МИНОБРНАУКИ РОССИИ

## Федеральное государственное автономное образовательное

## учреждение высшего образования

## «ЮЖНЫЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Институт компьютерных технологий и информационной безопасности

Направление подготовки (шифр, название) 09.03.04 “Программная инженерия”

**Отчет по практике**

обучающегося 3 курса

Фамилия Кузьменков

Имя Михаил

Отчество Валентинович

|  |  |  |
| --- | --- | --- |
| Обучающийся |  | Кузьменков М.В. |
|  | подпись | расшифровка подписи |

Место практики ООО “Программные технологии”

наименование профильной организации

Вид практики Производственная практика

Тип практики По получению профессиональных умений и опыта профессиональной деятельности

Способ проведения практики Стационарная

Форма проведения практики Дискретная (по видам практик)

Сроки прохождения практики *с* 10.02.2020 *по* 08.03.2020

|  |
| --- |
| Руководитель практики  от структурного подразделения ЮФУ |
| Родзина Ольга Николаевна |

ФИО, подпись

Техническое задание на разработку десктопного картографического приложения

1. Главный экран приложения должен представлять собой окно, в которое выводятся различные картографические элементы и условные знаки (параллели и меридианы, плитки карты, пользовательские точки).

2. Наборы одинаковых картографических элементов или условных знаков должны быть объединены в слои.

3. Должна быть предусмотрена возможность включать и отключать слои.

4. Параллели и меридианы должны быть окрашены в разные цвета, с плавным понижением их яркости от центра карты к её краям.

5. Должна быть предусмотрена возможность добавлять пользовательские точки на карту.

6. Плитки карты должны подгружаться с локального или удаленного сервера в ходе работы пользователя с приложением.

7. Должна быть реализованы базовые возможности управления картой (увеличение и уменьшение масштаба карты, перемещение по карте влево, вправо, вверх и вниз).

8. Должна быть предусмотрена возможность изменения размера главного экрана.

Срок реализации приложения: до 08.03.2020.

Обозначения и сокращения

В настоящем документе применяются следующие обозначения и сокращения:

ПО – программное обеспечение;

ТЗ – техническое задание;

ГИС – геоинформационная система.

Содержание

[Введение 6](#_Toc34720063)

[1 Общая информация о приложении 7](#_Toc34720064)

[1.1 Техническая информация 7](#_Toc34720065)

[2 Описание понятий и концепций предметной области 8](#_Toc34720066)

[2.1 Web Mercator 8](#_Toc34720067)

[2.1 Колбэк 8](#_Toc34720068)

[2.2 Плитка 9](#_Toc34720069)

[2.3 Другие понятия 10](#_Toc34720070)

[3 Архитектура приложения 11](#_Toc34720071)

[3.1 Модель, отрисовщик и контроллер 11](#_Toc34720072)

[3.2 Слои и мини-отрисовщики 12](#_Toc34720073)

[3.3 Плиточный слой 12](#_Toc34720074)

[3.4 Доступ к разделяемым ресурсам 13](#_Toc34720075)

[4 Описание модулей приложения 14](#_Toc34720076)

[4.1 Описание входной точки приложения 14](#_Toc34720077)

[4.2 Описание модели 14](#_Toc34720078)

[4.3 Описание слоев 14](#_Toc34720079)

[4.3.1 Описание плиточного слоя 15](#_Toc34720080)

[4.3.2 Описание слоя сетки параллелей и меридианов 16](#_Toc34720081)

[4.3.3 Описание слоя пользовательских точек 16](#_Toc34720082)

[4.4 Описание плиток 16](#_Toc34720083)

[4.5 Описание плиточных модулей 16](#_Toc34720084)

[4.5.1 Описание плиточных структур 17](#_Toc34720085)

[4.5.2 Описание плиточного кэша 17](#_Toc34720086)

[4.5.3 Плиточный загрузчик 18](#_Toc34720087)

[4.6 Описание отрисовщика 19](#_Toc34720088)

[4.7 Описание окон приложения 20](#_Toc34720089)

[4.7.1 Описание базового окна 20](#_Toc34720090)

[4.7.2 Описание окна отрисовщика 20](#_Toc34720091)

[4.7.3 Описание окна включения/отключения слоев 20](#_Toc34720092)

[4.8 Описание мини-отрисовщиков 21](#_Toc34720093)

[4.8.1 Описание базового мини-отрисовщика 21](#_Toc34720094)

[4.8.2 Описание мини-отрисовщик плиточного слоя 21](#_Toc34720095)

[4.8.3 Описание мини-отрисовщика слоя сетки параллелей и меридианов 22](#_Toc34720096)

[4.8.4 Описание мини-отрисовщика слоя пользовательских точек 23](#_Toc34720097)

[4.9 Описание вспомогательных библиотек 23](#_Toc34720098)

[5 Описание алгоритмов 24](#_Toc34720099)

[5.1 Алгоритм работы программы 24](#_Toc34720100)

[5.2 Алгоритм получения видимых отрисовщиком плиток 24](#_Toc34720101)

[5.3 Алгоритм получения видимых отрисовщиком линий сетки параллелей и меридианов 25](#_Toc34720102)

[5.4 Алгоритм формирования запроса на загрузку плитки 25](#_Toc34720103)

[5.5 Алгоритм частичной очистки кэша 25](#_Toc34720104)

[5.6 Алгоритм обработки запроса на загрузку плитки 26](#_Toc34720105)

[5.7 Алгоритм перерисовки изображения окна отрисовщика 26](#_Toc34720106)

[6 Тестирование 27](#_Toc34720107)

[Загрузка плиток 27](#_Toc34720108)

[Обращение к серверу 27](#_Toc34720109)

[Цвет сетки параллелей и меридианов 28](#_Toc34720110)

[Обработка исключений 28](#_Toc34720111)

[Стресс-тестирование 28](#_Toc34720112)

[7 Пользовательский интерфейс 29](#_Toc34720113)

[Заключение 30](#_Toc34720114)

[Список использованных источников 31](#_Toc34720115)

[Приложение А 32](#_Toc34720116)

[Приложение Б 33](#_Toc34720117)

[Приложение В 34](#_Toc34720118)

Введение

Целями производственной практики является получение знаний, умений и навыков в области разработки ПО, изучение Windows API и приобретение навыков разработки картографических приложений и геоинформационных систем.

Задачами практики является разработка десктопного картографического приложения и применение на практике навыков разработки на базе Windows API.

Картографические приложения и геоинформационные системы позволяют представить различные пространственные данные в удобном для принятия решений виде. Они способны выявлять закономерности в предметной области и производить анализ данных, что позволяет экономить ресурсы и контролировать состояние инфраструктуры. Поэтому, создание картографических приложения и геоинформационных систем является актуальными направлением в области разработки программного обеспечения.

