МИНОБРНАУКИ РОССИИ
федеральное государственное автономное образовательное учреждение
высшего образования
«Санкт-Петербургский политехнический университет Петра Великого»

Институт дополнительного образования

Высшая инженерная школа

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАЗРАБОТКА ПРОТОТИПА ПРОГРАММЫ ДЛЯ ПЛАНИРОВАНИЯ САДОВОГО УЧАСТКА

По программе профессиональной переподготовки:
«Разработчик прикладного программного обеспечения (Языки C и C++)»

Выполнил:
Захаров Игорь Сергеевич
Подпись_____

Руководитель:
ст. педагог, Абрамова Марина Геннадьевна
Подпись_____

Санкт-Петербург 2020

СОДЕРЖАНИЕ

Термины и определения

- Красные линии - линии, которые обозначают границы территорий общего пользования и подлежат установлению, изменению или отмене в документации по планировке территории.
- Интерактивные объекты – объекты разрабатываемого приложения, для которых определена реакция на действия пользователя и определена реакция при взаимодействии с другими объектами.
- Drag and drop – способ перемещения объектов путем «захвата» объекта с помощью мыши, переноса на нужное место и «бросания». [9]

ВВЕДЕНИЕ

При обустройстве своего садового участка важно знать, как правильно расположить постройки, чтобы избежать их переноса или сноса по решению суда. Для всех объектов, расположенных на участке, нормируются допустимые расстояния: до других сооружений, до границы участка, до красных линий улиц и проездов и т.д. Проблема расположения построек с учетом действующих нормативов особенно актуальна для владельцев небольших участков размером 0,04-0,06 га.

Целью данной работы является разработка прототипа программы для планирования расположения на садовом участке основных построек, с учетом допустимых расстояний между ними.

Первый раздел работы посвящен введению в предметную область. В нем рассмотрены нормы и правила застройки садовых участков, приведен обзор существующих решений для планирования садовых участков.

Во втором разделе формируются задачи, которые должны решаться с помощью разрабатываемого приложения и задаются граничные условия.

Третий раздел посвящен описанию архитектуры разрабатываемого приложения.

В четвертом разделе подробно описывается реализация интерфейса приложения, реализация основных модулей приложения и организация их взаимодействия, приводятся методы решения поставленных задач.

В пятом разделе описаны методы тестирования приложения.

1. ПЛАНИРОВАНИЕ ЗАСТРОЙКИ САДОВЫХ УЧАСТКОВ

1.1 Нормативная документация

При планировании застройки садового участка следует руководствоваться сводом правил СП 53.13330.2019 «ПЛАНИРОВКА И ЗАСТРОЙКА ТЕРРИТОРИИ ВЕДЕНИЯ ГРАЖДАНАМИ САДОВОДСТВА. ЗДАНИЯ И СООРУЖЕНИЯ (СНИП 30-02-97)». Данный документ на федеральном уровне устанавливает требования к общей планировке застройки территории садоводства, требования к застройке каждого участка, требования к устройству инженерных коммуникаций. Следует отметить, что уточненные нормы планирования застройки могут быть установлены на региональном уровне и могут носить, как обязательный, так и рекомендательный характер. [1]

Свод правил СП 53.13330.2019 устанавливает минимальные расстояния до красных линий улиц и проездов, до ограждения со стороны улиц (красные линии не всегда совпадают с границами участка), до границ соседнего участка, до объектов, расположенных на соседнем участке, а также минимальные расстояния между некоторыми объектами, расположенными на участке. Минимальные расстояния согласно п.6 СП 53.13330.2019 сведены в таблицу 1.1

При определении планировании расположения жилого дома на садовом участке следует руководствоваться СП 4.13130.2013 «Системы противопожарной защиты. ОГРАНИЧЕНИЕ РАСПРОСТРАНЕНИЯ ПОЖАРА НА ОБЪЕКТАХ ЗАЩИТЫ. Требования к объемно-планировочным и конструктивным решениям», который устанавливает минимальные допустимые расстояния между жилыми зданиями, в зависимости от их типа.

Таблица 1.1

Минимальные допустимые расстояния между объектами на садовом участке

Объекты	Минимальные расстояния, м.		
	Красная линия улицы	Красная линия проезда	Граница соседнего участка
Жилое строение(дом)	5	3	3
Помещение для содержания скота, птицы	5	5	4
Другие постройки (сарай, баня, теплица)	5	5	1
Стволы высокорослых деревьев (Высота кроны > 15 метров)	-	-	3
Стволы среднерослых деревьев (Высота кроны < 15 метров)	-	-	2
Кустарник	-	-	1
-	Ограждение участка со стороны улиц		
Компостное устройство	2		
Надворная уборная	2		
-	Жилой дом		
Отдельно стоящая баня	8		
Надворная уборная	8		
-	Колодец		
Компостное устройство	8		
Надворная уборная	8		

1.2 Приложения для планирования садового участка

Существует более 20 приложений, позволяющих составить план садового участка, разработать ландшафтный дизайн и составить смету проекта, не считая программных комплексов, предназначенных для общего проектирования. Для примера хотелось бы рассмотреть три из них:

- Garden Planner;

Возможности: предназначена для двухмерного проектирования ландшафтного дизайна садового участка. Содержит более 1000 декоративных и функциональных объектов для размещения на плане.

- X-Designer;

Возможности: позволяет составить план садового участка в 3D.

- Наш Сад 9.0 Рубин;

Возможности: имеет обширную базу готовых объектов, позволяет составлять двухмерные планы садовых участков, есть возможность отображения плана в 3D, автоматически составляются сметы.

Вышеперечисленное программное обеспечение позволяет составлять подробные планы садовых участков, разрабатывать ландшафтный дизайн, но учет минимальных допустимых расстояний между объектами пользователь должен производить самостоятельно, опираясь на нормативные документы.

Разрабатываемое приложение предназначено для первичного определения местоположения базовых объектов и позволяет визуально оценить их расстановку на двухмерном плане садового участка, а также отображает минимальные допустимые расстояния между объектами, согласно нормативной документации в процессе планирования участка. Для дальнейшей проработки плана садового участка следует воспользоваться одним из приложений для разработки ландшафтного дизайна или одной из систем автоматизированного проектирования и черчения.

2. ПОСТАНОВКА ЗАДАЧ

2.1 Граничные условия

СП 53.13330.2019 устанавливает нормы застройки не только отдельных участков, но и всего садоводства в целом. Для определения допустимых расстояний между объектами, при разработке прототипа программы планирования участка, следует руководствоваться п.6 СП 53.13330.2019. Противопожарные расстояния согласно таблице 1 СП 4.13130.2013 не учитываются. [1]

Кроме того, для разрабатываемого прототипа приложения установлены следующие ограничения:

1. Форма участка – прямоугольник или квадрат.

2. Размеры и расположение участка и соседних объектов задаются при создании нового проекта и не изменяются в дальнейшем.

3. Площадь жилой застройки не ограничивается

4. Указания допустимых расстояний между объектами носят рекомендательный характер в рамках данной программы.

2.2 Задачи

Для достижения поставленной цели, в процессе разработки приложения решаются следующие задачи:

1. Разработка архитектуры приложения

2. Разработка интерфейса приложения.

3. Разработка концепции динамического отображения допустимых расстояний между объектами.

4. Разработка классов и функций приложения, в соответствии с функциональными требованиями.

5. Проверка работоспособности приложения.

Функциональные требования:

- Возможность задания размеров участка и типов соседних объектов.
- Возможность размещения объектов на плане с помощью метода drag and drop.
- Возможность визуального отображения минимально допустимых расстояний между объектами.
- Возможность перемещения и вращения объектов.
- Возможность масштабирования плана садового участка.

3. ПРОЕКТИРОВАНИЕ

Для разработки прототипа приложения использовалась среда разработки программного обеспечения Qt. Язык программирования - C++. [2]

При рассмотрении возможных способов разработки приложения, выбор был сделан в пользу Qt Graphics View Framework (далее – «графическое представление»). Графическое представление основано на концепции «модель-представление» и предназначено для взаимодействия и управления большим количеством элементов, представляющих двумерные изображения. [3]

Возможности графического представления:

- Перемещение элементов.
- Изменение размера элементов.
- Вращение элементов.
- Обнаружение коллизий между элементами.

3.1 Описание архитектуры

На рисунке 3.1 представлена структурная схема разрабатываемого приложения.

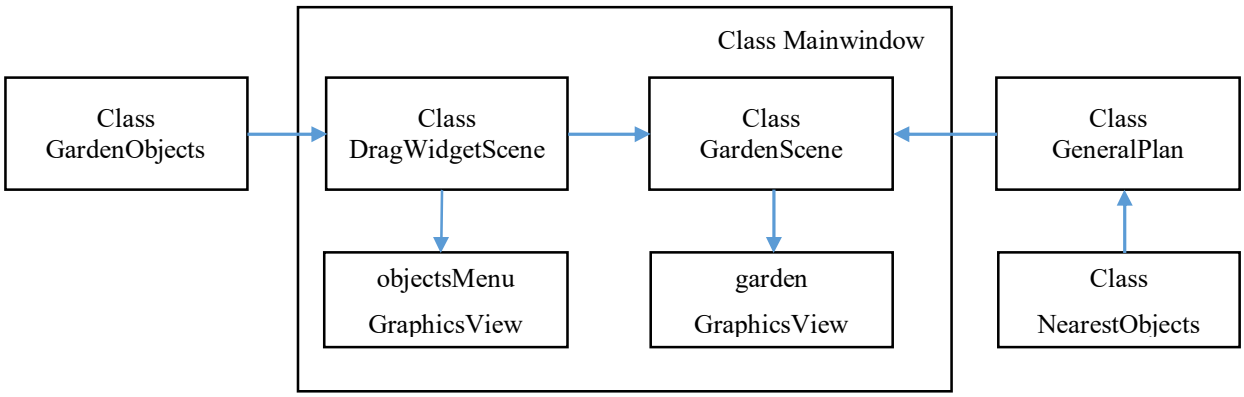

Рисунок 3.1 – Структурная схема приложения

В классе главного окна (класс MainWindow) динамически создаются два основных объекта разрабатываемого приложения:

1. Объект класса DragWidgetScene. Класс наследуется от класса QGraphicsScene и представляет собой контейнер объектов.

2. Объект класса GardenScene. Класс является основным классом приложения и также наследуется от класса QGraphicsScene. Данный класс представляет рабочее поле разрабатываемого приложения, в нем реализуются методы, сигналы и слоты, отвечающие за создание объектов, взаимодействие объектов. [4]

Объекты классов DragWidgetScene и GardenScene в конструкторе главного окна связываются с соответствующими представлениями objectMenu и garden, позволяющими визуализировать содержимое сцен.

Класс GeneralPlan наследуется от класса QGraphicsItem. Объект данного класса создается динамически при создании нового плана и предназначается для хранения параметров садового участка, соседних участков и фиксированных объектов, расположенных на этих участках. В данном классе определены четыре переменные – объекты класса NearestObjects, в которых хранятся параметры соседних участков; фиксированные объекты, расположенные на этих участках; запретные зоны данных объектов. Объект класса GeneralPlan добавляется на сцену GardenScene после задания пользователем параметров садового участка.

Абстрактный класс GardenObjects наследуется от класса QGraphicsObjects и содержит в себе переменные и методы, общие для интерактивных объектов. Объекты классов, производных от класса GardenObjects добавляются на сцену DragWidgetScene и при помощи метода drag and drop устанавливаются на сцену GardenScene.

Все разработанные классы являются инкапсулированными. Для классов, в которых необходим доступ к переменным, разработаны функции доступа – геттеры и сеттеры. [5]

Более подробно процесс задания параметров плана садового участка, создания и взаимодействия объектов будет рассмотрен в следующей главе.

4. РЕАЛИЗАЦИЯ

4.1 Интерфейс приложения

Интерфейс приложения разработан с помощью средства быстрой разработки интерфейсов приложений Qt Designer, который при помощи ряда инструментов позволяет создавать необходимые элементы интерфейса, такие как виджеты, кнопки, графические виды и т.д., а также задавать и редактировать их свойства. Файл интерфейса(формы) с расширением *.ui подключается к текущему проекту. [6]

В разрабатываемом приложении предусмотрены два отдельных файла формы:

- mainwindow.ui – форма, задающая интерфейс разрабатываемого приложения;
- dialoggeneralplan.ui – форма, задающая внешний вид диалогового окна задания параметров участка.

