Министерство цифрового развития, связи и массовых коммуникаций Российской Федерации **ФГБОУ ВО «СибГУТИ»**

 Кафедра **ТСиВС**

**Курсовая работа по дисциплине:**

«Визуальное программирование и человеко-машинное взаимодействие»

на тему: «Тепловая карта распространения радиосигнала»

**Выполнил**: Студент группы ИА-131 Нагорный С.Д.

**Проверил**: Ст. преп. каф. ТСиВС

Ахпашев Р.В.

**Оценка**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**Работа зачтена**: «\_\_\_» \_\_\_\_ 2023г.

г. Новосибирск 2023

**Содержание**

Задание.......................................................................................................3

Краткая теория по курсу............................................................................4

Создание сцены “QGraphicsScene”..........................................................5

Установка размеры карт и постройка стен..............................................7

Создание 12-ти базовых станций.............................................................8

Вычисление силы сигнала........................................................................9

Определение цвета и прозрачности.......................................................10

Создание графических элементов..........................................................12

Вывод........................................................................................................14

Список литературы..................................................................................15

**Задание**

Задача на данную курсовую работу состоит в создании программы для вывода окна приложения при помощи класса “QMainWindow”. Создать пиксельную карту 1000х1000. Случайным образом установить Точку доступа (базовую станцию) 5G New Radio. Отрисовать тепловую карту (закрасить каждый пиксель определенным цветом) для модели распространения радиосигнала из спецификации 3GPP TR 38.901, пункт 7.4. Несущая частота (fc) может быть в пределах от 2.1[GHz] до 6[GHz]. Также нужно было создать 12 базовых станций в офисе с выводом на экран их тепловых карт.

**Краткая теория по курсу**

Разработка ведется в среде программирования QT Creator.

Qt — это кроссплатформенный фреймворк для создания графических интерфейсов пользователя GUI. Помимо модулей для создания GUI, Qt содержит множество модулей для разработки прикладного программного обеспечения. Фреймворк разработан преимущественно на языке программирования C, некоторые компоненты используют QML и JavaScript.

1. Тепловые карты: тепловая карта (или тепловая диаграмма) - это графическое представление данных, в котором цветовая шкала используется для отображения значения каждой точки данных. Таким образом, тепловая карта позволяет визуализировать пространственное распределение значений данных.

2. Базовые станции: базовая станция - это устройство, используемое для передачи и приема сигнала в беспроводной связи. Она обычно устанавливается на высоте и используется для усиления сигнала мобильных устройств.

3. Визуализация данных: для создания тепловых карт в Qt Creator потребуется знание технологий визуализации данных, таких как использование графических элементов, цветовых градиентов и интерполяции для правильного отображения данных.

4. Использование Qt Creator: для разработки тепловых карт базовых станций потребуется знание инструментов и функциональности Qt Creator, таких как использование графического интерфейса для создания пользовательского интерфейса, работа с графикой и обработка данных.

5. Алгоритм Брезенхе́ма — это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками. Это один из старейших алгоритмов в машинной графике — он был разработан Джеком Элтоном Брезенхэмом в компании IBM в 1962 году. Алгоритм широко используется, в частности, для рисования линий на экране компьютера.

**Этапы разработки приложения**

**Этап 1 (Создание сцены “QGraphicsScene”)**

Создаем объект QGraphicsScene\* scene, который представляет собой область, на которой можно размещать графические элементы. Затем создаем объект QGraphicsView\* view, который отображает содержимое QGraphicsScene.

Далее мы устанавливаем параметры для отображения такие как:

* **setRenderHint** - сглаживание краев при рисовании.
* **setViewportUpdateMode** - устанавливает режим обновления виджета QGraphicsView, когда все содержимое QGraphicsScene перерисовывается при каждом обновлении.
* **setOptimizationFlag** - устанавливает флаг оптимизации, который указывает, что QGraphicsView не должен корректировать размеры графических элементов при использовании сглаживания краев.
* **setRenderHint** - устанавливает режим сглаживания краев при рисовании графических элементов в QGraphicsView.