В ходе учебной практики, было необходимо спроектировать, разработать и протестировать картографическое приложение.

1. Общая информация о приложении

Данное приложение позволяет просматривать карту земной поверхности.

Для удобства поверх карты местности приложение отображает сетку параллелей и меридианов. Расстояние между параллелями – примерно 14о широты, а между меридианами – 30о долготы.

Имеется возможность отмечать точки на карте. После окончания работы приложения точки не сохраняются.

Земная поверхность, сетка параллелей и меридианов и пользовательские точки объединены в слои, которые можно включать и отключать.

Для работы приложения требуется подключение к сети Интернет.

* 1. Техническая информация

Приложение написано на языке C++ с использованием технологии Windows API и библиотек STL, Boost.Asio и LodePNG. Разработка велась на компьютере с процессором 1.6 GHz, видеокартой 128 Mb и оперативной памятью 512 Mb. Необходимая операционная система – Windows 8.1.

1. Описание понятий и концепций предметной области

Для описания работы программы необходимо описать понятия и концепции предметной области.

* 1. Web Mercator

Web Mercator представляет собой аппроксимированное изображение земной поверхности на квадратной плоскости. Координатами любой точки на такой карте являются широта и долгота (Рисунок 1).



Рисунок 1 – Web Mercator

* 1. Колбэк

Колбэк – это функция, которая должна быть вызвана после наступления определенного события в программе. Это используется при асинхронной модели приложения, для того, чтобы служебные (внутренние) функции приложения не блокировали пользовательский интерфейс.

Примером может послужить загрузка большого объема данных, во время которого пользователь может продолжать работать с приложением, и после которого может отобразиться сообщение об успешной загрузке. В данном случае, колбэк – это вывод сообщения.

* 1. Плитка

Изображение земной поверхности формируется из квадратных плиток расширением 256 х 256 пикселей. На верхнем, нулевом, уровне приближения (level of dimension, lod) вся земная поверхность представлена в виде одной плитки. На первом уровне – в виде четырех плиток (размерность квадратного изображения земной поверхности – 2 х 2 плитки). На втором уровне – в виде 16 плиток (4 х 4 плитки), и так далее.

Координаты плитки – это совокупность ее уровня приближения, положения по координате Х и Y (в плитках). Например, координаты единственной плитки на нулевом уровне приближения будут 0-0-0. На первом уровне приближения, координаты плиток будут соответственно (слева на право, сверху вниз): 1-0-0, 1-1-0, 1-0-1 и 1-1-1. В общем случае, на N-ом уровне приближения разбиение на плитки будет выглядеть следующим образом:

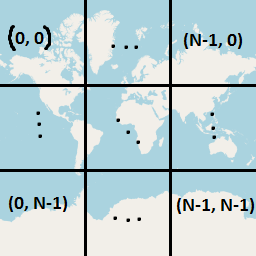


Рисунок 2 – Плиточные координаты N-ого уровня приближения

Плитки имеют плиточные координаты вида “lod-x-y”. Из координат такого вида можно сформировать запрос к плиточному серверу.

* 1. Другие понятия

*Слой* – это набор одинаковых элементов, например ресторанов или дорог. Объекты объединяются в слои для того, чтобы предоставить возможность пользователю просматривать только интересующую его информацию.

*Скейлирование* (от англ. Scale – Масштаб), в контексте данного приложения – это процесс перевода мировых координат в оконные. Этот процесс происходит на основе вычислений, в которых задействованы данные о местоположении объектов и данные о местоположение окна в мировых координатах, а так же данные о размерах окна и текущем уровне приближения.

1. Архитектура приложения

Любое картографическое приложение базируется на ряде стандартных архитектурных решений.

* 1. Модель, отрисовщик и контроллер

Приложение состоит из двух больших кластеров:

* + первый кластер – это модель. Она содержит в себе объекты реального мира и информацию о них. К объектам могут относиться различные географические объекты (плитки, сетка параллелей и меридианов) и пользовательские объекты (точки). К информации об объектах может относиться их местоположение, геометрические параметры и дополнительная информация. Основа модели – Web Mercator;
  + второй кластер – это отрисовщик. Его задача – отрисовывать изображения объектов и отображать информацию об объектах.

Оконный интерфейс приложения подразумевает, что пользователь взаимодействует с ним при помощи элементов окна, мыши и клавиатуры. Поэтому, так как отрисовщик выводит изображение тоже в окно, он совмещает в себе так же и функционал контроллера.

Можно представить, что окно приложения находится не только в пространстве экрана компьютера, но и в пространстве модели (мира). Отрисовщик в любой момент времени знает положение окна в мировом пространстве и меняет его в зависимости от действий пользователя.

Положение окна отрисовщика в мировом пространстве определяется координатами его верхнего левого и правого нижнего углов в мировом пространстве, а так же уровнем приближения.

Взаимодействие модели, отрисовщика и пользователя происходит следующим образом. При инициализации, отрисовщик, зная исходное положение своего окна в мировом пространстве, отрисовывает начальное изображение модели. Он ожидает действий пользователя. Когда пользователь совершает действий, например приближает карту или перемещается по карте с помощью мыши, отрисовщик меняет положение окна в мировом пространстве. Зная новое положение окна, отрисовщик перерисовывает изображение модели. Этот процесс продолжается до завершения работы приложения.

* 1. Слои и мини-отрисовщики

Объекты и информация о них хранятся в модели в виде слоев. Отрисовщик имеет для каждого слоя свой мини-отрисовщик. Во время отрисовки изображения модели, отрисовщик по очереди обращается к каждому мини-отрисовщику за изображением слоя, и затем комбинирует эти изображения в единое изображение.

Модель должна состоять из слоев и предоставлять доступ к ним извне.

* 1. Плиточный слой

Важной частью модели (и приложения в целом) является слой плиток карты. Дело в том, что плитки карты находятся на удаленном сервере (полная локальная копия всех плиток заняла бы свыше 1 Тб памяти). Поэтому, их нужно динамически подгружать с плиточного сервера и кэшировать. В приложении для этого существуют два класса, которые можно назвать плиточными загрузчиком и кэшем соответственно.

Загрузчик должен работать в отдельном потоке, чтобы процесс подгрузки плиток не блокировал пользовательский ввод. Чтобы иметь возможность завершить поток плиточного загрузчика, в его состав должен входить флаг активности, который может устанавливаться и сбрасываться извне. В потоке плиточного загрузчика должен вызываться метод обработки запроса на загрузку плитки.

Взаимодействие отрисовщика, модели, плиточных кэша и загрузчика происходит следующим образом. Отрисовщик запрашивает у модели плитки, которые должны быть отрисованы в окне для отображения текущего участка местности. Модель формирует список плиток и запрашивает их у кэша. Если в кэше имеется соответствующая плитка, то кэш возвращает ее. Если нет, то кэш запрашивает у загрузчика плитку, а сам возвращает изображение, замещающее отсутствующую плитку. Модель формирует из этих плиток массив и возвращает его отрисовщику, который в свою очередь отрисовывает их.

