МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«СЕВАСТОПОЛЬСКИЦ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

    Институт информационных технологий и управления в технических системах

*(название института полностью)*

Кафедра/департамент                  «Информационные системы»

*(наименование кафедры/департамента полностью)*

                            09.03.02 Информационные системы и технологии

*(код и наименование направления подготовки/специальности)*

                                   Информационные системы и технологии

*(наименование профиля/специальности)*

**КУРСОВАЯ РАБОТА / КУРСОВОЙ ПРОЕКТ**

по дисциплине

**Объектно-ориентированное программирование**

*(наименование дисциплины)*

на тему               **Маршрутное такси**

Выполнили: обучающиеся

группы:

*(инициалы, фамилия)*

«       »                                 20 22 г.

Научный руководитель:

*(инициалы, фамилия)*

«       »                                 20 22 г.

Оценка

«       »                                 20 22 г.

Севастополь

20   22

**АННОТАЦИЯ**

Назначением данной пояснительной записки является описание процесса разработки программы – имитационной модели работы сети маршрутных такси. В документе описывается процесс абстрагирования, выделения классов, их проектирование; выделения состояния объектов, потоков данных между объектами; определение жизненного цикла как всей программы. После проектирования описывается процесс создания программы на одном из языков программирования и тестирования готового продукта.

**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ 4](#_Toc115155091)

[ВВЕДЕНИЕ 5](#_Toc115155092)

[СПИСОК ИСПОЛНИТЕЛЕЙ 7](#_Toc115155093)

[1 ПОСТАНОВКА ЗАДАЧИ 8](#_Toc115155094)

[1.1 Цель разработки 8](#_Toc115155095)

[1.2 Описательная постановка задачи 8](#_Toc115155096)

[1.3 Условия выполнения и ограничения 9](#_Toc115155097)

[2 ПРОЕКТНОЕ РЕШЕНИЕ 10](#_Toc115155098)

[2.1 Абстрагирование и выделение объектов 10](#_Toc115155099)

[2.2 Построение диаграммы классов предметной области 11](#_Toc115155100)

[2.3 Построение информационной модели 12](#_Toc115155101)

[2.4 Описание жизненного цикла программы 12](#_Toc115155102)

[2.5 Описание жизненного цикла объектов 12](#_Toc115155103)

[2.6 Диаграмма переходов состояний 12](#_Toc115155104)

[2.7 Диаграмма потоков данных и действий 15](#_Toc115155105)

[3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ 16](#_Toc115155106)

[3.1 Обоснование выбора системы программирования 16](#_Toc115155107)

[3.2 Описание реализации основных классов и их методов 17](#_Toc115155108)

[3.3 Интерфейс пользователя 22](#_Toc115155109)

[3.4 Критерии качества программной системы 23](#_Toc115155110)

[ВЫВОДЫ 24](#_Toc115155111)

[ПЕРЕЧЕНЬ ССЫЛОК 25](#_Toc115155112)

[ПРИЛОЖЕНИЕ А 26](#_Toc115155113)

ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И УСЛОВНЫХ ОБОЗНАЧЕНИЙ

МТ – маршрутное такси

ООП – объектно-ориентированное программирование

ПДД – правила дорожного движения

СМО – система массового обслуживания

# **ВВЕДЕНИЕ**

В рамках настоящей курсовой работы ведется разработка программы на основании технического задания, выданного организацией Севастопольский государственный университет. Дата выдачи задания: 19.02.2022.

Целями курсовой работы являются изучение программирования на основе объектно-ориентированной технологии, приобретение навыков написания программ на языке с поддержкой объектно-ориентированного программирования (ООП) на примере написания программы для моделирования работы сети маршрутных такси (согласно варианту задания).

Для достижения каждой из поставленных целей были поставлены следующие задачи:

– выбор языка реализации, детализация постановки задачи;

– абстрагирование, разработка классов и их иерархии;

– написание текста программы на выбранном языке;

– разработка тестовых примеров;

– тестирование и отладка программы;

– разработка программных документов в соответствии с действующими стандартами.

Актуальностью написания программы является то, что при помощи данной модели возможно оценить среднее время нахождения пассажира маршрутного такси (МТ) в пути в зависимости от интенсивности пассажиропотока и количества МТ, находящихся на маршруте. Построение такого рода зависимости позволяет получить оптимальные значения среднего времени пассажира в пути и количества МТ на маршруте, устраивающие и пассажиров, и владельцев автопарка. Так как уменьшение количества МТ на маршруте позволяет владельцу автопарка экономить на топливе, обслуживании и заработной плате водителя, ему выгодно иметь на маршруте меньше МТ, но низкое их количество не будет удовлетворять требованиям пассажиров, которые не желают долго ждать МТ на остановке и хотят доехать до пункта назначения быстрее. При помощи разработанной программы имеется возможность найти эти оптимальные значения, устраивающие как пассажиров, так и владельцев МТ. Таким образом, решаемая задача актуальная имеет прямое практическое предназначение.

По окончанию выполнения курсовой работы должны быть получены навыки решения задач объектно-ориентированным подходом.

# **СПИСОК ИСПОЛНИТЕЛЕЙ**

Текущая работа была исполнена студентами:

1. Иванов Иван Иванович, группа ИС/б-20-2-о. Вклад: разработка идеи, документирование, разработка архитектуры приложения, написание кода, отладка, тестирование.

# **1 ПОСТАНОВКА ЗАДАЧИ**

## **1.1 Цель разработки**

Целью разработки является создание программы имитационной модели работы сети МТ.

## **1.2 Описательная постановка задачи**

В программе моделируется работа сети МТ на базе теории систем массового обслуживания (СМО).

По маршруту движется не более N МТ. Каждое МТ имеет 18 посадочных мест. Водители МТ не нарушают правила дорожного движения (ПДД) и не берут дополнительных пассажиров. На маршруте имеется L остановок, на каждой из которых случайным образом генерируется некоторое количество пассажиров. Количество пассажиров, скопившихся на остановке, пропорционально времени ожидания.

Для каждого пассажира случайным образом генерируется число остановок P, которое он хочет проехать P <= 2\*L/3. Время посадки/высадки 1 пассажира занимает 7 секунд. Время стоянки на остановке, если есть свободные места, 10 секунд, в противном случае МТ не останавливается.

МТ движется по маршруту со средней скоростью 40 км/ч (скорость с учетом светофоров, пробок и т.п.). Расстояние между всеми остановками заранее известно.

Необходимо промоделировать работу системы в течение задаваемого количества времени, определить следующие характеристики:

– процент занятости всех МТ на маршруте;

– среднее число пропускаемых МТ остановок за 1 прохождение маршрута;

– среднюю загруженность МТ;

– среднее время проезда пассажира.

## **1.3 Условия выполнения и ограничения**

Программа должна быть реализована на языке, полностью поддерживающем объектно-ориентированный подход. Программа должна поддерживать ввод исходных данных с клавиатуры, а также загрузку из файла. Должна быть возможность сохранения результатов работы программы в файл.

# **2 ПРОЕКТНОЕ РЕШЕНИЕ**

## **2.1 Абстрагирование и выделение объектов**

При анализе программы были выделены следующие объекты:

– маршрут;

– пассажир;

– автобусная остановка;

– автобус.

Автобус движется по маршруту от начала и до конца, может забирать и пассажиров с одной остановки и высаживать на другой. Автобусная остановка «пораждает» пассажиров через заданные промежутки времени.

При разработке были выделены следующие классы:

Entity (сущность). Базовый класс для любого объекта внутри системы.

Builder (строитель). Базовый класс для конструирования объекта при помощи паттерна «билдер». Такой подход был использован, поскольку в системе пораждается много однотипных объектов.

Route (маршрут). Базовый класс в модели. Маршруту принадлежат автобусы и остановки. Отвечает за жизненный цикл автобусов и остановок – пораждает (spawn) и уничтожает (despawn). Имеет протяженность. Для упрощения модели, внутренний «мир» упрощен до одномерного, то есть все остановки находятся на одной оси. Начало маршрута находится в точке 0.

Passenger (пассажир). Является проекцией реального пассажира в модели. Появляется на остановке, может сесть в автобус и переехать на другую остановку. После прибытия в пункт назначения записывает в статистику время в пути и уничтожается.