На рисунке 4.1 представлен интерфейс разрабатываемого приложения, состоящий из 3 основных частей:

1. Рабочее поле приложения.

Это основная область приложения, предназначенная для создания плана участка, размещения и изменения объектов на плане.

2. Панель объектов.

Область приложения, представляющая список объектов в графическом виде, доступных для расположения в рабочем поле приложения.

3. Панель инструментов.

Условно делится на 3 части и состоит из панели команд, панели инструментов объектов и панели вспомогательных функций. Все действия, представленные на панели инструментов продублированы в меню.

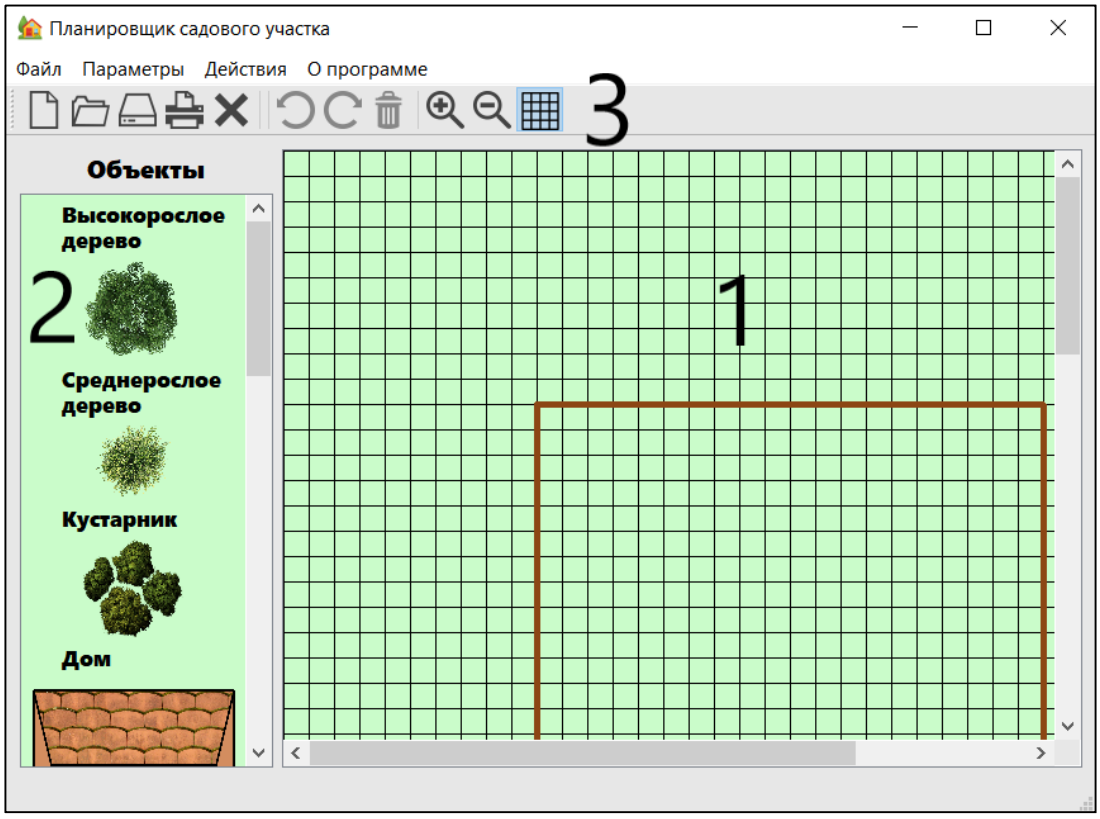

Рисунок 4.1 – Интерфейс приложения

Панель команд предоставляет такие действия как:

- создание нового плана;
- открытие существующего плана;
- сохранение плана;
- печать плана;
- закрытие программы.

Панель инструментов объектов задает доступные действия для объектов:

- вращение объекта против часовой стрелки на 10°;
- вращение объекта по часовой стрелке на 10°;
- удаление объекта.

Панель вспомогательных функций:

- увеличение плана участка (zoom in);
- уменьшение плана участка (zoom out);
- отображение размерной сетки;

Связь кнопок, расположенных на панели инструментов с предопределенными для этих кнопок действиями реализуется с помощью сигнально-слотовой системы, предоставляемой средой разработки Qt.

Во время запуска программы панель объектов не отображается, а в панели инструментов доступно только 3 действия – создание нового плана, открытие существующего плана и выход. При создании нового плана становится доступна панель объектов, все действия панели команд и панели вспомогательных функций. Панель инструментов объектов становится активна только при условии выделения интерактивного объекта, расположенного на плане.

4.2 Концепция динамического отображения допустимых расстояний между объектами

В основе концепции лежит принцип отображения допустимых расстояний в виде запретных зон, устанавливаемых для фиксированных и интерактивных объектов. При каких-либо действиях с интерактивным объектом, таких, как перемещение, поворот или изменение размера, в соответствующем слоте объекта класса GardenScene происходит последовательный перебор всех запретных зон, расположенных на плане объектов. При помощи системы флагов определяется, должен ли интерактивный объект взаимодействовать с какой-либо запретной зоной и устанавливается соответствующая подсветка запретных зон.

4.2.1 Типы объектов

Объекты в разрабатываемом приложении подразделяются на 2 типа: фиксированные и интерактивные.

Фиксированные объекты определяются согласно заданным параметрам при создании нового плана садового участка и устанавливаются на плане. Пользователь не имеет возможности каким-либо образом модифицировать данные объекты, для них запрещены такие действия, как выделение, перемещение, поворот и изменение размера. Так как данные объекты представлены в виде линий, то классы данных объектов наследуются от базового класса QGraphicsLineItem.

Каждый фиксированный объект имеет одну или несколько запретных зон, представляющих собой объекты класса RestrictedArea и являющиеся дочерними элементами фиксированных объектов.

Список фиксированных объектов представлен в таблице 4.1

Таблица 4.1

Список фиксированных объектов

Класс объекта	Описание	Количество запретных зон
1. StreetRedLine	Красная линия улицы	1
2. DriveWayRedLine	Красная линия проезда	2
3. Fence	Ограждение участка	1
4. NeighbourBorder	Граница соседнего участка	4

Интерактивные объекты добавляются на план садового участка пользователем, путем выбора их на панели объектов и переноса на план методом drag and drop. Данные объекты наследуются от разработанного абстрактного класса GardenObjects. Для них установлены флаги QGraphicsItem::ItemIsMovable и QGraphicsItem::ItemIsSelectable, что позволяет пользователю взаимодействовать с данными объектами. [8]

Запретные зоны имеют только те интерактивные объекты, для которых возможно взаимодействие между собой.

Интерактивные объекты, для которых возможно взаимодействие только с фиксированными объектами своих запретных зон не имеют.

Более подробно принцип взаимодействия описан в п.п. 4.2.4.

В таблице 4.2 представлен список интерактивных объектов.

Таблица 4.2

Список интерактивных объектов

Название класса объекта	Описание	Количество запретных зон
1. BigTree	Высокорослое дерево	-
2. Bush	Кустарник	-
3. Compost	Компостная яма	1
4. Glasshouse	Теплица	-
5. Henhouse	Курятник	-
6. House	Жилой дом	1
7. MidsizeTree	Среднерослое дерево	-
8. Sauna	Баня	1
9. Shed	Сарай	-
10. Watercloset	Уборная	1
11. Well	Колодец	-

4.2.2 Формирование объектов

Все интерактивные объекты наследуются от абстрактного класса GardenObjects, в котором определены переменные и методы, общие для интерактивных объектов. Класс GardenObjects в свою очередь наследуется от класса QGraphicsObject, что позволяет перегружать необходимые методы базового класса, такие как:

- int type() const

Данный метод позволяет задать для объекта пользовательский тип, что в дальнейшем устанавливает возможность определения установленного типа объекта на сцене, а также разрешает использовать qcraphics_cast для приведения типов.

- QRectF boundingRect() const

Метод задает область объекта, ограниченную прямоугольником.

- QPainterPath shape() const

С помощью данного метода устанавливается уточненная форма объекта. Это необходимо для более точной установки коллизий между объектами.

Если для интерактивного объекта определено наличие запретной зоны, то при установке данного объекта на план садового участка одновременно создается объект класса RestrictedArea. При этом определяется тип интерактивного объекта и в зависимости от него определяется форма запретной зоны. Например, для объекта типа «Колодец», форма запретной зоны будет представлять окружность, а для объекта типа «Баня» - прямоугольник со скругленными углами. Далее устанавливаются родительские отношения между интерактивным объектом и его запретной зоной. Объект класса RestrictedArea становится дочерним по отношению к интерактивному объекту. Благодаря системе Parent – Child, все перемещения и трансформации родительского объекта будут соответствующе отражаться на дочернем объекте, то есть интерактивный объект и объект класса RestrictedArea на сцене будут представлять собой единое целое.

Создание и установка объектов класса RestrictedArea для фиксированных объектов производится аналогично.

4.2.3 Система флагов

Для определения необходимости и типа взаимодействия объектов между собой, была разработана система флагов.

Система флагов представляет собой класс GardenFlags, в котором представлены флаги в виде перечисления (enum). Название флага для удобства использования состоит из названия объекта и минимального допустимого расстояния. Структура перечисления флагов представлена в таблице 4.3. [7]

Таблица 4.3

Структура флагов

Флаг	Значение	Описание
1. DriveWayRedLine5	0x1	Расстояние до красной линии проезда – 5 метров
2. DriveWayRedLine3	0x2	Расстояние до красной линии проезда – 3 метра
3. Fence2	0x4	Расстояние до забора – 2 метра
4. NeighbourBorder1	0x8	Расстояние до границы соседнего участка – 1 метр.
5. NeighbourBorder2	0x10	Расстояние до границы соседнего участка – 2 метра.
6. NeighbourBorder3	0x20	Расстояние до границы соседнего участка – 3 метра.
7. NeighbourBorder4	0x40	Расстояние до границы соседнего участка – 4 метра.
8. NeighbourHouse6	0x80	Расстояние до дома соседа – 6 метров
9. NeighbourHouse8	0x100	Расстояние до дома соседа – 8 метров
10. NeighbourHouse10	0x200	Расстояние до дома соседа – 10 метров
11. NeighbourHouse12	0x400	Расстояние до дома соседа – 12 метров
12. NeighbourHouse15	0x800	Расстояние до дома соседа – 15 метров
13. NeighbourBuilding4	0x1000	Расстояние до постройки соседа – 4 метра
14. MyHouse8	0x2000	Расстояние до жилого дома – 8 метров
15. Compost8	0x4000	Расстояние до компостной ямы – 8 метров
16. WaterCloset8	0x8000	Расстояние до уборной – 8 метров
17. Well8	0x10000	Расстояние до колодца – 8 метров
18. Sauna8	0x20000	Расстояние до бани – 8 метров
19. StreetRedLine5	0x40000	Расстояние до красной линии улицы – 5 метров

Для того, чтобы набор флагов стал доступен для метаобъектной системы, используются макросы: Q_DECLARE_FLAGS(Options,Option) и Q_DECLARE_OPERATORS_FOR_FLAGS(GardenFlags::Options), где Option – имя перечисления, а Options – имя набора флагов.

GardenFlag:Options задается в виде константы для каждого интерактивного объекта. Например:

static const GardenFlags::Options c_saunaFlags(GardenFlags::StreetRedLine5 |GardenFlags::DriveWayRedLine5 | GardenFlags::NeighbourBorder1 | GardenFlags::MyHouse8);

В данном примере создается набор флагов для объектов типа «Баня». Для данного типа объектов установлены флаги, обозначающие все минимальные допустимые расстояния, которые устанавливаются согласно таблице минимальных расстояний (см. табл. 1.1).

GardenFlag:Option представляет собой один флаг и задается только для объектов класса RestrictedArea. Например, объект класса Sauna имеет одну запретную зону (RestrictedArea), для которой устанавливается флаг GardenFlags::Sauna8 путем присваивания данного значения переменной, объявленной в классе RestrictedArea.