* 1. Доступ к разделяемым ресурсам

Так как отрисовщик и плиточный загрузчик работают в разных потоках, но в процессе своей работы обращаются к одним и тем же ресурсам, в приложении предусмотрена защита от “data races”, когда два и более потока стараются перезаписать/прочитать одну и ту же область памяти. Эта защита реализована в виде мьютекса.

Конкретно, разделяемыми ресурсами являются массив плиток плиточного кэша и очередь запросов на загрузку плиток плиточного загрузчика. Обращение к разделяемым ресурсам происходит во время обработки плиточным загрузчиком запроса на загрузку плитки и во время обработки плиточным слоем запроса на получение видимых отрисовщиком плиток. Таким образом, ссылка на мьютекс должна находиться в плиточном загрузчике и в плиточном слое.

Других потоков и разделяемых ресурсов в программе нет.

1. Описание модулей приложения

В ходе разработки приложения, были разработаны различные модули, структуры данных, классы и вспомогательная библиотека. Перед ознакомлением с данным разделом рекомендуется ознакомиться с приложениями А и B, в которых приведены схема подключения файлов и структура файловой директории соответственно. Они позволяют наглядно увидеть взаимосвязь между модулями.

* 1. Описание входной точки приложения

Входная точка представлена в приложении функцией “wWinMain”. Здесь происходит инициализация мьютекса доступа к элементам массива кэша и элементам очереди загрузчика, самого загрузчика, модели и отрисовщика, а так же запуск потока загрузчика и цикла обработки сообщений отрисовщика.

* 1. Описание модели

Модель представлена в приложении классом “MapModel”. В приложении имеется единственный экземпляр данного класса, который находится во входной точке приложения. В его состав входят слои и методы, предоставляющие доступ к слоям.

* 1. Описание слоев

Слои представлены в приложении базовым классом BaseLayer (а так же наследованными от него классами “TileLayer”, “LatLonGridLayer” и “UserPointsLayer”, речь о которых пойдет в пунктах данного подраздела). В приложении нет ни одного экземпляра класса BaseLayer. В его состав входят флаг видимости слоя и методы взаимодействия с ним.

* + 1. Описание плиточного слоя

Плиточный слой представлен в приложении классом “TileLayer”, который является наследником класса “BaseLayer”. В приложении имеется единственный экземпляр данного класса, который находится в модели. В его состав входят:

* + плиточный кэш;
  + массив пикселей изображения, замещающего массив пикселей изображения плитки, отсутствующей в плиточном кэше;
  + ссылка на мьютекс доступа к массиву плиток плиточного кэша и очереди запросов плиточного загрузчика;
  + метод для получения видимых отрисовщиком плиток.

Ниже приведен листинг заголовочного файла данного класса:

class TileLayer : public BaseLayer

{

public:

enum class Status

{

Clean,

Dirty

};

public:

TileLayer(

TileServices::TileRequester& tileRequester,

std::mutex& tileServicesMutex

) noexcept ;

private:

void \_InitTilePlaceholder() noexcept;

public:

std::vector<Tile> GetVisibleTiles(

RectWM wmWindow, unsigned lod,

Status& status) noexcept;

private:

TileServices::TileCache \_tileCache;

std::vector<unsigned char> \_tilePlaceholder;

std::mutex& \_tileServicesMutex;

};

Рисунок 3 – Листинг заголовочного файла класса “TileLayer”

* + 1. Описание слоя сетки параллелей и меридианов

Слой сетки параллелей и меридианов представлен в приложении классом “LatLonGridLayer”, который является наследником класса “BaseLayer”. В приложении имеется единственный экземпляр данного класса, который находится в модели. В его состав входят:

* + массив линий сетки параллелей и меридианов;
  + метод получения видимых отрисовщиком линий сетки параллелей и меридианов.
    1. Описание слоя пользовательских точек

Слой пользовательских точек представлен в приложении классом “UserPointsLayer”, который является наследником класса “BaseLayer”. В приложении имеется единственный экземпляр данного класса, который находится в модели. В его состав входят:

* + массив пользовательских точек;
  + метод получения видимых пользовательских точек.
  1. Описание плиток

Плитки представлены в приложении классом “Tile”. Объекты этого класса хранятся в плиточном кэше, а их копии попадают в массив видимых отрисовщиком плиток. В его состав входят:

– структура для описания ключа плитки;

* + ключ плитки;

– массив пикселей изображения плитки;

– время последнего использования (отрисовки) плитки.

* 1. Описание плиточных модулей

Плиточные модули – это совокупность модулей, связанных с процессом доставки плиток от плиточного сервера в модель.

* + 1. Описание плиточных структур

Структуры данных, необходимые для реализации манипуляций с плитками находятся в отдельном заголовочном файле. Среди них, основной является структура “TileSlot”. В ее состав входят:

* + статус плитки. Находясь в кэше, плитка может находится в одном из трех состояний: “свободна”, “загружается” и “занята”. Свободная плитка может быть перезаписана. Когда свободных и загружающихся плиток в кэше не остается, то происходит частичная очистка кэша;
  + ключ плитки;
  + изображение плитки;
  + время последнего использования плитки.
    1. Описание плиточного кэша

Плиточный кэш представлен в программе классом “TileCache”. В приложении имеется единственный экземпляр данного класса, который находится в плиточном слое. В его состав входят:

* + массив плиток;
  + массив указателей на свободные плитки;
  + массив вида “ключ-значение” указателей на занятые плитки (в роли ключа выступает ключ занятой плитки, а в виде значения выступает указатель на занятую плитку);
  + ссылка на плиточный загрузчик;
  + метод доступа к плитке по ключу;
  + метод добавления запроса на загрузку плитки в очередь запросов плиточного загрузчика. Вызывается, если при вызове метода доступа к плитке по ключу нужной плитки не оказалось;
  + методы частичной и полной очистки кэша. Частичная очистка кэша вызывается тогда, когда размер массива плиток кэша достигает заранее заданного лимита. Во время частичной очистки, свободными помечаются те плитки, которые давно не использовались.
    1. Плиточный загрузчик

Плиточный загрузчик представлен в программе классом “TileRequester”. В приложении имеется единственный экземпляр данного класса, который находится в точке входа приложения и работает в отдельном потоке [1]. В его состав входят:

* + очередь запросов на загрузку плиток;
  + поля и методы для установления и поддержания соединения с плиточным сервером;
  + метод обработки запроса на загрузку плитки;
  + методы для формирования запроса к плиточному серверу и обработки ответа плиточного сервера;
  + флаг активности;
  + мьютекс для доступа к массиву плиток плиточного кэша и очереди запросов на загрузку плиток.