PassengerBuilder (конструктор пассажира). Класс для создания экземпляра пассажира с заданными параметрами.

Busstop (автобусная остановка). Является проекцией реальной остановки в модели. Имеет свои координаты на маршруте (расстояние от начала маршрута). Отвечает за создание и уничтожение пассажиров.

BusstopBuilder (конструктор остановки). Класс для создания экземпляра остановки с заданными параметрами.

Bus (автобус). Является проекцией реального автобуса в модели. Имеет посадочные места, на которые могут быть размещены пассажиры. Движется от начала маршрута до конца по описанному в задаче алгоритму. При достижении конца маршрута, записывает свои показатели в статистику и уничтожается. Пока пассажир находится в автобусе, отвечает за его жизненный цикл (уничтожает при закрытии программы).

BusBuilder (конструктор автобуса). Класс для создания экземпляра автобуса с заданными параметрами.

ExperimentInputData (исходные данные эксперимента). Класс-контейнер для хранения исходных данных.

ExperimentOutputData (выходные данные эксперимента). Класс-контейнер для хранения выходных данных.

Statcollector (сборщик статистики). Предназначен для записи показателей объектов системы и для получения средних значений функционирования системы по полученной выборке записей.

## **2.2 Построение диаграммы классов предметной области**

На основании предыдущего раздела построено графическое отображение иерархии классов, представленное на рисунке 1.



Рисунок 1 – Диаграмма классов предметной области

## **2.3 Построение информационной модели**

Диаграмма классов представлена на рисунке 2.

## **2.4 Описание жизненного цикла программы**

Диаграмма жизненного цикла программы представлена на рисунке 3.

## **2.5 Описание жизненного цикла объектов**

Диаграмма жизненного цикла объектов представлена на рисунке 4.

## **2.6 Диаграмма переходов состояний**

Диаграмма переходов состояний представлена на рисунке 5 для класса пассажира и на рисунке 6 для класса автобус.



Рисунок 2 – Информационная модель



Рисунок 3 – Жизненный цикл программы



Рисунок 4 – Жизненный цикл объектов



Рисунок 5 – Диаграмма переходов состояний объекта класса Passenger



Рисунок 6 – Диаграмма переходов состояний объекта класса Bus

## **2.7 Диаграмма потоков данных и действий**

Диаграмма потоков данных и действий представлена на рисунке 7.



Рисунок 7 – Диаграмма потоков данных и действий

# **3 ПРОГРАММНАЯ РЕАЛИЗАЦИЯ**

## **3.1 Обоснование выбора системы программирования**

Было принято решение реализовать проект на языке программирования С++. Преимущества языка программирования:

– работа на низком уровне, что позволяет написать оптимизированную программу;

– типовые математические вычислительные задачи, такие как численное решение дифференциальных уравнений, как правило решаются при помощи этого языка благодаря его производительности;

– поддержка множества стилей программирования: процедурное, объектно-ориентированное и обобщенное программирования);

– большое сообщество.

Недостатки:

– сложный язык (из-за его больших возможностей и низкоуровневого доступа достаточно легко ошибиться);

– плохая поддержка модульности;

– неинтуитивные преобразования типов.

В качестве среды программирования была выбрана Visual Studio 2022 Community. Причина выбора:

– удобство написания и отладки кода;

– приятный пользовательский интерфейс;

– выделение ошибок и предупреждений при написании кода;

– автоматическое генерирование «бойлерплейт» кода;

– бесплатная лицензия.

## **3.2 Описание реализации основных классов и их методов**

В ходе разработки были созданы классы, аналогичные описанным в разделе 2.1: Entity, Builder, Route, Passenger, PassengerBuilder, Busstop, BusstopBuilder, Bus, BusBuilder, ExperimentInputData, ExperimentOutputData, Statcollector.

Базовым классом любого объекта внутри системы (Bus, Passenger, Busstop, Route) является класс Entity. Он имеет два виртуальных метода в качестве интерфейса: метод tick, вызываемый через заданные отрезки времени на протяжении всего процесса моделирования, и деструктор для освобождения ресурсов после использования класса. В каждом дочернем классе эти методы переопределены. Деструктор отвечает только за освобождение ресурсов. Метод tick определяет поведения объекта внутри системы с течением времени. Течение времени обозначается периодическим вызовом этого метода. В качестве параметра он принимает промежуток времени, который прошел с момента последнего вызова этого метода. Таким образом, засекая внутри объекта времени, можно определить его состояние, например, для пассажира – завершилась ли посадка в автобус.

Passenger дополнительно содержит следующие методы:

– getId – для получения уникального идентификатора пассажира;

– setStartBusstop – для определения отправной точки (как правило задается остановкой, на который он появился и является ссылкой на нее);

– getStartBusstop – для получения ссылки на отправную остановку;

– setTargetBusstop – для определения пункта назначения (конечной точки), также задается остановкой, на который появился пассажир, равен ссылке на следующую случайную остановку, выбранную исходя из условий задачи;

– getTargetBusstop – для получения ссылки на остановку назначения пассажира;

– startBoarding – для запуска процесса посадки/высадки пассажира. При первом вызове функции считается, что началась посадка, при втором вызове – высадка;

– isBoardingEnded – для проверки, закончилась ли посадка/высадка;

– isOnTrip – для проверки статуса пассажира – едет или еще ждет автобус;

– getTripTime – для определения прошедшего времени с момента посадки в автобус;

– getTotalTime – для получения прошедшего времени с момента появления на остановке (создания пассажира).

Bus дополнительно содержит следующие методы:

– getId – для получения уникального идентификатора автобуса;

– isNeedToDelete – для получения статуса автобуса, когда автобус доезжает до конца маршрута, он помечается «к удалению», данный метод позволяет определить этот статус у автобуса;

– getPosition – для получения текущих координат автобуса;

– setParentRoute – для задания маршрута, на котором едет автобус;

– getPassedBusstops – получение списка остановок, мимо которых автобус проехал, либо останавливался;

– getVisitedBusstops – получение списка остановок, на которых автобус останавливался;

– getTransitState – получение статуса движения автобуса;

– getPassengers – получение списка пассажира со свободными местами;

– getPassengerCount – получение количества пассажиров, находящихся в автобусе в данный момент времени;

– getFreeSeatCount – получение количества свободных мест в автобусе;

– needToGetOff – для определения, надо ли кому-то из салона выйти на текущей остановке;

– sitOnTheBus – для определения, сидит ли указанный пассажир в автобусе.

Busstop дополнительно содержит следующие методы:

– getId – для получения уникального идентификатора остановки;

– getPosotion – для получения координат остановки на маршруте;

– setParentRoute – для задания маршрута, которому принадлежит остановка;

– getParentRoute – для определения маршрута, которому принадлежит остановка;

– setPassengerBuilder – для задания билдера для пассажиров на остановке;

– getPassengerBuilder – для определения билдера, используемого для создания пассажиров на данной остановке;

– haveWaiters – для определения, есть ли на остановке ожидающие;

– getWaitingPassengers – получение списка ожидающих пассажиров;

– addArrival – для добавления пассажира на остановку в статусе прибывшего;

– spawnPassenger – метод создания пассажира на остановке;

– despawnPassenger – метод для удаления пассажира.

Route дополнительно содержит следующие методы:

– setLength – для задания общей протяженности маршрута;

– getLength – для получения общей протяженности маршрута;

– setBusCount – для задания максимального количества автобусов, одновременно находящихся на маршруте;

– getBusCount – для получения максимального количества автобусов, одновременно находящихся на маршруте;

– setBusBuilder – для задания билдера автобусов;

– getBusBuilder – для получения билдера автобусов;

– getBusArray – для получения списка автобусов на маршруте;

– getBusstopArray – для получения списка остановок на маршруте;

– setStatcollector – для задания класса, собирающего статистику;

– getStatcollector – для получения класса, собирающего статистику;

– getBusstopIndex – для получения порядкового номера остановки по указателю на нее;

– getBusstopCount – для получения количества остановок на маршруте;

– getBusstop – для получения указателя на остановку по ее порядковому номеру;

– despawnBus – уничтожение автобуса;

– spawnBus – создание автобуса.

