: %d: ", Car->ID); printCrossCell(tmp);

if (c->y - Car->crossNextCell.y > 0){

// printf("true\n");

RESULT\_1 = true;

break;

}else{

// printf("false\n");

RESULT\_1 = false;

break;

}

}

}

tmp = \*c;

// printf("++y:\n");

for (tmp.y += 1; tmp.y < CROSS\_SIDE; tmp.y++){

Car = getCarByCrossCell(&tmp);

if (Car == NULL/\* || Car == OCCUPYING\_CAR\*/){

// printf("null car\n");

continue;

}

if (Car == OCCUPYING\_CAR){

// printf("occupy car\n");

continue;

}

if (Car->moveDir == getLeftMoveDir(dir) || Car->moveDir == getRightMoveDir(dir)){

// printf("problem ID: %d, ", Car->ID); printCrossCell(tmp);

if (Car->crossNextCell.y - c->y > 0){

// printf("true\n");

RESULT\_2 = true;

break;

}else{

// printf("false\n");

RESULT\_2 = false;

break;

}

}

}

}

return (RESULT\_1 && RESULT\_2);//in case when a car touches the board of cross

}

GLvoid countCarVelocityAndNextCell(car\* Car, cross\_cell\* c){

if (Car->moveDir == NORTH){

Car->velocity = Car->crossCurrCell.y - c->y;

if (Car->velocity > MAX\_CROSS\_VELOCITY){

Car->velocity = MAX\_CROSS\_VELOCITY;

}

Car->crossNextCell = \*c;

Car->crossNextCell.y = Car->crossCurrCell.y - Car->velocity;

}

else if (Car->moveDir == SOUTH){

Car->velocity = c->y - Car->crossCurrCell.y;

if (Car->velocity > MAX\_CROSS\_VELOCITY){

Car->velocity = MAX\_CROSS\_VELOCITY;

}

Car->crossNextCell = \*c;

Car->crossNextCell.y = Car->crossCurrCell.y + Car->velocity;

}

else if (Car->moveDir == EAST){

Car->velocity = c->x - Car->crossCurrCell.x;

if (Car->velocity > MAX\_CROSS\_VELOCITY){

Car->velocity = MAX\_CROSS\_VELOCITY;

}

Car->crossNextCell = \*c;

Car->crossNextCell.x = Car->crossCurrCell.x + Car->velocity;

}

else if (Car->moveDir == WEST){

Car->velocity = Car->crossCurrCell.x - c->x;

if (Car->velocity > MAX\_CROSS\_VELOCITY){

Car->velocity = MAX\_CROSS\_VELOCITY;

}

Car->crossNextCell = \*c;

Car->crossNextCell.x = Car->crossCurrCell.x - Car->velocity;

}

// if (Car->velocity == \_0\_CELL\_){

// printf("\_0\_CELL\_");

// printf("ID: %d, ", Car->ID);

// printCrossCell(Car->crossCurrCell);

// printCarCharacter(Car);

// }

}

bool isCrossCellsEqual(cross\_cell\* c, cross\_cell\* d){

return (c->crossNum == d->crossNum && c->x == c->x && c->y == d->y);

}

bool isInCrossBoards(cross\_cell\* c)

{

return (c->x >= 0 && c->x < CROSS\_SIDE && c->y >= 0 && c->y < CROSS\_SIDE);

}

GLvoid transformCrossCellIntoRLC(RLC\* rlc, car\* Car)

{

rlc->road = crosses[Car->crossCurrCell.crossNum].exitRoadsIndexes[Car->moveDir];

rlc->line = getLineOfAppearingOnRoadFromCross(Car);

rlc->cell = 0;

}

GLint getLineOfAppearingOnRoadFromCross(car\* Car)

{

if (Car->moveDir == NORTH) return getCrossExit(Car->crossCurrCell.x, Car->moveDir);

if (Car->moveDir == SOUTH) return getCrossExit(Car->crossCurrCell.x, Car->moveDir);

if (Car->moveDir == EAST) return getCrossExit(Car->crossCurrCell.y, Car->moveDir);

if (Car->moveDir == WEST) return getCrossExit(Car->crossCurrCell.y, Car->moveDir);

}

//....................................................................................................................

GLvoid transformRLCIntoCrossCell(cross\_cell\* c, car\* Car)

{

c->crossNum = roads[Car->currCell.road].endCrossNum;

if (Car->moveDir == NORTH)

{

c->x = getCrossEnter(Car->currCell.line, Car->moveDir);

c->y = CROSS\_SIDE - 1;

}

else if (Car->moveDir == SOUTH)

{

c->x = getCrossEnter(Car->currCell.line, Car->moveDir);

c->y = 0;

}

else if (Car->moveDir == EAST)

{

c->y = getCrossEnter(Car->currCell.line, Car->moveDir);

c->x = 0;

}

else

{

c->y = getCrossEnter(Car->currCell.line, Car->moveDir);

c->x = CROSS\_SIDE - 1;

}

}

//....................................................................................................................

GLvoid addCross(GLint crossIndex, GLfloat start\_x, GLfloat start\_y, GLint\* enterRoadIndexes, GLint\* exitRoadIndexes)

{

CROSS\_SIDE = (NUMBER\_OF\_LINES + 1) \* 2;

NUMBER\_OF\_CROSS\_CELLS = CROSS\_SIDE \* CROSS\_SIDE;

HALF\_CROSS\_SIDE = CROSS\_SIDE / 2;

MAX\_CROSS\_DIGIT = (NUMBER\_OF\_CROSSES - 1);

MAX\_CROSS\_CELL\_DIGIT = (NUMBER\_OF\_CROSS\_CELLS - 1);

#ifdef DEBUG

DEFAULT\_FOV = 60;

vec2 top\_left = { roads[enterRoadIndexes[2]].startLineCoord, roads[enterRoadIndexes[1]].startLineCoord };

vec2 bottom\_right = { roads[enterRoadIndexes[3]].startLineCoord, roads[enterRoadIndexes[0]].startLineCoord };

cellsVertices = malloc(sizeof(float) \* 2 \* 5 \* (CROSS\_SIDE + NUMBER\_OF\_CELLS \* 2 + 1) \* 2);

for (int i = 0; i < CROSS\_SIDE + NUMBER\_OF\_CELLS \* 2 + 1; i++) {

float line\_vertices[10] = {

top\_left[0], top\_left[1] - CELL\_WIDTH \* i, 1.0f, 0.0f, 0.0f,

bottom\_right[0], top\_left[1] - CELL\_WIDTH \* i, 1.0f, 0.0f, 0.0f

};

memcpy(&cellsVertices[2 \* 5 \* i], line\_vertices, 2 \* 5 \* sizeof(float));

}

for (int i = 0; i < CROSS\_SIDE + NUMBER\_OF\_CELLS \* 2 + 1; i++) {

float line\_vertices[10] = {

top\_left[0] + CELL\_WIDTH \* i, top\_left[1], 1.0f, 0.0f, 0.0f,

top\_left[0] + CELL\_WIDTH \* i, bottom\_right[1], 1.0f, 0.0f, 0.0f

};

memcpy(&cellsVertices[2 \* 5 \* (CROSS\_SIDE + NUMBER\_OF\_CELLS \* 2 + 1) + 2 \* 5 \* i], line\_vertices, 2 \* 5 \* sizeof(float));

}

#endif

crosses[crossIndex].cells = (car \*\*)malloc(sizeof(car \*) \* NUMBER\_OF\_CROSS\_CELLS);

if (crosses[crossIndex].cells == NULL) {

printf("malloc failed on crosses cells on index %d", crossIndex);

exit(1);

}

setCrossProperties(crossIndex, enterRoadIndexes, exitRoadIndexes);

}

GLvoid setCrossProperties(GLint crossIndex, GLint\* enterRoadIndexes, GLint\* exitRoadIndexes)

{

cross\* Cross = &crosses[crossIndex];

for (int i = 0; i < NUMBER\_OF\_CROSS\_ROADS; i++){

Cross->enterRoadsIndexes[i] = enterRoadIndexes[i];

Cross->exitRoadsIndexes[i] = exitRoadIndexes[i];

Cross->enterRoadsPtrs[i] = &roads[enterRoadIndexes[i]];

Cross->exitRoadsPtrs[i] = &roads[exitRoadIndexes[i]];

}

for (GLint i = 0; i < NUMBER\_OF\_CROSS\_CELLS; i++){

Cross->cells[i] = NULL;

}

Cross->carsEndingManeuver.head = NULL;

Cross->carsEndingManeuver.tail = NULL;

Cross->carsEndingManeuver.qauntity = 0;

Cross->carsArriving.head = NULL;

Cross->carsArriving.tail = NULL;

Cross->carsArriving.qauntity = 0;

// filling roads with data about cross

for (int i = 0; i < NUMBER\_OF\_CROSS\_ROADS; i++){

roads[enterRoadIndexes[i]].isEndCross = true;

// roads[enterRoadIndexes[i]].isBeginCross = false;//this property is to be set outside the foo

roads[enterRoadIndexes[i]].endCrossNum = crossIndex;

roads[enterRoadIndexes[i]].endCross = Cross;

roads[exitRoadIndexes[i]].beginCross = Cross;

roads[exitRoadIndexes[i]].isBeginCross = true;

// roads[exitRoadIndexes[i]].isEndCross = false;

}

for (int i = 0; i < NUMBER\_OF\_CROSS\_ROADS; i++){

GLint index = getFreeTrafficLightIndex();

roads[enterRoadIndexes[i]].traffic\_light\_index = index;

roads[enterRoadIndexes[i]].traffic\_light\_ptr = &lights[index];

// printf("dir: %d, enter: %d, light\_index: %d\n", roads[enterRoadIndexes[i]].dir, enterRoadIndexes[i], index);

}

setDefaultTrafficLightProperties();

}

#include <dbg.h>

#include <cross.h>

#include <cars.h>

GLvoid printGrid(GLint crossIndex){

printf("\nCROSS %d\n", crossIndex);

for (int y = 0; y < CROSS\_SIDE; y++){

for (int i = 0; i < CROSS\_SIDE; i++) printf("\_\_\_\_");

printf("\n");

for (int x = 0; x < CROSS\_SIDE; x++){

cross\_cell c = {crossIndex, x, y};

car\* Car = getCarByCrossCell(&c);

if (Car == OCCUPYING\_CAR){

printf("| \* ");

}

else if (Car == NULL){

printf("| ");

}

else{

if (Car->ID < 10) printf("| %d ", Car->ID);

else if (Car->ID < 100) printf("| %d", Car->ID);

}

}

printf("|\n");

}

for (int i = 0; i < CROSS\_SIDE; i++) printf("\_\_\_\_");

printf("\n");

}// Standard

#include <string.h>

// Custom modules

#include <gui.h>

#include <render.h>

#include <cars.h>

#include <road.h>

#include "GLFW/glfw3.h"

#include "algorithms.h"

#include "camera.h"

#include "map.h"

#include "traffic\_density.h"

#include <rlc.h>

#include <texture.h>

#include <shader.h>

#include <dbg.h>

// External

#if defined(\_WIN32) || defined(WIN32)

#include <dirent/dirent.h>

#else

#include <dirent.h>

#endif

#define NK\_ASSERT(a)

#define MAX\_VERTEX\_BUFFER 512 \* 1024

#define MAX\_ELEMENT\_BUFFER 128 \* 1024

#define NK\_INCLUDE\_FIXED\_TYPES

#define NK\_INCLUDE\_STANDARD\_IO

#define NK\_INCLUDE\_STANDARD\_VARARGS

#define NK\_INCLUDE\_DEFAULT\_ALLOCATOR

#define NK\_INCLUDE\_VERTEX\_BUFFER\_OUTPUT

#define NK\_INCLUDE\_FONT\_BAKING

#define NK\_INCLUDE\_DEFAULT\_FONT

#include <nuklear/nuklear.h>

#include <nuklear/nuklear\_glfw\_gl3.h>

#include <style.c>

char userSaveName[MAX\_BUFFER\_SIZE];

int activeFileIndex = 0;

int curr\_step = 0;

config initConfig = {

50,

3,

100,

SEVERAL\_CROSSES,

};

void initGUI()

{

if (isSaveMenuActive) {

showSaveMenu();

return;

}

if (isLoadMenuActive) {

showLoadMenu();

return;

}

if (isInitMenuActive) {

showInitMenu();

return;

}

if (isHelpMenuActive) {

showHelpMenu();

return;

}

if (nk\_begin(context, "PauseMenu", nk\_rect(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT), 0))

{

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "Main Menu", NK\_TEXT\_CENTERED);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 85);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_button\_label(context, "New Model"))

{

isInitMenuActive = true;

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 5, 1);

if (isInit) {

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 85);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_button\_label(context, "Save Model"))

{

strcpy(userSaveName, "");

isSaveMenuActive = true;

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 5, 1);

}

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 85);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_button\_label(context, "Load Model"))

{

isLoadMenuActive = true;

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 5, 1);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 85);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_button\_label(context, "Help"))

isHelpMenuActive = true;

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 5, 1);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 85);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_button\_label(context, "Quit"))

{

exit(0);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_layout\_row\_dynamic(context, 30, 1);

nk\_label(context, "Theme", NK\_TEXT\_CENTERED);

static int op = THEME\_DARK;

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 53);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_option\_label(context, "Black", op == THEME\_BLACK))

{

op = THEME\_BLACK;

set\_style(context, op);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 53);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_option\_label(context, "Blue", op == THEME\_BLUE))

{

op = THEME\_BLUE;

set\_style(context, op);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 53);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_option\_label(context, "Dark", op == THEME\_DARK))

{

op = THEME\_DARK;

set\_style(context, op);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 53);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_option\_label(context, "Red", op == THEME\_RED))

{

op = THEME\_RED;

set\_style(context, op);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_begin(context, NK\_STATIC, 30, 2);

nk\_layout\_row\_push(context, (float)WINDOW\_WIDTH / 2 - 53);

nk\_spacer(context);

nk\_layout\_row\_push(context, 150);

if (nk\_option\_label(context, "White", op == THEME\_WHITE))

{

op = THEME\_WHITE;

set\_style(context, op);

}

nk\_layout\_row\_end(context);

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_layout\_row\_dynamic(context, 30, 1);

nk\_label(context, "Credits", NK\_TEXT\_CENTERED);

float parent\_ratio[] = {0.285f, 0.43f, 0.285f};

float child\_ratio[] = {0.01f, 0.98f, 0.01f};

nk\_layout\_row(context, NK\_DYNAMIC, 244, 3, parent\_ratio);

nk\_spacer(context);

if (nk\_group\_begin(context, "HelpInfo", NK\_WINDOW\_BORDER|NK\_WINDOW\_NO\_SCROLLBAR)) {