QGraphicsScene\* scene = new QGraphicsScene();

QGraphicsView\* view = new QGraphicsView(scene, this);

view->setRenderHint(QPainter::Antialiasing);

view->setDragMode(QGraphicsView::ScrollHandDrag);

view->setViewportUpdateMode(QGraphicsView::FullViewportUpdate);

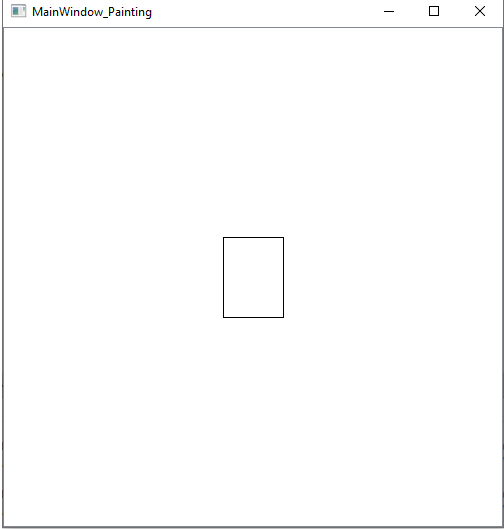
view->setOptimizationFlag(QGraphicsView::DontAdjustForAntialiasing, true);

view->setViewportUpdateMode(QGraphicsView::SmartViewportUpdate);

view->setRenderHint(QPainter::Antialiasing, true);

view->setRenderHint(QPainter::SmoothPixmapTransform, true);

setCentralWid

**Этап 2 (Установка размеры карт и постройка стен)**

Задаем размеры карты 1400х700. Далее начинаем рисовать офисную комнату, где будут располагаться наши базовые станции. Создаем двумерный вектор “walls” , который представляет стены на карте. Каждый элемент в “walls” является булевым значением, где “true” обозначает наличие стены, а “false” - отсутствие.

Циклы for заполняют границы карты (первый и последний столбцы и строки) значениями true, показатель конца карты. Также заполняют внутренние границы карты значениями true, чтобы создать офисную комнату.

int mapWidth = 1400;

int mapHeight = 700;

std::vector<std::vector<bool>>walls(mapWidth,std::vector<bool>(mapHeight, false));

for(int x = 0; x < mapWidth; x++){

walls[x][0] = true;

walls[x][mapHeight-1] = true;

}

for(int y = 0; y < mapHeight; y++){

walls[0][y] = true;

walls[mapWidth-1][y] = true;

}

for(int x = 98; x < 1302; x++){

walls[x][100] = true;

walls[x][600] = true;

}

for(int y = 100; y < 600; y++){

walls[98][y] = true;

walls[1302][y] = true;}



**Этап 3 (Создание 12-ти базовых станций)**

Данная часть используется для создания 12-и базовых станций. Переменные **cellPosX** и **cellPosY** используются для определения позиции каждой базовой станции на карте.

std::vector<std::vector<float>>signalStrengths(mapWidth,std::vector<float>(mapHeight, 0.0));

for(int i = 0; i < 12; i++){

int cellPosX, cellPosY;

if(i < 2) {

if(i == 0){

cellPosX = 200;

cellPosY = 250;

}

else if(i == 1){

cellPosX = 400;

cellPosY = 250;

}

}

else if(i < 4) {

if(i == 2){

cellPosX = 200;

cellPosY = mapHeight - 250;

}

else if(i == 3){

cellPosX = 400;

cellPosY = mapHeight - 250;

}

}

**Этап 4 (Вычисление силы сигнала)**

Этот код используется для вычисления силы сигнала в каждой точке на карте, исходя из расположения базовой станции и расстояния до нее.

Первый цикл выполняется для переменной x от 0 до mapWidth. Второй цикл выполняется для переменной y от 0 до mapHeight. Таким образом, мы перебираем каждую точку на карте.

Внутри циклов мы вычисляем расстояние между текущей базовой станцией (заданной переменными cellPosX и cellPosY) и текущей точкой на карте (заданной переменными x и y). Для этого мы используем формулу расстояния между двумя точками в двумерном пространстве:

float distance = sqrt(pow(cellPosX-x, 2) + pow(cellPosY-y, 2));

Затем мы вычисляем силу сигнала на основе расстояния, используя формулу:

float signalStrength = 1.0/(1.0 + pow(distance/50.0, 2));

Здесь мы используем обратную функцию квадратного корня, чтобы получить значение, которое уменьшается с увеличением расстояния между базовой станцией и точкой на карте. Мы также используем значение 50.0 в знаменателе, чтобы определить, как быстро уменьшается сила сигнала с увеличением расстояния.

В результате выполнения этого кода мы получаем массив signalStrengths, который содержит значения силы сигнала в каждой точке на карте, исходя из расположения базовой станции и расстояния до нее.

for(int x = 0; x < mapWidth; x++){

for(int y = 0; y < mapHeight; y++){

float distance = sqrt(pow(cellPosX-x, 2) + pow(cellPosY-y, 2));

float signalStrength = 1.0/(1.0 + pow(distance/50.0, 2));

signalStrengths[x][y] += signalStrength;

}

}

**Этап 5 (Определение цвета и прозрачности)**

Сначала мы первым циклом перебираем каждую точку на карте, как в предыдущей части кода.