Ниже приведен листинг заголовочного файла данного класса:

class TileRequester

{

// Creation section.

public:

TileRequester(std::mutex& tileServicesMutex);

void SetPCleanSlots(std::map<TileKey, TileSlot\*>\* pCleanSlots);

private:

void \_EstablishServerConnection();

// Requesting section.

public:

PushStatus PushTileRequest(const TileRequest& tileRequest);

void ProcessTileRequest();

private:

void \_RefreshConnection();

void \_MakeRequest(std::string requestString);

void \_ProcessResponseStatus(boost::asio::streambuf& responseBuf);

void \_ProcessResponseHeaders(boost::asio::streambuf& responseBuf, std::stringstream& imageBuf);

void \_ReadResponseData(boost::asio::streambuf& responseBuf, std::stringstream& imageBuf);

// Requesting control section.

public:

void StartRequesting() noexcept;

void StopRequesting() noexcept;

bool IsRequesting() noexcept;

// Destroying section.

public:

~TileRequester();

// State cection.

private:

std::stack<TileRequest> \_requests;

std::set<TileKey> \_requestsKeys;

bool \_requesting;

std::mutex& \_tileServicesMutex;

boost::asio::io\_service \_serverService;

boost::asio::ip::tcp::socket \_serverSocket;

boost::asio::ip::tcp::endpoint \_serverEndpoint;

std::map<TileKey, TileSlot\*>\* \_pCleanSlots;

};

Рисунок 4 – Листинг заголовочного файла класса “TileRequester”

* 1. Описание отрисовщика

Отрисовщик представлен в программе классом “MapView”. В приложении имеется единственный экземпляр данного класса, который находится в окне отрисовщика. В его состав входят:

– ссылка на модель;

– информационные поля состояния и местоположения окна;

– список мини-отрисовщиков;

– метод перерисовки изображения в окне отрисовщика;

– методы управления положением и состоянием окна. Они вызываются из окна отрисовщика, когда пользователь совершает определенные действия;

* 1. Описание окон приложения

В этом подразделе описаны окна приложения. Они представлены в программе базовым классом “BaseWindow” и его наследником “MapViewWindow”.

* + 1. Описание базового окна

Базовое окно представлено в программе классом “BaseWindow”. В приложении нет ни одного экземпляра данного класса [2, 3]. Он является основой для классов других окон. Его состав специфичен и не представляет интереса для данного отчета.

* + 1. Описание окна отрисовщика

Окно отрисовщика представлено в программе классом “MapViewWindow”, который является наследником класса “BaseWindow”. В приложении имеется единственный экземпляр данного класса, находящийся в точке входа приложения. В его состав входят:

* + отрисовщик;

– обработчик сообщений. Он идентифицирует сообщения, извлекает нужные параметры и вызывает соответствующие методы отрисовщика.

* + 1. Описание окна включения/отключения слоев

Окно включения/отключения слоев представлено в программе классом “LayerPanelWindow”, который является наследником класса “BaseWindow”. В приложении имеется единственный экземпляр данного класса, находящийся в окне отрисовщика. В его состав входят:

* + методы инициализации элементов окна (labels, checkboxes);
  + обработчик сообщений;
  + указатель на отрисовщик;
  + хэндл окна отрисовщика.
  1. Описание мини-отрисовщиков

В данном разделе описаны мини-отрисовщики. В приложении они представлены базовым классом “BaseRenderer” и его наследниками “TileRenderer”, “LatLonGridRenderer” и “UserPointsRenderer”.

* + 1. Описание базового мини-отрисовщика

Базовый мини-отрисовщик представлен в программе классом “BaseRenderer”. В приложении нет ни одного экземпляра данного класса. Он является основой для других мини-отрисовщиков. В его состав входят:

* + флаг активности;
  + нескольких, общих для всех мини-отрисовщиков, вспомогательных методов (скейлирование точки и отрезка);
  + ряда указателей на параметры состояния и положения окна отрисовщика.
    1. Описание мини-отрисовщик плиточного слоя

Мини-отрисовщик плиточного слоя представлен в программе классом “TileRenderer”, который является наследником класса “BaseRenderer”. В приложении он представлен в единственном экземпляре, который находится в отрисовщике. В его состав входят:

* + указатель на плиточный слой модели;
  + метод получения изображения плиточного слоя при заданном состоянии и положении окна отрисовщика;
  + вспомогательные методы отрисовки плиточного слоя.

Ниже приведен листинг заголовочного файла данного класса:

class TileRenderer : public BaseRenderer

{

public:

TileRenderer() noexcept;

TileRenderer(

TileLayer\* tileLayer,

HWND\* mvHwnd,

RectInt\* mvUiWindow,

RectWM\* mvWmWindow,

double\* mvScaleLat,

double\* mvScaleLon,

unsigned\* mvScaleLevel

) noexcept;

public:

HBITMAP GetTileLayerBitmap(const std::vector<unsigned char>& whiteBitmapPixels) noexcept;

private:

void \_DrawTiles(HDC tileLayerHdc) noexcept;

void \_DrawTile(HDC tileLayerHdc, const Tile& tile) noexcept;

void \_DrawTileBitmap(HDC tileLayerHdc, HBITMAP hTileBitmap, int bmpOriginX, int bmpOriginY) noexcept;

private:

TileLayer\* \_tileLayer;

Рисунок 5 – Листинг заголовочного файла класса “TileRequester”

* + 1. Описание мини-отрисовщика слоя сетки параллелей и меридианов

Мини-отрисовщик слоя сетки параллелей и меридианов представлен в программе классом “LatLonGridRenderer”, который является наследником класса “BaseRenderer”. В приложении он представлен в единственном экземпляре, который находится в отрисовщике. В его состав входят:

* + указатель на слой сетки параллелей и меридианов модели;
  + метод получения изображения слоя сетки параллелей и меридианов при заданном состоянии и положении окна отрисовщика;
  + вспомогательные методы отрисовки слой сетки параллелей и меридианов.
    1. Описание мини-отрисовщика слоя пользовательских точек

Мини-отрисовщик слоя пользовательских точек представлен в программе классом “UserPointsRenderer”, который является наследником класса “BaseRenderer”. В приложении он представлен в единственном экземпляре, который находится в отрисовщике. В его состав входят:

* + указатель на слой пользовательских точек модели;
  + метод получения изображения слоя пользовательских точек при заданном состоянии и положении окна отрисовщика;
  + вспомогательные методы отрисовки слоя пользовательских точек.
  1. Описание вспомогательных библиотек

В приложении имеется маленькая рукописная геометрическая библиотека, содержащая в себе примитивы и методы работы с ними (определение характера взаимного расположения) и небольшая сторонняя графическая библиотека для преобразование изображения из PNG-формата в массив пикселей LodePNG.