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_layout\_row(context, NK\_DYNAMIC, 20, 3, child\_ratio);

nk\_spacer(context);

nk\_label(context, "ÐŸÐ¾Ð´Ð³Ð¾Ñ‚Ð¾Ð²Ð¸Ð»Ð¸:", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, " Ð§ÑƒÐ¿Ñ€Ð¸Ñ†ÐºÐ¸Ð¹ Ð”ÐµÐ½Ð¸Ñ Ð’Ð»Ð°Ð´Ð¼Ð¸Ñ€Ð¾Ð²Ð¸Ñ‡", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, " Ð“Ð¾Ð¿Ñ‚Ð° Ð’Ð»Ð°Ð´Ð¸Ð¼Ð¸Ñ€ Ð”Ð¼Ð¸Ñ‚Ñ€Ð¸ÐµÐ²Ð¸Ñ‡", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, " 5151003/30002", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Ð˜Ð½ÑÑ‚Ð¸Ñ‚ÑƒÑ‚ ÐºÐ¾Ð¼Ð¿ÑŒÑŽÑ‚ÐµÑ€Ð½Ñ‹Ñ… Ð½Ð°ÑƒÐº Ð¸ ÐºÐ¸Ð±ÐµÑ€Ð±ÐµÐ·Ð¾Ð¿Ð°ÑÐ½Ð¾ÑÑ‚Ð¸", NK\_TEXT\_CENTERED);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Ð’Ñ‹ÑÑˆÐ°Ñ ÑˆÐºÐ¾Ð»Ð° ÐºÐ¸Ð±ÐµÑ€Ð±ÐµÐ·Ð¾Ð¿Ð°ÑÐ½Ð¾ÑÑ‚Ð¸", NK\_TEXT\_CENTERED);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Ð¡ÐŸÐ±ÐŸÐ£ Ð¸Ð¼. ÐŸÐµÑ‚Ñ€Ð° Ð’ÐµÐ»Ð¸ÐºÐ¾Ð³Ð¾", NK\_TEXT\_CENTERED);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "2024", NK\_TEXT\_CENTERED);

nk\_spacer(context);

nk\_group\_end(context);

}

nk\_spacer(context);

}

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

void initFont()

{

struct nk\_font\_atlas\* atlas;

struct nk\_font\* jb;

struct nk\_font\_config jb\_cfg;

jb\_cfg = nk\_font\_config(24);

jb\_cfg.oversample\_h = 1;

jb\_cfg.oversample\_v = 1;

jb\_cfg.range = nk\_font\_cyrillic\_glyph\_ranges();

nk\_glfw3\_font\_stash\_begin(&glfw, &atlas);

jb = nk\_font\_atlas\_add\_from\_file(atlas, "../resources/fonts/JetBrainsMono-Bold.ttf", 24, &jb\_cfg);

nk\_glfw3\_font\_stash\_end(&glfw);

nk\_style\_set\_font(context, &jb->handle);

}

void showSaveMenu()

{

if (nk\_begin(context, "SaveMenu", nk\_rect(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT), 0))

{

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "Save Model", NK\_TEXT\_CENTERED);

nk\_layout\_row\_dynamic(context, 25, 1);

nk\_label(context, "Enter save name.", NK\_TEXT\_CENTERED);

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 40, 3);

nk\_spacer(context);

nk\_flags event = nk\_edit\_string\_zero\_terminated(context, NK\_EDIT\_BOX | NK\_EDIT\_AUTO\_SELECT, userSaveName, sizeof(userSaveName), nk\_filter\_ascii);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 40, 5);

nk\_spacer(context);

nk\_spacer(context);

if (nk\_button\_label(context, "Save"))

{

save();

}

}

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

void showLoadMenu()

{

DIR\* directory;

struct dirent\* entry;

directory = opendir("../saves");

char saves[MAX\_SAVES][MAX\_BUFFER\_SIZE];

int save\_counter = 0;

if (directory == NULL)

{

printf("Error opening directory\n");

isLoadMenuActive = false;

return;

}

while ((entry = readdir(directory)) != NULL)

{

memcpy(saves[save\_counter], entry->d\_name, MAX\_BUFFER\_SIZE);

save\_counter++;

}

if (nk\_begin(context, "LoadMenu", nk\_rect(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT), 0))

{

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "Load Model", NK\_TEXT\_CENTERED);

nk\_layout\_row\_dynamic(context, 300, 3);

nk\_spacer(context);

if (nk\_group\_begin(context, "Save files", NK\_WINDOW\_BORDER))

{

if (save\_counter == 2)

{

nk\_layout\_row\_dynamic(context, 30, 1);

nk\_label(context, "No saves found", NK\_TEXT\_CENTERED);

}

else

{

for (int i = 0; i < save\_counter; i++)

{

if (strcmp("..", saves[i]) == 0 || strcmp(".", saves[i]) == 0)

continue;

nk\_layout\_row\_dynamic(context, 30, 1);

if (nk\_option\_label(context, saves[i], i == activeFileIndex))

{

activeFileIndex = i;

}

}

}

nk\_group\_end(context);

}

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 35, 5);

nk\_spacer(context);

nk\_spacer(context);

if (nk\_button\_label(context, "Load"))

{

if (strcmp("..", saves[activeFileIndex]) == 0 || strcmp(".", saves[activeFileIndex]) == 0)

return;

load(saves[activeFileIndex]);

}

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 35, 5);

nk\_spacer(context);

nk\_spacer(context);

if (save\_counter > 2)

{

if (nk\_button\_label(context, "Delete"))

{

char fullname[MAX\_BUFFER\_SIZE + 10] = "../saves/";

strcat(fullname, saves[activeFileIndex]);

remove(fullname);

}

}

}

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

void save()

{

char fullname[MAX\_BUFFER\_SIZE + 10] = "../saves/";

strcat(fullname, userSaveName);

strcat(fullname, ".bin");

FILE\* saveFile = fopen(fullname, "wb");

if (saveFile == NULL)

{

printf("Error: unable to save the model. Check if you have the save folder\n\n");

return;

}

fwrite(&initConfig, sizeof(initConfig), 1, saveFile);

RLC\* occupiedCells = (RLC \*)malloc(sizeof(RLC) \* NUMBER\_OF\_ROADS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_CELLS);

int occupiedCellsCounter = 0;

for (int i = 0; i < NUMBER\_OF\_ROADS; i++)

for (int j = 0; j < NUMBER\_OF\_LINES + 1; j++)

for(int k = 0; k < NUMBER\_OF\_CELLS; k++)

if (roads[i].lines[j].cells[k] == OCCUPYING\_CAR) {

occupiedCells[occupiedCellsCounter].road = i;

occupiedCells[occupiedCellsCounter].line = j;

occupiedCells[occupiedCellsCounter].cell = k;

occupiedCellsCounter++;

}

else {

occupiedCells[occupiedCellsCounter].road = -1;

occupiedCells[occupiedCellsCounter].line = -1;

occupiedCells[occupiedCellsCounter].cell = -1;

occupiedCellsCounter++;

}

fwrite(cars, sizeof(car) \* MAX\_CARS, 1, saveFile);

fwrite(occupiedCells, sizeof(RLC) \* NUMBER\_OF\_ROADS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_CELLS, 1, saveFile);

fwrite(&freeCars, sizeof(int), 1, saveFile);

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES) {

for (int i = 0; i < NUMBER\_OF\_CROSSES; i++) {

int q\_count = crosses[i].carsEndingManeuver.qauntity;

fwrite(&q\_count, sizeof(int), 1, saveFile);

q\_item \* item\_ptr = crosses[i].carsEndingManeuver.head;

while (q\_count > 0) {

fwrite(&item\_ptr->value->ID, sizeof(int), 1, saveFile);

item\_ptr = item\_ptr->next;

q\_count--;

}

q\_count = crosses[i].carsArriving.qauntity;

fwrite(&q\_count, sizeof(int), 1, saveFile);

item\_ptr = crosses[i].carsArriving.head;

while (q\_count > 0) {

fwrite(&item\_ptr->value->ID, sizeof(int), 1, saveFile);

item\_ptr = item\_ptr->next;

q\_count--;

}

int \* cellsID = (int \*)malloc(sizeof(int) \* NUMBER\_OF\_CROSS\_CELLS);

for (int cell = 0; cell < NUMBER\_OF\_CROSS\_CELLS; cell++)

if (crosses[i].cells[cell]) {

if (crosses[i].cells[cell] == OCCUPYING\_CAR) {

cellsID[cell] = -3;

continue;

}

cellsID[cell] = crosses[i].cells[cell]->ID;

}

else {

cellsID[cell] = -2;

}

//cellsID[cell] = crosses[i].cells[cell] ? crosses[i].cells[cell]->ID : -1;

fwrite(cellsID, sizeof(int) \* NUMBER\_OF\_CROSS\_CELLS, 1, saveFile);

free(cellsID);

}

fwrite(lights, sizeof(traffic\_light) \* NUMBER\_OF\_TRAFFIC\_LIGHTS, 1, saveFile);

}

//fwrite(&densityData, sizeof(int) \* NUMBER\_OF\_ROADS, 1, saveFile);

free(occupiedCells);

fclose(saveFile);

isSaveMenuActive = false;

}

void load(char\* fileName)

{

char fullname[MAX\_BUFFER\_SIZE + 10] = "../saves/";

strcat(fullname, fileName);

FILE\* saveFile = fopen(fullname, "rb");

if (saveFile == NULL)

{

printf("Error: failed to open save file while loading\n\n");

return;

}

fread(&initConfig, sizeof(initConfig), 1, saveFile);

init(saveFile);

isLoadMenuActive = false;

}

void showInitMenu() {

if (nk\_begin(context, "InitMenu", nk\_rect(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT), 0))

{

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "New Model", NK\_TEXT\_CENTERED);

nk\_layout\_row\_dynamic(context, 25, 1);

char cars\_label[MAX\_BUFFER\_SIZE];

sprintf(cars\_label, "Max amount of cars: %d", initConfig.max\_cars);

nk\_label(context, cars\_label, NK\_TEXT\_LEFT);

nk\_slider\_int(context, 1, &initConfig.max\_cars, 1000, 1);

nk\_layout\_row\_dynamic(context, 25, 1);

char lines\_label[MAX\_BUFFER\_SIZE];

sprintf(lines\_label, "Number of lines: %d", initConfig.lines);

nk\_label(context, lines\_label, NK\_TEXT\_LEFT);

nk\_slider\_int(context, 1, &initConfig.lines, 64, 1);

nk\_layout\_row\_dynamic(context, 25, 1);

char spawn\_label[MAX\_BUFFER\_SIZE];

sprintf(spawn\_label, "Cars spawn frequency: %d%%", initConfig.spawn\_frequency);

nk\_label(context, spawn\_label, NK\_TEXT\_LEFT);

nk\_slider\_int(context, 0, &initConfig.spawn\_frequency, 100, 1);

nk\_layout\_row\_dynamic(context, 25, 1);

nk\_label(context, "Choose map type:", NK\_TEXT\_LEFT);

if (nk\_option\_label(context, "Vertical Road", initConfig.map\_type == VERTICAL)) initConfig.map\_type = VERTICAL;

if (nk\_option\_label(context, "Horizontal Road", initConfig.map\_type == HORIZONTAL)) initConfig.map\_type = HORIZONTAL;

if (nk\_option\_label(context, "Cross", initConfig.map\_type == CROSS)) initConfig.map\_type = CROSS;

if (nk\_option\_label(context, "Multiple Crosses", initConfig.map\_type == SEVERAL\_CROSSES)) initConfig.map\_type = SEVERAL\_CROSSES;

nk\_layout\_row\_dynamic(context, 40, 5);

nk\_spacer(context);

nk\_spacer(context);

if (nk\_button\_label(context, "Create a new model"))

{

init(NULL);

}

}

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

void init (FILE\* saveFile) {

glm\_mat3\_identity\_array(carTransformMatrixes, 1000);

for (int i = 0; i < 1000; i++)

glm\_translate2d(carTransformMatrixes[i], (vec2){100.0, 100.0});

#ifdef DEBUG

dbgIsCellsInit = false;

dbgVerticesInit = false;

#endif

if (isInit) {

free(densityData);

for (int i = 0; i < NUMBER\_OF\_ROADS; i++) {

for(int j = 0; j < NUMBER\_OF\_LINES; j++) {

free(roads[i].lines[j].cells);

}

free(roads[i].lines);

}

free(lineVertices);

free(cars);

free(roadIndices);

free(roadVertices);

free(roads);

free(lights);

for (int i = 0; i < NUMBER\_OF\_CROSSES; i++) {

free(crosses[i].cells);

q\_delete(&crosses[i].carsArriving);

q\_delete(&crosses[i].carsEndingManeuver);

}

free(crosses);

free(checkedCars);

free(skipCarsFromCross);

free(carAddingQueue);

free(userCarsPtrs);

glDeleteVertexArrays(1, &roadVAO);

glDeleteVertexArrays(1, &carVAO);

glDeleteVertexArrays(1, &lineVAO);

glDeleteVertexArrays(1, &backgroundVAO);

glDeleteBuffers(1, &roadVBO);

glDeleteBuffers(1, &carVBO);

glDeleteBuffers(1, &carInstanceVBO);

glDeleteBuffers(1, &lineVBO);

glDeleteBuffers(1, &backgroundVBO);

glDeleteBuffers(1, &roadEBO);

glDeleteBuffers(1, &carEBO);

glDeleteBuffers(1, &backgroundEBO);

}

traffic\_light\_adding\_index = -1;

midLinesCounter = 0;

densityDataInit = false;

alghorithmsInit = false;

cameraInit = false;

setMap(initConfig.map\_type, initConfig.lines, initConfig.max\_cars, initConfig.spawn\_frequency);

setCarsToDefault();

initBackground();

initRoads();

initLines();

initCars();

step\_count = 0;

if (saveFile) {

RLC \* occupiedCells = (RLC \*)malloc(sizeof(RLC) \* NUMBER\_OF\_CELLS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS);

fread(cars, sizeof(car) \* MAX\_CARS, 1, saveFile);

fread(occupiedCells, sizeof(RLC) \* NUMBER\_OF\_ROADS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_CELLS, 1, saveFile);

fread(&freeCars, sizeof(int), 1, saveFile);

for (int i = 0; i < NUMBER\_OF\_CELLS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS; i++) {

if (occupiedCells[i].road != -1)

roads[occupiedCells[i].road].lines[occupiedCells[i].line].cells[occupiedCells[i].cell] = OCCUPYING\_CAR;