Внутри циклов мы получаем значение силы сигнала для текущей точки на карте из массива signalStrengths:

float signalStrength = signalStrengths[x][y];

Затем мы определяем цвет на основе силы сигнала, используя функцию fromHsvF класса QColor. Здесь мы используем значение 0.6 - signalStrength\*0.6 в качестве значения оттенка, чтобы получить цвет, который становится темнее с увеличением силы сигнала:

QColor color = QColor::fromHsvF(0.6 - signalStrength\*0.6, 1.0, 1.0);

Затем мы определяем прозрачность на основе силы сигнала. Здесь мы используем значение 255 в качестве максимального значения прозрачности и значение 75 в качестве максимального значения, чтобы ограничить прозрачность для более сильных сигналов:

int alpha = std::min(75, static\_cast<int>(signalStrength\*255));

Затем мы создаем прямоугольник для текущей точки на карте, используя класс QGraphicsRectItem и задавая координаты и размеры прямоугольника:

QGraphicsRectItem\* rect = new QGraphicsRectItem(x, y, 1, 1);

Затем мы устанавливаем цвет и прозрачность прямоугольника, используя методы setBrush и setOpacity:

rect->setBrush(QBrush(color));

rect->setOpacity(alpha/255.0);

В результате выполнения этого кода мы получаем карту с точками, которые имеют цвет и прозрачность, отображающие силу сигнала в каждой точке на карте.

for(int x = 0; x < mapWidth; x++){

for(int y = 0; y < mapHeight; y++){

float signalStrength = signalStrengths[x][y];

QColor color = QColor::fromHsvF(0.6 - signalStrength\*0.6, 1.0, 1.0);

int alpha = std::min(75, static\_cast<int>(signalStrength\*255));

QGraphicsRectItem\* rect = new QGraphicsRectItem(x, y, 1, 1);

rect->setBrush(QBrush(color));

rect->setOpacity(alpha/255.0);

scene->addItem(rect);

}

}

**Вывод**:



**Этап 6 (Создание графических элементов)**

По окончании кода будут отображены все графические элементы.

Первый цикл создает стены на сцене, пробегая каждую клетку карты. Если в клетке есть стена (walls[x][y] == true), создается прямоугольный элемент QGraphicsRectItem размером 1x1 пиксель в координатах (x,y) с черной кистью, который добавляется на сцену.

Второй цикл создает базовые станции на сцене. Он просматривает каждую базовую станцию в списке baseStations и создает элемент QGraphicsEllipseItem, представляющий круг с центром в координатах (bs->x, bs->y) и радиусом bs->radius. Элементу устанавливается красная кисть и добавляется на сцену.

Затем происходит вычисление силы сигнала в каждой точке на карте. Создается двумерный вектор signalStrengths, хранящий значения силы сигнала в каждой точке. Внешний цикл проходит по всем x-координатам, а внутренний цикл - по всем y-координатам. Для каждой точки вычисляется суммарная сила сигнала от всех базовых станций. Внутренний цикл проходит по всем базовым станциям в списке baseStations и вычисляет расстояние между текущей точкой и базовой станцией. Затем вычисляется сила сигнала от этой базовой станции по формуле signalStrength += bs->power/(distance\*distance), где bs->power - мощность базовой станции, distance - расстояние между точкой и базовой станцией. Результат записывается в соответствующую ячейку вектора signalStrengths.

После этого происходит отображение карты силы сигнала на сцене. Внешний цикл проходит по всем x-координатам, а внутренний цикл - по всем y-координатам. Для каждой точки вычисляется значение силы сигнала из вектора signalStrengths. Затем вычисляется цвет для этой точки на основе значения силы сигнала. Цвет выбирается из палитры оттенков синего, начиная с насыщенного синего цвета (hue=0) и заканчивая бледным голубым цветом (hue=0.6). Чем больше значение силы сигнала, тем ближе цвет к насыщенному синему. Затем создается элемент QGraphicsRectItem размером 1x1 пиксель в координатах (x,y) с установленной кистью с выбранным цветом и прозрачностью, соответствующей значению силы сигнала, который добавляется на сцену.