Геометрическая библиотека включает в себя:

* + структуры целочисленных примитивов и примитивов с плавающей точкой (точка, линия, отрезок и прямоугольник);
  + классы исключений;
  + функции сравнения чисел с плавающей точкой;
  + функции определения валидности примитивов;
  + функции определения взаимного расположения примитивов;
  + функции определения видимости примитивов (в прямоугольнике).

1. Описание алгоритмов

В данном разделе приведены описания алгоритмов работы программы в целом и отдельных ее модулей в частности. Листинги функций и методов, реализующих данные алгоритмы, находятся в приложении В.

* 1. Алгоритм работы программы

Алгоритм работы программы реализуется функцией “wWinMain” входной точки приложения и состоит из следующей последовательности действий:

1. инициализировать мьютекс доступа к элементам массива кэша и элементам очереди загрузчика, самого загрузчика, модели и отрисовщика;
2. запустить поток загрузчика и цикла обработки сообщений отрисовщика;
3. обрабатывать сообщения, подгружать плитки и обрабатывать исключения, таким образом обеспечивая работу приложения и взаимодействие с пользователем;
4. перед завершением работы приложения, переключить флаг активности загрузчика, чтобы его поток смог завершиться.
   1. Алгоритм получения видимых отрисовщиком плиток

Получение видимых отрисовщиком плиток реализуется методом “GetVisibleTiles” класса “TileLayer” и происходит следующим образом. Отрисовщик подает на вход положение своего окна в мировом пространстве. Модель переводит эти координаты в плиточные координаты и запрашивает указатели на изображения соответствующих плиток у плиточного кэша. Формируется массив пар вида “ключ-указатель”, который затем возвращается отрисовщику.

* 1. Алгоритм получения видимых отрисовщиком линий сетки параллелей и меридианов

Получение видимых отрисовщиком линий сетки параллелей и меридианов реализуется методом “GetVisibleLatLonGrid” класса “LatLonGridLayer” и происходит следующим образом. Просматривается массив линий сетки параллелей и меридианов, определяя для каждой линии её видимость в окне отрисовщика и добавляя видимые линии в результирующий массив, который затем возвращается отрисовщику.

Таким же образом происходит и получение видимых отрисовщиком пользовательских точек.

* 1. Алгоритм формирования запроса на загрузку плитки

Формирование запроса на загрузку плитки реализуется методом “MakeTileRequest ” плиточного кэша и происходит следующим образом:

1. если все плитки заняты, произвести частичную очистку кэша;
2. если нет свободных плиток, ничего запрос не отправлять, иначе, выбрать одну из свободных плиток, указать ее в запросе на загрузку плитки и отправить запрос плиточному загрузчику;
3. если запроса на данную плитку еще не поступало, то пометить выбранную плитку занятой, иначе, оставить плитку свободной.
   1. Алгоритм частичной очистки кэша

Алгоритм частичной очистки кэша реализуется методом “CleanUp” класса “TileCache” и работает следующим образом:

1. указатели из массива указателей на занятые плитки переносятся во вспомогательный массив указателей;
2. массив указателей на занятые плитки полностью очищается;
3. вспомогательный массив указателей сортируется по возрастанию времени последнего использования плиток;
4. до тех пор, пока размер массива указателей на занятые плитки не достигнет определенного заранее заданного значения, указатели, находящиеся в конце вспомогательного массива указателей, перемещаются в массив указателей на занятые плитки.
5. если во вспомогательном массиве указателей остались указатели, то соответствующие им плитки помечаются как свободные.
   1. Алгоритм обработки запроса на загрузку плитки

Обработка плиточным загрузчиком запроса на загрузку плитки реализуется методом “ProcessRequest” класса “TileRequester” и происходит следующим образом:

1. если запросов в очереди нет, то ничего делать;
2. извлечь очередной запрос на плитку из очереди;
3. сформировать и отправить запрос к серверу на загрузку плитки;
4. обработать ответ сервера и ошибки, если таковые возникли;
5. преобразовать полученное изображение плитки в формате PNG в массив пикселей этого изображения;
6. записать данные загруженной плитки в отведенный для нее плитку в массиве плиток кэша;
7. добавить в массив указателей кэша на занятые плитки указатель на загруженную плитку.
   1. Алгоритм перерисовки изображения окна отрисовщика

Перерисовка изображения окна отрисовщика реализуется методом “Redraw” класса “MapView” и происходит следующим образом. При необходимости, происходит очистка экрана. Затем, каждый активный мини-отрисовщик возвращает изображение своего слоя, после чего эти изображения выводятся в окно отрисовщика, накладываясь друг на друга.

Необходимость очищать экран возникает в следующих случаях:

* + состояние, местоположение и (или) размеры окна были изменены таким образом, что перерисовка плиточного слоя не сможет покрыть всю область окна отрисовщика;
  + плиточный слой неактивен.

1. Тестирование

В ходе разработки приложения возникали определенные проблемы и неполадки. Для их выявления проводилось ручное тестирование программы, в ходе которого выявились следующие проблемы:

* + цвет плиток отображается некорректно;
  + плитки появляются на экране не сразу;
  + плиточный сервер не отвечал на запросы;
  + цвет параллелей и меридианов оказался не подходящим.

Часть данных проблем была решена.

**Цвет**

Приложение некорректно отображает цвет изображений. Связано это с тем, что пиксели изображений представлены в формате RGBA, а отрисовщик воспринимает их как BRGA, или наоборот – доподлинно это установить не удалось, как и разрешить данную проблему.

Загрузка плиток

Во время приближения и отдаления карты, слой плиток может отображаться не сразу, так как приложению нужно определенное время, чтобы подгрузить их изображения с плиточного сервера. Это относится скорее к особенностям приложения, но проблемная составляющая этого вопроса заключается в том, чтобы ускорить процесс загрузки/отображения данных и, тем самым, сделать пользовательский опыт более приятным. Для этого, был проведен ряд оптимизаций структур данных и алгоритмов приложения, а так же запланированы дальнейшие их улучшения.

Обращение к серверу

Забавный и поучительный случай произошел во время разработки модуля загрузки плиток. Из-за неудачного дизайнерского решения процесса запроса-загрузки плиток, очередь загрузчика постоянно была заполнена запросами и он без остановки обращался к серверу за плитками. Сервер, в свою очередь, воспринял это (вполне обосновано) за DDOS-атаку, и заблокировал IP компьютера, на котором велась разработка.

Дизайн был изменен и проблема решена. На данный момент, IP компьютера все еще заблокирован на основном сервере, поэтому, загрузка плиток производится с резервного сервера. В дальнейшем, планируется связаться с владельцами основного сервера, объяснить ситуацию, принести извинения и сделать запрос на разблокировку IP.