Для определения, с какими запретными зонами должен взаимодействовать объект, используется метод bool QFlags::testFlag(Enum flag) const, который возвращает true, если данный флаг установлен в наборе флагов.

4.2.4 Взаимодействие объектов

Проверка пересечений объектов осуществляется с помощью метода сцены QGraphicsItem::collidesWithItem(const QGraphicsItem *other, Qt::ItemSelectionMode mode = Qt::IntersectsItemShape) const.

Взаимодействие объектов будет продемонстрировано с помощью двух примеров, представленных на рисунках 4.2 и 4.3

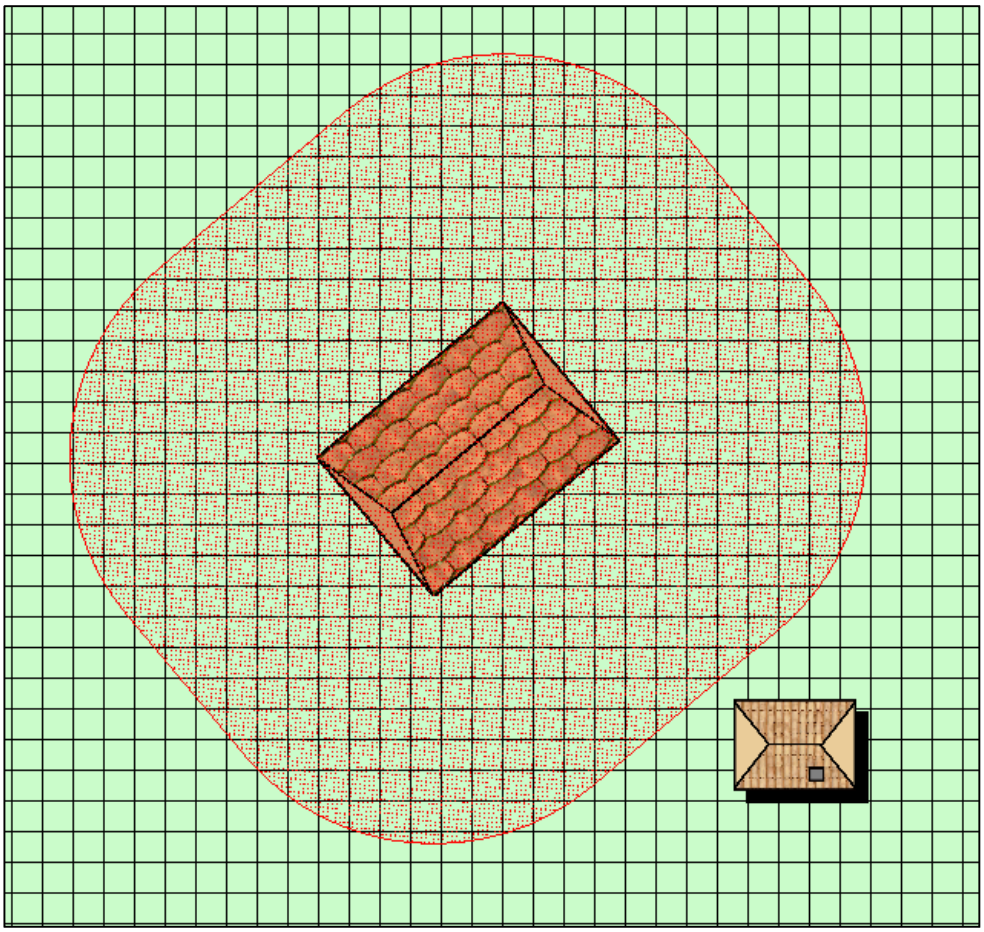

Рисунок 4.2 – Пример взаимодействия объектов №1

В случае, показанном на рисунке 4.2 объект «Баня» выделен, соответственно для него должны отображаться все запретные зоны тех объектов, с которыми нормируются минимальные расстояния. При выделении объекта, сцена генерирует сигнал selectionChanged(), который связан с разработанным слотом сцены slotSelectItems(). В данном слоте происходит последовательная проверка флагов всех запретных зон, установленных на плане на соответствие набору флагов, определенного для объекта «Баня». В данном случае флаг запретной зоны объекта «Жилой дом» содержится в наборе флагов объекта «Баня». Данная запретная зона подсвечивается штриховкой точками красного цвета и определяет область, при установке в которую объекта «Баня» минимальное расстояние 8 метров не будет выдержано.

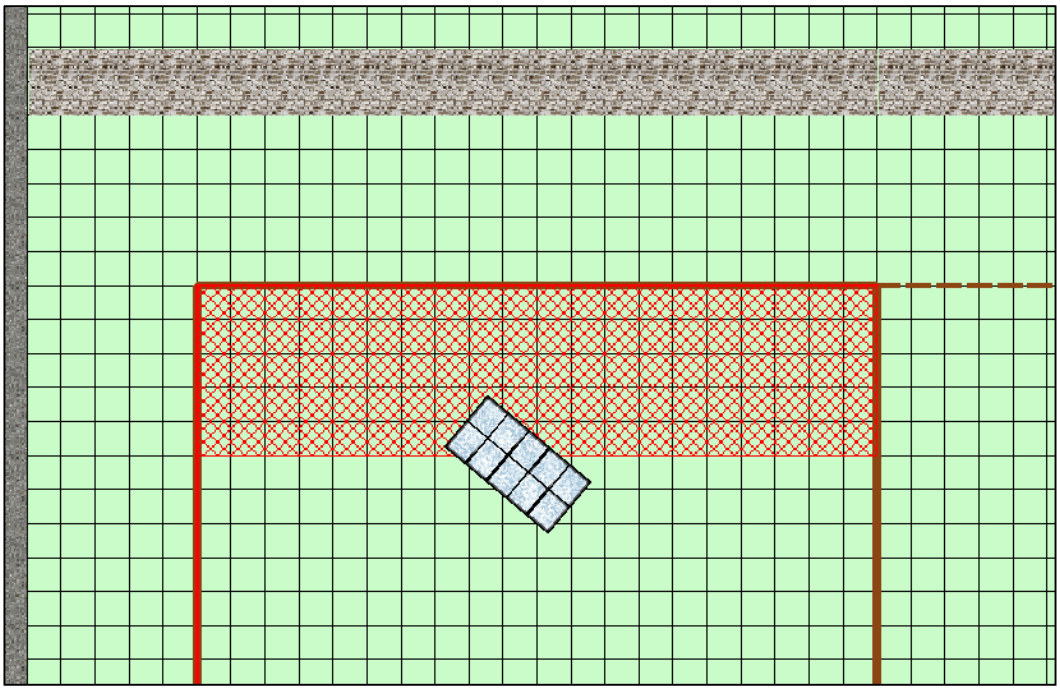

Рисунок 4.3 – Пример взаимодействия объектов №2

В случае, показанном на рисунке 4.3 объект «Теплица» установлен на плане (объект не выделен). При этом вызывается разработанный метод сцены setCollidingAreas(). В данном методе последовательно перебираются все, находящиеся на сцене объекты, определяются коллизии между объектами и запретными зонами с помощью метода сцены collidesWithItem(), определяется вхождение флага запретной зоны в набор флагов объекта. Объект «Теплица» имеет в своем наборе флагов DriveWayRedLine5 флаг, определяющий минимальное расстояние 5 метров до красной зоны проезда. Так как факт пересечения объектом запретной зоны с флагом DriveWayRedLine5 установлен, то данная запретная зона подсвечивается перекрестной штриховкой красного цвета. Данная зона будет подсвечиваться до того момента, пока пользователь не уберет объект из этой зоны.

4.3 Реализация базовых функций

4.3.1 Задание параметров садового участка

При создании нового плана садового участка вызывается диалоговое окно, представленное на рисунке 4.2

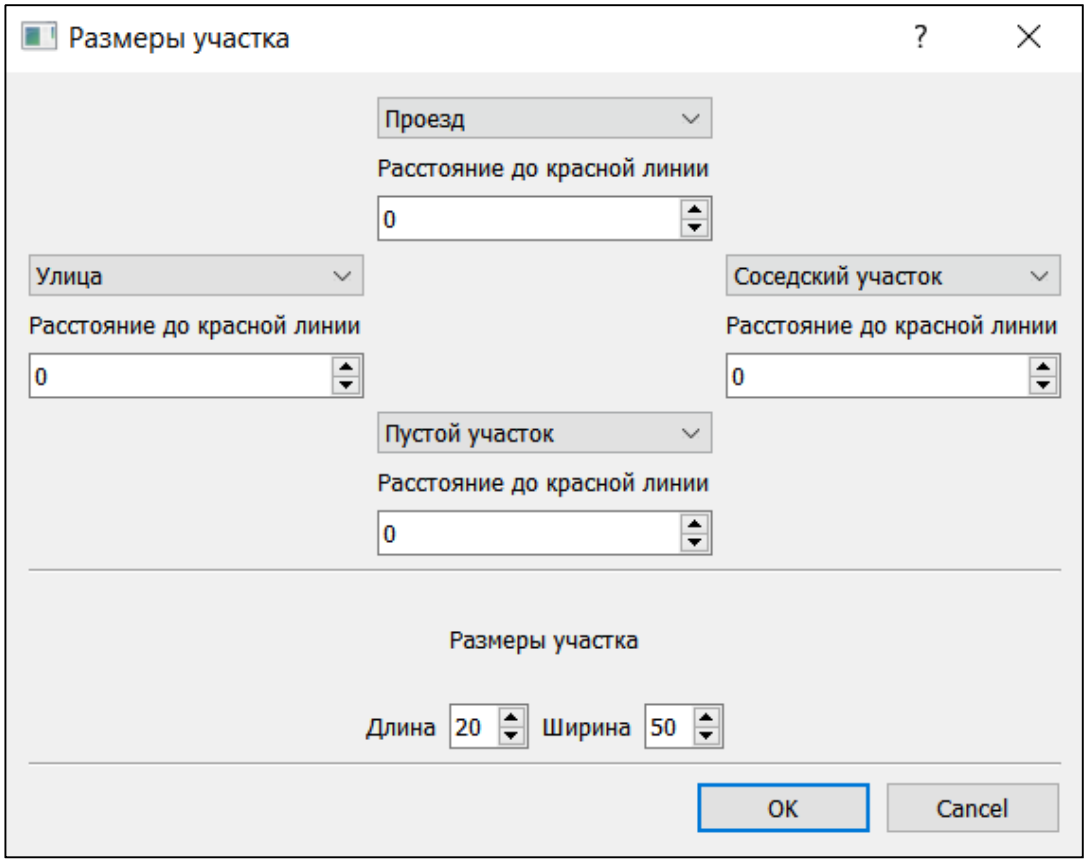

Рисунок 4.2 – Диалоговое окно задания параметров участка

В данном окне задаются основные параметры создаваемого плана:

- размеры участка;
- типы соседних участков;
- расстояния до красных линий;

После задания всех необходимых параметров и нажатия кнопки «ОК», вызывается разработанный слот сцены GardenScene slotSetGeneralPlan(), в котором:

- создается новый объект класса GeneralPlan и в него передаются параметры, полученные из диалогового окна;
- в объекте класса GeneralPlan задаются параметры соседних участков, вычисляется общий размер плана, задаются параметры визуального оформления соседних участков (отрисовка улиц, проездов, границ участков);
- план устанавливается на сцену GardenScene;
- вызывается разработанный метод сцены setFixedRestrictedAreas(), который, с помощью полученных данных от объекта GeneralPlan, устанавливает фиксированные объекты и связанные с ними запретные зоны на сцену GardenScene;
- генерируются сигналы для разблокировки панели команд и панели вспомогательных функций;
- генерируется сигнал для отображения панели объектов;
- генерируется сигнал для отображения размерной сетки.

Размерная сетка и фон рабочего поля задаются посредством перегруженного метода базового класса сцены void QGraphicsScene::drawBackground(QPainter *painter, const QRectF &rect).