Наконец, устанавливаются размеры сцены и показывается вид.

for(int x = 0; x < mapWidth; x++){

for(int y = 0; y < mapHeight; y++){

if(walls[x][y]){

QGraphicsRectItem\* wall = new QGraphicsRectItem(x, y, 1, 1);

wall->setBrush(QBrush(Qt::black));

scene->addItem(wall);

}

}

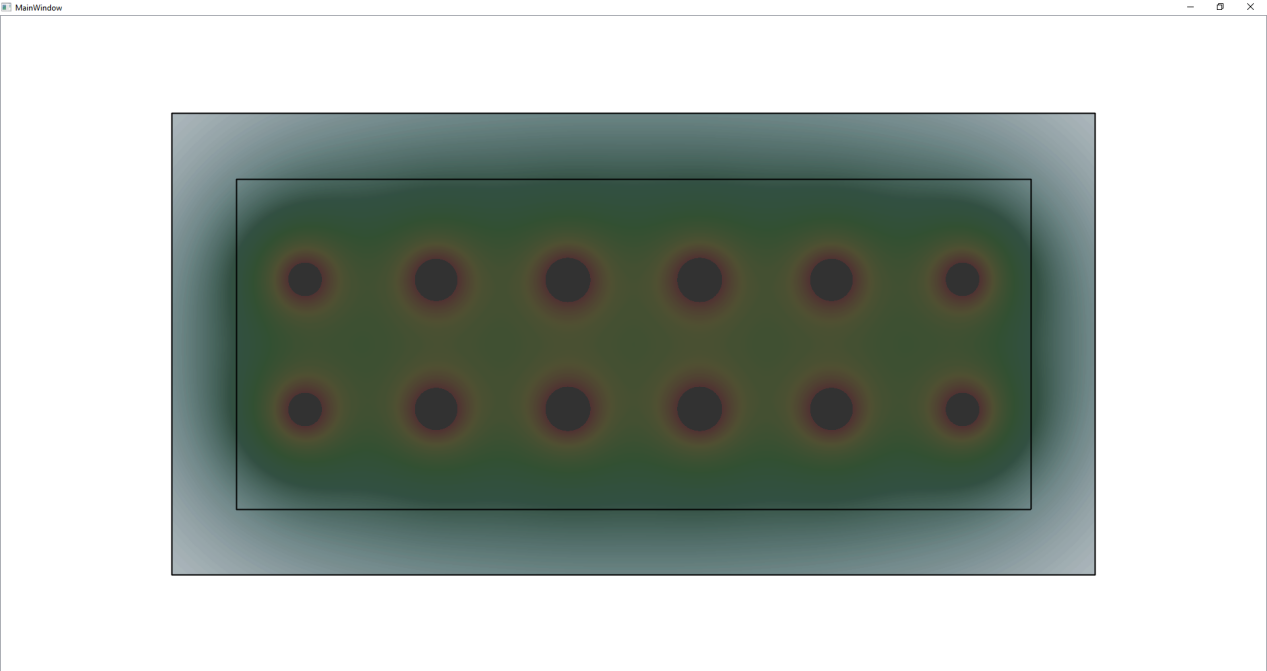
}

view->setSceneRect(0, 0, mapWidth, mapHeight);

view->show();

}

Итоговый результат наших действий:



**Вывод**:

В ходе выполнения данной работы была разработана программа для создания карты силы сигнала между 12-ю базовыми станциями и точками на карте. Были использованы различные математические формулы для вычисления расстояния и силы сигнала, а также библиотека Qt для создания графического интерфейса.

Данная программа может быть полезна в области телекоммуникаций для определения качества связи между базовыми станциями и точками на карте, а также для создания тепловых карт для анализа качества связи на всей территории.

В целом, выполнение данной работы позволило получить практические навыки в области программирования и математики, а также расширить свой кругозор в области телекоммуникаций.

**GitHub:**

****

**Список литературы**

1. 3rd Generation Partnership Project; Technical Specification Group Radio Access Network; Study on channel model for frequencies from 0.5 to 100 GHz
2. Компьютерная графика. Рисование Qt (объект QPainter). Графики функций С++ \\ [\*ссылка\*](https://dzen.ru/a/X_2WN_I4Wj37HZJN)
3. Документация QtCreator \\ [\*ссылка\*](https://doc.qt.io/qt-5/qvector.html)
4. Реализация алгоритма Брезенхэма. [Online]. Доступно: [https://ru.wikibooks.org/wiki/Реализации\_алгоритмов/ Алгоритм\_Брезенхэма#Для C++](https://ru.wikibooks.org/wiki/)
5. <https://www.3gpp.org/ftp/Specs/archive/38_series/38.901/38901-h00.zip> - скачивание архива