В процессе моделирования одни экземпляры классов создаются, а другие уничтожаются и так до тех пор, пока не закончится заданное внутреннее время моделирования. Например, остановка (Busstop) постоянно с заданной периодичностью порождает пассажиров (Passenger). Для того, чтобы каждый раз не передавать в конструктор класса множество параметров о его состоянии, было принято решение использовать паттерн «билдер» для создания экземпляров классов. Классы билдеры имеют общий родительский класс Builder с единственным виртуальным методом build, позволяющий получить экземпляр указанного класса. В дочерних классах этот метод переопределен, чтобы создавались экземпляры конкретных типов. Помимо этого, каждый дочерний класс содержит методы для передачи параметров в билдер.

PassengerBuilder содержит дополнительно метод:

– setBoardingTimeLimit – для задания времени посадки/высадки пассажира.

BusBuilder содержит следующие дополнительные методы:

– setPosition – для определения координат автобуса, из которых он начинает движение. По умолчанию – 0;

– setVelocity – для задания средней скорости движения по маршруту;

– setTransit – для задания статуса движения (движется/стоит, по умолчанию – движется);

– setStopTimeLimit – для задания минимального времени нахождения на остановке, если остановился на ней;

– setPassengersLimit – для задания количества мест в автобусе.

BusstopBuilder содержит следующие дополнительные методы:

– setPosition – для задания координат остановки на маршруте;

– setPassengerBuilder – для задания класса-билдера для создания экземпляров пассажиров;

– setParentRoute – для задания ссылки на маршрут, которому будет принадлежать остановка;

– setSpawnEnable – для задания разрешения на появления на остановке пассажиров (например, на конечной остановке пассажира не будут садиться, только выходить, значит там нет необходимости создавать пассажиров);

– setLowSpawnTime – для задания минимального интервала времени, через который появится очередной пассажир на остановке;

– setHighSpawnTime – для задания максимального интервала времени, через который появится очередной пассажир на остановке.

Запуск моделирования осуществляется в процедурном стиле путем вызова функции doExperiment, на вход принимающую класс-контейнер с исходными данными эксперимента ExperimentInputData и возвращающего класс-контейнер с результатами моделирования ExperimentOutputData. Указанные классы-контейнеры не содержат методов и служат исключительно для удобного обмена данными между моделирующей частью программы и интерфейсом.

При проектировании архитектуры приложения было уделено внимание минимальной связности классов между собой, что благоприятно сказывается на поддержке кода, его расширении и модификации. В связи с чем объекты системы могут функционировать отдельно друг от друга. Поэтому был добавлен класс Statcollector, ссылка на который присутствует практически у каждого объекта системы. Класс предназначен для сбора статистики о параметрах функционирования объектов системы. Каждый объект системы самостоятельно обращается к нему и передает информацию о себе, которая в дальнейшем обрабатывается и переносится в результаты моделирования.

В него входят следующие методы для записи статистики:

– addMissedStops для передачи количества пропущенных остановок автобусом;

– addTripTime для добавления времени пассажира в пути;

– addTotalTime для добавления общего времени пути пассажира включительно время ожидания на остановке;

– addBusLoad для добавления процента загрузки автобуса в текущий момент времени.

А также следующие методы для обработки записанных данных:

– getAvgMessedStopsCount – для получения среднего количества пропускаемых автобусами остановок;

– getAvgTripTime – для получения среднего времени в пути для пассажиров;

– getAvgTotalTime – для получения среднего времени в пути для пассажиров включая время ожидания на остановке;

– getAvgBusLoad – для получения процента загруженности указанного автобуса;

– getAvgAllBusLoad – для получения процента загруженности всех автобусов;

– getBusStat – для получения статистики по автобусам (процент загруженности).

## **3.3 Интерфейс пользователя**

На рисунке 8 показан внешний вид программы. Управление программой осуществляется через консоль путем выбора номера команды. Всего в программе доступно 5 команд:

– ввести начальные данные из консоли;

– загрузить данные из файла;

– показать результаты моделирования в консоли;

– сохранить результаты моделирования в файл;

– выйти из программы.

****Рисунок 8 – Внешний вид окна программы

## **3.4 Критерии качества программной системы**

К основным критериям качества программной системы были отнесены:

– функциональность;

– отказоустойчивость;

– быстродействие;

– модифицируемость;

– расширяемость;

– гибкость.

# **ВЫВОДЫ**

В ходе выполнения данного курсового проекта было проведено абстрагирование, разработаны классы и их иерархии. Также был написан код программы на языке программирования C++, результатом чего является понимание особенностей реализации объектно-ориентированного подхода при разработке программ при помощи данного языка программирования. Вся разработка велась в среде Visual Studio 2022 Community, что способствует получению практических навыком работы с данной программой и ее инструментами тестирования и отладки. При разработке были реализованы все основные концепции ООП (инкапсуляция, наследование и полиморфизм), а также использованы некоторые паттерны проектирования (стратегия, билдер).

Заключающим этапом стала разработка пояснительной записки, в которой описаны все пункты и шаги при разработке, также описана структура программы и ее классы. Полученные навыки могут быть применены для дальнейшей реализации задач, которые подразумевают применение ООП в своей разработке.

# **ПЕРЕЧЕНЬ ССЫЛОК**

1. Бьёрн Страуструп, Язык программирования C++. Краткий курс. – М.: Вильямс, 2019 – 320 с. – ISBN: 978-5-907144-12-5.

2. Гамма Э., Хелм Р., Джонсон Р., Влиссидес Дж., Приемы объектно-ориентированного проектирования. Паттерны проектирования. – СПб: Питер, 2001 – 368 с. – ISBN: 5-272-00355-1.

3. Добавить книгу преподавателя по предмету.

# **ПРИЛОЖЕНИЕ А**

Исходный код, разбитый на файлы

entity.h

#pragma once

// базовый класс для всех сущностей в модели

class Entity {

public:

// виртуальный деструктор

virtual ~Entity();

// метод вызываемый через отрезок delta секунд (для изменения состояния модели с дискретным шагом по времени)

virtual void tick(double delta);

};

builder.h

#pragma once

#include "entity.h"

class Builder {

public:

virtual Entity\* build();

};

passenger.h

#pragma once

#include "entity.h"

#include "builder.h"

class Passenger : public Entity {

protected:

int id;

// стартовая остановка (с которой пассажир сажидся в автобус)

Entity\* startBusstop = nullptr;

// целевая остановка (куда пассажиру надо дораться)

Entity\* targetBusstop = nullptr;

// счетчик времени в пути

double tripTimer = 0.0;

// счетчик всего времени (ожидание + поездка)

double totalTime = 0.0;

// счетчик времени посадки/высадки

double boardingTimer = 0.0;

// флаг того, что пассажир находится в пути

bool onTrip = false;

// флаг того, что пассажир садится/выходит

bool isBoarding = false;

// время посадки/высадки пассажира

double boardingTimeLimit;

public:

virtual void tick(double delta);

int getId();

void setStartBusstop(Entity\* busstop);

Entity\* getStartBusstop();

void setTargetBusstop(Entity\* busstop);

Entity\* getTargetBusstop();

// метод используется для посадки и высадки. первое использование - посадка

// далее считается, что пассажир находится в пути

// второе использование - высадка, считается, что пассажир приехал в пункт назначения

void startBoarding();

bool isBoardingEnded();

bool isOnTrip();

double getTripTime();

double getTotalTime();

friend class PassengerBuilder;

};

class PassengerBuilder : public Builder {

protected:

int idCounter = 0;

double boardingTimeLimit;

public:

virtual Entity\* build();

PassengerBuilder\* setBoardingTimeLimit(double boardingTime);

};

busstop.h

#pragma once

#include "entity.h"

#include "builder.h"

#include <vector>

using namespace std;

class Busstop : public Entity {

protected:

int id;

// координата остановки на маршруте

double position;

// сслыка на маршрут, на котором находится остановка

Entity\* parentRoute;

// массив ожидающих пассажиров

vector<Entity\*> waitingPassengers;

// массив прибывших пассажиров

vector<Entity\*> arrivalPassengers;

// ссылка на билдер пассажиров

Builder\* passengerBuilder;

// разрешает появление пассажиров на остановке

bool spawnEnable = false;