Цвет сетки параллелей и меридианов

В техническом задании указано, что параллели должны быть синими, меридианы красными, а от центра к краям должен быть плавный градиент к прозрачному цвету. В ходе тестирования, было принято отказаться от данного пункта, так как данный слой, по своему предназначению, не должен привлекать к себе слишком много внимания. Цвет параллелей и меридианов был изменен на темно-серый.

Обработка исключений

В приложении действует механизм обработки исключений. Благодаря ему, на этапе отладки были обнаружены несколько серьезных ошибок. После исправления данных ошибок, приложение находится в полностью рабочем состоянии [4].

Стресс-тестирование

В ходе стресс-тестирования были обнаружены и устранены мелкие баги. На текущий момент, приложение корректно работает при любом объеме и характере пользовательской активности.

1. Пользовательский интерфейс

Главный экран приложения представляет собой окно, в которое выводятся различные картографические элементы (Рисунок 3).

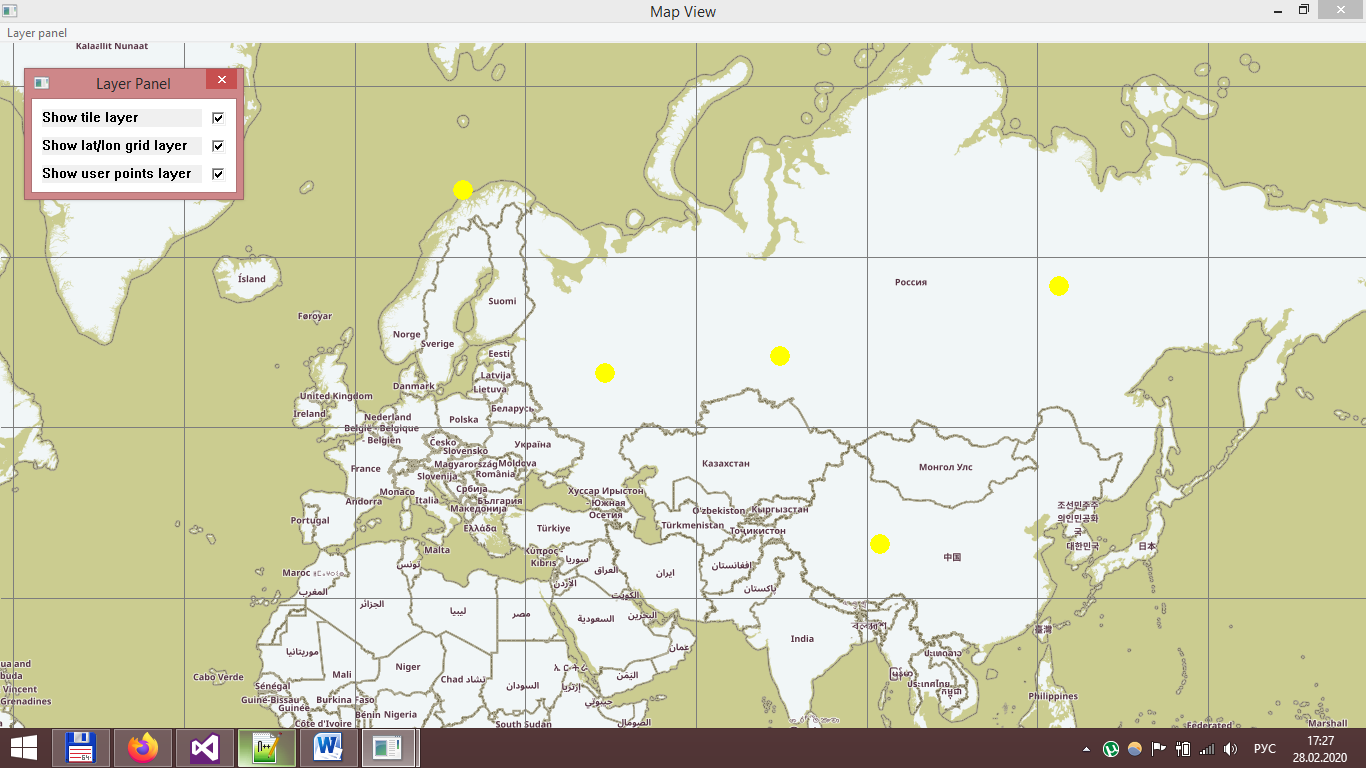


Рисунок 6 – Главный экран приложения

Взаимодействие с картой осуществляется при помощи мыши. Для того, чтобы перемещаться по карте вдоль сторон света, необходимо “перетаскивать” ее в соответствующем направлении. Чтобы приближать и отдалять карту, необходимо прокручивать колесико мыши, от себя и к себе соответственно. Чтобы отметить пользовательскую точку, необходимо дважды щелкнуть левой кнопкой мыши в нужное место.

Чтобы вызвать меню включения/отключения слоев, необходимо кликнуть по соответствующему пункту меню. Чтобы включить/отключить слой, необходимо установить/снять галочку напротив соответствующего пункта.

Заключение

В ходе практики, были выполнены задачи и достигнуты цели практики. Были получены знания, умения и навыки в области разработки ПО, изучена технология Windows API и приобретены навыки разработки картографических приложений и геоинформационных систем. Было разработано десктопное картографическое приложение и были применены на практике навыки разработки приложений на базе Windows API. Были приобретены новые знания, умения и навыки программирования и работы в коллективе.

Данное приложение, в виду тематики, имеет огромный потенциал расширения и дополнения, который наверняка будет реализовываться при выполнении последующих творческих заданий и при написании дипломной работы.

Сравнивать данное учебное приложение с реальными аналогами не имеет смыла, так как они на много порядков превосходят его в сложности, возможностях и качестве реализации.

Список использованных источников

1. John Torjo. Boost.Asio C++ Network Programming / Изд. Packt Publishing, 2013 – 142 c.
2. Microsoft. Get Started with Win32 and C++ / <https://docs.microsoft.com/en-us/windows/win32/learnwin32/learn-to-program-for-windows>, 2018
3. Brook Miles. theForger’s Win32 API Programming Tutorial / <http://winprog.org/tutorial/>, 1998-2019
4. Bjerne Stroustrup. The C++ Programming Language, Fourth Edition / Изд. Addison-Wesley, 2013 – 1346 c.