4.3.2 Размещение объектов на плане методом drag and drop

Перемещение уже установленных объектов на плане реализуется посредством метода drag and drop с помощью встроенной возможности сцены QGraphicsScene. Иная ситуация возникает, когда требуется переместить объекты с панели объектов и установить их на сцену. Для этого в классе DragWidgetScene перегружаются два метода базового класса – MousePressEvent и MouseMoveEvent, а в классе GardenScene – DragEnterEvent и DropEvent. [9]

Последовательность действий следующая:

- в событии MousePressEvent устанавливается точка нажатия и тип объекта, на который нажатие было произведено;
- в событии MouseMoveEvent создается новый объект типа QDrag и новый объект типа QMimeData;
- тип объекта для переноса устанавливается в виде QString в объект QMimeData;
- объект QMimeData устанавливается в объект QDrag;
- в объекте QDrag устанавливается изображение, полученное разработанным методом QPixmap getPixmap() из объекта, на котором было произведено нажатие левой кнопки мыши;
- объект переносится на сцену GardenScene;
- в момент вхождения объекта в область графического представления сцены GardenScene, срабатывает событие DragEnterEvent;
- определяется тип объекта в контейнере QMimeData;
- отображаются существующие запретные зоны для данного объекта;
- в момент отпускания кнопки мыши срабатывает событие DropEvent;
- снова определяется тип объекта в контейнере QMimeData;
- динамически создается объект класса, соответствующего идентифицированному типу;
- объект добавляется на сцену и смещается на позицию курсора;
- создаются запретные зоны (если они определены для данного типа объекта) и устанавливаются родительские связи между ними и объектом.

4.3.3 Действия над объектами и планом

Так как класс сцены предоставляет готовые методы для действий над объектами, а класс графического представления предоставляет методы для действий над отображением сцены, то необходимо только разработать

соответствующие слоты, связанные с сигналами от кнопок на панели инструментов. [6], [10]

Список реализованных методов представлен в таблице 4.4.

Таблица 4.4

Методы для реализации действий над объектами и планом

Действие	Слот	Метод базового класса
Действия над объектами		
1. Поворот объекта против часовой стрелки на 10°	slotRotationLeft	qreal QGraphicsItem::rotation() const
2. Поворот объекта по часовой стрелке на 10°	slotRotationRight	qreal QGraphicsItem::rotation() const
3. Удаление объекта	slotDeleteItem	void QGraphicsScene::removeItem(QGraphicsItem *item)
Действия над планом		
4. Увеличение	slotZoomIn	void QGraphicsView::scale(qreal sx, qreal sy)
5. Уменьшение	slotZoomOut	void QGraphicsView::scale(qreal sx, qreal sy)

5. ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Частично тестирование разрабатываемого прототипа программы проводилось непосредственно в ходе разработки. При создании новых объектов сразу же проверялась корректность их отображения на плане и правильность их взаимодействия с уже разработанными объектами.

Кроме того, по окончанию разработки проводилось функциональное тестирование, в ходе которого проверялось соответствие приложения функциональным требованиям:

Корректность отображения визуального оформления плана при различных конфигурациях своего и соседских участков;

Правильность установки фиксированных объектов при различных конфигурациях соседских участков;

Корректность переноса каждого из объектов методом drag and drop;

Корректность установки запретных зон объектов;

Правильность отображения взаимодействий между интерактивными и фиксированными объектами, при различных действиях пользователя;

Правильность работы кнопок на панели инструментов относительно каждого интерактивного элемента и плана в целом.

В процессе тестирования было выявлено несколько проблем, представленных в таблице 5.1

Таблица 5.1

Выявленные проблемы

Проблема	Причина	Решение
1. Не отображается взаимодействие с запретной зоной красной линии улицы.	Некорректное значение флага StreetRedLine5 = 0x0.	Флагу StreetRedLine5 присвоено значение 0x40000.
2. При переносе объекта с панели объектов, запретные зоны для данного объекта оставались активными при выносе объекта за пределы окна приложения.	Не перегружен метод обработки события DragLeaveEvent в классе GardenScene.	DragLeaveEvent перегружен, в нем производятся необходимые действия.

ЗАКЛЮЧЕНИЕ

В работе представлены результаты разработки прототипа программы для планирования садового участка. Программное обеспечение разработано в среде Qt Framework, с использованием среды разработки Qt Designer, а также Qt GraphicsView Framework для работы с двухмерной графикой.

В процессе работы были получены следующие результаты:

- разработана архитектура приложения;
- разработан интерфейс приложения;
- разработана концепция динамического отображения минимально допустимых расстояний между объектами;
- разработаны необходимые классы и функции, в соответствии с функциональными требованиями;
- проведена проверка работоспособности приложения.

При разработке были исследованы и применены возможности Qt GraphicsView Framework, в частности методы для работы со сценой, ее представлениями и отдельными элементами.

Для дальнейшего развития проекта необходимо разработать свод новых функциональных требований, в которые могут входить:

- возможность сохранения плана в файл, открытия из файла и печати плана;
- возможность изменения размера объектов;
- учет противопожарных расстояний между постройками.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. СП 53.13330.2019. Планировка и застройка территории ведения гражданами садоводства. Здания и сооружения (СНиП 30-02-97* Планировка и застройка территорий садоводческих (дачных) объединений граждан, здания и сооружения). Введ. 15.04.2020.

2. Абрамова М.Г. Прикладное программирование на C++ с ипользованием Qt (базовый уровень). Учебное пособие по курсу. /М.Г.Абрамова, Н.Н.Костина, М.И.Полубенцева. - СПб.: Высшая инженерная школа СПбГПУ, 2019 – 86 с.

3. Костина Н.Н. Курс: DEV-QT20. Прикладное программирование на C++ с использованием QT. Углубленное изучение. Учебное пособие. /Костина Н.Н, Полубенцева М.И. – СПб.: Высшая инженерная школа СПбГПУ, 2020 – 656 с.

4. Саммерфилд. М. Qt. Профессиональное программирование. Разработка кроссплатформенных приложений на C++. – СПб.: Символ-Плюс, 2011. – 560 с.

5. Страуструп Б. Язык программирования C++, 3-е изд. – СПб.: «Невский диалект» - «Издательство БИНОМ», 1999. – 991 с.

6. Шлее М. Qt 5.10 Профессиональное программирование на C++. – СПб.: БХВ-Петербург, 2019. – 1072 с.: ил. – (в подлиннике).

7. Бешенов, Алексей. Программирование с Qt. Часть 4. Алгоритмы. Флаги и биты. [Электронный документ] (https://www.ibm.com/developerworks/ru/library/l-qt_4/) Проверено 23.06.2020.

8. Легоцкой, Евгений. Qt/C++ - Урок 027. Полиморфизм в Qt на примере геометрических фигур в QGraphicsScene [Электронный документ]. (https://evileg.com/ru/post/90/) Проверено 11.06.2020.

9. Перетаскивание (Drag and drop) [Электронный документ] (http://doc.crossplatform.ru/qt/4.5.0/dnd.html) Проверено 20.06.2020.

10. Qt Garphics View Framework – темная сторона. Часть 2: [Электронный документ] (https://habr.com/ru/post/183432/). Проверено 25.06.2020.

Приложение 1. Исходный код разработанного ПО

Constants.h

```
#ifndef CONSTANTS_H
#define CONSTANTS_H
#include <QGraphicsItem>
#include "gardenflags.h"

// Константы, задающие размер ячейки, дистанции до фиксированных объектов и
декоративных элементов
static const int c_cellSize = 20;
static const int c_roadWidth = 5*c_cellSize;
static const int c_roadDistance = 5*c_cellSize;
static const int c_driveWayWidth = 2*c_cellSize;
static const int c_emptyObjectSize = 10*c_cellSize;
static const int c_streetObjectSize = 10*c_cellSize;
static const int c_driveWayObjectSize = 8*c_cellSize;
static const int c_neighbourObjectSize = 18*c_cellSize;
//Вспомогательные перечисления
enum class NearestObjectsType {EMPTY, STREET, DRIVEWAY, NEIGHBOUR};
enum class Position {LEFT,RIGHT,TOP,BOTTOM};
enum class Direction {LEFT,RIGHT,TOP,BOTTOM};
//Задание пользовательских типов объектов
enum{RESTRICTED_AREA_TYPE = QGraphicsItem::UserType+1,
    STREET_RED_LINE_TYPE = QGraphicsItem::UserType+2,
    DRIVEWAY_RED_LINE_TYPE = QGraphicsItem::UserType+3,
    FENCE_TYPE = QGraphicsItem::UserType+4,
    NEIGHBOUR_BORDER_TYPE = QGraphicsItem::UserType+5,
    BIG_TREE_TYPE = QGraphicsItem::UserType+6,
    HOUSE_TYPE = QGraphicsItem::UserType+7,
    MIDSIZE_TREE_TYPE = QGraphicsItem::UserType+8,
    BUSH_TYPE = QGraphicsItem::UserType+9,
    SAUNA_TYPE = QGraphicsItem::UserType+10,
    SHED_TYPE = QGraphicsItem::UserType+11,
    GLASSHOUSE_TYPE = QGraphicsItem::UserType+12,
    HENHOUSE_TYPE = QGraphicsItem::UserType+13,
    WATERCLOSET_TYPE = QGraphicsItem::UserType+14,
    COMPOST_TYPE = QGraphicsItem::UserType+15,
    WELL_TYPE = QGraphicsItem::UserType+16};

//Флаги объектов
static const GardenFlags::Options c_treeFlags(GardenFlags::NeighbourBorder3);
static const GardenFlags::Options
c_midsizeTreeFlags(GardenFlags::NeighbourBorder2);
static const GardenFlags::Options c_bushFlags(GardenFlags::NeighbourBorder1);
static const GardenFlags::Options c_houseFlags(GardenFlags::NeighbourBorder3 |
                                               GardenFlags::StreetRedLine5 |
                                               GardenFlags::DriveWayRedLine3 |
                                               GardenFlags::Sauna8 |
                                               GardenFlags::WaterCloset8 |
                                               GardenFlags::MyHouse8);
static const GardenFlags::Options c_saunaFlags(GardenFlags::StreetRedLine5 |
                                               GardenFlags::DriveWayRedLine5 |
                                               GardenFlags::NeighbourBorder1 |
                                               GardenFlags::MyHouse8);
static const GardenFlags::Options c_shedFlags(GardenFlags::StreetRedLine5 |
                                              GardenFlags::DriveWayRedLine5 |
                                              GardenFlags::NeighbourBorder1);
```

```
static const GardenFlags::Options c_glasshouseFlags(GardenFlags::StreetRedLine5
|
                                                    GardenFlags::DriveWayRedLine5 |
                                                    GardenFlags::NeighbourBorder1);
static const GardenFlags::Options c_henhouseFlags(GardenFlags::StreetRedLine5 |
                                                    GardenFlags::DriveWayRedLine5 |
                                                    GardenFlags::NeighbourBorder4);
static const GardenFlags::Options c_waterclosetFlags(GardenFlags::Fence2 |
                                                    GardenFlags::MyHouse8 |
                                                    GardenFlags::NeighbourBorder2 |
                                                    GardenFlags::Well8);
static const GardenFlags::Options c_compostFlags(GardenFlags::Fence2 |
                                                    GardenFlags::NeighbourBorder2 |
                                                    GardenFlags::Well8);
static const GardenFlags::Options c_wellFlags(GardenFlags::Compost8 |
                                                    GardenFlags::WaterCloset8);
```

#endif // CONSTANTS_H

Mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include "gardenscene.h"
#include "dialoggeneralplan.h"
#include "dragwidgetscene.h"
#include "gardenflags.h"

QT_BEGIN_NAMESPACE
namespace Ui { class MainWindow; }
QT_END_NAMESPACE

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    MainWindow(QWidget *parent = nullptr);
    ~MainWindow();

private:
    Ui::MainWindow *ui;

public slots:
    void slotShowMenu();     //Отображение панели объектов
    void slotZoomIn();       //Увеличение плана
    void slotZoomOut();      //Уменьшение плана
    void slotEnableObjectTools();    //Активация панели инструментов
    void slotDisableObjectTools();   //Дезактивация панели инструментов
    void slotEnableCommandMenu();    //Активация меню команд
    void slotExit();     //Выход