// пассажир появляется через случайный промежуток времени между двумя данными временами

// минимальное время появления пассажира

double lowSpawnTime = 0.0;

// макисмальное время появления пассажира

double highSpawnTime = 0.0;

// таймер появления пассажира

double spawnTimer;

// через это время появится пасажир

double currentSpawnTime;

// passenger spawn speed

public:

virtual void tick(double delta);

int getId();

double getPosition();

void setParentRoute(Entity\* route);

Entity\* getParentRoute();

void setPassengerBuilder(Builder\* builder);

Builder\* getPassengerBuilder();

// есть ли на остановке пассажиры, ожидающие автобус

bool haveWaiters();

vector<Entity\*>\* getWaitingPassengers();

// добавление прибывшего пассажира на остановку

void addArrival(Entity\* passenger);

// создание пассажира на остановке с заданием остановки, куда он хочет уехать

void spawnPassenger(Entity\* targetBusstop);

void despawnPassenger(Entity\* passenger);

friend class BusstopBuilder;

};

class BusstopBuilder : public Builder {

protected:

double position = 0.0;

int idCounter = 0;

Builder\* passengerBuilder;

Entity\* parentRoute;

bool spawnEnable;

double lowSpawnTime;

double highSpawnTime;

public:

virtual Entity\* build();

BusstopBuilder\* setPosition(double pos);

BusstopBuilder\* setPassengerBuilder(Builder\* builder);

BusstopBuilder\* setParentRoute(Entity\* route);

BusstopBuilder\* setSpawnEnable(bool flag);

BusstopBuilder\* setLowSpawnTime(double time);

BusstopBuilder\* setHighSpawnTime(double time);

};

bus.h

#pragma once

#include "entity.h"

#include "builder.h"

#include <vector>

using namespace std;

// класс маршрутного такси (в программе - автобус)

class Bus : public Entity {

protected:

int id;

// координата автобуса от начала маршрута (км)

double position;

// скорость автобуса (км/ч)

double velocity;

// флаг, показывающий, движется автобус или стоит

bool inTransit;

// сслыка на маршрут, по которому движется автобус

Entity\* parentRoute;

// флаг, показывающий, что автобус подлежит удалению

bool needToDelete;

// массив остановок, мимо которых проехал автобус (в том числе посещенные им)

vector<Entity\*> passedBusstops;

// предел ожидания на остановке в секундах

double stopTimeLimit;

// счетчик времени остановки

double stopTimer;

// массив остановок, на которых автобус останавливался

vector<Entity\*> visitedBusstops;

// массив пассажиров

vector<Entity\*> passengers;

// получение номера первого свободного места

int getFreeSitIndex();

// average load

public:

virtual ~Bus();

virtual void tick(double delta);

int getId();

bool isNeedToDelete();

double getPosition();

void setParentRoute(Entity\* route);

vector<Entity\*>\* getPassedBusstops();

vector<Entity\*>\* getVisitedBusstops();

bool getTransitState();

vector<Entity\*>\* getPassengers();

// функция для подсчета количества пассажиров в автобусе

int getPassengerCount();

// количество свободных мест в автобусе

int getFreeSeatCount();

// надо ли кому-то из автобуса выйти на указанной остановке

bool needToGetOff(Entity\* busstop);

// сидит ли указанный пассажир в автобусе

bool sitOnTheBus(Entity\* passenger);

friend class BusBuilder;

};

// паттерн билдер для создания автобуса

class BusBuilder : public Builder {

protected:

double position = 0.0;

double velocity = 0.0;

bool inTransit = false;

bool needToDelete = false;

int idCounter = 0;

double stopTimeLimit = 0.0;

// задает количество мест в автобусе. не хранится в классе автобуса,

// нужна для задания размера массива пассажиров в автобусе

int passengersLimit = 0;

public:

BusBuilder\* setPosition(double pos);

BusBuilder\* setVelocity(double vel);

BusBuilder\* setTransit(bool t);

BusBuilder\* setStopTimeLimit(double stl);

BusBuilder\* setPassengersLimit(int pl);

virtual Entity\* build();

};

route.h

#pragma once

#include "entity.h"

#include "builder.h"

#include "statcollector.h"

#include <vector>

using namespace std;

class Route : public Entity {

protected:

// длина маршрута (км)

double length;

// массив автобусов, которые находятся на маршруте

vector<Entity\*> busArray;

// количество автобусов на маршруте (N)

int busCount;

// ссылка на билдер автобусов

Builder\* busBuilder;

// массив автобусных остановок

vector<Entity\*> busstopArray;

// сборщик статистики

Statcollector\* statcollector;

public:

virtual ~Route();

virtual void tick(double delta);

void setLength(double l);

double getLength();

void setBusCount(int c);

int getBusCount();

void setBusBuilder(Builder\* builder);

Builder\* getBusBuilder();

vector<Entity\*>\* getBusArray();

vector<Entity\*>\* getBusstopArray();

void setStatcollector(Statcollector\* sc);

Statcollector\* getStatcollector();

// порядковый номер остановки

int getBusstopIndex(Entity\* busstop);

// количество остановок на маршруте

int getBusstopCount();

// остановка по номеру

Entity\* getBusstop(int index);

void despawnBus(Entity\* ptr);

void spawnBus();

};

experiment.h

#pragma once

#include <vector>

#include <map>

using namespace std;

class ExperimentInputData {

public:

// скорость движения автобуса (км/ч)

double busVelocity;

// время ожидания автобуса на остановке (с)

double busstopWaitingTime;

// количество мест в автобусе

int seatLimit;

// количество автобусов на маршруте

int busCount;

// время посадки/высадки пассажира (с)

double boardingTime;

// начальное число пассажиров на остановке генерируется между:

int initPassengersLowCount;

int initPassengersHighCount;

// очередное пассажир на остановке появится в случайное время (с) между:

double passengerLowSpawnTime;

double passengerHighSpawnTime;

// массив координат остановок на маршруте (км)

vector<double> busstops;

// шаг моделирования

double dt;

// время моделирования

double modelingTime;

};

class ExperimentOutputData {

public:

// среднее коичество пропускаемых остановок

double missedBusstops;

// среднее время пассажира в пути (c)

double tripTime;

// среднее время пассажира в пути с учетом времени ожидания (c)

double totalTime;

// средняя загруженность по автобусам (%)

map<int, double> busLoad;

// средня загруженность всех автобусов (%)

double allBusLoad;

// время выполнения эксперимента (c)

double time;

};

// функция для проведения моделирования

ExperimentOutputData doExperiment(ExperimentInputData inputData);

random.h

#pragma once

#include <stdlib.h>

// сброс генератора случайных чисел

void randomize();

// генерирование случайного числа от нуля до lim

template<class T>

inline T rnd(T lim)

{

return (T)(((double)rand() / (double)RAND\_MAX) \* (double)lim);

}

// генерирование случайного числа от lowLim до upLim

template<class T>

inline T rnd(T lowLim, T upLim)

{

return lowLim + rnd(upLim - lowLim);

}

inputreader.h

#pragma once

#include "experiment.h"

ExperimentInputData read(const char\* filename);

outputwriter.h

#pragma once

#include "experiment.h"

// запись результатов моделирования в файл

void write(const char\* filename, const ExperimentInputData& inputData, const ExperimentOutputData& outputData);

statcollector.h

#pragma once

#include <vector>

#include <map>

using namespace std;

class Statcollector {

protected:

// массив со значениями количества пропущенных остановок

vector<int> missedStopsCount;

// массив со значениями времени в пути

vector<double> tripTime;

// массив со значениями общего времени проезда для пассажира (ожидание + дорога)

vector<double> totalTime;

// массив значений загруженности по id автобуса

map<int, vector<double>> busLoad;

// общая функция для вычисления среднего арифмпетического по значениям массива

template<class T>

double getAvg(vector<T>\* v) {

if (v->size() == 0) {

return 0.0;

}

double sum = 0.0;

for (T val: \*v) {

sum += (double)val;

}

return sum / (double)v->size();

}

public:

void addMissedStops(int ms);

void addTripTime(double t);

void addTotalTime(double t);

void addBusLoad(int id, double load);

// полчениче числа среднего количества пропускаемых остановок

double getAvgMessedStopsCount();