}

for (int i = 0; i < MAX\_CARS; i++) {

if(cars[i].isActive) {

if (cars[i].nextCell.road != NEXT\_CELL\_IS\_ON\_CROSS) {

//setBornCar(&cars[i], i, cars[i].currCell);

roads[cars[i].currCell.road].lines[cars[i].currCell.line].cells[cars[i].currCell.cell] = &cars[i];

} else {

if (cars[i].crossCurrCell.crossNum != -1)

setCarOnCross(cars[i].crossCurrCell, &cars[i], i);

}

}

}

free(occupiedCells);

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES) {

for (int i = 0; i < NUMBER\_OF\_CROSSES; i++) {

int q\_count;

fread(&q\_count, sizeof(int), 1, saveFile);

while(q\_count > 0) {

int carID;

fread(&carID, sizeof(int), 1, saveFile);

q\_append(&cars[carID], &crosses[i].carsEndingManeuver);

q\_count--;

}

fread(&q\_count, sizeof(int), 1, saveFile);

while(q\_count > 0) {

int carID;

fread(&carID, sizeof(int), 1, saveFile);

q\_append(&cars[carID], &crosses[i].carsArriving);

q\_count--;

}

int \* cellsID = (int \*)malloc(sizeof(int) \* NUMBER\_OF\_CROSS\_CELLS);

fread(cellsID, sizeof(int) \* NUMBER\_OF\_CROSS\_CELLS, 1, saveFile);

for (int cell = 0; cell < NUMBER\_OF\_CROSS\_CELLS; cell++) {

if (cellsID[cell] != -2) {

if (cellsID[cell] == -3) {

crosses[i].cells[cell] = OCCUPYING\_CAR;

continue;

}

crosses[i].cells[cell] = &cars[cellsID[cell]];

}

else

crosses[i].cells[cell] = NULL;

}

free(cellsID);

}

traffic\_light \* newLights = (traffic\_light \*)malloc(sizeof(traffic\_light) \* NUMBER\_OF\_TRAFFIC\_LIGHTS);

fread(newLights, sizeof(traffic\_light) \* NUMBER\_OF\_TRAFFIC\_LIGHTS, 1, saveFile);

for (int i = 0; i < NUMBER\_OF\_TRAFFIC\_LIGHTS; i++) {

lights[i].changeTimer = newLights[i].changeTimer;

lights[i].color = newLights[i].color;

}

free(newLights);

}

// free(densityData);

// densityData = (int\* )malloc(sizeof(int) \* NUMBER\_OF\_ROADS);

// fread(&densityData, sizeof(int) \* NUMBER\_OF\_ROADS, 1, saveFile);

fclose(saveFile);

lastTime = glfwGetTime();

timer = lastTime;

deltaTime = 0.0f;

endPauseTime = 0.0f;

}

isInitMenuActive = false;

isInit = true;

paused = false;

}

void showInfo() {

int width = 720;

int height = 320;

if (nk\_begin(context,

"Info",

nk\_rect((float)WINDOW\_WIDTH/2 - 110,

(float)WINDOW\_HEIGHT/2 - 110,

width,

height),

NK\_WINDOW\_BORDER|NK\_WINDOW\_MOVABLE|NK\_WINDOW\_CLOSABLE))

{

char carsInfo[MAX\_BUFFER\_SIZE];

char timeInfo[MAX\_BUFFER\_SIZE];

char stepInfo[MAX\_BUFFER\_SIZE];

char trafficDensityInfo[MAX\_BUFFER\_SIZE];

char carsOnLine[MAX\_BUFFER\_SIZE];

sprintf(carsInfo, "Total cars: %d", MAX\_CARS - freeCars);

sprintf(timeInfo, "Time: %.2f", glfwGetTime());

sprintf(stepInfo, "Updates: %d", step\_count);

nk\_layout\_row\_dynamic(context, 20, 1);

nk\_label(context, carsInfo, NK\_TEXT\_LEFT);

nk\_label(context, timeInfo, NK\_TEXT\_LEFT);

nk\_label(context, stepInfo, NK\_TEXT\_LEFT);

if (isLinePicked) {

if (step\_count && step\_count != curr\_step) {

sprintf(trafficDensityInfo, "Traffic Density: %d", getDensityData(pickedRLC.road));

sprintf(carsOnLine, "Cars on Line: %d", getCarsNumOnLine(pickedRLC.road, pickedRLC.line));

}

nk\_layout\_row\_dynamic(context, 20, 1);

nk\_label(context, trafficDensityInfo, NK\_TEXT\_LEFT);

nk\_label(context, carsOnLine, NK\_TEXT\_LEFT);

} else {

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "Pick a line to get more info.", NK\_TEXT\_LEFT);

}

}

if (nk\_window\_is\_hidden(context, "Info")) {

isShowInfo = false;

return;

}

if (nk\_window\_is\_hovered(context) && glfwGetMouseButton(window, GLFW\_MOUSE\_BUTTON\_LEFT) == GLFW\_PRESS)

nk\_window\_set\_position(context, "Info", nk\_vec2(mousePosX - (float)width / 2, mousePosY));

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

void showHelpMenu() {

if (nk\_begin(context, "HelpMenu", nk\_rect(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT), 0)) {

nk\_layout\_row\_dynamic(context, 50, 1);

nk\_label(context, "Help", NK\_TEXT\_CENTERED);

float parent\_ratio[] = {0.15f, 0.7f, 0.15f};

float child\_ratio[] = {0.01f, 0.98f, 0.01f};

nk\_layout\_row(context, NK\_DYNAMIC, 330, 3, parent\_ratio);

nk\_spacer(context);

if (nk\_group\_begin(context, "HelpInfo", NK\_WINDOW\_BORDER|NK\_WINDOW\_NO\_SCROLLBAR)) {

nk\_layout\_row\_dynamic(context, 10, 1);

nk\_layout\_row(context, NK\_DYNAMIC, 20, 3, child\_ratio);

nk\_spacer(context);

nk\_label(context, "Use WASD to move camera.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Use mouse wheel to zoom in and out.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Left click on road to set a car.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Right click on road to set a crashed car.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "Press ESC to open or close the pause menu.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 20, 1);

nk\_layout\_row(context, NK\_DYNAMIC, 20, 3, child\_ratio);

nk\_spacer(context);

nk\_label(context, "Press '`' or '~' to open or close the info window.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "When the info window is opened, left click on the line to pick it.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_label(context, "You can move around the info window by clicking on it's top panel.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_spacer(context);

nk\_layout\_row\_dynamic(context, 20, 1);

nk\_layout\_row(context, NK\_DYNAMIC, 20, 3, child\_ratio);

nk\_spacer(context);

nk\_label(context, "Press ESC to exit this menu.", NK\_TEXT\_LEFT);

nk\_spacer(context);

nk\_group\_end(context);

}

nk\_spacer(context);

}

nk\_end(context);

nk\_glfw3\_render(&glfw, NK\_ANTI\_ALIASING\_ON, MAX\_VERTEX\_BUFFER, MAX\_ELEMENT\_BUFFER);

}

#include <log.h>

#include <string.h>

#include <stdio.h>

#include <stdbool.h>

// log\_key \_LOG\_KEY\_ = PLAY;

// log\_key \_LOG\_KEY\_ = RECORD;

log\_key \_LOG\_KEY\_ = NO;

FILE\* f = NULL;

// char fileName[100] = "log.txt";

// char fileName[100] = "situation.txt";

char fileName[100] = "a.txt";

GLint fileSize;

GLvoid openFile(){

if (\_LOG\_KEY\_ == NO){

return;

}

if (\_LOG\_KEY\_ == PLAY){

f = fopen(fileName, "r");

fseek(f, 0, SEEK\_END);

fileSize = ftell(f);

rewind(f);

}else if (\_LOG\_KEY\_ == RECORD){

f = fopen(fileName, "w");

}

if (f == NULL){

printf("opening %s failed\n", fileName);

exit(-1);

}

}

GLvoid closeFile(){

if (\_LOG\_KEY\_ == NO)

return;

fclose(f);

}

bool readFile(log\_data\* data){

char buf[20] = {'\0'};

if (ftell(f) == fileSize){

printf("ftell(f) == fileSize\n");

return false;

}

fgets(buf, 20, f);

if (buf[strlen(buf) - 1] == '\n'){

buf[strlen(buf) - 1] = '\0';

}

if (strncmp(buf, "=", 1) == 0){

return false;

}

char \*istr = strtok(buf, SEP);

data->road = atoi(istr);

istr = strtok(NULL, SEP);

data->line = atoi(istr);

istr = strtok(NULL, SEP);

data->cell = atoi(istr);

istr = strtok(NULL, SEP);

data->velocity = atoi(istr);

return true;

}

GLvoid writeFile(log\_data\* data){

fprintf(f, "%d %d %d %d\n", data->road, data->line, data->cell, data->velocity);

}

GLvoid printFileEmptyStep(){

fprintf(f, "\n");

}

GLvoid printFileStepEnd(){

fprintf(f, "=\n");

// printf("fileSize: %ld\n", ftell(f));

}

// External

#include <glad/glad.h>

#include <GLFW/glfw3.h>

#include <cglm/cglm.h>

#define NK\_ASSERT(a)

#define MAX\_VERTEX\_BUFFER 512 \* 1024

#define MAX\_ELEMENT\_BUFFER 128 \* 1024

#define NK\_INCLUDE\_FIXED\_TYPES

#define NK\_INCLUDE\_STANDARD\_IO

#define NK\_INCLUDE\_STANDARD\_VARARGS

#define NK\_INCLUDE\_DEFAULT\_ALLOCATOR

#define NK\_INCLUDE\_VERTEX\_BUFFER\_OUTPUT

#define NK\_INCLUDE\_FONT\_BAKING

#define NK\_INCLUDE\_DEFAULT\_FONT

#define NK\_IMPLEMENTATION

#define NK\_GLFW\_GL3\_IMPLEMENTATION

#include <nuklear/nuklear.h>

#include <nuklear/nuklear\_glfw\_gl3.h>

#include <stb/stb\_image.h>

// Custom modules

#include <macros.h>

#include <shader.h>

#include <render.h>

#include <camera.h>

#include <road.h>

#include <rlc.h>

#include <direction.h>

#include <cross.h>

#include <map.h>

#include <cars.h>

#include <gui.h>

#include <algorithms.h>

#include <search\_tools.h>

#include <traffic\_light.h>

#include <traffic\_density.h>

#include <texture.h>

#include <log.h>

int main()

{

openFile();

initGL();

genShader(&shaderProgram, "vertex\_shader.glsl", "fragment\_shader.glsl");

genShader(&carShader, "vCar.glsl", "fCar.glsl");

genShader(&backgroundShader, "vBackground.glsl", "fBackground.glsl");

initTextures();

initFont();

glfwSetTime(0.0f);

lastTime = glfwGetTime();

timer = lastTime;

while (!glfwWindowShouldClose(window)) {

if (paused || !isInit)

initGUI();

if (!paused) {

if (isInit)

glfwSetTime(glfwGetTime() - getPauseTime());

else

glfwSetTime(0.0f);

currTime = glfwGetTime();

deltaTime += (currTime - lastTime) / limitFPS;

lastTime = currTime;

endPauseTime = 0;

if (isInit) {

while (deltaTime >= 1.0) {

update();

deltaTime--;

}

render();

}

}

else

endPauseTime = glfwGetTime();

nk\_glfw3\_new\_frame(&glfw);

glfwSwapBuffers(window);

glfwPollEvents();

}

closeFile();

quit();

}

// External

#include <glad/glad.h>

// Standard

#include <stdbool.h>

#include <stdio.h>

#include <stdlib.h>

// Custom modules

#include <macros.h>

#include <map.h>

#include <direction.h>

#include <road.h>

#include <cross.h>

#include <cars.h>

#include <traffic\_light.h>

#include <traffic\_density.h>

int NUMBER\_OF\_TRAFFIC\_LIGHTS;

int NUMBER\_OF\_CROSSES;

int NUMBER\_OF\_CROSS\_CELLS;

int NUMBER\_OF\_ROADS;

int NUMBER\_OF\_CELLS;

int NUMBER\_OF\_LINES;

int MAX\_CARS;

int SPAWN\_FREQUENCY;

int MAP\_TYPE;

float DEFAULT\_FOV;

float ROAD\_WIDTH;

float HALF\_ROAD\_WIDTH;

traffic\_light\* lights;

cross\* crosses;

road\* roads;

unsigned int roadVAO, roadVBO, roadEBO;

float\* roadVertices;

int\* roadIndices;

unsigned int lineVAO, lineVBO;

float\* lineVertices;

car \* cars;

int \* densityData;

//a road gets a direction, an edge state

void setMap(int map\_type, int lines, int max\_cars, int spawn\_frequency)

{

if (map\_type == VERTICAL || map\_type == HORIZONTAL) {

NUMBER\_OF\_ROADS = 2;

NUMBER\_OF\_CELLS = 40;

} else if (map\_type == CROSS) {

NUMBER\_OF\_CROSSES = 1;

NUMBER\_OF\_TRAFFIC\_LIGHTS = 4 \* NUMBER\_OF\_CROSSES;

NUMBER\_OF\_ROADS = 8;

NUMBER\_OF\_CELLS = 20;

}

else if (map\_type == SEVERAL\_CROSSES){

NUMBER\_OF\_CROSSES = 4;

NUMBER\_OF\_TRAFFIC\_LIGHTS = 4 \* NUMBER\_OF\_CROSSES;

NUMBER\_OF\_ROADS = 24;

NUMBER\_OF\_CELLS = 30;

}

MAP\_TYPE = map\_type;

NUMBER\_OF\_LINES = lines;

MAX\_CARS = max\_cars;

SPAWN\_FREQUENCY = spawn\_frequency;

ROAD\_WIDTH = CELL\_LENGTH \* (NUMBER\_OF\_LINES + 1.0f);

HALF\_ROAD\_WIDTH = ROAD\_WIDTH / 2.0f;

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES){

crosses = (cross \*)malloc(sizeof(cross) \* NUMBER\_OF\_CROSSES);

if (crosses == NULL) {

printf("malloc failed on crosses");

exit(1);

}

lights = (traffic\_light \*)malloc(sizeof(traffic\_light) \* NUMBER\_OF\_TRAFFIC\_LIGHTS);

if (lights == NULL) {

printf("malloc failed on lights");

exit(1);