};
```

#endif // MAINWINDOW_H

Gardenscene.h

```
#ifndef GARDENSCENE_H
#define GARDENSCENE_H

#include <QWidget>
#include <QGraphicsScene>
#include <QGraphicsItem>
#include <QDrag>
#include <QMimeData>
#include <QGraphicsSceneDragDropEvent>
#include <QGraphicsEffect>
#include <QGraphicsDropShadowEffect>
#include "generalplan.h"
#include "dialoggeneralplan.h"
#include "FixedObjects/streetredline.h"
#include "FixedObjects/drivewayredline.h"
#include "FixedObjects/fence.h"
#include "FixedObjects/neighbourborder.h"
#include "restrictedarea.h"
#include "Objects/bigtree.h"
#include "Objects/house.h"
#include "Objects/midsizetree.h"
#include "Objects/bush.h"
#include "Objects/sauna.h"
#include "Objects/shed.h"
#include "Objects/glasshouse.h"
#include "Objects/henhouse.h"
#include "Objects/watercloset.h"
#include "Objects/compost.h"
#include "Objects/well.h"

class GardenScene : public QGraphicsScene
{
    Q_OBJECT
    QRect m_scene_size; //Размер сцены
    bool grid;

public:
    explicit GardenScene(QWidget *parent = nullptr);
    void setFixedRestrictedAreas(GeneralPlan* plan); //Запретные зоны
фиксированных объектов
    void setCollidingAreas();        //Отображение пересечений
    void clearRestrictedAreas();     //Очистка подсвеченных зон

signals:
    signalEnableGrid(bool);
    signalShowMenu();
    signalEnableObjectTools();
    signalDisableObjectTools();
    signalEnableCommandMenu();

public slots:
    void slotSetGeneralPlan();  //Задание генерального плана
    void slotSetGrid(bool);     //установка размерной сетки
    void slotSelectItems();     //Отображение запретных зон для выбранного
объекта
    void slotRotationLeft();    //Поворот влево
    void slotRotationRight();   //Поворот вправо
    void slotDeleteItem();      //Удаление элемента
```

```
    // QGraphicsScene interface
protected:
    virtual void drawBackground(QPainter *painter, const QRectF &rect);

    // QGraphicsScene interface
protected:
    virtual void dragEnterEvent(QGraphicsSceneDragDropEvent *event);

    // QGraphicsScene interface
protected:
    virtual void dragMoveEvent(QGraphicsSceneDragDropEvent *event);

    // QGraphicsScene interface
protected:
    virtual void dropEvent(QGraphicsSceneDragDropEvent *event);

    // QGraphicsScene interface
protected:
    virtual void dragLeaveEvent(QGraphicsSceneDragDropEvent *event);
};

        #endif // GARDENSCENE_H
```

Dragwidgetscene.h

```
#ifndef DRAGWIDGETSCENE_H
#define DRAGWIDGETSCENE_H

#include <QWidget>
#include <QApplication>
#include <QDrag>
#include <QMimeData>
#include <QGraphicsScene>
#include <QGraphicsSceneMouseEvent>
#include "Objects/bigtree.h"
#include "Objects/house.h"
#include "Objects/midsizetree.h"
#include "Objects/bush.h"
#include "Objects/sauna.h"
#include "Objects/shed.h"
#include "Objects/glasshouse.h"
#include "Objects/henhouse.h"
#include "Objects/watercloset.h"
#include "Objects/compost.h"
#include "Objects/well.h"
#include <QDebug>
#include "Constants.h"

class DragWidgetScene : public QGraphicsScene
{
    Q_OBJECT
    QRect m_scene_size;
    QPointF m_dragStart;
    QGraphicsItem* currentItem;
public:
    explicit DragWidgetScene(QWidget *parent = nullptr);
```

```
signals:


    // QGraphicsScene interface
protected:
    virtual void mousePressEvent(QGraphicsSceneMouseEvent *event);
    virtual void mouseMoveEvent(QGraphicsSceneMouseEvent *event);
    virtual void mouseReleaseEvent(QGraphicsSceneMouseEvent *event);
    virtual void drawBackground(QPainter *painter, const QRectF &rect);
};
```

#endif // DRAGWIDGETSCENE_H

Generalplan.h

```
#ifndef GENERALPLAN_H
#define GENERALPLAN_H
#include <QWidget>
#include <QGraphicsItem>
#include <QPainter>
#include <QSize>
#include <QLine>
#include <QDebug>
#include "nearestobjects.h"
#include "Constants.h"



class GeneralPlan: public QGraphicsItem
{

    QSize m_size_garden_site;                       //Размер участка
    QSize m_total_size;
    NearestObjects m_left_object = NearestObjects(Position::LEFT);     //Объект
слева от участка
    NearestObjects m_right_object = NearestObjects(Position::RIGHT);    //Объект
справа от участка
    NearestObjects m_top_object = NearestObjects(Position::TOP);      //Объект
сверху от участка
    NearestObjects m_bottom_object = NearestObjects(Position::BOTTOM);
//Объект снизу от участка
    QPixmap pixmap;
    QRect m_garden_coordinates;

public:
    GeneralPlan();


    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);

    void adjustSize();
    NearestObjects & getLeftObject();
    NearestObjects & getRightObject();
    NearestObjects & getTopObject();
    NearestObjects & getBottomObject();
```

```
    void setLeftObjectData(uint distance,uint type,uint sideSize);
    void setRightObjectData(uint distance,uint type,uint sideSize);
    void setTopObjectData(uint distance,uint type,uint sideSize);
    void setBottomObjectData(uint distance,uint type,uint sideSize);

    void setSizeGardenSite(QSize);
    QSize getTotalSize();
    void setGardenCoordinates();
    QRect getGardenCoordinates();

    QLine getLeftSideLine();
    QLine getRightSideLine();
    QLine getTopSideLine();
    QLine getBottomSideLine();

    QLine getLeftSideFence();
    QLine getRightSideFence();
    QLine getTopSideFence();
    QLine getBottomSideFence();

};

        #endif // GENERALPLAN_H
```

Nearestobjects.h

```
#ifndef NEARESTOBJECTS_H
#define NEARESTOBJECTS_H
#include<QSize>
#include<QLine>
#include<QDebug>
#include "Constants.h"



class NearestObjects
{

public:


    uint m_distance;     //Расстояние до красной линии
    QSize m_size;        //Размер объекта
    NearestObjectsType m_type;        //Тип объекта
    Position m_object_position;
    Direction m_forbidden_area_direction; //Направление запретной зоны


public:
   NearestObjects();
   NearestObjects(Position forbidden_area_direction);

   void setObjectData(uint distance,uint type,uint sideSize);
   uint getDistance();
   QSize getSize();
   NearestObjectsType getType();
   Direction getForbiddenAreaDirection();
```

```
};
```

#endif // NEARESTOBJECTS_H

Restrictedarea.h

```
#ifndef RESTRICTEDAREA_H
#define RESTRICTEDAREA_H
#include <QGraphicsItem>
#include <QPen>
#include <QPainter>
#include "gardenflags.h"
#include "Constants.h"

enum class displayMode{EMPTY, BORDER,FILLED};

class RestrictedArea: public QGraphicsItem
{
    QRect m_rect_Area;
    uint m_Radius;
    GardenFlags::Option m_flag;
    displayMode m_mode;

public:
    explicit RestrictedArea();
    RestrictedArea(QRect rect, GardenFlags::Option flag);
    RestrictedArea(QRect rect, GardenFlags::Option flag, int radius);
    void setMode(displayMode);
    GardenFlags::Option getFlag();
    displayMode getMode();


    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual QPainterPath shape() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = RESTRICTED_AREA_TYPE };
    int type() const override{return Type;}
};

#endif // RESTRICTEDAREA_H
```

Gardenflags.h

```
/*
Класс флагов для определения запрещенных зон
для различных объектов.
*/
#ifndef GARDENFLAGS_H
#define GARDENFLAGS_H

#include <QFlags>
```

```
class GardenFlags
{
public:
    enum Option
    {

        DriveWayRedLine5 = 0x1,
        DriveWayRedLine3 = 0x2,
        Fence2 = 0x4,
        NeighbourBorder1 = 0x8,
        NeighbourBorder2 = 0x10,
        NeighbourBorder3 = 0x20,
        NeighbourBorder4 = 0x40,
        NeighbourHouse6 = 0x80,
        NeighbourHouse8 = 0x100,
        NeighbourHouse10 = 0x200,
        NeighbourHouse12 = 0x400,
        NeighbourHouse15 = 0x800,
        NeighbourBuilding4 = 0x1000,
        MyHouse8 = 0x2000,
        Compost8 = 0x4000,
        WaterCloset8 = 0x8000,
        Well8 = 0x10000,
        Sauna8 = 0x20000,
        StreetRedLine5 = 0x40000
    };

    Q_DECLARE_FLAGS(Options,Option)
    GardenFlags();
};

Q_DECLARE_OPERATORS_FOR_FLAGS(GardenFlags::Options)
#endif // GARDENFLAGS_H
```

Dialoggeneralplan.h

```
#ifndef DIALOGGENERALPLAN_H
#define DIALOGGENERALPLAN_H

#include <QDialog>

namespace Ui {
class DialogGeneralPlan;
}

class DialogGeneralPlan : public QDialog
{
    Q_OBJECT
    uint m_width;
    uint m_height;
    uint m_left_object_type;
    uint m_left_object_distance;
    uint m_top_object_type;
    uint m_top_object_distance;
    uint m_right_object_type;
    uint m_right_object_distance;
    uint m_bottom_object_type;
    uint m_bottom_object_distance;

public:
    explicit DialogGeneralPlan(QWidget *parent = nullptr);
    ~DialogGeneralPlan();
```

```
private:
    Ui::DialogGeneralPlan *ui;
public slots:
    void slotSetWidth(int);
    void slotSetHeight(int);
    void slotSetLeftObjectType(int);
    void slotSetLeftObjectDistance(int);
    void slotSetTopObjectType(int);
    void slotSetTopObjectDistance(int);
    void slotSetRightObjectType(int);
    void slotSetRightObjectDistance(int);
    void slotSetBottomObjectType(int);
    void slotSetBottomObjectDistance(int);

    int getWidth();
    int getHeight();
    int getLeftObjectType();
    int getLeftObjectDistance();
    int getTopObjectType();
    int getTopObjectDistance();
    int getRightObjectType();
    int getRightObjectDistance();
    int getBottomObjectType();
    int getBottomObjectDistance();

};

#endif // DIALOGGENERALPLAN_H
```

Drivewayredline.h

```
#ifndef DRIVEWAYREDLINE_H
#define DRIVEWAYREDLINE_H
#include <QGraphicsItem>
#include <QPen>
#include <QPainter>
#include "Constants.h"
#include "restrictedarea.h"

class DriveWayRedLine: public QGraphicsLineItem
{
    QLine m_line;
    QColor m_color;
    uint m_penWidth;
    QPen pen;
    QRect m_rect5;
    RestrictedArea* m_restrictedArea5;
    QRect m_rect3;
    RestrictedArea* m_restrictedArea3;
    QVector<RestrictedArea*> restricted_areas;
public:
    DriveWayRedLine();
    DriveWayRedLine(QLine line, Direction direction);
    QVector<RestrictedArea*> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = DRIVEWAY_RED_LINE_TYPE };
```

```
    int type() const override{return Type;}
};

#endif // DRIVEWAYREDLINE_H
```

Fence.h

```
#ifndef FENCE_H
#define FENCE_H

#include <QGraphicsItem>
#include <QPen>
#include <QPainter>
#include "Constants.h"
#include "restrictedarea.h"

class Fence: public QGraphicsLineItem
{
    QLine m_line;
    QPen pen;
    QRect m_rect2;
    RestrictedArea* m_restrictedArea2;
    QVector<RestrictedArea*> restricted_areas;
public:
    Fence();
    Fence(QLine line, Direction direction);
    QVector<RestrictedArea *> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = FENCE_TYPE };
    int type() const override{return Type;}
};

#endif // FENCE_H
```

Neighbourborder.h

```
#ifndef NEIGHBOURBORDER_H
#define NEIGHBOURBORDER_H

#include <QGraphicsItem>
#include <QPen>
#include <QPainter>
#include "Constants.h"
#include "restrictedarea.h"

class NeighbourBorder: public QGraphicsLineItem
{
    QLine m_line;
    QPen pen;
    QRect m_rect1;
    RestrictedArea* m_restrictedArea1;
    QRect m_rect2;
    RestrictedArea* m_restrictedArea2;
    QRect m_rect3;
    RestrictedArea* m_restrictedArea3;
    QRect m_rect4;
```