// получение среднего времени в пути

double getAvgTripTime();

// получение среднего общего времени

double getAvgTotalTime();

// средний процент загруженности автобуса с указанным id

double getAvgBusLoad(int id);

// процент загруженности всех автобусов

double getAvgAllBusLoad();

// получение среднего процента загруженности с id автобусов

map<int, double> getBusStat();

};

entity.cpp

#include "entity.h"

Entity::~Entity()

{

}

void Entity::tick(double delta)

{

}

builder.cpp

#include "builder.h"

Entity\* Builder::build()

{

return nullptr;

}

passenger.cpp

#include "passenger.h"

void Passenger::tick(double delta)

{

// увеличиваем счетчик всего времени

totalTime += delta;

// если в пути

if (onTrip) {

// увеличиваем время в пути на шаг моделирования

tripTimer += delta;

}

// если осуществляется посадка/высадка

if (isBoarding) {

// увеличиваем таймер

boardingTimer += delta;

// если время посадки/высадки прошло

if (boardingTimer > boardingTimeLimit) {

// завершаем посадку/высадку

isBoarding = false;

// меняем статус "в пути" на обратный (ждет - в пути - сошел)

onTrip = !onTrip;

}

}

}

int Passenger::getId()

{

return id;

}

void Passenger::setStartBusstop(Entity\* busstop)

{

startBusstop = busstop;

}

Entity\* Passenger::getStartBusstop()

{

return startBusstop;

}

void Passenger::setTargetBusstop(Entity\* busstop)

{

targetBusstop = busstop;

}

Entity\* Passenger::getTargetBusstop()

{

return targetBusstop;

}

void Passenger::startBoarding()

{

isBoarding = true;

boardingTimer = 0.0;

}

bool Passenger::isBoardingEnded()

{

return !isBoarding;

}

bool Passenger::isOnTrip()

{

return onTrip;

}

double Passenger::getTripTime()

{

return tripTimer;

}

double Passenger::getTotalTime()

{

return totalTime;

}

Entity\* PassengerBuilder::build()

{

Passenger\* passenger = new Passenger();

passenger->id = idCounter++;

passenger->startBusstop = nullptr;

passenger->targetBusstop = nullptr;

passenger->tripTimer = 0.0;

passenger->totalTime = 0.0;

passenger->boardingTimer = 0.0;

passenger->onTrip = false;

passenger->isBoarding = false;

passenger->boardingTimeLimit = boardingTimeLimit;

return passenger;

}

PassengerBuilder\* PassengerBuilder::setBoardingTimeLimit(double boardingTime)

{

boardingTimeLimit = boardingTime;

return this;

}

bus.cpp

#include "bus.h"

#include "route.h"

#include "busstop.h"

#include "passenger.h"

int Bus::getFreeSitIndex()

{

int result = -1;

for (int i = 0; i < passengers.size(); i++) {

if (passengers[i] == nullptr) {

result = i;

break;

}

}

return result;

}

Bus::~Bus()

{

// управление жизненным циклом пассажиров

// пока они едут в автобусе, либо заходят/выходят, за них "отвечает" автобус

for (Entity\* passengerEntity : passengers) {

if (passengerEntity) {

delete passengerEntity;

}

}

}

void Bus::tick(double delta)

{

Route\* route = (Route\*)parentRoute;

// сбор статистики по загруженности

if (route->getStatcollector()) {

route->getStatcollector()->addBusLoad(id, 1.0 - (double)getFreeSeatCount() / (double)passengers.size());

}

// пока пассажир в автобусе, за его моделирование "отвечает" автобус

for (Entity\* passenger : passengers) {

if (passenger) {

passenger->tick(delta);

}

}

if (inTransit) {

// с течением времени изменяем координату автобуса

// в соответствии с его скоростью и прошедшим временем

position += velocity \* (delta / 3600.0); // delta в секуднах переводим в часы

// просматриваем информацию по каждой остановке

for (Entity\* busstopEntity : \*route->getBusstopArray()) {

Busstop\* busstop = (Busstop\*)busstopEntity;

// смотрим по координатам - проехал ли остановку?

if (position > busstop->getPosition()) {

// если остановка не в списке пройденных

if (find(passedBusstops.begin(), passedBusstops.end(), busstop) == passedBusstops.end()) {

// добавляем в список пройденных

passedBusstops.push\_back(busstop);

// останавливаемся на остановке если надо кому-то выйти на ней,

// либо если есть свободные места и на остановке ожидает пассажир

if (needToGetOff(busstop) ||

(getFreeSeatCount() > 0 && busstop->haveWaiters())) {

// добавляем остановку в список остановок, на которых автобус останавливался

visitedBusstops.push\_back(busstop);

// обнуляем таймер остановки

stopTimer = 0.0;

// задаем состояние автобуска как "не в движении" (на остановке)

inTransit = false;

}

}

}

}

}

else {

// увеличиваем таймер остановки на интервал моделирования

stopTimer += delta;

// контроль высадки, затем посадки

// флаг, показывающий, что идет процесс посадки/высадки

bool boardingFlag = false;

// проверяем, осуществляется ли в данный момент посадка/высадка

for (Entity\* passengerEntity : passengers) {

if (passengerEntity) {

Passenger\* passenger = (Passenger\*)passengerEntity;

boardingFlag |= !passenger->isBoardingEnded();

}

}

// если никто не садится/выходит, проверяем, кому надо выйти

if (!boardingFlag) {

// получаем текущую остановку

Busstop\* currentBusstop = (Busstop\*)visitedBusstops[visitedBusstops.size() - 1];

if (currentBusstop) {

// освобождаем места в автобусе

for (int i = 0; i < passengers.size(); i++) {

Passenger\* passenger = (Passenger\*)passengers[i];

if (passenger) {

if (passenger->isBoardingEnded() && !passenger->isOnTrip()) {

passengers[i] = nullptr;

}

}

}

// ищем пассажира, которому надо выйти на текущей остановке

for (Entity\* passengerEntity : passengers) {

if (passengerEntity) {

Passenger\* passenger = (Passenger\*)passengerEntity;

if (passenger->getTargetBusstop() == currentBusstop) {

// запускаем процесс высадки

passenger->startBoarding();

boardingFlag = true;

// добавляем в список прибывших

currentBusstop->addArrival(passenger);

break;

}

}

}

// если все, кто хотел, вышли из автобуса, начинаем посадку

if (!boardingFlag && (getFreeSeatCount() > 0)) {

if (currentBusstop->haveWaiters()) {

// начинаем процесс посадки у первого пассажира в очереди на остановке

Passenger\* passenger = (Passenger\*)(\*currentBusstop->getWaitingPassengers())[0];

// удаляем с остановки

currentBusstop->getWaitingPassengers()->erase(currentBusstop->getWaitingPassengers()->begin());

// сажаем пассажира на свободное место

passengers[getFreeSitIndex()] = passenger;

// запускаем процесс посадки

passenger->startBoarding();

boardingFlag = true;

}

}

}

}

// когда посадка/высадка завершена и вышел предел ожидания, автобус движется дальше

if (!boardingFlag && (stopTimer > stopTimeLimit)) {

// задаем состояние автобуса как "в движении"

inTransit = true;

}

}

if (inTransit) {

// проверка делается, когда автобус отъехал от последней остановки

// проверка на выход за пределы маршрута и деспавн автобуса

if (position > route->getLength()) {

// помечаем автобус для удаления с маршрута

needToDelete = true;

}

}

}

int Bus::getId()

{

return id;

}

bool Bus::isNeedToDelete()

{

return needToDelete;

}

double Bus::getPosition()

{

return position;

}

void Bus::setParentRoute(Entity\* route)

{

parentRoute = route;

}

vector<Entity\*>\* Bus::getPassedBusstops()

{

return &passedBusstops;

}

vector<Entity\*>\* Bus::getVisitedBusstops()

{

return &visitedBusstops;

}

bool Bus::getTransitState()

{

return inTransit;

}

vector<Entity\*>\* Bus::getPassengers()

{

return &passengers;

}

int Bus::getPassengerCount()

{

int passengersCount = 0;

for (int i = 0; i < passengers.size(); i++) {

if (passengers[i]) {

passengersCount++;

}

}

return passengersCount;