}

}

roads = (road \*)malloc(sizeof(road) \* NUMBER\_OF\_ROADS);

if (roads == NULL) {

printf("malloc failed on roads");

exit(1);

}

roadVertices = (float \*)malloc(sizeof(float) \* NUMBER\_OF\_ROADS \* 4 \* 5);

if (roadVertices == NULL) {

printf("malloc failed on roadVertices");

exit(1);

}

roadIndices = (int \*)malloc(sizeof(int) \* NUMBER\_OF\_ROADS \* 6);

if (roadIndices == NULL) {

printf("malloc failed on roadIndices");

exit(1);

}

cars = (car \*)malloc(sizeof(car) \* MAX\_CARS);

if (cars == NULL) {

printf("malloc failed on cars");

exit(1);

}

densityData = (int\*)calloc(NUMBER\_OF\_ROADS, sizeof(int));

if (densityData == NULL) {

printf("malloc failed with densityData");

exit(1);

}

if (MAP\_TYPE == CROSS)

lineVertices = (float \*)malloc(sizeof(float) \* (NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 4 \* 5 \* 2));

else if (map\_type == SEVERAL\_CROSSES)

lineVertices = (float \*)malloc(sizeof(float) \* (NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 12 \* 5 \* 2));

else

lineVertices = (float \*)malloc(sizeof(float) \* (NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 5 \* 2));

if (lineVertices == NULL) {

printf("malloc failed on lineVertices");

exit(1);

}

switch (MAP\_TYPE)

{

case VERTICAL:

DEFAULT\_FOV = 45.0f;

addRoad(0, HALF\_ROAD\_WIDTH, -1.0f, NORTH);

addRoad(1, -HALF\_ROAD\_WIDTH, 1.0f, SOUTH);

break;

case HORIZONTAL:

DEFAULT\_FOV = 38.0f;

addRoad(1, 1.0f, HALF\_ROAD\_WIDTH, WEST);

addRoad(0, -1.0f, -HALF\_ROAD\_WIDTH, EAST);

break;

case CROSS:

DEFAULT\_FOV = 28.0f + NUMBER\_OF\_LINES \* 1.2;

GLfloat crossCenterX = 0.0f;

GLfloat crossCenterY = 0.0f;

setRoadsAroundCross(crossCenterX, crossCenterX);

GLint enterRoadIndexes[] = {0, 1, 2, 3};

GLint exitRoadIndexes[] = {4, 5, 6, 7};

addCross(0, crossCenterX, crossCenterY, enterRoadIndexes, exitRoadIndexes);

break;

case SEVERAL\_CROSSES:

DEFAULT\_FOV = 70.0f;

GLfloat roadLength = NUMBER\_OF\_CELLS \* CELL\_LENGTH;

GLfloat roadLengthHalf = roadLength / 2;

GLfloat roadWidth = (NUMBER\_OF\_LINES + 1) \* CELL\_LENGTH;

GLfloat crossIndent = roadLengthHalf + roadWidth;

GLfloat innerRoadHalf = roadLengthHalf + roadWidth / 2;

GLfloat outerRoadHalf = roadLengthHalf + roadWidth / 2 + roadWidth;

GLfloat nearRoadStart = roadLengthHalf + roadWidth \* 2;

GLfloat farRoadStart = roadLengthHalf + roadWidth \* 2 + roadLength;

GLint enterRoadIndexes0[] = {4, 9, 18, 23};

GLint enterRoadIndexes1[] = {16, 21, 14, 19};

GLint enterRoadIndexes2[] = {12, 17, 2, 7};

GLint enterRoadIndexes3[] = {0, 5, 6, 11};

GLint exitRoadIndexes0[] = { 8, 5, 22, 19};

GLint exitRoadIndexes1[] = {20, 17, 18, 15};

GLint exitRoadIndexes2[] = {16, 13, 6, 3};

GLint exitRoadIndexes3[] = { 4, 1, 10, 7};

addRoad(0, outerRoadHalf, -farRoadStart, NORTH);

addRoad(1, innerRoadHalf, -nearRoadStart, SOUTH);

addRoad(2, -farRoadStart, -outerRoadHalf, EAST);

addRoad(3, -nearRoadStart, -innerRoadHalf, WEST);

addRoad(4, outerRoadHalf, -roadLengthHalf, NORTH);

addRoad(5, innerRoadHalf, roadLengthHalf, SOUTH);

addRoad(6, -roadLengthHalf, -outerRoadHalf, EAST);

addRoad(7, roadLengthHalf, -innerRoadHalf, WEST);

addRoad(8, outerRoadHalf, nearRoadStart, NORTH);

addRoad(9, innerRoadHalf, farRoadStart, SOUTH);

addRoad(10, nearRoadStart, -outerRoadHalf, EAST);

addRoad(11, farRoadStart, -innerRoadHalf, WEST);

addRoad(12, -innerRoadHalf, -farRoadStart, NORTH);

addRoad(13, -outerRoadHalf, -nearRoadStart, SOUTH);

addRoad(14, -farRoadStart, innerRoadHalf, EAST);

addRoad(15, -nearRoadStart, outerRoadHalf, WEST);

addRoad(16, -innerRoadHalf, -roadLengthHalf, NORTH);

addRoad(17, -outerRoadHalf, roadLengthHalf, SOUTH);

addRoad(18, -roadLengthHalf, innerRoadHalf, EAST);

addRoad(19, roadLengthHalf, outerRoadHalf, WEST);

addRoad(20, -innerRoadHalf, nearRoadStart, NORTH);

addRoad(21, -outerRoadHalf, farRoadStart, SOUTH);

addRoad(22, nearRoadStart, innerRoadHalf, EAST);

addRoad(23, farRoadStart, outerRoadHalf, WEST);

addCross(0, crossIndent, crossIndent, enterRoadIndexes0, exitRoadIndexes0);

addCross(1, -crossIndent, crossIndent, enterRoadIndexes1, exitRoadIndexes1);

addCross(2, -crossIndent, -crossIndent, enterRoadIndexes2, exitRoadIndexes2);

addCross(3, crossIndent, -crossIndent, enterRoadIndexes3, exitRoadIndexes3);

break;

}

#ifdef DEBUG

DEFAULT\_FOV = 60.0f;

#endif

for (int i = 0; i < NUMBER\_OF\_ROADS; i++)

{

setLines(i);

}

}

//tempt foo. will be used only for the map with one cross.

GLvoid setRoadsAroundCross(GLfloat start\_x, GLfloat start\_y)

{

GLfloat roadLength = CELL\_LENGTH \* NUMBER\_OF\_CELLS;

GLfloat halfRoadIndent = ((GLfloat)(NUMBER\_OF\_LINES + 1) / 2) \* CELL\_LENGTH;

GLfloat crossIndent = (NUMBER\_OF\_LINES + 1) \* CELL\_LENGTH;

addRoad(0, start\_x + halfRoadIndent, start\_y - crossIndent - roadLength, NORTH);

addRoad(1, start\_x - halfRoadIndent, start\_y + crossIndent + roadLength, SOUTH);

addRoad(2, start\_x - crossIndent - roadLength, start\_y - halfRoadIndent, EAST);

addRoad(3, start\_x + crossIndent + roadLength, start\_y + halfRoadIndent, WEST);

addRoad(4, start\_x + halfRoadIndent, start\_y + crossIndent, NORTH);

addRoad(5, start\_x - halfRoadIndent, start\_y - crossIndent, SOUTH);

addRoad(6, start\_x + crossIndent, start\_y - halfRoadIndent, EAST);

addRoad(7, start\_x - crossIndent, start\_y + halfRoadIndent, WEST);

}

// Standard

#include "cglm/cglm.h"

#include <time.h>

#include <string.h>

// External

#include <stb/stb\_image.h>

// Custom modules

#include <algorithms.h>

#include <camera.h>

#include <cars.h>

#include <map.h>

#include <render.h>

#include <road.h>

#include <search\_tools.h>

#include <shader.h>

#include <texture.h>

#define NK\_ASSERT(a)

#define MAX\_VERTEX\_BUFFER 512 \* 1024

#define MAX\_ELEMENT\_BUFFER 128 \* 1024

#define NK\_INCLUDE\_FIXED\_TYPES

#define NK\_INCLUDE\_STANDARD\_IO

#define NK\_INCLUDE\_STANDARD\_VARARGS

#define NK\_INCLUDE\_DEFAULT\_ALLOCATOR

#define NK\_INCLUDE\_VERTEX\_BUFFER\_OUTPUT

#define NK\_INCLUDE\_FONT\_BAKING

#define NK\_INCLUDE\_DEFAULT\_FONT

#include <nuklear/nuklear.h>

#include <nuklear/nuklear\_glfw\_gl3.h>

GLuint WINDOW\_WIDTH = 1280;

GLuint WINDOW\_HEIGHT = 960;

GLchar WINDOW\_NAME[] = "LOST HIGHWAY";

bool paused = true;

bool isInit = false;

bool isInitMenuActive = false;

bool isSaveMenuActive = false;

bool isLoadMenuActive = false;

bool isHelpMenuActive = false;

bool isShowInfo = false;

bool isLinePicked = false;

RLC pickedRLC;

float mousePosX = 0.0;

float mousePosY = 0.0;

GLFWwindow \*window;

struct nk\_glfw glfw = {0};

struct nk\_context \*context;

GLuint isCarLoc, projectionLoc, modelLoc;

GLdouble limitFPS = 1.0 / FPS;

GLdouble lastTime;

GLdouble deltaTime = 0, currTime = 0, endPauseTime = 0;

GLdouble timer;

unsigned int backgroundVBO, backgroundVAO, backgroundEBO;

#ifdef DEBUG

GLuint cellsVBO, cellsVAO;

GLfloat \*cellsVertices;

bool dbgIsCellsInit = false;

bool dbgVerticesInit = false;

#endif

GLvoid scrollCallback(GLFWwindow \*window, double xoffset, double yoffset) {

if (yoffset < 0.0) {

cameraPos[0] = 0.0;

cameraPos[1] = 0.0;

}

cameraFOV -= (float)yoffset;

if (cameraFOV < 1.0f) {

cameraFOV = 1.0f;

}

if (cameraFOV > DEFAULT\_FOV) {

cameraFOV = DEFAULT\_FOV;

}

}

GLvoid framebufferSizeCallback(GLFWwindow \*window, int width, int height) {

WINDOW\_HEIGHT = height;

WINDOW\_WIDTH = width;

glViewport(0, 0, width, height);

}

GLvoid keyCallback(GLFWwindow \*window, int key, int scancode, int action, int mods) {

if (key == GLFW\_KEY\_ESCAPE && action == GLFW\_PRESS) {

if (isSaveMenuActive) {

isSaveMenuActive = false;

} else if (isLoadMenuActive) {

isLoadMenuActive = false;

} else if (isInitMenuActive) {

isInitMenuActive = false;

} else if (isHelpMenuActive) {

isHelpMenuActive = false;

} else {

paused = !paused;

}

}

if (key == GLFW\_KEY\_F6 && action == GLFW\_PRESS)

glfwSetWindowShouldClose(window, true);

if (key == GLFW\_KEY\_GRAVE\_ACCENT && action == GLFW\_PRESS)

isShowInfo = !isShowInfo;

}

GLvoid cursorPositionCallback(GLFWwindow \*window, double xpos, double ypos) {

mousePosX = xpos;

mousePosY = ypos;

}

void mouseButtonCallback(GLFWwindow \*window, int button, int action, int mods) {

if (action == GLFW\_PRESS && !paused) {

float normalizedX = (2.0f \* mousePosX / WINDOW\_WIDTH) - 1.0f;

float normalizedY = 1.0f - (2.0f \* mousePosY / WINDOW\_HEIGHT);

glUseProgram(carShader);

setProjection(carShader);

setView(carShader);

glUseProgram(shaderProgram);

setProjection(shaderProgram);

setView(shaderProgram);

mat4 inverseProjectionView;

glm\_mat4\_mul(projection, view, inverseProjectionView);

glm\_mat4\_inv(inverseProjectionView, inverseProjectionView);

vec4 clipPos = {normalizedX, normalizedY, 0.0f, 1.0f};

vec4 viewPos;

glm\_mat4\_mulv(inverseProjectionView, clipPos, viewPos);

vec4 worldPos;

glm\_vec4\_divs(viewPos, viewPos[3], worldPos);

dot\_coord mousePos;

mousePos.x = worldPos[0];

mousePos.y = worldPos[1];

RLC rlc;

if (!getRLCbyDot(&rlc, &mousePos)) {

return;

}

else printRLC(rlc, "clicked");

if (button == GLFW\_MOUSE\_BUTTON\_LEFT && !isShowInfo){

appendRLCinCarAddingQueue(rlc);

}

if (button == GLFW\_MOUSE\_BUTTON\_RIGHT && !isShowInfo){

appendRLCinCarAddingQueue\_CRUSH(rlc);

}

if (button == GLFW\_MOUSE\_BUTTON\_LEFT && isShowInfo) {

pickedRLC = rlc;

isLinePicked = true;

}

}

}

GLvoid processKeyboardInput() {

if (glfwGetKey(window, GLFW\_KEY\_W) == GLFW\_PRESS) {

moveCamera(UP);

}

if (glfwGetKey(window, GLFW\_KEY\_S) == GLFW\_PRESS) {

moveCamera(DOWN);

}

if (glfwGetKey(window, GLFW\_KEY\_A) == GLFW\_PRESS) {

moveCamera(LEFT);

}

if (glfwGetKey(window, GLFW\_KEY\_D) == GLFW\_PRESS) {

moveCamera(RIGHT);

}

}

GLvoid initRoads() {

glGenVertexArrays(1, &roadVAO);

glGenBuffers(1, &roadVBO);

glGenBuffers(1, &roadEBO);

glBindVertexArray(roadVAO);

float \* newRoadVertices = (float \*)malloc(sizeof(float) \* NUMBER\_OF\_ROADS \* 4 \* 5);

memcpy(newRoadVertices, roadVertices, sizeof(float) \* NUMBER\_OF\_ROADS \* 4 \* 5);

if (MAP\_TYPE == CROSS) {

newRoadVertices[0 \* 4 \* 5 + 11] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[0 \* 4 \* 5 + 16] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[5 \* 4 \* 5 + 1] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[5 \* 4 \* 5 + 6] += CROSS\_SIDE \* CELL\_WIDTH;