```
    RestrictedArea* m_restrictedArea4;

    QVector<RestrictedArea*> restricted_areas;
public:
    NeighbourBorder();
    NeighbourBorder(QLine line, Direction direction);
    QVector<RestrictedArea*> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    virtual QRectF boundingRect() const;
    enum { Type = NEIGHBOUR_BORDER_TYPE };
    int type() const override{return Type;}



};

#endif // NEIGHBOURBORDER_H
```

Streetredline.h

```
#ifndef STREETREDLINE_H
#define STREETREDLINE_H
#include <QGraphicsItem>
#include <QPen>
#include <QPainter>
#include "Constants.h"
#include "restrictedarea.h"

class StreetRedLine: public QGraphicsLineItem
{
    QLine m_line;
    QColor m_color;
    uint m_penWidth;
    QPen pen;
    QRect m_rect5;
    RestrictedArea* m_restrictedArea5;
    QVector<RestrictedArea*> restricted_areas;


public:
    StreetRedLine();
    StreetRedLine(QLine line, Direction direction);
    QVector<RestrictedArea*> getRestrictedAreas();


    // QGraphicsItem interface
public:
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    virtual QRectF boundingRect() const;
    enum { Type = STREET_RED_LINE_TYPE };
    int type() const override{return Type;}

};

#endif // STREETREDLINE_H
```

Bigtree.h

```
#ifndef BIGTREE_H
#define BIGTREE_H

#include <QPainter>
#include <QDebug>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"

class BigTree : public GardenObjects
{
    Q_OBJECT
    int m_radius;

public:
    explicit BigTree();
    QPixmap getPixmap();
    QPoint getPoint();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = BIG_TREE_TYPE };
    int type() const override{return Type;}

    // QGraphicsItem interface
public:
    virtual QPainterPath shape() const;

};

#endif // BIGTREE_H
```

Bush.h

```
#ifndef BUSH_H
#define BUSH_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include <QDebug>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"

class Bush: public GardenObjects
{
    Q_OBJECT
    int m_radius;

public:
    explicit Bush();
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
```

```
    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual QPainterPath shape() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = BUSH_TYPE };
    int type() const override{return Type;}
};

#endif // BUSH_H
```

Compost.h

```
#ifndef COMPOST_H
#define COMPOST_H

#include <QPainter>
#include <QDebug>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"
#include "restrictedarea.h"

class Compost: public GardenObjects
{
    Q_OBJECT
    uint m_width;
    uint m_height;
    RestrictedArea* m_compostArea8;
    QVector<RestrictedArea*> restricted_areas;

public:
    Compost();
    QVector<RestrictedArea*> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = COMPOST_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
};

#endif // COMPOST_H
```

Gardenobjects.h

```
#ifndef GARDENOBJECTS_H
#define GARDENOBJECTS_H
```

```
#include <QObject>
#include <QGraphicsObject>
#include <QPainter>


class GardenObjects: public QGraphicsObject
{
    Q_OBJECT
public:
    QPoint m_center_Point;
    QPixmap pixmap;
    QRect m_rect;
public:
    GardenObjects() {};
    virtual QPixmap getPixmap() = 0;
    virtual QPoint getPoint() = 0;
};

#endif // GARDENOBJECTS_H
```

Glasshouse.h

```
#ifndef GLASSHOUSE_H
#define GLASSHOUSE_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include "gardenobjects.h"
#include "Constants.h"

class Glasshouse : public GardenObjects
{
    Q_OBJECT

    uint m_width;
    uint m_height;
public:
    explicit Glasshouse();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = GLASSHOUSE_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
};

#endif // GLASSHOUSE_H
```

Henhouse.h

```
#ifndef HENHOUSE_H
#define HENHOUSE_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include "gardenobjects.h"
#include "Constants.h"

class Henhouse: public GardenObjects
{
    Q_OBJECT

    uint m_width;
    uint m_height;
public:
    Henhouse();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = HENHOUSE_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
};

#endif // HENHOUSE_H
```

House.h

```
#ifndef HOUSE_H
#define HOUSE_H

#include <QPainter>
#include <QDebug>
#include <QFile>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"
#include "restrictedarea.h"

class House : public GardenObjects
{
    Q_OBJECT
    uint m_width;
    uint m_height;
    RestrictedArea* m_houseArea8;
    QVector<RestrictedArea*> restricted_areas;
public:
    explicit House();
```

```
    QPixmap getPixmap();
    QPoint getPoint();
    QVector<RestrictedArea*> getRestrictedAreas();

public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = HOUSE_TYPE };
    int type() const override{return Type;}
};

#endif // HOUSE_H
```

Midsizetree.h

```
#ifndef MIDSIZETREE_H
#define MIDSIZETREE_H

#include <QPainter>
#include <QDebug>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"
class MidsizeTree : public GardenObjects
{
    Q_OBJECT

    double m_radius;

public:
    explicit MidsizeTree();
    QPixmap getPixmap();
    QPoint getPoint();


    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual QPainterPath shape() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = MIDSIZE_TREE_TYPE };
    int type() const override{return Type;}
};

#endif // MIDSIZETREE_H
```

Sauna.h

```
#ifndef SAUNA_H
#define SAUNA_H

#include <QPainter>
#include <QDebug>
#include "gardenobjects.h"
#include "Constants.h"
#include "gardenflags.h"
#include "restrictedarea.h"
```

```
class Sauna : public GardenObjects
{
    Q_OBJECT

    uint m_width;
    uint m_height;
    RestrictedArea* m_saunaArea8;
    QVector<RestrictedArea*> restricted_areas;
public:
    explicit Sauna();
    QPixmap getPixmap();
    QPoint getPoint();
    QVector<RestrictedArea*> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = SAUNA_TYPE };
    int type() const override{return Type;}
};

#endif // SAUNA_H
```

Shed.h

```
#ifndef SHED_H
#define SHED_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include "gardenobjects.h"
#include "Constants.h"


class Shed : public GardenObjects
{
    Q_OBJECT

    uint m_width;
    uint m_height;

public:
    explicit Shed();


    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = SHED_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
```

```
    virtual QPoint getPoint();
};

#endif // SHED_H
```

Watercloset.h

```
#ifndef WATERCLOSET_H
#define WATERCLOSET_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include "gardenobjects.h"
#include "restrictedarea.h"
#include "Constants.h"

class Watercloset: public GardenObjects
{
    Q_OBJECT
    uint m_width;
    uint m_height;
    RestrictedArea* m_waterclosetArea8;
    QVector<RestrictedArea*> restricted_areas;
public:
    explicit Watercloset();
    QVector<RestrictedArea*> getRestrictedAreas();

    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = WATERCLOSET_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
};

#endif // WATERCLOSET_H
```

Well.h

```
#ifndef WELL_H
#define WELL_H

#include <QObject>
#include <QGraphicsObject>
#include <QPainter>
#include "gardenobjects.h"
#include "restrictedarea.h"
#include "Constants.h"

class Well : public GardenObjects
```

```
{
    Q_OBJECT
    int m_radius;
    RestrictedArea* m_wellArea8;
    QVector<RestrictedArea*> restricted_areas;
public:
    explicit Well();
    QVector<RestrictedArea*> getRestrictedAreas();
    // QGraphicsItem interface
public:
    virtual QRectF boundingRect() const;
    virtual QPainterPath shape() const;
    virtual void paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget);
    enum { Type = WELL_TYPE };
    int type() const override{return Type;}

    // GardenObjects interface
public:
    virtual QPixmap getPixmap();
    virtual QPoint getPoint();
};

#endif // WELL_H
```

Main.cpp

```
#include "mainwindow.h"

#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();
    return a.exec();
}
```

Mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

MainWindow::MainWindow(QWidget *parent)
    : QMainWindow(parent)
    , ui(new Ui::MainWindow)
{
    ui->setupUi(this);



    //Создаем сцену и связываем с представлением
    GardenScene* scene = new GardenScene();
    ui->gardenGraphicsView->setScene(scene);
```

```
    ui->gardenGraphicsView-
>setViewportUpdateMode(QGraphicsView::FullViewportUpdate);


    DragWidgetScene* objectsMenuScene = new DragWidgetScene();
    ui->objectsMenuGraphicsView->setScene(objectsMenuScene);
    ui->label->hide();
    ui->objectsMenuGraphicsView->hide();
    qDebug()<<ui->objectsMenuGraphicsView->viewport()->size();

    ui->action_rotateLeft->setDisabled(true);
    ui->action_rotateRight->setDisabled(true);
    ui->action_Delete->setDisabled(true);
    ui->action_zoomIn->setDisabled(true);
    ui->action_zoomOut->setDisabled(true);
    ui->action_print->setDisabled(true);
    ui->action_saveAs->setDisabled(true);
    ui->action_save->setDisabled(true);


connect(scene,SIGNAL(signalEnableObjectTools()),this,SLOT(slotEnableObjectTools(
)));

connect(scene,SIGNAL(signalDisableObjectTools()),this,SLOT(slotDisableObjectTool
s()));

connect(scene,SIGNAL(signalEnableCommandMenu()),this,SLOT(slotEnableCommandMenu(
)));

    connect(ui-
>action_new,SIGNAL(triggered(bool)),scene,SLOT(slotSetGeneralPlan()));
    connect(ui-
>action_grid,SIGNAL(triggered(bool)),scene,SLOT(slotSetGrid(bool)));
    connect(scene,SIGNAL(signalEnableGrid(bool)),ui-
>action_grid,SLOT(setEnabled(bool)));
    connect(ui-
>action_rotateLeft,SIGNAL(triggered(bool)),scene,SLOT(slotRotationLeft()));
    connect(ui-
>action_rotateRight,SIGNAL(triggered(bool)),scene,SLOT(slotRotationRight()));
    connect(ui-
>action_Delete,SIGNAL(triggered(bool)),scene,SLOT(slotDeleteItem()));
    connect(scene,SIGNAL(signalShowMenu()),this,SLOT(slotShowMenu()));
    connect(ui->action_zoomIn,SIGNAL(triggered(bool)),this,SLOT(slotZoomIn()));
    connect(ui-
>action_zoomOut,SIGNAL(triggered(bool)),this,SLOT(slotZoomOut()));
    connect(ui->action_exit,SIGNAL(triggered(bool)),this, SLOT(slotExit()));
}

MainWindow::~MainWindow()
{
    delete ui;
}

void MainWindow::slotShowMenu()
{
    ui->objectsMenuGraphicsView->show();
    ui->label->show();
}

void MainWindow::slotZoomIn()
{
    ui->gardenGraphicsView->scale(1.1,1.1);
}
```

```
void MainWindow::slotZoomOut()
{

    ui->gardenGraphicsView->scale(0.9,0.9);
}

void MainWindow::slotEnableObjectTools()
{
    ui->action_rotateLeft->setEnabled(true);
    ui->action_rotateRight->setEnabled(true);
    ui->action_Delete->setEnabled(true);
}

void MainWindow::slotDisableObjectTools()
{
    ui->action_rotateLeft->setDisabled(true);
    ui->action_rotateRight->setDisabled(true);
    ui->action_Delete->setDisabled(true);

}

void MainWindow::slotEnableCommandMenu()
{
    ui->action_zoomIn->setEnabled(true);
    ui->action_zoomOut->setEnabled(true);
    ui->action_print->setEnabled(true);
    ui->action_saveAs->setEnabled(true);
    ui->action_save->setEnabled(true);
}

void MainWindow::slotExit()
{
    this->close();
}
```