}

int Bus::getFreeSeatCount()

{

return (int)passengers.size() - getPassengerCount();

}

bool Bus::needToGetOff(Entity\* busstop)

{

bool result = false;

for (Entity\* passengerEntity : passengers) {

if (passengerEntity) {

Passenger\* passenger = (Passenger\*)passengerEntity;

result |= (passenger->getTargetBusstop() == busstop);

}

}

return result;

}

bool Bus::sitOnTheBus(Entity\* passenger)

{

return find(passengers.begin(), passengers.end(), passenger) != passengers.end();

}

BusBuilder\* BusBuilder::setPosition(double pos)

{

position = pos;

return this;

}

BusBuilder\* BusBuilder::setVelocity(double vel)

{

velocity = vel;

return this;

}

BusBuilder\* BusBuilder::setTransit(bool t)

{

inTransit = t;

return this;

}

BusBuilder\* BusBuilder::setStopTimeLimit(double stl)

{

stopTimeLimit = stl;

return this;

}

BusBuilder\* BusBuilder::setPassengersLimit(int pl)

{

passengersLimit = pl;

return this;

}

Entity\* BusBuilder::build()

{

Bus\* bus = new Bus();

bus->inTransit = inTransit;

bus->velocity = velocity;

bus->position = position;

bus->needToDelete = needToDelete;

bus->stopTimeLimit = stopTimeLimit;

bus->id = idCounter++;

bus->getPassengers()->resize(passengersLimit);

// делаем все места свободными

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

(\*bus->getPassengers())[i] = nullptr;

}

return bus;

}

busstop.cpp

#include "busstop.h"

#include "route.h"

#include "passenger.h"

#include "random.h"

Entity\* BusstopBuilder::build()

{

Busstop\* busstop = new Busstop();

busstop->position = position;

busstop->passengerBuilder = passengerBuilder;

busstop->parentRoute = parentRoute;

busstop->id = idCounter++;

busstop->spawnEnable = spawnEnable;

busstop->lowSpawnTime = lowSpawnTime;

busstop->highSpawnTime = highSpawnTime;

busstop->spawnTimer = 0.0;

busstop->currentSpawnTime = rnd(lowSpawnTime, highSpawnTime);

return busstop;

}

BusstopBuilder\* BusstopBuilder::setPosition(double pos)

{

position = pos;

return this;

}

BusstopBuilder\* BusstopBuilder::setPassengerBuilder(Builder\* builder)

{

passengerBuilder = builder;

return this;

}

BusstopBuilder\* BusstopBuilder::setParentRoute(Entity\* route)

{

parentRoute = route;

return this;

}

BusstopBuilder\* BusstopBuilder::setSpawnEnable(bool flag)

{

spawnEnable = flag;

return this;

}

BusstopBuilder\* BusstopBuilder::setLowSpawnTime(double time)

{

lowSpawnTime = time;

return this;

}

BusstopBuilder\* BusstopBuilder::setHighSpawnTime(double time)

{

highSpawnTime = time;

return this;

}

void Busstop::tick(double delta)

{

Route\* route = (Route\*)parentRoute;

if (spawnEnable) {

spawnTimer += delta;

if (spawnTimer > currentSpawnTime) {

// сбрасываем таймер

currentSpawnTime = rnd(lowSpawnTime, highSpawnTime);

spawnTimer = 0.0;

// генерируем нового пассажира

int selfIndex = route->getBusstopIndex(this);

// пассажиру надо будет проехать случайно число остановок (не более 2/3 от всего количества остановок)

int passCount = 1 + rnd(min(route->getBusstopCount() - selfIndex - 1, 2 \* route->getBusstopCount() / 3));

Entity\* target = route->getBusstop(selfIndex + passCount);

// проверка на ошибочную генерацию пункта назначения

if (target == this) {

throw "same busstop!";

}

spawnPassenger(target);

}

}

// удаляем прибывших пассажиров со сбором статистики по ним

for (int i = 0; i < arrivalPassengers.size(); i++) {

Passenger\* passenger = (Passenger\*)arrivalPassengers[i];

// удаляем пассажира только когда высадка полностью завершилась

if (passenger->isBoardingEnded()) {

despawnPassenger(passenger);

i--;

}

}

}

int Busstop::getId()

{

return id;

}

double Busstop::getPosition()

{

return position;

}

void Busstop::setParentRoute(Entity\* route)

{

parentRoute = route;

}

Entity\* Busstop::getParentRoute()

{

return parentRoute;

}

void Busstop::setPassengerBuilder(Builder\* builder)

{

passengerBuilder = builder;

}

Builder\* Busstop::getPassengerBuilder()

{

return passengerBuilder;

}

bool Busstop::haveWaiters()

{

return waitingPassengers.size() > 0;

}

vector<Entity\*>\* Busstop::getWaitingPassengers()

{

return &waitingPassengers;

}

void Busstop::addArrival(Entity\* passenger)

{

arrivalPassengers.push\_back(passenger);

}

void Busstop::spawnPassenger(Entity\* targetBusstop)

{

// создаем пассажира при помощи билдера

Passenger\* passenger = (Passenger\*)passengerBuilder->build();

// задаем начальную остановку

passenger->setStartBusstop(this);

// задаем конечную для него остановку

passenger->setTargetBusstop(targetBusstop);

// добавлем в массив ожидающих пассажиров на остановке

waitingPassengers.push\_back(passenger);

}

void Busstop::despawnPassenger(Entity\* passenger)

{

// удаление пассажира аналогично удалению автобуса

auto ptrItem = find(arrivalPassengers.begin(), arrivalPassengers.end(), passenger);

if (ptrItem != arrivalPassengers.end()) {

Entity\* entity = \*ptrItem;

// сбор статистики по пассажиру

Passenger\* passenger = (Passenger\*)entity;

Route\* route = (Route\*)parentRoute;

if (route->getStatcollector()) {

route->getStatcollector()->addTripTime(passenger->getTripTime());

route->getStatcollector()->addTotalTime(passenger->getTotalTime());

}

delete entity;

arrivalPassengers.erase(ptrItem);

}

}

route.cpp

#include "route.h"

#include "bus.h"

Route::~Route()

{

// очищаем память из-под ресурсов

for (Entity\* bus : busArray) {

delete bus;

}

for (Entity\* busstop : busstopArray) {

delete busstop;

}

}

void Route::tick(double delta)

{

// ищем автобусы с отметкой для удаления и удаляем их

for (Entity\* busEntity : busArray) {

Bus\* bus = (Bus\*)busEntity;

if (bus->isNeedToDelete()) {

despawnBus(bus);

}

}

// добавляем автобусы, если на маршруте их меньше, чем надо

while (busArray.size() < busCount) {

spawnBus();

}

// вызывваем "тик" для каждого автобуса

for (Entity\* bus : busArray) {

bus->tick(delta);

}

// и для каждой остановки

for (Entity\* busstopEntity : busstopArray) {

busstopEntity->tick(delta);

}

}

void Route::setLength(double l)

{

length = l;

}

double Route::getLength()

{

return length;

}

void Route::setBusCount(int c)

{

busCount = c;

}

int Route::getBusCount()

{

return busCount;

}

void Route::setBusBuilder(Builder\* builder)

{

busBuilder = builder;

}

Builder\* Route::getBusBuilder()

{

return busBuilder;

}

vector<Entity\*>\* Route::getBusArray()

{

return &busArray;

}

vector<Entity\*>\* Route::getBusstopArray()

{

return &busstopArray;

}

void Route::setStatcollector(Statcollector\* sc)

{

statcollector = sc;

}

Statcollector\* Route::getStatcollector()

{

return statcollector;

}

int Route::getBusstopIndex(Entity\* busstop)

{

int result = -1;

for (int i = 0; i < busstopArray.size(); i++) {

if (busstopArray[i] == busstop) {

result = i;

break;

}

}

return result;

}

int Route::getBusstopCount()

{

return (int)busstopArray.size();

}

Entity\* Route::getBusstop(int index)

{

return busstopArray[index];

}

void Route::despawnBus(Entity\* ptr)

{

// удаление автобуса

// найдем его в массиве атобусов

auto ptrItem = find(busArray.begin(), busArray.end(), ptr);

if (ptrItem != busArray.end()) {

// удаляем данные

Entity\* entity = \*ptrItem;