}

if (MAP\_TYPE == SEVERAL\_CROSSES) {

newRoadVertices[0 \* 4 \* 5 + 11] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[0 \* 4 \* 5 + 16] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[1 \* 4 \* 5 + 1] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[1 \* 4 \* 5 + 6] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[4 \* 4 \* 5 + 11] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[4 \* 4 \* 5 + 16] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[5 \* 4 \* 5 + 1] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[5 \* 4 \* 5 + 6] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[12 \* 4 \* 5 + 11] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[12 \* 4 \* 5 + 16] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[13 \* 4 \* 5 + 1] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[13 \* 4 \* 5 + 6] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[16 \* 4 \* 5 + 11] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[16 \* 4 \* 5 + 16] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[17 \* 4 \* 5 + 1] += CROSS\_SIDE \* CELL\_WIDTH;

newRoadVertices[17 \* 4 \* 5 + 6] += CROSS\_SIDE \* CELL\_WIDTH;

}

glBindBuffer(GL\_ARRAY\_BUFFER, roadVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(GLfloat) \* NUMBER\_OF\_ROADS \* 4 \* 5,

newRoadVertices, GL\_STATIC\_DRAW);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, roadEBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(GLint) \* 6 \* NUMBER\_OF\_ROADS,

roadIndices, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)0);

glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)(2 \* sizeof(GLfloat)));

glEnableVertexAttribArray(1);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

free(newRoadVertices);

}

GLvoid initLines() {

glGenVertexArrays(1, &lineVAO);

glGenBuffers(1, &lineVBO);

glBindVertexArray(lineVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, lineVBO);

if (MAP\_TYPE == CROSS) {

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(float) \*

(NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 4 \* 5 \* 2),

lineVertices, GL\_STATIC\_DRAW);

} else if (MAP\_TYPE == SEVERAL\_CROSSES) {

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(float) \*

(NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 12 \* 5 \* 2),

lineVertices, GL\_STATIC\_DRAW);

} else {

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(float) \* (NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + 5 \* 2),

lineVertices, GL\_STATIC\_DRAW);

}

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)0);

glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)(2 \* sizeof(GLfloat)));

glEnableVertexAttribArray(1);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

}

GLvoid initCars() {

glGenBuffers(1, &carInstanceVBO);

glGenVertexArrays(1, &carVAO);

glGenBuffers(1, &carVBO);

glGenBuffers(1, &carEBO);

glBindVertexArray(carVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, carInstanceVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(mat3) \* 1000, carTransformMatrixes,

GL\_DYNAMIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, carVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(float) \* 4 \* 4, carVertices,

GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 4 \* sizeof(float),

(GLvoid \*)0);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 4 \* sizeof(float),

(GLvoid \*)(2 \* sizeof(float)));

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, carEBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(GLint) \* 6, carIndices,

GL\_STATIC\_DRAW);

glBindBuffer(GL\_ARRAY\_BUFFER, carInstanceVBO);

glVertexAttribPointer(2, 3, GL\_FLOAT, GL\_FALSE, sizeof(mat3),

(GLvoid \*)(sizeof(float) \* 0));

glVertexAttribPointer(3, 3, GL\_FLOAT, GL\_FALSE, sizeof(mat3),

(GLvoid \*)(sizeof(float) \* 3));

glVertexAttribPointer(4, 3, GL\_FLOAT, GL\_FALSE, sizeof(mat3),

(GLvoid \*)(sizeof(float) \* 6));

glEnableVertexAttribArray(0);

glEnableVertexAttribArray(1);

glEnableVertexAttribArray(2);

glEnableVertexAttribArray(3);

glEnableVertexAttribArray(4);

glVertexAttribDivisor(0, 0);

glVertexAttribDivisor(1, 0);

glVertexAttribDivisor(2, 1);

glVertexAttribDivisor(3, 1);

glVertexAttribDivisor(4, 1);

glBindVertexArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

}

GLvoid render() {

glClearColor(0.28f, 0.55f, 0.24f, 1.0f);

glClear(GL\_COLOR\_BUFFER\_BIT);

glBindTexture(GL\_TEXTURE\_2D, backgroundTexture);

glUseProgram(backgroundShader);

setProjection(backgroundShader);

setView(backgroundShader);

glBindVertexArray(backgroundVAO);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0);

if (MAP\_TYPE == CROSS || MAP\_TYPE == SEVERAL\_CROSSES){

for (int i = 0; i < NUMBER\_OF\_TRAFFIC\_LIGHTS; i++) {

if (lights[i].color == RED)

glBindTexture(GL\_TEXTURE\_2D, redLight);

if (lights[i].color == GREEN)

glBindTexture(GL\_TEXTURE\_2D, greenLight);

if (lights[i].color == YELLOW)

glBindTexture(GL\_TEXTURE\_2D, yellowLight);

glBindVertexArray(lights[i].VAO);

glDrawElements(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0);

}

}

glUseProgram(shaderProgram);

processKeyboardInput();

setProjection(shaderProgram);

setView(shaderProgram);

glBindVertexArray(roadVAO);

glDrawElements(GL\_TRIANGLES, NUMBER\_OF\_ROADS \* 6, GL\_UNSIGNED\_INT, 0);

glBindVertexArray(lineVAO);

if (MAP\_TYPE == CROSS)

glDrawArrays(GL\_LINES, 0, NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 2 + 2 \* 4);

else if (MAP\_TYPE == SEVERAL\_CROSSES)

glDrawArrays(GL\_LINES, 0, NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 2 + 2 \* 12);

else

glDrawArrays(GL\_LINES, 0, NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 2 + 2);

#ifdef DEBUG

dbgRenderCells();

#endif

glUseProgram(carShader);

glBindTexture(GL\_TEXTURE\_2D, carTexture);

setView(carShader);

setProjection(carShader);

glBindVertexArray(carVAO);

for (int i = 0; i < MAX\_CARS; i++)

if (cars[i].isActive && !cars[i].isCrushed)

moveCarOnScreen(i);

glBindBuffer(GL\_ARRAY\_BUFFER, carInstanceVBO);

glBufferSubData(GL\_ARRAY\_BUFFER, 0, sizeof(mat3) \* MAX\_CARS, carTransformMatrixes);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, carEBO);

glDrawElementsInstanced(GL\_TRIANGLES, 6, GL\_UNSIGNED\_INT, 0, MAX\_CARS);

if (isShowInfo && isInit)

showInfo();

}

GLvoid quit() {

free(densityData);

for (int i = 0; i < NUMBER\_OF\_ROADS; i++) {

for(int j = 0; j < NUMBER\_OF\_LINES; j++) {

free(roads[i].lines[j].cells);

}

free(roads[i].lines);

}

free(lineVertices);

free(cars);

free(roadIndices);

free(roadVertices);

free(roads);

free(lights);

for (int i = 0; i < NUMBER\_OF\_CROSSES; i++) {

free(crosses[i].cells);

q\_delete(&crosses[i].carsArriving);

q\_delete(&crosses[i].carsEndingManeuver);

}

free(crosses);

free(checkedCars);

free(skipCarsFromCross);

free(carAddingQueue);

free(userCarsPtrs);

glDeleteVertexArrays(1, &roadVAO);

glDeleteVertexArrays(1, &carVAO);

glDeleteVertexArrays(1, &lineVAO);

glDeleteVertexArrays(1, &backgroundVAO);

glDeleteBuffers(1, &roadVBO);

glDeleteBuffers(1, &carVBO);

glDeleteBuffers(1, &carInstanceVBO);

glDeleteBuffers(1, &lineVBO);

glDeleteBuffers(1, &backgroundVBO);

glDeleteBuffers(1, &roadEBO);

glDeleteBuffers(1, &carEBO);

glDeleteBuffers(1, &backgroundEBO);

glDeleteTextures(1, &carTexture);

glDeleteTextures(1, &backgroundTexture);

glDeleteTextures(1, &redLight);

glDeleteTextures(1, &greenLight);

glDeleteTextures(1, &yellowLight);

nk\_glfw3\_shutdown(&glfw);

glfwTerminate();

exit(0);

}

GLdouble getPauseTime() {

GLdouble pauseTime = endPauseTime - lastTime;

if (pauseTime > 0) {

return pauseTime;

} else {

return 0;

}

}

GLvoid initGL() {

srand(time(NULL));

glfwInit();

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MAJOR, 3);

glfwWindowHint(GLFW\_CONTEXT\_VERSION\_MINOR, 3);

glfwWindowHint(GLFW\_RESIZABLE, false);

glfwWindowHint(GLFW\_OPENGL\_PROFILE, GLFW\_OPENGL\_CORE\_PROFILE);

window = glfwCreateWindow(WINDOW\_WIDTH, WINDOW\_HEIGHT, WINDOW\_NAME, NULL, NULL);

glfwMakeContextCurrent(window);

int iconWidth, iconHeight;

int channels;

unsigned char \*pixels = stbi\_load("../resources/icons/icon.png",

&iconWidth,

&iconHeight,

&channels,

4);

GLFWimage icons[1];

icons[0].width = iconWidth;

icons[0].height = iconHeight;

icons[0].pixels = pixels;

glfwSetWindowIcon(window, 1, icons);

gladLoadGLLoader((GLADloadproc)glfwGetProcAddress);

glViewport(0, 0, WINDOW\_WIDTH, WINDOW\_HEIGHT);

context = nk\_glfw3\_init(&glfw, window, NK\_GLFW3\_INSTALL\_CALLBACKS);

glfwSwapInterval(1);

glfwSetKeyCallback(window, keyCallback);

glfwSetFramebufferSizeCallback(window, framebufferSizeCallback);

glfwSetScrollCallback(window, scrollCallback);

glfwSetCursorPosCallback(window, cursorPositionCallback);

glfwSetMouseButtonCallback(window, mouseButtonCallback);

glfwSetErrorCallback(errorCallback);

stbi\_image\_free(icons[0].pixels);

}

GLfloat getScreenVelocity(car \*Car) {

return (GLfloat)(Car->velocity \* VELOCITY\_MULTIPLIER) / FPS;

}

GLfloat getScreenVelocityShift(car \*Car) {

DIRECTION carRoadDir = roads[Car->currCell.road].dir;

if (carRoadDir == WEST || carRoadDir == EAST)

return (GLfloat)(\_1\_CELL\_ \* getDirMultiplier(Car->moveDir) \* (-Car->roadDirMultiplier) \* VELOCITY\_MULTIPLIER) / FPS;

else

return (GLfloat)(\_1\_CELL\_ \* getDirMultiplier(Car->moveDir) \* (Car->roadDirMultiplier) \* VELOCITY\_MULTIPLIER) / FPS;

}

GLvoid moveCarOnScreen(GLint carIndex) {

car \*Car = &cars[carIndex];

GLfloat screenVelocity;

if (Car->move == FORWARD) {

screenVelocity = getScreenVelocity(Car);

Car->realPos += screenVelocity;

} else {

screenVelocity = getScreenVelocityShift(Car);

}

if (Car->move == FORWARD)

glm\_translate2d\_y(carTransformMatrixes[carIndex], screenVelocity);

else

glm\_translate2d\_x(carTransformMatrixes[carIndex], screenVelocity);

}

#ifdef DEBUG

GLvoid dbgInitCells() {

glGenVertexArrays(1, &cellsVAO);

glGenBuffers(1, &cellsVBO);

glBindVertexArray(cellsVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, cellsVBO);

if (MAP\_TYPE == CROSS) {

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(float) \* 2 \* 5 \*

(NUMBER\_OF\_CELLS \* 2 + CROSS\_SIDE + 1) \* 2,

cellsVertices, GL\_STATIC\_DRAW);

} else {

glBufferData(GL\_ARRAY\_BUFFER,

sizeof(float) \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS \*

NUMBER\_OF\_CELLS \* 5 \* 8,

cellsVertices, GL\_STATIC\_DRAW);

}

glVertexAttribPointer(0, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)0);

glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(GLfloat),

(GLvoid \*)(2 \* sizeof(GLfloat)));

glEnableVertexAttribArray(1);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

glBindVertexArray(0);

}

GLvoid dbgRenderCells() {

if (!dbgIsCellsInit) {

dbgInitCells();

dbgIsCellsInit = true;

}

glBindVertexArray(cellsVAO);

glDrawArrays(

GL\_LINES, 0,

(2 \* 5 \* 8 \* NUMBER\_OF\_CELLS \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS));

}

#endif

void initBackground() {

float backgroundX = ((2.0f \* 1818 / WINDOW\_HEIGHT) - 1.0f) \* 10.0f;

float backgroundY = (1.0f - (2.0f \* 1080 / WINDOW\_WIDTH)) \* 10.0f;

float backgroundVertices[5 \* 4] = {

backgroundX, backgroundY, 0.0f, 5.0f, 5.0f,

backgroundX, -backgroundY, 0.0f, 5.0f, 0.0f,

-backgroundX, -backgroundY, 0.0f, 0.0f, 0.0f,

-backgroundX, backgroundY, 0.0f, 0.0f, 5.0f

};

unsigned int backgroundIndices[3 \* 2] = {

0, 1, 3,

1, 2, 3

};

glGenVertexArrays(1, &backgroundVAO);

glGenBuffers(1, &backgroundVBO);

glGenBuffers(1, &backgroundEBO);

glBindVertexArray(backgroundVAO);

glBindBuffer(GL\_ARRAY\_BUFFER, backgroundVBO);

glBufferData(GL\_ARRAY\_BUFFER, sizeof(backgroundVertices), backgroundVertices, GL\_STATIC\_DRAW);

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, backgroundEBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(backgroundIndices), backgroundIndices, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glEnableVertexAttribArray(1);

}

void errorCallback(int e, const char \*d) {

printf("Error %d: %s\n", e, d);

}

// Standard

#include <stdlib.h>

#include <string.h>

// Custom modules

#include <road.h>

#include <map.h>

#include <cars.h>

#include <algorithms.h>

#include <render.h>

int midLinesCounter = 0;

DIRECTION getRoadDir(car\* Car)

{

return ((roads + Car->currCell.road)->dir);

}

GLint getDirMultiplier(DIRECTION dir)

{

if (dir == NORTH || dir == EAST) return 1;

else if (dir == SOUTH || dir == WEST) return -1;

else return 0;

}

DIRECTION getLeftMoveDir(DIRECTION dir)

{

if (dir == NORTH) return WEST;

else if (dir == SOUTH) return EAST;

else if (dir == WEST) return SOUTH;

else if (dir == EAST) return NORTH;

else return NONE;

}

DIRECTION getRightMoveDir(DIRECTION dir)

{

if (dir == NORTH) return EAST;

else if (dir == SOUTH) return WEST;

else if (dir == WEST) return NORTH;

else if (dir == EAST) return SOUTH;

else return NONE;