Gardenscene.cpp

```
#include "gardenscene.h"

GardenScene::GardenScene(QWidget *parent) : QGraphicsScene(parent)
{
    grid = true;

    connect(this,SIGNAL(selectionChanged()),this,SLOT(slotSelectItems()));

}

void GardenScene::setFixedRestrictedAreas(GeneralPlan* plan)
{
    if(plan->getLeftObject().getType()==NearestObjectsType::STREET)
    {
        StreetRedLine* streetLeftRedLine = new StreetRedLine(plan-
>getLeftSideLine(),
                                                             plan-
>getLeftObject().getForbiddenAreaDirection());
        this->addItem(streetLeftRedLine);
        QVector<RestrictedArea*> tmp = streetLeftRedLine->getRestrictedAreas();
        for (auto var : tmp) {
```

```
            var->setParentItem(streetLeftRedLine);
        }
        Fence* fenceLeftLine = new Fence(plan->getLeftSideFence(),plan-
>getLeftObject().getForbiddenAreaDirection());
        this->addItem(fenceLeftLine);
        QVector<RestrictedArea*> tmp2 = fenceLeftLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceLeftLine);
        }
        this->update();
    }

    if(plan->getRightObject().getType()==NearestObjectsType::STREET)
    {
        StreetRedLine* streetRightRedLine = new StreetRedLine(plan-
>getRightSideLine(),
                                                              plan-
>getRightObject().getForbiddenAreaDirection());
        this->addItem(streetRightRedLine);
        QVector<RestrictedArea*> tmp = streetRightRedLine->getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(streetRightRedLine);
        }
        Fence* fenceRightLine = new Fence(plan->getRightSideFence(),plan-
>getRightObject().getForbiddenAreaDirection());
        this->addItem(fenceRightLine);
        QVector<RestrictedArea*> tmp2 = fenceRightLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceRightLine);
        }
        this->update();
    }

    if(plan->getTopObject().getType()==NearestObjectsType::STREET)
    {
        StreetRedLine* streetTopRedLine = new StreetRedLine(plan-
>getTopSideLine(),
                                                            plan-
>getTopObject().getForbiddenAreaDirection());
        this->addItem(streetTopRedLine);
        QVector<RestrictedArea*> tmp = streetTopRedLine->getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(streetTopRedLine);
        }
        Fence* fenceTopLine = new Fence(plan->getTopSideFence(),plan-
>getTopObject().getForbiddenAreaDirection());
        this->addItem(fenceTopLine);
        QVector<RestrictedArea*> tmp2 = fenceTopLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceTopLine);
        }
        this->update();
    }

    if(plan->getBottomObject().getType()==NearestObjectsType::STREET)
    {
        StreetRedLine* streetBottomRedLine = new StreetRedLine(plan-
>getBottomSideLine(),
                                                               plan-
>getBottomObject().getForbiddenAreaDirection());
        this->addItem(streetBottomRedLine);
        QVector<RestrictedArea*> tmp = streetBottomRedLine-
>getRestrictedAreas();
```

```
        for (auto var : tmp) {
            var->setParentItem(streetBottomRedLine);
        }
        Fence* fenceBottomLine = new Fence(plan->getBottomSideFence(),plan-
>getBottomObject().getForbiddenAreaDirection());
        this->addItem(fenceBottomLine);
        QVector<RestrictedArea*> tmp2 = fenceBottomLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceBottomLine);
        }
        this->update();
    }

    if(plan->getLeftObject().getType()==NearestObjectsType::DRIVEWAY)
    {
        DriveWayRedLine* driveWayLeftRedLine = new DriveWayRedLine(plan-
>getLeftSideLine(),
                                                                   plan-
>getLeftObject().getForbiddenAreaDirection());
        this->addItem(driveWayLeftRedLine);
        QVector<RestrictedArea*> tmp = driveWayLeftRedLine-
>getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(driveWayLeftRedLine);
        }
        Fence* fenceLeftLine = new Fence(plan->getLeftSideFence(),plan-
>getLeftObject().getForbiddenAreaDirection());
        this->addItem(fenceLeftLine);
        QVector<RestrictedArea*> tmp2 = fenceLeftLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceLeftLine);
        }
        this->update();
    }

    if(plan->getRightObject().getType()==NearestObjectsType::DRIVEWAY)
    {
        DriveWayRedLine* driveWayRightRedLine = new DriveWayRedLine(plan-
>getRightSideLine(),
                                                                   plan-
>getRightObject().getForbiddenAreaDirection());
        this->addItem(driveWayRightRedLine);
        QVector<RestrictedArea*> tmp = driveWayRightRedLine-
>getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(driveWayRightRedLine);
        }
        Fence* fenceRightLine = new Fence(plan->getRightSideFence(),plan-
>getRightObject().getForbiddenAreaDirection());
        this->addItem(fenceRightLine);
        QVector<RestrictedArea*> tmp2 = fenceRightLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceRightLine);
        }
        this->update();
    }

    if(plan->getTopObject().getType()==NearestObjectsType::DRIVEWAY)
    {
        DriveWayRedLine* driveWayTopRedLine = new DriveWayRedLine(plan-
>getTopSideLine(),
                                                                   plan-
>getTopObject().getForbiddenAreaDirection());
```

```
        this->addItem(driveWayTopRedLine);
        QVector<RestrictedArea*> tmp = driveWayTopRedLine->getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(driveWayTopRedLine);
        }
        Fence* fenceTopLine = new Fence(plan->getTopSideFence(),plan-
>getTopObject().getForbiddenAreaDirection());
        this->addItem(fenceTopLine);
        QVector<RestrictedArea*> tmp2 = fenceTopLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceTopLine);
        }
        this->update();
    }

    if(plan->getBottomObject().getType()==NearestObjectsType::DRIVEWAY)
    {
        DriveWayRedLine* driveWayBottomRedLine = new DriveWayRedLine(plan-
>getBottomSideLine(),
                                                                     plan-
>getBottomObject().getForbiddenAreaDirection());
        this->addItem(driveWayBottomRedLine);
        QVector<RestrictedArea*> tmp = driveWayBottomRedLine-
>getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(driveWayBottomRedLine);
        }
        Fence* fenceBottomLine = new Fence(plan->getBottomSideFence(),plan-
>getBottomObject().getForbiddenAreaDirection());
        this->addItem(fenceBottomLine);
        QVector<RestrictedArea*> tmp2 = fenceBottomLine->getRestrictedAreas();
        for (auto var : tmp2) {
            var->setParentItem(fenceBottomLine);
        }
        this->update();
    }

    if(plan->getLeftObject().getType()==NearestObjectsType::NEIGHBOUR)
    {
        NeighbourBorder* neighbourBorderLeft = new NeighbourBorder(plan-
>getLeftSideLine(),
                                                                   plan-
>getLeftObject().getForbiddenAreaDirection());
        this->addItem(neighbourBorderLeft);
        QVector<RestrictedArea*> tmp = neighbourBorderLeft-
>getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(neighbourBorderLeft);
        }

        this->update();
    }

    if(plan->getRightObject().getType()==NearestObjectsType::NEIGHBOUR)
    {
        NeighbourBorder* neighbourBorderRight = new NeighbourBorder(plan-
>getRightSideLine(),
                                                                    plan-
>getRightObject().getForbiddenAreaDirection());
        this->addItem(neighbourBorderRight);
        QVector<RestrictedArea*> tmp = neighbourBorderRight-
>getRestrictedAreas();
        for (auto var : tmp) {
```

```
            var->setParentItem(neighbourBorderRight);
        }
        this->update();
    }

    if(plan->getTopObject().getType()==NearestObjectsType::NEIGHBOUR)
    {
        NeighbourBorder* neighbourBorderTop = new NeighbourBorder(plan-
>getTopSideLine(),
                                                                  plan-
>getTopObject().getForbiddenAreaDirection());
        this->addItem(neighbourBorderTop);
        QVector<RestrictedArea*> tmp = neighbourBorderTop->getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(neighbourBorderTop);
        }
        this->update();
    }

    if(plan->getBottomObject().getType()==NearestObjectsType::NEIGHBOUR)
    {
        NeighbourBorder* neighbourBorderBottom = new NeighbourBorder(plan-
>getBottomSideLine(),
                                                                  plan-
>getBottomObject().getForbiddenAreaDirection());
        this->addItem(neighbourBorderBottom);
        QVector<RestrictedArea*> tmp = neighbourBorderBottom-
>getRestrictedAreas();
        for (auto var : tmp) {
            var->setParentItem(neighbourBorderBottom);
        }
        this->update();
    }



}

void GardenScene::setCollidingAreas()
{
    this->clearRestrictedAreas();

    foreach (auto item , this->items())
    {
        if(item->type()==BIG_TREE_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_treeFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
             }
        }
```

```
        if(item->type()==MIDSIZE_TREE_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_midsizeTreeFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
             }
        }
        if(item->type()==BUSH_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_bushFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
             }
        }
        if(item->type()==HOUSE_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_houseFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
             }
        }
        if(item->type()==SAUNA_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
```

```
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_saunaFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==SHED_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_shedFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==GLASSHOUSE_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_glasshouseFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==HENHOUSE_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
```

```
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_henhouseFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==WATERCLOSET_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_waterclosetFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==COMPOST_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
                    if(c_compostFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
        if(item->type()==WELL_TYPE)
        {
            foreach (auto otherItem , this->items())
            {
                if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                {
                    RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                    GardenFlags::Option flag = tmp->getFlag();
```

```
                    if(c_wellFlags.testFlag(flag))
                    {
                        if(otherItem->collidesWithItem(item))
                        {
                            tmp->setMode(displayMode::FILLED);
                        }
                    }
                }
            }
        }
    }
    this->update();
}

void GardenScene::clearRestrictedAreas()
{
    foreach (auto item , this->items())
    {
        if (item->type()==RESTRICTED_AREA_TYPE)
        {
            RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(item);
            tmp->setMode(displayMode::EMPTY);
        }
        else if (item->graphicsEffect())
        {
            item->setGraphicsEffect(nullptr);
        }
    }
    this->update();
}

void GardenScene::slotSetGeneralPlan()
{

    GeneralPlan* plan = new GeneralPlan();
    DialogGeneralPlan* dialog = new DialogGeneralPlan();
    ////////////////////////////////////////////////////////
    //Задаем параметры генерального плана через диалоговое окно
    ////////////////////////////////////////////////////////
    if(dialog->exec()==QDialog::Accepted)
    {

        plan->setLeftObjectData(dialog->getLeftObjectDistance()*c_cellSize,
                                                            dialog->getLeftObjectType(),
                                                            dialog-
>getHeight()*c_cellSize);

        plan->setTopObjectData(dialog->getTopObjectDistance()*c_cellSize,
                                                            dialog->getTopObjectType(),
                                                            dialog-
>getWidth()*c_cellSize);
        plan->setRightObjectData(dialog->getRightObjectDistance()*c_cellSize,
                                                            dialog-
>getRightObjectType(),
                                                            dialog-
>getHeight()*c_cellSize);
        plan->setBottomObjectData(dialog->getBottomObjectDistance()*c_cellSize,
                                                            dialog-
>getBottomObjectType(),
                                                            dialog-
>getWidth()*c_cellSize);
        plan->setSizeGardenSite(QSize(dialog->getWidth()*c_cellSize,dialog-
>getHeight()*c_cellSize));
```

```
        plan->adjustSize();
        plan->setGardenCoordinates();


        m_scene_size = QRect(0,0,plan->getTotalSize().width(),plan-
>getTotalSize().height());
        this->setSceneRect(m_scene_size); //Размеры сцены
        this->addItem(plan);
    ////////////////////////////////////////////
    //Задаем параметры фиксированных запретных зон
    ////////////////////////////////////////////


    this->setFixedRestrictedAreas(plan);

    ////////////////////////////


    emit signalEnableGrid(true);
    emit signalEnableCommandMenu();
    emit signalShowMenu();
    ////////////////////////////////////////////////////////////////////


    }
}

void GardenScene::slotSetGrid(bool b)
{
    grid = b;
    this->update();
}

void GardenScene::slotSelectItems()
{
    this->clearRestrictedAreas();
    if(this->selectedItems().size())
    {
        emit signalEnableObjectTools();
        foreach (auto item , this->selectedItems())
        {
            if(item->type()==BIG_TREE_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_treeFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
```

```
            if(item->type()==MIDSIZE_TREE_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_midsizeTreeFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==BUSH_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_bushFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==HOUSE_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_houseFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
```

```
                }
            }
            if(item->type()==SAUNA_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_saunaFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==SHED_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_shedFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==GLASSHOUSE_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_glasshouseFlags.testFlag(flag))
```

```
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==HENHOUSE_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_henhouseFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==WATERCLOSET_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_waterclosetFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==COMPOST_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
```

```
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_compostFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
            if(item->type()==WELL_TYPE)
            {
                QGraphicsDropShadowEffect* effect = new
QGraphicsDropShadowEffect();
                effect->setColor(Qt::black);
                item->setGraphicsEffect(effect);

                foreach (auto otherItem , this->items())
                {
                    //Важно!! Проверка на пересечение с собственным child
                    if (otherItem->type()==RESTRICTED_AREA_TYPE && otherItem-
>parentItem()!=item)
                    {
                        RestrictedArea* tmp =
qgraphicsitem_cast<RestrictedArea*>(otherItem);
                        GardenFlags::Option flag = tmp->getFlag();
                        if(c_wellFlags.testFlag(flag))
                        {
                            tmp->setMode(displayMode::BORDER);
                        }
                    }
                }
            }
        }
    }
    else
    {
        emit signalDisableObjectTools();
        this->setCollidingAreas();
    }
    this->update();
}

void GardenScene::slotRotationLeft()
{
    foreach (auto item , this->items())
    {
        if (item->isSelected())
        {
            item->setTransformOriginPoint(item->boundingRect().center());
            qreal angle = item->rotation();
            item->setRotation(angle-10);

        }

    }
    this->update();
}

void GardenScene::slotRotationRight()
{
    foreach (auto item , this->items())
```

```
    {
        if (item->isSelected())
        {
            item->setTransformOriginPoint(item->boundingRect().center());
            qreal angle = item->rotation();
            item->setRotation(angle+10);

        }

    }
    this->update();
}

void GardenScene::slotDeleteItem()
{
    foreach (auto item , this->items())
    {
        if (item->isSelected())
        {
            this->removeItem(item);
        }

    }
    this->update();
}



void GardenScene::drawBackground(QPainter *painter, const QRectF &rect)
{
    ////////////////////////////////////////
    /// Отрисовка фона
    ////////////////////////////////////////

    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(QColor(202,252,202,255));
    painter->fillRect(rect,brush);

    ////////////////////////////////////////
    /// Отрисовка сетки
    ////////////////////////////////////////
    if(grid)
    {
        brush.setStyle(Qt::NoBrush);
        QPen pen(Qt::SolidLine); //Тип линии сетки
        pen.setColor(QColor(Qt::black));  //Цвет линии сетки 210,220,240,255
        pen.setWidth(0);
        painter->setBrush(brush);
        painter->setPen(pen);
        for (int i = 0; i <= m_scene_size.width(); i+=c_cellSize)
        {
            painter->drawLine(i,0,i,m_scene_size.height());
        }
        for (int i = 0; i <= m_scene_size.height(); i+=c_cellSize)
        {
            painter->drawLine(0,i,m_scene_size.width(),i);
        }
    }
```

```
}


void GardenScene::dragEnterEvent(QGraphicsSceneDragDropEvent *event)
{
    event->setAccepted(true);
    QList<QGraphicsItem*> itemList = this->items();
    if(event->mimeData()->text().toInt()==BIG_TREE_TYPE)
    {
       for (auto var : itemList)
       {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_treeFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
       }
    }
    else if(event->mimeData()->text().toInt()==MIDSIZE_TREE_TYPE)
    {
       for (auto var : itemList)
       {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_midsizeTreeFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
       }
    }
    else if(event->mimeData()->text().toInt()==BUSH_TYPE)
    {
       for (auto var : itemList)
       {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_bushFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
       }
    }
    else if(event->mimeData()->text().toInt()==HOUSE_TYPE)
    {
        for (auto var : itemList)
        {
             if (var->type()==RESTRICTED_AREA_TYPE)
             {
                 RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                 GardenFlags::Option flag = tmp->getFlag();
                 if(c_houseFlags.testFlag(flag))
                 {
                     tmp->setMode(displayMode::FILLED);
```

```
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==SAUNA_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_saunaFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==SHED_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_shedFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==GLASSHOUSE_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_glasshouseFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==HENHOUSE_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_henhouseFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
```

```
    }
    else if(event->mimeData()->text().toInt()==WATERCLOSET_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_waterclosetFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==COMPOST_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_compostFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    else if(event->mimeData()->text().toInt()==WELL_TYPE)
    {
        for (auto var : itemList)
        {
            if (var->type()==RESTRICTED_AREA_TYPE)
            {
                RestrictedArea* tmp = qgraphicsitem_cast<RestrictedArea*>(var);
                GardenFlags::Option flag = tmp->getFlag();
                if(c_wellFlags.testFlag(flag))
                {
                    tmp->setMode(displayMode::FILLED);
                }
            }
        }
    }
    this->update();
}


void GardenScene::dragMoveEvent(QGraphicsSceneDragDropEvent *event)
{
}


void GardenScene::dropEvent(QGraphicsSceneDragDropEvent *event)
{
    QList<QGraphicsItem*> itemList = this->items();
    if(event->mimeData()->text().toInt()==BIG_TREE_TYPE)
    {
       BigTree* tree = new BigTree();
       tree->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
```

```
        this->addItem(tree);
        QPointF tmp = event->scenePos()-tree->getPoint();
        tree->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();
    }
    else if(event->mimeData()->text().toInt()==MIDSIZE_TREE_TYPE)
    {
        MidsizeTree* tree = new MidsizeTree();
        tree->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(tree);
        QPointF tmp = event->scenePos()-tree->getPoint();
        tree->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();
    }
    else if(event->mimeData()->text().toInt()==BUSH_TYPE)
    {
        Bush* bush = new Bush();
        bush->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(bush);
        QPointF tmp = event->scenePos()-bush->getPoint();
        bush->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();
    }

    else if(event->mimeData()->text().toInt()==HOUSE_TYPE)
    {
        House* house = new House();
        house->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(house);
        QPointF tmp = event->scenePos()-house->getPoint();
        house->setPos(tmp);
        QVector<RestrictedArea*> areas = house->getRestrictedAreas();
        for (auto var : areas)
        {
            var->setParentItem(house);
        }
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==SAUNA_TYPE)
    {
        Sauna* sauna = new Sauna();
        sauna->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(sauna);
        QPointF tmp = event->scenePos()-sauna->getPoint();
        sauna->setPos(tmp);
        QVector<RestrictedArea*> areas = sauna->getRestrictedAreas();
        for (auto var : areas)
        {
            var->setParentItem(sauna);
        }
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
```

```
    else if(event->mimeData()->text().toInt()==SHED_TYPE)
    {
        Shed* shed = new Shed();
        shed->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(shed);
        QPointF tmp = event->scenePos()-shed->getPoint();
        shed->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==GLASSHOUSE_TYPE)
    {
        Glasshouse* glasshouse = new Glasshouse();
        glasshouse->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(glasshouse);
        QPointF tmp = event->scenePos()-glasshouse->getPoint();
        glasshouse->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==HENHOUSE_TYPE)
    {
        Henhouse* henhouse = new Henhouse();
        henhouse->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(henhouse);
        QPointF tmp = event->scenePos()-henhouse->getPoint();
        henhouse->setPos(tmp);
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==WATERCLOSET_TYPE)
    {
        Watercloset* wc = new Watercloset();
        wc->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(wc);
        QPointF tmp = event->scenePos()-wc->getPoint();
        wc->setPos(tmp);

        QVector<RestrictedArea*> areas = wc->getRestrictedAreas();
        for (auto var : areas)
        {
            var->setParentItem(wc);
        }
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==COMPOST_TYPE)
    {
        Compost* compost = new Compost();
        compost->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(compost);
        QPointF tmp = event->scenePos()-compost->getPoint();
        compost->setPos(tmp);
```

```
        QVector<RestrictedArea*> areas = compost->getRestrictedAreas();
        for (auto var : areas)
        {
            var->setParentItem(compost);
        }
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    else if(event->mimeData()->text().toInt()==WELL_TYPE)
    {
        Well* well = new Well();
        well->setFlags(QGraphicsItem::ItemIsMovable |
QGraphicsItem::ItemIsSelectable);
        this->addItem(well);
        QPointF tmp = event->scenePos()-well->getPoint();
        well->setPos(tmp);

        QVector<RestrictedArea*> areas = well->getRestrictedAreas();
        for (auto var : areas)
        {
            var->setParentItem(well);
        }
        this->clearRestrictedAreas();
        this->setCollidingAreas();

    }
    this->update();
}


void GardenScene::dragLeaveEvent(QGraphicsSceneDragDropEvent *event)
{

    this->clearRestrictedAreas();
    this->setCollidingAreas();
    QGraphicsScene::dragLeaveEvent(event);
}
```