// перед удалением собираем статистику

Bus\* bus = (Bus\*)entity;

// генерируем ошибку, если удаляется автобус, из которого вышли не все пассажиры

if (bus->getPassengerCount() > 0) {

throw "passenger on the bus!";

}

if (getStatcollector()) {

getStatcollector()->

addMissedStops((int)bus->getPassedBusstops()->size() - (int)bus->getVisitedBusstops()->size());

}

delete entity;

// удаляем элемент из массива

busArray.erase(ptrItem);

}

}

void Route::spawnBus()

{

// добавляем в массив автобусов новый, созданный при помощи билдера

Entity\* newBus = busBuilder->build();

// задаем текущий маршрут родительским для нового автобуса

((Bus\*)newBus)->setParentRoute(this);

busArray.push\_back(newBus);

}

experiment.cpp

#include "experiment.h"

#include "route.h"

#include "bus.h"

#include "busstop.h"

#include "passenger.h"

#include "random.h"

#include <stdio.h>

#include <stdlib.h>

#include <time.h>

ExperimentOutputData doExperiment(ExperimentInputData inputData)

{

clock\_t start, end;

start = clock();

ExperimentOutputData result;

Route route;

BusBuilder busBuilder;

BusstopBuilder busstopBuilder;

Statcollector statcollector;

PassengerBuilder passengerBuilder;

// начинаем с начала маршрута

busBuilder.setPosition(0.0);

// в начальное точке движемся (если первая точка - остановка, то надо false)

busBuilder.setTransit(true);

// скорость автобуса

busBuilder.setVelocity(inputData.busVelocity);

// ожидание пассажира на остановке

busBuilder.setStopTimeLimit(inputData.busstopWaitingTime);

// количество мест в автобусе

busBuilder.setPassengersLimit(inputData.seatLimit);

route.setBusBuilder(&busBuilder);

// количество автобусов на маршруте

route.setBusCount(inputData.busCount);

route.setStatcollector(&statcollector);

// время посадки/высадки пассажира

passengerBuilder.setBoardingTimeLimit(inputData.boardingTime);

busstopBuilder.setParentRoute(&route);

busstopBuilder.setPassengerBuilder(&passengerBuilder);

busstopBuilder.setHighSpawnTime(inputData.passengerLowSpawnTime);

busstopBuilder.setLowSpawnTime(inputData.passengerHighSpawnTime);

busstopBuilder.setSpawnEnable(true);

// создание остановок на маршруте

for (int i = 0; i < inputData.busstops.size() - 1; i++) {

route.getBusstopArray()->push\_back((Busstop\*)busstopBuilder

.setPosition(inputData.busstops[i])->build());

}

// на конечной остановке не появляются пассажиры

route.getBusstopArray()->push\_back((Busstop\*)busstopBuilder

.setPosition(inputData.busstops[inputData.busstops.size() - 1])->setSpawnEnable(false)->build());

// определяем общую длину маршрута

route.setLength(inputData.busstops[inputData.busstops.size() - 1]);

// изначально размещаем некоторе количество пассажиров на остановках

for (int i = 0; i < route.getBusstopCount() - 1; i++) {

// генерируем число пассажиров

int count = rnd(inputData.initPassengersLowCount, inputData.initPassengersHighCount);

Busstop\* busstop = (Busstop\*)route.getBusstopArray()->at(i);

for (int j = 0; j < count; j++) {

// пассажиру надо будет проехать случайно число остановок (не более 2/3 от всего количества остановок)

int passCount = 1 + rnd(min(route.getBusstopCount() - i - 1, 2 \* route.getBusstopCount() / 3));

Entity\* target = route.getBusstop(i + passCount);

// размещаем пассажира на остановке

busstop->spawnPassenger(target);

}

}

// процесс моделирования

for (double t = 0.0; t < inputData.modelingTime; t += inputData.dt) {

route.tick(inputData.dt);

}

// сохраняем результаты эксперимента

result.missedBusstops = statcollector.getAvgMessedStopsCount();

result.tripTime = statcollector.getAvgTripTime();

result.totalTime = statcollector.getAvgTotalTime();

for (auto i : statcollector.getBusStat()) {

result.busLoad.insert(pair<int, double>(i.first, i.second \* 100.0));

}

result.allBusLoad = statcollector.getAvgAllBusLoad() \* 100.0;

end = clock();

// вычисляем время проведения эксперимента

result.time = ((double)end - start) / ((double)CLOCKS\_PER\_SEC);

return result;

}

statcollector.cpp

#include "statcollector.h"

void Statcollector::addMissedStops(int ms)

{

missedStopsCount.push\_back(ms);

}

void Statcollector::addTripTime(double t)

{

tripTime.push\_back(t);

}

void Statcollector::addTotalTime(double t)

{

totalTime.push\_back(t);

}

void Statcollector::addBusLoad(int id, double load)

{

if (busLoad.find(id) == busLoad.end()) {

busLoad.insert(pair<int, vector<double>>(id, {}));

}

busLoad[id].push\_back(load);

}

double Statcollector::getAvgMessedStopsCount()

{

return getAvg(&missedStopsCount);

}

double Statcollector::getAvgTripTime()

{

return getAvg(&tripTime);

}

double Statcollector::getAvgTotalTime()

{

return getAvg(&totalTime);

}

double Statcollector::getAvgBusLoad(int id)

{

if (busLoad.find(id) == busLoad.end()) {

return 0.0;

}

return getAvg(&busLoad[id]);

}

double Statcollector::getAvgAllBusLoad()

{

double result = 0.0;

int count = 0;

for (auto i : busLoad) {

result += getAvgBusLoad(i.first);

count++;

}

if (count > 0) {

result = result / (double)count;

}

return result;

}

map<int, double> Statcollector::getBusStat()

{

map<int, double> result;

for (auto i : busLoad) {

result.insert(pair<int, double>(i.first, getAvg(&i.second)));

}

return result;

}

random.cpp

#include "random.h"

#include <time.h>

void randomize()

{

srand((unsigned int)time(0));

}

inputreader.cpp

#include "inputreader.h"

#include <iostream>

#include <fstream>

ExperimentInputData read(const char\* filename)

{

ExperimentInputData result;

// открываем файл для чтения

ifstream f(filename);

char buf[64];

if (f) {

// читаем до конца файла

while (!f.eof()) {

// считываем название параметра

f >> buf;

// проверяем, какой параметр считали

if (strcmp(buf, "busVelocity") == 0) {

f >> result.busVelocity;

}

if (strcmp(buf, "busstopWaitingTime") == 0) {

f >> result.busstopWaitingTime;

}

if (strcmp(buf, "seatLimit") == 0) {

f >> result.seatLimit;

}

if (strcmp(buf, "busCount") == 0) {

f >> result.busCount;

}

if (strcmp(buf, "boardingTime") == 0) {

f >> result.boardingTime;

}

if (strcmp(buf, "initPassengersLowCount") == 0) {

f >> result.initPassengersLowCount;

}

if (strcmp(buf, "initPassengersHighCount") == 0) {

f >> result.initPassengersHighCount;

}

if (strcmp(buf, "passengerLowSpawnTime") == 0) {

f >> result.passengerLowSpawnTime;

}

if (strcmp(buf, "passengerHighSpawnTime") == 0) {

f >> result.passengerHighSpawnTime;

}

if (strcmp(buf, "busstops") == 0) {

int busstopCount;

f >> busstopCount;

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

double pos;

f >> pos;

result.busstops.push\_back(pos);

}

}

if (strcmp(buf, "dt") == 0) {

f >> result.dt;

}

if (strcmp(buf, "modelingTime") == 0) {

f >> result.modelingTime;

}

}

f.close();

}

else {

// если файл не найден

throw "file not found!";

}

return result;

}

outputwriter.cpp

#include "outputwriter.h"

void print\_time(FILE\* file, double sec) {

int h = (int)sec / 3600;

sec -= h \* 3600.0;

int m = (int)sec / 60;

sec -= m \* 60.0;

fprintf(file, "%i h %i m %f sec", h, m, sec);

}

void write(const char\* filename, const ExperimentInputData& inputData, const ExperimentOutputData& outputData)