}

car\*\* getFirstCellPtr(RLC rlc) {

return roads[rlc.road].lines[rlc.line].cells;

}

GLvoid addRoad(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, DIRECTION dir)

{

// road properties

roads[roadIndex].dir = dir;

roads[roadIndex].isBeginCross = false; // by default

roads[roadIndex].isEndCross = false; // by default

setRoadBoards(roadIndex, start\_x, start\_y);

setEdgeState(roadIndex, start\_x, start\_y, dir);

GLfloat length = CELL\_LENGTH \* NUMBER\_OF\_CELLS;

setRoad(roadIndex, start\_x, start\_y, length, dir);

}

GLvoid setRoadBoards(GLint roadIndex, GLfloat start\_x, GLfloat start\_y)

{

switch (roads[roadIndex].dir)

{

case NORTH:

{

roads[roadIndex].stem = start\_x;

roads[roadIndex].startLineCoord = start\_y;

roads[roadIndex].endLineCoord = start\_y + (NUMBER\_OF\_CELLS \* CELL\_LENGTH);

break;

}

case SOUTH:

{

roads[roadIndex].stem = start\_x;

roads[roadIndex].startLineCoord = start\_y;

roads[roadIndex].endLineCoord = start\_y - (NUMBER\_OF\_CELLS \* CELL\_LENGTH);

break;

}

case EAST:

{

roads[roadIndex].stem = start\_y;

roads[roadIndex].startLineCoord = start\_x;

roads[roadIndex].endLineCoord = start\_x + (NUMBER\_OF\_CELLS \* CELL\_LENGTH);

break;

}

case WEST:

{

roads[roadIndex].stem = start\_y;

roads[roadIndex].startLineCoord = start\_x;

roads[roadIndex].endLineCoord = start\_x - (NUMBER\_OF\_CELLS \* CELL\_LENGTH);

break;

}

}

// printf("dir: %d, endLIneCoord: %f, startLineCoord: %f\n", roads[roadIndex].dir, roads[roadIndex].endLineCoord, roads[roadIndex].startLineCoord);

}

GLvoid setEdgeState(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, DIRECTION dir)

{

switch (dir){

case NORTH:{

if (start\_y + 1.0f < ERROR\_LIMIT){

roads[roadIndex].isEdge = true;

}else{

roads[roadIndex].isEdge = false;

}

break;

}

case SOUTH:{

if (start\_y - 1.0f < ERROR\_LIMIT){

roads[roadIndex].isEdge = true;

}else{

roads[roadIndex].isEdge = false;

}

break;

}

case EAST:{

if (start\_x + 1.0f < ERROR\_LIMIT){

roads[roadIndex].isEdge = true;

}else{

roads[roadIndex].isEdge = false;

}

break;

}

case WEST:{

if (start\_x - 1.0f < ERROR\_LIMIT){

roads[roadIndex].isEdge = true;

}else{

roads[roadIndex].isEdge = false;

}

break;

}

}

}

GLvoid setRoad(GLint roadIndex, GLfloat start\_x, GLfloat start\_y, GLfloat length, DIRECTION dir)

{

GLint indeces[] =

{

0 + (roadIndex \* 4), 2 + (roadIndex \* 4), 3 + (roadIndex \* 4),

0 + (roadIndex \* 4), 1 + (roadIndex \* 4), 3 + (roadIndex \* 4)

};

memcpy(&roadIndices[roadIndex \* 6], indeces, sizeof(GLint) \* 6);

if (dir == NORTH)

{

GLfloat verticies[] =

{

start\_x + HALF\_ROAD\_WIDTH, start\_y, 0.31f, 0.31f, 0.31f,

start\_x - HALF\_ROAD\_WIDTH, start\_y, 0.31f, 0.31f, 0.31f,

start\_x + HALF\_ROAD\_WIDTH, start\_y + length, 0.31f, 0.31f, 0.31f,

start\_x - HALF\_ROAD\_WIDTH, start\_y + length, 0.31f, 0.31f, 0.31f

};

memcpy(&roadVertices[roadIndex \* 4 \* 5], verticies, sizeof(GLfloat) \* 4 \* 5);

}

else if (dir == SOUTH)

{

GLfloat verticies[] =

{

start\_x + HALF\_ROAD\_WIDTH, start\_y, 0.31f, 0.31f, 0.31f,

start\_x - HALF\_ROAD\_WIDTH, start\_y, 0.31f, 0.31f, 0.31f,

start\_x + HALF\_ROAD\_WIDTH, start\_y - length, 0.31f, 0.31f, 0.31f,

start\_x - HALF\_ROAD\_WIDTH, start\_y - length, 0.31f, 0.31f, 0.31f

};

memcpy(&roadVertices[roadIndex \* 4 \* 5], verticies, sizeof(GLfloat) \* 4 \* 5);

}

else if (dir == EAST)

{

GLfloat verticies[] =

{

start\_x, start\_y + HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x, start\_y - HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x + length, start\_y + HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x + length, start\_y - HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f

};

memcpy(&roadVertices[roadIndex \* 4 \* 5], verticies, sizeof(GLfloat) \* 4 \* 5);

}

else if (dir == WEST)

{

GLfloat vertices[] =

{

start\_x, start\_y + HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x, start\_y - HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x - length, start\_y + HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f,

start\_x - length, start\_y - HALF\_ROAD\_WIDTH, 0.31f, 0.31f, 0.31f

};

memcpy(&roadVertices[roadIndex \* 4 \* 5], vertices, sizeof(GLfloat) \* 4 \* 5);

}

}

GLvoid setLines(GLint roadIndex)

{

roads[roadIndex].lines = (line \*)malloc(sizeof(line) \* (NUMBER\_OF\_LINES + 1));

if (roads[roadIndex].lines == NULL) {

printf("malloc failed on lines on road %d", roadIndex);

exit(1);

}

for (int i = 0; i < NUMBER\_OF\_LINES + 1; i++) {

roads[roadIndex].lines[i].cells = (car \*\*)malloc(sizeof(car \*) \* NUMBER\_OF\_CELLS);

if (roads[roadIndex].lines[i].cells == NULL) {

printf("malloc failed on cells on road %d line %d", roadIndex, i);

exit(1);

}

}

GLfloat stride = CELL\_WIDTH;

int i;

if (roads[roadIndex].dir == NORTH || roads[roadIndex].dir == SOUTH)

{

GLfloat x, y1, y2;

if (roads[roadIndex].dir == SOUTH) {

stride \*= -1;

x = roadVertices[0 + 4 \* 5 \* roadIndex];

}

else

x = roadVertices[0 + 4 \* 5 \* roadIndex] - ROAD\_WIDTH;

y1 = roadVertices[1 + 4 \* 5 \* roadIndex];

y2 = roadVertices[11 + 4 \* 5 \* roadIndex];

for (i = 0; i < NUMBER\_OF\_LINES; i++)

{

x += stride;

GLfloat vertices[] =

{

x, y1, 1.0f, 1.0f, 1.0f,

x, y2, 1.0f, 1.0f, 1.0f

};

memcpy(&lineVertices[i \* 5 \* 2 + roadIndex \* 5 \* NUMBER\_OF\_LINES \* 2], vertices, sizeof(float) \* 5 \* 2);

for (int cell = 0; cell < NUMBER\_OF\_CELLS; cell++) {

roads[roadIndex].lines[i].cells[cell] = NULL;

}

roads[roadIndex].lines[i].carSpawnCoord = x;

}

if (MAP\_TYPE == VERTICAL && roadIndex == 0) {

float vertices[] =

{

x - stride \* NUMBER\_OF\_LINES, y1, 0.0f, 0.0f, 0.0f,

x - stride \* NUMBER\_OF\_LINES, y2, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

}

if (MAP\_TYPE == CROSS && roadIndex < 4) {

float vertices[] =

{

x - stride \* NUMBER\_OF\_LINES, y1, 0.0f, 0.0f, 0.0f,

x - stride \* NUMBER\_OF\_LINES, y2, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + midLinesCounter \* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

midLinesCounter++;

}

if (MAP\_TYPE == SEVERAL\_CROSSES && (

roadIndex == 1 ||

roadIndex == 5 ||

roadIndex == 9 ||

roadIndex == 13 ||

roadIndex == 17 ||

roadIndex == 21

)) {

float vertices[] =

{

x - stride \* NUMBER\_OF\_LINES, y1, 0.0f, 0.0f, 0.0f,

x - stride \* NUMBER\_OF\_LINES, y2, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + midLinesCounter \* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

midLinesCounter++;

}

for (int cell = 0; cell < NUMBER\_OF\_CELLS; cell++) {

roads[roadIndex].lines[i].cells[cell] = NULL;

}

roads[roadIndex].lines[i].carSpawnCoord = x + stride;

#ifdef DEBUG

if (MAP\_TYPE != CROSS) {

if (!dbgVerticesInit) {

cellsVertices = malloc(sizeof(float) \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS \* NUMBER\_OF\_CELLS \* 5 \* 8);

dbgVerticesInit = true;

}

DEFAULT\_FOV = 60;

stride = fabs(stride);

x = roadVertices[0 + 4 \* 5 \* roadIndex] - ROAD\_WIDTH;

y1 = roadVertices[1 + 4 \* 5 \* roadIndex];

y2 = roadVertices[11 + 4 \* 5 \* roadIndex];

int dir\_multiplier;

if (roads[roadIndex].dir == NORTH)

dir\_multiplier = 1;

else

dir\_multiplier = -1;

for (i = 0; i < NUMBER\_OF\_LINES + 1; i++) {

x += stride;

for(int j = 0; j < NUMBER\_OF\_CELLS; j++) {

float cellVertices[] = {

x, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x, y1 + ((j + 1) \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x - CELL\_WIDTH, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x - CELL\_WIDTH, y1 + ((j + 1) \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x - CELL\_WIDTH, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x - CELL\_WIDTH, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f,

x, y1 + (j \* CELL\_LENGTH) \* dir\_multiplier, 1.0f, 0.0f, 0.0f

};

int index = roadIndex \* 5 \* 8 \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_CELLS + i \* 5 \* 8 \* NUMBER\_OF\_CELLS + j \* 5 \* 8;

memcpy(&cellsVertices[index], cellVertices, sizeof(GLfloat) \* 5 \* 8);

}

}

}

#endif

}

if (roads[roadIndex].dir == EAST || roads[roadIndex].dir == WEST)

{

GLfloat y, x1, x2;

if (roads[roadIndex].dir == EAST) {

y = roadVertices[1 + 4 \* 5 \* roadIndex];

stride \*= -1;

}

else

y = roadVertices[1 + 4 \* 5 \* roadIndex] - ROAD\_WIDTH;

x1 = roadVertices[0 + 4 \* 5 \* roadIndex];

x2 = roadVertices[10 + 4 \* 5 \* roadIndex];

for (i = 0; i < NUMBER\_OF\_LINES; i++)

{

y += stride;

GLfloat vertices[] =

{

x1, y, 1.0f, 1.0f, 1.0f,

x2, y, 1.0f, 1.0f, 1.0f

};

memcpy(&lineVertices[i \* 5 \* 2 + roadIndex \* 5 \* NUMBER\_OF\_LINES \* 2], vertices, sizeof(float) \* 5 \* 2);

for (int cell = 0; cell < NUMBER\_OF\_CELLS; cell++) {

roads[roadIndex].lines[i].cells[cell] = NULL;

}

roads[roadIndex].lines[i].carSpawnCoord = y;

}

if (MAP\_TYPE == HORIZONTAL && roadIndex == 0) {

GLfloat vertices[] =

{

x1, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f,

x2, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

}

if (MAP\_TYPE == CROSS && roadIndex < 4) {

GLfloat vertices[] =

{

x1, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f,

x2, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + midLinesCounter \* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

midLinesCounter++;

}

if (MAP\_TYPE == SEVERAL\_CROSSES && (

roadIndex == 2 ||

roadIndex == 7 ||

roadIndex == 10 ||

roadIndex == 14 ||

roadIndex == 18 ||

roadIndex == 22

)) {

if (roadIndex == 6) {

GLfloat vertices[] =

{

x1, y + stride, 0.0f, 0.0f, 0.0f,

x2, y + stride, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + midLinesCounter\* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

} else {

GLfloat vertices[] =

{

x1, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f,

x2, y - stride \* NUMBER\_OF\_LINES, 0.0f, 0.0f, 0.0f

};

memcpy(&lineVertices[NUMBER\_OF\_LINES \* NUMBER\_OF\_ROADS \* 5 \* 2 + midLinesCounter\* 5 \* 2], vertices, sizeof(float) \* 5 \* 2);

}

midLinesCounter++;

}

for (int cell = 0; cell < NUMBER\_OF\_CELLS; cell++) {

roads[roadIndex].lines[i].cells[cell] = NULL;

}

roads[roadIndex].lines[i].carSpawnCoord = y + stride;

#ifdef DEBUG

if (MAP\_TYPE != CROSS) {

if (!dbgVerticesInit) {

cellsVertices = malloc(sizeof(float) \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_ROADS \* NUMBER\_OF\_CELLS \* 5 \* 8);

dbgVerticesInit = true;

}

DEFAULT\_FOV = 60;

stride = fabs(stride);

y = roadVertices[1 + 4 \* 5 \* roadIndex] - ROAD\_WIDTH;

x1 = roadVertices[0 + 4 \* 5 \* roadIndex];

x2 = roadVertices[10 + 4 \* 5 \* roadIndex];

int dir\_multiplier;

if (roads[roadIndex].dir == EAST)

dir\_multiplier = 1;

else

dir\_multiplier = -1;

for (i = 0; i < NUMBER\_OF\_LINES + 1; i++) {

y += stride;

for(int j = 0; j < NUMBER\_OF\_CELLS; j++) {

float cellVertices[] = {

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y, 1.0f, 0.0f, 0.0f,

x1 + ((j + 1) \* CELL\_LENGTH) \* dir\_multiplier, y, 1.0f, 0.0f, 0.0f,

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y - CELL\_WIDTH, 1.0f, 0.0f, 0.0f,

x1 + ((j + 1) \* CELL\_LENGTH) \* dir\_multiplier, y - CELL\_WIDTH, 1.0f, 0.0f, 0.0f,

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y - CELL\_WIDTH, 1.0f, 0.0f, 0.0f,