Приложение А

Схема подключения заголовочных файлов

Схема подключения заголовочных файлов в исходном коде приложения выглядит следующим образом:

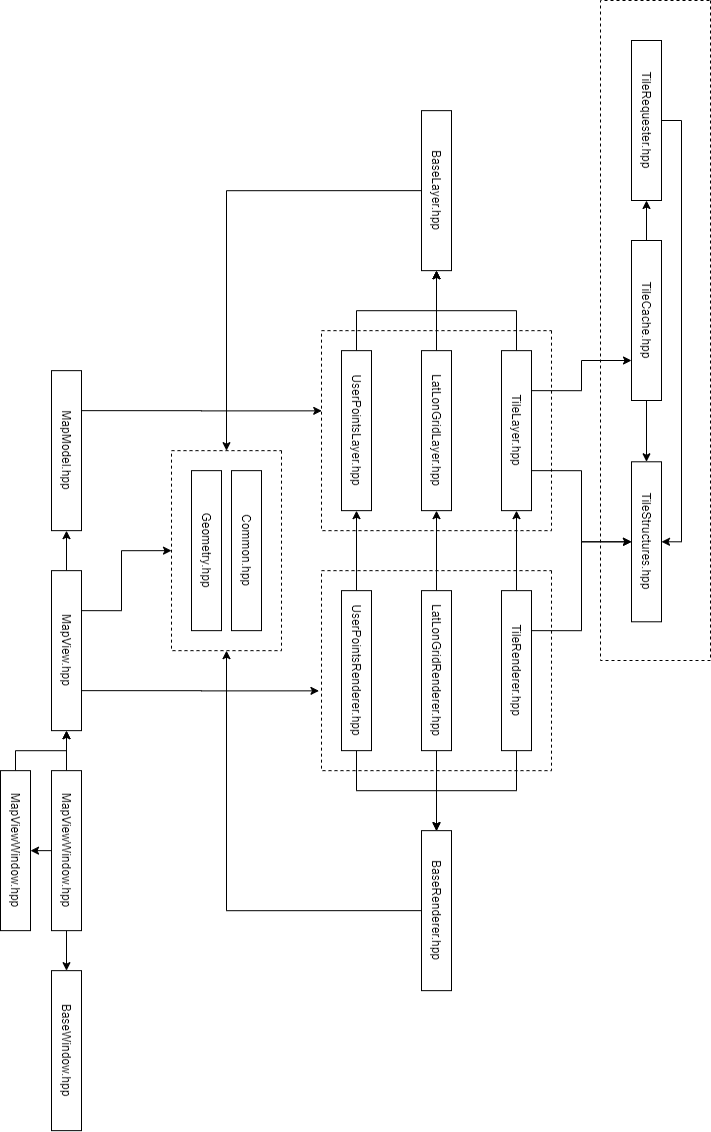


Рисунок А.1 – Схема подключения заголовочных файлов

Приложение Б

СТруктура файловой директории

Структура файловой директории исходных кодов приложения выглядит следующим образом:

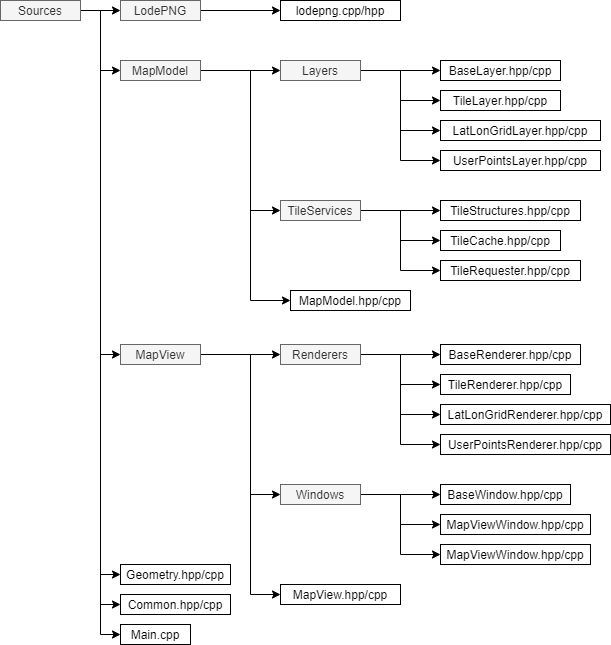


Рисунок Б.1 – Структура файловой директории исходных кодов

Приложение В

листинги алгоритмов

В данном приложении приведены листинги функций и методов, реализующих алгоритмы, описанные в разделе “Описание алгоритмов”.

**В.1 Листинг алгоритма работы программы**

Работа программы происходит в функции “wWinMain”, которая находится во входной точке приложения. Ее листинг выглядит следующим образом:

int WINAPI wWinMain(

HINSTANCE hInstance,

HINSTANCE,

PWSTR pCmsLine,

int nCmdShow

)

{

// Initialize map model.

std::mutex tileServicesMutex;

TileServices::TileRequester tileRequester(tileServicesMutex);

MapControl::MapModel mapModel(tileRequester, tileServicesMutex);

// Initialize map view/controller.

MapViewWindow mapViewWindow(mapModel);

if (!mapViewWindow.Create(

L"Map View Window", WS\_OVERLAPPEDWINDOW,

mvDefW, mvDefH,

0, 0

))

{

return 0;

}

mapViewWindow.GiveHwndToMapView();

ShowWindow(mapViewWindow.Window(), nCmdShow);

// Initialize tile requesting thread.

auto requestingProcess = [&tileRequester, &tileServicesMutex]()

{

while (tileRequester.IsRequesting())

{

tileRequester.ProcessTileRequest();

std::this\_thread::yield();

}

};

auto requestingThread = std::async(requestingProcess);

// Initialize window message loop.

MSG msg = {};

while (GetMessage(&msg, NULL, 0, 0))

{

if (msg.message == WM\_KEYDOWN)

{

mapViewWindow.HandleKeyDown(msg);

}

TranslateMessage(&msg);

DispatchMessage(&msg);

std::this\_thread::yield();

}

tileRequester.StopRequesting();

return 0;

}

**В.2 Листинг алгоритма получения видимых отрисовщиком плиток**

Получение видимых отрисовщиком плиток происходит в методе “GetVisibleTiles”, который находится в плиточном слое. Его листинг выглядит следующим образом:

std::vector<TilePointer> TileLayer::GetVisibleTiles(RectWM wmWindow, unsigned lod, Status& status) noexcept

{

// Set up x/y limits.

static const int minTileXY = 0;

int maxTileXY = SafePow(2, lod) - 1;

// Tile w/h in degrees.

double tileWDgs = wmFullWDgs / static\_cast<double>(

maxTileXY + 1

);

double tileHDgs = wmFullHDgs / static\_cast<double>(

maxTileXY + 1

);

// Calculate x-y boundaries of visible tiles.

unsigned leftBorderXTls = static\_cast<unsigned>(

max(static\_cast<int>(

(wmWindow.origin.lon - wmMinLon) / tileWDgs), minTileXY)

);

unsigned rightBorderXTls = static\_cast<unsigned>(

min(static\_cast<int>(

(wmWindow.extrem.lon - wmMinLon) / tileWDgs), maxTileXY)