{

FILE\* file = nullptr;

fopen\_s(&file, filename, "w");

if (file) {

// пишем исходные данные

fprintf(file, "INPUT DATA:\n");

fprintf(file, "bus velocity: %.2f km/h\n", inputData.busVelocity);

fprintf(file, "busstop waiting time: %.2f sec\n", inputData.busstopWaitingTime);

fprintf(file, "bus seat count: %i\n", inputData.seatLimit);

fprintf(file, "bus count limit: %d\n", inputData.busCount);

fprintf(file, "boarding time: %.2f sec\n", inputData.boardingTime);

fprintf(file, "init busstop passenger count between %i and %i\n", inputData.initPassengersLowCount, inputData.initPassengersHighCount);

fprintf(file, "passenger spawn time between ");

print\_time(file, inputData.passengerLowSpawnTime);

fprintf(file, " and ");

print\_time(file, inputData.passengerHighSpawnTime);

fprintf(file, "\n");

fprintf(file, "busstops (%i):\n", (int)inputData.busstops.size());

for (int i = 0; i < inputData.busstops.size(); i++) {

fprintf(file, "%.2f km, ", inputData.busstops[i]);

}

fprintf(file, "\n");

fprintf(file, "route length: %.2f km\n", inputData.busstops[inputData.busstops.size() - 1]);

fprintf(file, "modeling time: ");

print\_time(file, inputData.modelingTime);

fprintf(file, "\n");

fprintf(file, "dt = %.2f sec\n", inputData.dt);

// пишем результаты

fprintf(file, "\nmodeling ends in: %.4f sec\n\n", outputData.time);

fprintf(file, "RESULTS:\n");

fprintf(file, "average missed stops count: %f\n", outputData.missedBusstops);

fprintf(file, "average passenger trip time: ");

print\_time(file, outputData.tripTime);

fprintf(file, " (%f sec)\n", outputData.tripTime);

fprintf(file, "average passenger total time: ");

print\_time(file, outputData.totalTime);

fprintf(file, " (%f sec)\n", outputData.totalTime);

fprintf(file, "bus load by id:\n");

for (auto i : outputData.busLoad) {

fprintf(file, "#%d: %.2f%%\n", i.first, i.second);

}

fprintf(file, "all bus load is: %.2f%%\n", outputData.allBusLoad);

fclose(file);

}

}

main.cpp

#include "experiment.h"

#include "inputreader.h"

#include "outputwriter.h"

#include "random.h"

#include <iostream>

using namespace std;

bool doCalculations(ExperimentInputData& inp, ExperimentOutputData& out) {

bool result = false;

try {

// осуществляем вычисления

printf\_s("calculating...\n");

out = doExperiment(inp);

printf\_s("done in %.4f second(s)\n", out.time);

result = true;

}

catch (...) {

}

return result;

}

void print\_time(double sec) {

int h = (int)sec / 3600;

sec -= h \* 3600.0;

int m = (int)sec / 60;

sec -= m \* 60.0;

printf\_s("%i h %i m %f sec", h, m, sec);

}

int main() {

int result = 0;

// сбрасываем генератор случайных чисел

randomize();

// входные и выходные данные для моделирования

ExperimentInputData inp;

ExperimentOutputData out;

int option = 0;

bool haveResults = false;

char inpfile[256];

char outfile[256];

do {

// печатаем меню

printf\_s("\n\nchoose option:\n");

printf\_s("[1] - enter data from this,\n");

printf\_s("[2] - enter data from file,\n");

printf\_s("[3] - show results,\n");

printf\_s("[4] - save result to file,\n");

printf\_s("[5] - exit.\n");

cin >> option;

// очищаем экран

system("cls");

int busstopCount;

double busstopPosition;

switch (option) {

case 1:

result = 0;

printf\_s("enter input data.\n");

printf\_s("bus velocity (km/h): ");

cin >> inp.busVelocity;

printf\_s("busstop waiting time (sec): ");

cin >> inp.busstopWaitingTime;

printf\_s("seat limit: ");

cin >> inp.seatLimit;

printf\_s("bus count: ");

cin >> inp.busCount;

printf\_s("boarding time (sec): ");

cin >> inp.boardingTime;

printf\_s("init passengers low count: ");

cin >> inp.initPassengersLowCount;

printf\_s("init passengers high count: ");

cin >> inp.initPassengersHighCount;

printf\_s("passenger low spawn time (sec): ");

cin >> inp.passengerLowSpawnTime;

printf\_s("passenger high spawn time (sec): ");

cin >> inp.passengerHighSpawnTime;

printf\_s("busstop count: ");

cin >> busstopCount;

inp.busstops.clear();

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

printf\_s("busstop #%i position (km): ", i);

cin >> busstopPosition;

inp.busstops.push\_back(busstopPosition);

}

printf\_s("modeling time step (sec): ");

cin >> inp.dt;

printf\_s("total modeling time (sec): ");

cin >> inp.modelingTime;

if (!doCalculations(inp, out)) { // если произошла ошибка во время расчетов

result = -2;

printf\_s("error while calculating...\n");

haveResults = false;

}

else {

haveResults = true;

}

break;

case 2:

result = 0;

printf\_s("enter input data filename: ");

cin >> inpfile;

try {

printf\_s("reading from '%s'...\n", inpfile);

inp = read(inpfile);

}

catch (...) {

// ошибка чтения входного файла

result = -1;

printf\_s("error while file opening...\n");

}

// нет ошибок чтения, значит считаем

if (result == 0) {

if (!doCalculations(inp, out)) { // если произошла ошибка во время расчетов

result = -2;

printf\_s("error while calculating...\n");

haveResults = false;

}

else {

haveResults = true;

}

}

break;

case 3:

if (haveResults) {

// печатаем результаты в консоли

// пишем исходные данные

printf\_s( "INPUT DATA:\n");

printf\_s( "bus velocity: %.2f km/h\n", inp.busVelocity);

printf\_s( "busstop waiting time: %.2f sec\n", inp.busstopWaitingTime);

printf\_s( "bus seat count: %i\n", inp.seatLimit);

printf\_s( "bus count limit: %d\n", inp.busCount);

printf\_s( "boarding time: %.2f sec\n", inp.boardingTime);

printf\_s( "init busstop passenger count between %i and %i\n", inp.initPassengersLowCount, inp.initPassengersHighCount);

printf\_s( "passenger spawn time between ");

print\_time(inp.passengerLowSpawnTime);

printf\_s( " and ");

print\_time(inp.passengerHighSpawnTime);

printf\_s( "\n");

printf\_s( "busstops (%i):\n", (int)inp.busstops.size());

for (int i = 0; i < inp.busstops.size(); i++) {

printf\_s( "%.2f km, ", inp.busstops[i]);

}

printf\_s( "\n");

printf\_s( "route length: %.2f km\n", inp.busstops[inp.busstops.size() - 1]);

printf\_s( "modeling time: ");

print\_time(inp.modelingTime);

printf\_s( "\n");

printf\_s( "dt = %.2f sec\n", inp.dt);

// пишем результаты

printf\_s( "\nmodeling ends in: %.4f sec\n\n", out.time);

printf\_s( "RESULTS:\n");

printf\_s( "average missed stops count: %f\n", out.missedBusstops);

printf\_s( "average passenger trip time: ");

print\_time(out.tripTime);

printf\_s( " (%f sec)\n", out.tripTime);

printf\_s( "average passenger total time: ");

print\_time(out.totalTime);

printf\_s( " (%f sec)\n", out.totalTime);

printf\_s( "bus load by id:\n");

for (auto i : out.busLoad) {

printf\_s( "#%d: %.2f%%\n", i.first, i.second);

}

printf\_s( "all bus load is: %.2f%%\n", out.allBusLoad);

}

else {

printf\_s("there is not any modeling results. please enter input data.\n");

}

break;

case 4:

if (haveResults) {

printf\_s("enter output file name: ");

cin >> outfile;

// сохраняем результаты вычислений в файл

write(outfile, inp, out);

printf("result saved into '%s'\n", outfile);

}

else {

printf\_s("there is not any modeling results. please enter input data.\n");

}

break;

case 5:

break;

default:

printf\_s("wrong option!\n");

}

} while (option != 5);

system("pause");

return result;

}