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y, 1.0f, 0.0f, 0.0f,

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y - CELL\_WIDTH, 1.0f, 0.0f, 0.0f,

x1 + (j \* CELL\_LENGTH) \* dir\_multiplier, y, 1.0f, 0.0f, 0.0f

};

int index = roadIndex \* 5 \* 8 \* (NUMBER\_OF\_LINES + 1) \* NUMBER\_OF\_CELLS + i \* 5 \* 8 \* NUMBER\_OF\_CELLS + j \* 5 \* 8;

memcpy(&cellsVertices[index], cellVertices, sizeof(GLfloat) \* 5 \* 8);

}

}

}

#endif

}

}

GLint isFurhterThanEndLine(car\* Car, road\* Road)

{

if (Road->dir == NORTH || Road->dir == EAST){

if (Car->realPos >= Road->endLineCoord){

//printf("isFurther\n");

return 1;

}else{

// printf("realpos: %f, endLine: %f\n", Car->realPos, Road->endLineCoord);

// printCarProperties(Car->currCell);

}

}else{

if (Car->realPos <= Road->endLineCoord)

return 1;

}

return 0;

}

GLint isOutOfScreenSpace(GLfloat realPos)

{

if (realPos <= -1.0f || realPos >= 1.0f)

return 1;

return 0;

}

int getCarsNumOnLine(int roadIndex, int lineIndex) {

int counter = 0;

for (int i = 0; i < NUMBER\_OF\_CELLS; i++) {

car \* cellPtr = roads[roadIndex].lines[lineIndex].cells[i];

if (cellPtr && cellPtr->isActive)

counter++;

}

return counter;

}

GLvoid printRoadProperties(GLint i){

road\* r = &roads[i];

printf("isBeginCross: %d\n isEndCross: %d\n nedCrossNum: %d\n\n",

r->isBeginCross, r->isEndCross, r->endCrossNum);

}

#include <search\_tools.h>

// Standard

#include <stdbool.h>

#include <stdlib.h>

#include <stdio.h>

#include <road.h>

#include <algorithms.h>

#include <map.h>

dot\_coord \_DOT\_ = {NO\_COORD, NO\_COORD};

GLfloat absFloat(GLfloat a)

{

if (a < 0) return (-1) \* a;

else return a;

}

bool getRLCbyDot(RLC\* rlc, dot\_coord\* dot)

{

if (dot->x == NO\_COORD || dot->y == NO\_COORD)

{

return false;

}

bool is\_in = false;

GLint roadIndex = getRoadIndex(dot);

if (roadIndex == NO\_ROAD\_INDEX)

{

return false;

}

GLint lineIndex = getLineIndex(dot, roadIndex);

if (lineIndex == NO\_LINE\_INDEX)

{

return false;

}

GLint cellIndex = getCellIndex(dot, roadIndex, lineIndex);

if (cellIndex == NO\_LINE\_INDEX)

{

return false;

}

rlc->road = roadIndex;

rlc->line = lineIndex;

rlc->cell = cellIndex;

return true;

}

GLint getRoadIndex(dot\_coord\* dot)

{

GLint roadIndex = NO\_ROAD\_INDEX;

bool is\_in = false;

for (roadIndex = 0; roadIndex < NUMBER\_OF\_ROADS; roadIndex++)

{

if (is\_in = isInRoadRange(roadIndex, dot))

{

break;

}

}

if (is\_in)

{

return roadIndex;

}

else

{

return NO\_ROAD\_INDEX;

}

}

//If the chosen dot belong to the line confining the cell, then isInRoadRange() returns FALSE.

bool isInRoadRange(GLint roadIndex, dot\_coord\* dot)

{

road\* Road = &roads[roadIndex];

if (Road->dir == NORTH || Road->dir == SOUTH) {

if (!(dot->x > Road->stem - HALF\_ROAD\_WIDTH && dot->x < Road->stem + HALF\_ROAD\_WIDTH)) {

return false;

}

} else {

if (!(dot->y > Road->stem - HALF\_ROAD\_WIDTH && dot->y < Road->stem + HALF\_ROAD\_WIDTH)) {

return false;

}

}

switch (Road->dir)

{

case NORTH:

{

if (dot->y > Road->startLineCoord && dot->y < Road->endLineCoord)

{

return true;

}

else return false;

}

case SOUTH:

{

if (dot->y < Road->startLineCoord && dot->y > Road->endLineCoord)

{

return true;

}

else return false;

}

case EAST:

{

if (dot->x > Road->startLineCoord && dot->x < Road->endLineCoord)

{

return true;

}

else return false;

}

case WEST:

{

if (dot->x < Road->startLineCoord && dot->x > Road->endLineCoord)

{

return true;

}

else return false;

}

}

}

GLint getLineIndex(dot\_coord\* dot, GLint roadIndex)

{

GLfloat leftCheek = getRoadLeftCheek(roadIndex);

GLfloat recess;

DIRECTION dir = roads[roadIndex].dir;

if (dir == NORTH || dir == SOUTH)

{

recess = absFloat(dot->x - leftCheek);

}

else

{

recess = absFloat(dot->y - leftCheek);

}

GLint index = recess / CELL\_WIDTH;

// printf("leftCheek: %f, recess: %f, index: %d\n", leftCheek, recess, index);

if (index >= 0 && index < NUMBER\_OF\_LINES + 1)

{

return index;

}

else

{

return NO\_LINE\_INDEX;

}

}

GLfloat getRoadLeftCheek(GLint roadIndex)

{

switch (roads[roadIndex].dir)

{

case NORTH:

{

return roads[roadIndex].stem - HALF\_ROAD\_WIDTH;

}

case SOUTH:

{

return roads[roadIndex].stem + HALF\_ROAD\_WIDTH;

}

case EAST:

{

return roads[roadIndex].stem + HALF\_ROAD\_WIDTH;

}

case WEST:

{

return roads[roadIndex].stem - HALF\_ROAD\_WIDTH;

}

}

}

GLint getCellIndex(dot\_coord\* dot, GLint roadIndex, GLint lineIndex)

{

GLfloat recess;

DIRECTION dir = roads[roadIndex].dir;

if (dir == NORTH || dir == SOUTH)

{

recess = absFloat(dot->y - roads[roadIndex].startLineCoord);

}

else

{

recess = absFloat(dot->x - roads[roadIndex].startLineCoord);

}

GLint index = recess / CELL\_LENGTH;

if (index >= 0 && index < NUMBER\_OF\_CELLS)

{

return index;

}

else

{

return NO\_CELL\_INDEX;

}

}

GLvoid clearDot(dot\_coord\* dot)

{

dot->x = NO\_COORD;

dot->y = NO\_COORD;

}

// Standard

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

// Custom modules

#include <shader.h>

int shaderProgram, carShader, backgroundShader;

char\* getShaderContent(const GLchar\* fileName)

{

FILE\* shaderFile;

long size = 0;

char\* shaderContent;

shaderFile = fopen(fileName, "r");

if (shaderFile == NULL)

printf("Failed to open %s\n", fileName);

fseek(shaderFile, 0, SEEK\_END);

size = ftell(shaderFile);

rewind(shaderFile);

shaderContent = memset(malloc(size + 1), '\0', size + 1);

if (shaderContent == NULL) {

printf("malloc failed with shaderContent");

exit(1);

}

fread(shaderContent, 1, size, shaderFile);

fclose(shaderFile);

return shaderContent;

}

void genShader(int \* shader, char \* vertexShaderName, char \* fragmentShaderName)

{

char vertexShaderFullName[128] = "../source/shaders/";

char fragmentShaderFullName[128] = "../source/shaders/";

strcat(vertexShaderFullName, vertexShaderName);

strcat(fragmentShaderFullName, fragmentShaderName);

const char\* vertexShaderSource = getShaderContent(vertexShaderFullName);

const char\* fragmentShaderSource = getShaderContent(fragmentShaderFullName);

GLuint vertexShader;

vertexShader = glCreateShader(GL\_VERTEX\_SHADER);

glShaderSource(vertexShader, 1, &vertexShaderSource, NULL);

glCompileShader(vertexShader);

GLuint fragmentShader;

fragmentShader = glCreateShader(GL\_FRAGMENT\_SHADER);

glShaderSource(fragmentShader, 1, &fragmentShaderSource, NULL);

glCompileShader(fragmentShader);

\*shader = glCreateProgram();

glAttachShader(\*shader, vertexShader);

glAttachShader(\*shader, fragmentShader);

glLinkProgram(\*shader);

glDeleteShader(vertexShader);

glDeleteShader(fragmentShader);

}

enum theme {THEME\_BLACK, THEME\_WHITE, THEME\_RED, THEME\_BLUE, THEME\_DARK};

static void

set\_style(struct nk\_context \*ctx, enum theme theme)

{

struct nk\_color table[NK\_COLOR\_COUNT];

if (theme == THEME\_WHITE) {

table[NK\_COLOR\_TEXT] = nk\_rgba(70, 70, 70, 255);

table[NK\_COLOR\_WINDOW] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_HEADER] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_BORDER] = nk\_rgba(0, 0, 0, 255);

table[NK\_COLOR\_BUTTON] = nk\_rgba(185, 185, 185, 255);

table[NK\_COLOR\_BUTTON\_HOVER] = nk\_rgba(170, 170, 170, 255);

table[NK\_COLOR\_BUTTON\_ACTIVE] = nk\_rgba(160, 160, 160, 255);

table[NK\_COLOR\_TOGGLE] = nk\_rgba(150, 150, 150, 255);

table[NK\_COLOR\_TOGGLE\_HOVER] = nk\_rgba(120, 120, 120, 255);

table[NK\_COLOR\_TOGGLE\_CURSOR] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_SELECT] = nk\_rgba(190, 190, 190, 255);

table[NK\_COLOR\_SELECT\_ACTIVE] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_SLIDER] = nk\_rgba(190, 190, 190, 255);

table[NK\_COLOR\_SLIDER\_CURSOR] = nk\_rgba(80, 80, 80, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_HOVER] = nk\_rgba(70, 70, 70, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_ACTIVE] = nk\_rgba(60, 60, 60, 255);

table[NK\_COLOR\_PROPERTY] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_EDIT] = nk\_rgba(150, 150, 150, 255);

table[NK\_COLOR\_EDIT\_CURSOR] = nk\_rgba(0, 0, 0, 255);

table[NK\_COLOR\_COMBO] = nk\_rgba(175, 175, 175, 255);

table[NK\_COLOR\_CHART] = nk\_rgba(160, 160, 160, 255);

table[NK\_COLOR\_CHART\_COLOR] = nk\_rgba(45, 45, 45, 255);

table[NK\_COLOR\_CHART\_COLOR\_HIGHLIGHT] = nk\_rgba( 255, 0, 0, 255);

table[NK\_COLOR\_SCROLLBAR] = nk\_rgba(180, 180, 180, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR] = nk\_rgba(140, 140, 140, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_HOVER] = nk\_rgba(150, 150, 150, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_ACTIVE] = nk\_rgba(160, 160, 160, 255);

table[NK\_COLOR\_TAB\_HEADER] = nk\_rgba(180, 180, 180, 255);

nk\_style\_from\_table(ctx, table);

} else if (theme == THEME\_RED) {

table[NK\_COLOR\_TEXT] = nk\_rgba(190, 190, 190, 255);

table[NK\_COLOR\_WINDOW] = nk\_rgba(30, 33, 40, 215);

table[NK\_COLOR\_HEADER] = nk\_rgba(181, 45, 69, 220);

table[NK\_COLOR\_BORDER] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_BUTTON] = nk\_rgba(181, 45, 69, 255);

table[NK\_COLOR\_BUTTON\_HOVER] = nk\_rgba(190, 50, 70, 255);

table[NK\_COLOR\_BUTTON\_ACTIVE] = nk\_rgba(195, 55, 75, 255);

table[NK\_COLOR\_TOGGLE] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_TOGGLE\_HOVER] = nk\_rgba(45, 60, 60, 255);

table[NK\_COLOR\_TOGGLE\_CURSOR] = nk\_rgba(181, 45, 69, 255);

table[NK\_COLOR\_SELECT] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_SELECT\_ACTIVE] = nk\_rgba(181, 45, 69, 255);

table[NK\_COLOR\_SLIDER] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_SLIDER\_CURSOR] = nk\_rgba(181, 45, 69, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_HOVER] = nk\_rgba(186, 50, 74, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_ACTIVE] = nk\_rgba(191, 55, 79, 255);

table[NK\_COLOR\_PROPERTY] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_EDIT] = nk\_rgba(51, 55, 67, 225);

table[NK\_COLOR\_EDIT\_CURSOR] = nk\_rgba(190, 190, 190, 255);

table[NK\_COLOR\_COMBO] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_CHART] = nk\_rgba(51, 55, 67, 255);

table[NK\_COLOR\_CHART\_COLOR] = nk\_rgba(170, 40, 60, 255);

table[NK\_COLOR\_CHART\_COLOR\_HIGHLIGHT] = nk\_rgba( 255, 0, 0, 255);

table[NK\_COLOR\_SCROLLBAR] = nk\_rgba(30, 33, 40, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR] = nk\_rgba(64, 84, 95, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_HOVER] = nk\_rgba(70, 90, 100, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_ACTIVE] = nk\_rgba(75, 95, 105, 255);

table[NK\_COLOR\_TAB\_HEADER] = nk\_rgba(181, 45, 69, 220);

nk\_style\_from\_table(ctx, table);