);

unsigned upBorderYTls = static\_cast<unsigned>(

max(static\_cast<int>(

(wmWindow.origin.lat - wmMinLat) / tileHDgs), minTileXY)

);

unsigned downBorderYTls = static\_cast<unsigned>(

min(static\_cast<int>(

(wmWindow.extrem.lat - wmMinLat) / tileHDgs), maxTileXY)

);

// Grab visible tiles.

std::vector<TilePointer> visibleTiles;

status = Status::Clean;

std::unique\_lock<std::mutex> tileServicesLock(\_tileServicesMutex);

for (unsigned x = leftBorderXTls; x <= rightBorderXTls; ++x)

{

for (unsigned y = upBorderYTls; y <= downBorderYTls; ++y)

{

TileKey visibleTileKey = TileKey{ lod, x, y };

std::vector<unsigned char> const\* visibleTileImage = \_tileCache.GetTileImage(visibleTileKey);

if (visibleTileImage)

{

visibleTiles.push\_back(TilePointer{ visibleTileKey, visibleTileImage } );

}

else

{

visibleTiles.push\_back(TilePointer{ visibleTileKey, &\_tilePlaceholder });

status = Status::Dirty;

}

}

}

return visibleTiles;

**В.3 Листинг алгоритма получения видимых отрисовщиком линий сетки параллелей и меридианов Код маленький, может перенести его в текст отчета?**

Получение видимых отрисовщиком линий сетки параллелей и меридианов происходит в методе “GetVisibleLatLonGrid”, который находится в слое сетки параллелей и меридианов. Его листинг выглядит следующим образом:

std::vector<SectWM> LatLonGridLayer::GetVisibleLatLonGrid(

RectWM wmWindow

) noexcept

{

std::vector<SectWM> visibleLatLonGrid;

for (SectWM& latLonGridSection : \_latLonGrid)

{

if (IsVisible(latLonGridSection, wmWindow))

{

visibleLatLonGrid.push\_back(latLonGridSection);

}

}

return visibleLatLonGrid;

}

**В.4 Листинг алгоритма формирования запроса на загрузку плитки**

Формирование запроса на загрузку плитки происходит в методе “MakeTileRequest” плиточного кэша. Его листинг выглядит следующим образом:

void TileCache::\_MakeTileRequest(const TileKey& tileKey) noexcept

{

if (\_cleanSlots.size() == SlotsQuantity)

{

\_CleanUp();

}

if (\_dirtySlots.empty())

{

return;

}

TileSlot\* pTileSlot = \_dirtySlots.front();

TileRequest tileRequest{ tileKey, pTileSlot };

PushStatus pushStatus = \_requester.PushTileRequest(tileRequest);

if (pushStatus == PushStatus::Pushed)

{

\_dirtySlots.pop();

pTileSlot->status = TileStatus::InProgress;

}

}

**В.5 Листинг алгоритма частичной очистки кэша**

Частичная очистка кэша происходит в методе “CleanUp”, который находится в плиточном кэше. Его листинг выглядит следующим образом:

void TileCache::\_CleanUp() noexcept

{

std::vector<TileSlot\*> cleanSlots = MapToVector(\_cleanSlots);

\_cleanSlots.clear();

std::sort(cleanSlots.begin(), cleanSlots.end(), &SlotSort);

while (\_cleanSlots.size() < CleanUpRest && !cleanSlots.empty())

{

TileSlot\* tileSlot = cleanSlots.back();

cleanSlots.pop\_back();

\_cleanSlots.insert(std::make\_pair(tileSlot->key, tileSlot));

}

if (cleanSlots.empty())

{

return;

}

for (TileSlot\* tileSlot : cleanSlots)

{

tileSlot->status = TileStatus::Dirty;

\_dirtySlots.push(tileSlot);

}

}

**В.6 Листинг алгоритма обработки запроса на загрузку плитки**

Обработка запроса на загрузку плитки происходит в методе “ProcessRequest”, который находится в плиточном загрузчике. Его листинг выглядит следующим образом:

void TileRequester::ProcessTileRequest()

{

std::unique\_lock<std::mutex> tileServicesLock(\_tileServicesMutex);

if (!\_serverSocket.is\_open())

{

throw SocketNotOpened();

}

if (\_requests.empty())

{

return;

}

TileRequest request = \_requests.top();

\_requests.pop();

\_requestsKeys.erase(request.key);

boost::asio::streambuf responseBuf;

std::stringstream imageBuf;

// Get tile image from server.

\_RefreshConnection();

\_MakeRequest(TileKeyToRequestString(request.key));

\_ProcessResponseStatus(responseBuf);

\_ProcessResponseHeaders(responseBuf, imageBuf);

\_ReadResponseData(responseBuf, imageBuf);

std::vector<unsigned char> tileImage = StringToVector(imageBuf.str());

// Update slot.

request.slot\_ptr->image = DecodePng(tileImage);

request.slot\_ptr->key = request.key;

request.slot\_ptr->status = TileStatus::Clean;

request.slot\_ptr->timestamp = std::chrono::milliseconds(0);

\_pCleanSlots->insert(std::make\_pair(request.key, request.slot\_ptr));

}

**В.7 Листинг алгоритма перерисовки изображения окна отрисовщика**

Перерисовка изображения окна отрисовщика происходит в методе “Redraw”, который находится в отрисовщике. Его листинг выглядит следующим образом:

void MapView::Redraw()

{

PAINTSTRUCT wPs;

HDC wHdc = BeginPaint(\_hWnd, &wPs);

if (\_necessaryToClearScreen)

{

\_ClearScreen(wHdc, wPs);

\_necessaryToClearScreen = false;

}

// Draw layers.

if (\_tileRenderer.GetActivity())

{

HBITMAP tileLayerBitmap = \_tileRenderer.GetTileLayerBitmap(\_transparentBitmapPixels);

\_DrawLayerBitmap(wHdc, tileLayerBitmap);

DeleteObject(tileLayerBitmap);

}

else

{

// When tile layer enabled, screen clearing isn't necessary.

\_ClearScreen(wHdc, wPs);

}

if (\_latLonGridRenderer.GetActivity())

{

HBITMAP hLlgLayerBitmap = \_latLonGridRenderer.GetLatLonGridLayerBitmap(\_transparentBitmapPixels);

\_DrawLayerBitmap(wHdc, hLlgLayerBitmap);

DeleteObject(hLlgLayerBitmap);

}

if (\_userPointsRenderer.GetActivity())

{

HBITMAP hUpLayerBitmap = \_userPointsRenderer.GetUserPointsLayerBitmap(\_transparentBitmapPixels);

\_DrawLayerBitmap(wHdc, hUpLayerBitmap);

DeleteObject(hUpLayerBitmap);

}

EndPaint(\_hWnd, &wPs);

}