} else if (theme == THEME\_BLUE) {

table[NK\_COLOR\_TEXT] = nk\_rgba(20, 20, 20, 255);

table[NK\_COLOR\_WINDOW] = nk\_rgba(202, 212, 214, 215);

table[NK\_COLOR\_HEADER] = nk\_rgba(137, 182, 224, 220);

table[NK\_COLOR\_BORDER] = nk\_rgba(140, 159, 173, 255);

table[NK\_COLOR\_BUTTON] = nk\_rgba(137, 182, 224, 255);

table[NK\_COLOR\_BUTTON\_HOVER] = nk\_rgba(142, 187, 229, 255);

table[NK\_COLOR\_BUTTON\_ACTIVE] = nk\_rgba(147, 192, 234, 255);

table[NK\_COLOR\_TOGGLE] = nk\_rgba(177, 210, 210, 255);

table[NK\_COLOR\_TOGGLE\_HOVER] = nk\_rgba(182, 215, 215, 255);

table[NK\_COLOR\_TOGGLE\_CURSOR] = nk\_rgba(137, 182, 224, 255);

table[NK\_COLOR\_SELECT] = nk\_rgba(177, 210, 210, 255);

table[NK\_COLOR\_SELECT\_ACTIVE] = nk\_rgba(137, 182, 224, 255);

table[NK\_COLOR\_SLIDER] = nk\_rgba(177, 210, 210, 255);

table[NK\_COLOR\_SLIDER\_CURSOR] = nk\_rgba(137, 182, 224, 245);

table[NK\_COLOR\_SLIDER\_CURSOR\_HOVER] = nk\_rgba(142, 188, 229, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_ACTIVE] = nk\_rgba(147, 193, 234, 255);

table[NK\_COLOR\_PROPERTY] = nk\_rgba(210, 210, 210, 255);

table[NK\_COLOR\_EDIT] = nk\_rgba(210, 210, 210, 225);

table[NK\_COLOR\_EDIT\_CURSOR] = nk\_rgba(20, 20, 20, 255);

table[NK\_COLOR\_COMBO] = nk\_rgba(210, 210, 210, 255);

table[NK\_COLOR\_CHART] = nk\_rgba(210, 210, 210, 255);

table[NK\_COLOR\_CHART\_COLOR] = nk\_rgba(137, 182, 224, 255);

table[NK\_COLOR\_CHART\_COLOR\_HIGHLIGHT] = nk\_rgba( 255, 0, 0, 255);

table[NK\_COLOR\_SCROLLBAR] = nk\_rgba(190, 200, 200, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR] = nk\_rgba(64, 84, 95, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_HOVER] = nk\_rgba(70, 90, 100, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_ACTIVE] = nk\_rgba(75, 95, 105, 255);

table[NK\_COLOR\_TAB\_HEADER] = nk\_rgba(156, 193, 220, 255);

nk\_style\_from\_table(ctx, table);

} else if (theme == THEME\_DARK) {

table[NK\_COLOR\_TEXT] = nk\_rgba(210, 210, 210, 255);

table[NK\_COLOR\_WINDOW] = nk\_rgba(57, 67, 71, 215);

table[NK\_COLOR\_HEADER] = nk\_rgba(51, 51, 56, 220);

table[NK\_COLOR\_BORDER] = nk\_rgba(46, 46, 46, 255);

table[NK\_COLOR\_BUTTON] = nk\_rgba(48, 83, 111, 255);

table[NK\_COLOR\_BUTTON\_HOVER] = nk\_rgba(58, 93, 121, 255);

table[NK\_COLOR\_BUTTON\_ACTIVE] = nk\_rgba(63, 98, 126, 255);

table[NK\_COLOR\_TOGGLE] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_TOGGLE\_HOVER] = nk\_rgba(45, 53, 56, 255);

table[NK\_COLOR\_TOGGLE\_CURSOR] = nk\_rgba(48, 83, 111, 255);

table[NK\_COLOR\_SELECT] = nk\_rgba(57, 67, 61, 255);

table[NK\_COLOR\_SELECT\_ACTIVE] = nk\_rgba(48, 83, 111, 255);

table[NK\_COLOR\_SLIDER] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_SLIDER\_CURSOR] = nk\_rgba(48, 83, 111, 245);

table[NK\_COLOR\_SLIDER\_CURSOR\_HOVER] = nk\_rgba(53, 88, 116, 255);

table[NK\_COLOR\_SLIDER\_CURSOR\_ACTIVE] = nk\_rgba(58, 93, 121, 255);

table[NK\_COLOR\_PROPERTY] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_EDIT] = nk\_rgba(50, 58, 61, 225);

table[NK\_COLOR\_EDIT\_CURSOR] = nk\_rgba(210, 210, 210, 255);

table[NK\_COLOR\_COMBO] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_CHART] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_CHART\_COLOR] = nk\_rgba(48, 83, 111, 255);

table[NK\_COLOR\_CHART\_COLOR\_HIGHLIGHT] = nk\_rgba(255, 0, 0, 255);

table[NK\_COLOR\_SCROLLBAR] = nk\_rgba(50, 58, 61, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR] = nk\_rgba(48, 83, 111, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_HOVER] = nk\_rgba(53, 88, 116, 255);

table[NK\_COLOR\_SCROLLBAR\_CURSOR\_ACTIVE] = nk\_rgba(58, 93, 121, 255);

table[NK\_COLOR\_TAB\_HEADER] = nk\_rgba(48, 83, 111, 255);

nk\_style\_from\_table(ctx, table);

} else {

nk\_style\_default(ctx);

}

}

#include <texture.h>

#include <glad/glad.h>

#include <stb/stb\_image.h>

#if defined(\_WIN32) || defined(WIN32)

#include <dirent/dirent.h>

#else

#include <dirent.h>

#endif

#include <stdbool.h>

#include <string.h>

const char texture\_path[128] = "../resources/textures/";

unsigned int carTexture, backgroundTexture, redLight, greenLight, yellowLight;

void initTextures() {

loadTexture("car.png", &carTexture);

loadTexture("back.png", &backgroundTexture);

// traffic light

loadTexture("red.png", &redLight);

loadTexture("green.png", &greenLight);

loadTexture("yellow.png", &yellowLight);

}

void loadTexture(char \*image\_name, unsigned int \*texture) {

stbi\_set\_flip\_vertically\_on\_load(true);

unsigned char \*texture\_pixels;

int texture\_width, texture\_height, texture\_channels;

char full\_path[128];

strcpy(full\_path, texture\_path);

strcat(full\_path, image\_name);

texture\_pixels = stbi\_load(full\_path, &texture\_width, &texture\_height,

&texture\_channels, 0);

GLenum format;

if (texture\_channels == 1)

format = GL\_RED;

else if (texture\_channels == 3)

format = GL\_RGB;

else if (texture\_channels == 4)

format = GL\_RGBA;

glGenTextures(1, texture);

glBindTexture(GL\_TEXTURE\_2D, \*texture);

glTexImage2D(GL\_TEXTURE\_2D, 0, format, texture\_width, texture\_height, 0,

format, GL\_UNSIGNED\_BYTE, texture\_pixels);

glGenerateMipmap(GL\_TEXTURE\_2D);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_S,

format == GL\_RGBA ? GL\_CLAMP\_TO\_EDGE : GL\_MIRRORED\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_WRAP\_T,

format == GL\_RGBA ? GL\_CLAMP\_TO\_EDGE : GL\_MIRRORED\_REPEAT);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MIN\_FILTER,

GL\_LINEAR\_MIPMAP\_LINEAR);

glTexParameteri(GL\_TEXTURE\_2D, GL\_TEXTURE\_MAG\_FILTER, GL\_LINEAR);

// Freeing allocated space for image

stbi\_image\_free(texture\_pixels);

}

#include <stdio.h>

#include <stdlib.h>

#include <string.h>

#include <traffic\_density.h>

bool densityDataInit = false;

GLvoid increaseDensityData(GLint roadIndex)

{

densityData[roadIndex] += 1;

}

GLvoid decreaseDensityData(GLint roadIndex)

{

densityData[roadIndex] -= 1;

}

GLint getDensityData(GLint roadIndex)

{

return densityData[roadIndex];

}

GLvoid printDensity(){

for (GLint i = 0; i < NUMBER\_OF\_ROADS; i++){

printf("r%d: %d\n", i, getDensityData(i));

}

}#include <traffic\_light.h>

#include <macros.h>

#include <cars.h>

#include <road.h>

#include <map.h>

#include <stdbool.h>

#include <stdio.h>

GLint timer\_digit = 0;

GLint traffic\_light\_adding\_index = -1;

bool yellowRedChange = false;

const int RED\_TIME = 6;

const int YELLOW\_TIME = 2;

const int GREEN\_TIME = 6;

GLvoid changeLightsColor(){

// return;

for (GLint i = 0; i <= traffic\_light\_adding\_index; i++){

if (--lights[i].changeTimer > 0){

continue;

}

if (lights[i].color == GREEN){

lights[i].color = YELLOW;

lights[i].changeTimer = YELLOW\_TIME;

}

else if (lights[i].color == YELLOW){

lights[i].color = RED;

lights[i].changeTimer = RED\_TIME;

yellowRedChange = true;

}

else if(lights[i].color == RED){

lights[i].color = GREEN;

lights[i].changeTimer = GREEN\_TIME;

}

}

// printLights();

}

COLOR getLightColor(GLint lightIndex){

return lights[lightIndex].color;

}

COLOR getLightColorByCar(car\* Car){

return lights[roads[Car->currCell.road].traffic\_light\_index].color;

}

GLint getFreeTrafficLightIndex(){

return ++traffic\_light\_adding\_index;

}

GLvoid setDefaultTrafficLightProperties(){

for (int roadIndex = 0; roadIndex < NUMBER\_OF\_ROADS; roadIndex++){

if (!roads[roadIndex].isEndCross){

continue;

}

if (roads[roadIndex].dir == NORTH || roads[roadIndex].dir == SOUTH){

lights[roads[roadIndex].traffic\_light\_index].color = GREEN;

lights[roads[roadIndex].traffic\_light\_index].changeTimer = GREEN\_TIME;

}

else{

lights[roads[roadIndex].traffic\_light\_index].color = RED;

lights[roads[roadIndex].traffic\_light\_index].changeTimer = RED\_TIME ;

}

// Render

glGenVertexArrays(1, &lights[roads[roadIndex].traffic\_light\_index].VAO);

glGenBuffers(1, &lights[roads[roadIndex].traffic\_light\_index].VBO);

glGenBuffers(1, &lights[roads[roadIndex].traffic\_light\_index].EBO);

glBindVertexArray(lights[roads[roadIndex].traffic\_light\_index].VAO);

glBindBuffer(GL\_ARRAY\_BUFFER, lights[roads[roadIndex].traffic\_light\_index].VBO);

float size = CELL\_LENGTH \* 1.5;

float height = size \* 2;

float width = size \* 0.7;

float x = roads[roadIndex].stem + HALF\_ROAD\_WIDTH;

float y = roads[roadIndex].endLineCoord;

DIRECTION dir = roads[roadIndex].dir;

if (dir == NORTH) {

y -= height;

float vertices[] = {

// positions // texture coords

x + width, y + height, 1.0f, 1.0f, 1.0f,

x + width, y, 1.0f, 1.0f, 0.0f,

x, y, 1.0f, 0.0f, 0.0f,

x, y + height, 1.0f, 0.0f, 1.0f

};

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

}

if (dir == SOUTH) {

x -= HALF\_ROAD\_WIDTH \* 2 + CELL\_LENGTH;

float vertices[] = {

// positions // texture coords

x + width, y, 1.0f, 1.0f, 1.0f,

x + width, y + height, 1.0f, 1.0f, 0.0f,

x, y + height, 1.0f, 0.0f, 0.0f,

x, y, 1.0f, 0.0f, 1.0f

};

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

}

if (dir == EAST) {

x -= HALF\_ROAD\_WIDTH \* 2 + CELL\_LENGTH;

y -= height;

float vertices[] = {

// positions // texture coords

y + height, x + width, 1.0f, 1.0f, 1.0f,

y + height, x, 1.0f, 0.0f, 1.0f,

y, x, 1.0f, 0.0f, 0.0f,

y, x + width, 1.0f, 1.0f, 0.0f

};

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

}

if (dir == WEST) {

float vertices[] = {

// positions // texture coords

y, x + width, 1.0f, 1.0f, 1.0f,

y, x, 1.0f, 0.0f, 1.0f,

y + height, x, 1.0f, 0.0f, 0.0f,

y + height, x + width, 1.0f, 1.0f, 0.0f

};

glBufferData(GL\_ARRAY\_BUFFER, sizeof(vertices), vertices, GL\_STATIC\_DRAW);

}

unsigned int indices[3 \* 2] = {

0, 1, 3,

1, 2, 3

};

glBindBuffer(GL\_ELEMENT\_ARRAY\_BUFFER, lights[roads[roadIndex].traffic\_light\_index].EBO);

glBufferData(GL\_ELEMENT\_ARRAY\_BUFFER, sizeof(indices), indices, GL\_STATIC\_DRAW);

glVertexAttribPointer(0, 3, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)0);

glEnableVertexAttribArray(0);

glVertexAttribPointer(1, 2, GL\_FLOAT, GL\_FALSE, 5 \* sizeof(float), (void\*)(3 \* sizeof(float)));

glEnableVertexAttribArray(1);

glBindVertexArray(0);

glBindBuffer(GL\_ARRAY\_BUFFER, 0);

}

}

GLvoid printLights(){

printf("\t\t\t\t");

for (int i = 0; i < 4; i+=4){

// printf("light ");

if (i == 0) printf("NORTH");

else if (i == 1) printf("SOUTH");

else if (i == 2) printf("EAST");

else if (i == 3) printf("WEST");

printf(":");

if (lights[i].color == RED){

printf("RED");

}

else if (lights[i].color == YELLOW){

printf("YELLOW");

}

else if (lights[i].color == GREEN){

printf("GREEN");

}

printf("\t");

}

printf("\n");

}

#version 330 core

out vec4 FragColor;

in vec2 texCoord;

uniform sampler2D ourTexture;

void main()

{

FragColor = texture(ourTexture, texCoord);

}

#version 330 core

out vec4 FragColor;

in vec2 texCoord;

in vec3 color;

uniform sampler2D ourTexture;

void main()

{

vec4 texColor = texture(ourTexture, texCoord);

if (texColor.a < 0.1)

discard;

FragColor = texColor \* vec4(color, 1.0f);

}

#version 330 core

out vec4 FragColor;

in vec3 color;

void main()

{

FragColor = vec4(color, 1.0);

}

#version 330 core

layout (location = 0) in vec3 aPos;

layout (location = 1) in vec2 aTexCoord;

out vec2 texCoord;

uniform mat4 projection;

uniform mat4 view;

void main()

{

gl\_Position = projection \* view \* vec4(aPos, 1.0);

texCoord = aTexCoord;

}

#version 330 core

layout (location = 0) in vec2 aPos;

layout (location = 1) in vec2 aTexCoord;

layout (location = 2) in mat3 aTransform;

out vec2 texCoord;

out vec3 color;

uniform mat4 projection;

uniform mat4 view;

float random(float seed)

{

return fract(sin(seed) \* 238.5453);

}

void main()

{

gl\_Position = projection \* view \* vec4(aTransform \* vec3(aPos, 1.0), 1.0);

texCoord = aTexCoord;

color = vec3(random(gl\_InstanceID), random(cos(gl\_InstanceID)), random(gl\_InstanceID \* sin(gl\_InstanceID)));

}

#version 330 core

layout (location = 0) in vec2 aPos;

layout (location = 1) in vec3 aColor;

out vec3 color;

uniform mat4 projection;

uniform mat4 view;

void main()

{

color = aColor;

gl\_Position = projection \* view \* vec4(aPos, 1.0, 1.0);

}