Dragwidgetscene.cpp

```
#include "dragwidgetscene.h"

DragWidgetScene::DragWidgetScene(QWidget *parent) : QGraphicsScene(parent)
{
    m_scene_size = QRect(0,0,120,1500);
    this->setSceneRect(m_scene_size);
    QFont font("Segoe UI Black", 10);

    QGraphicsTextItem* textHigh = this->addText("Высокорослое",font);
    //textHigh->setPos();
    QGraphicsTextItem* textTree1 = this->addText("дерево",font);
    textTree1->setPos(0,20);
    BigTree* tree = new BigTree();
    tree->setPos(60 - tree->getPoint().x(),50);
    this->addItem(tree);

    QGraphicsTextItem* textMid = this->addText("Среднерослое",font);
```

```
textMid->setPos(0,130);
QGraphicsTextItem* textTree2 = this->addText("дерево",font);
textTree2->setPos(0,150);
MidsizeTree* tree2 = new MidsizeTree();
tree2->setPos(60 - tree2->getPoint().x(),180);
this->addItem(tree2);

QGraphicsTextItem* textBush = this->addText("Кустарник",font);
textBush->setPos(0,240);
Bush* bush = new Bush();
bush->setPos(60 - bush->getPoint().x(),270);
this->addItem(bush);


QGraphicsTextItem* textHouse = this->addText("Дом",font);
textHouse->setPos(0,350);
House* house = new House();
this->addItem(house);
house->setPos(60-house->getPoint().x(),390);

QGraphicsTextItem* textSauna = this->addText("Баня",font);
textSauna->setPos(0,520);
Sauna* sauna = new Sauna();
this->addItem(sauna);
sauna->setPos(60-sauna->getPoint().x(),560);

QGraphicsTextItem* textShed = this->addText("Гараж/сарай",font);
textShed->setPos(0,630);
Shed* shed = new Shed();
this->addItem(shed);
shed->setPos(60-shed->getPoint().x(),670);

QGraphicsTextItem* textglasshouse = this->addText("Теплица",font);
textglasshouse->setPos(0,730);
Glasshouse* glasshouse = new Glasshouse();
this->addItem(glasshouse);
glasshouse->setPos(60-glasshouse->getPoint().x(),770);

QGraphicsTextItem* texthenhouse = this->addText("Курятник",font);
texthenhouse->setPos(0,830);
Henhouse* henhouse = new Henhouse();
this->addItem(henhouse);
henhouse->setPos(60-henhouse->getPoint().x(),870);

QGraphicsTextItem* textWC = this->addText("Уборная",font);
textWC->setPos(0,940);
Watercloset* watercloset = new Watercloset();
this->addItem(watercloset);
watercloset->setPos(60-watercloset->getPoint().x(),980);

QGraphicsTextItem* textCompost = this->addText("Компостная яма",font);
textCompost->setPos(0,1040);
Compost* compost = new Compost();
this->addItem(compost);
compost->setPos(60-compost->getPoint().x(),1080);

QGraphicsTextItem* textWell = this->addText("Колодец",font);
textWell->setPos(0,1140);
Well* well = new Well();
this->addItem(well);
well->setPos(60-well->getPoint().x(),1180);
```

```
}


void DragWidgetScene::mousePressEvent(QGraphicsSceneMouseEvent *event)
{
    currentItem = this->itemAt(event->scenePos(),QTransform());
    qDebug()<<event->scenePos();
    if(currentItem != nullptr)
    {
        qDebug()<<currentItem->type();
        m_dragStart = event->pos();
    }
    else QGraphicsScene::mousePressEvent(event);
}

void DragWidgetScene::mouseMoveEvent(QGraphicsSceneMouseEvent *event)
{
    if(currentItem != nullptr)
    {
        if( event->buttons() & Qt::LeftButton)
        {
                QDrag* drag = new QDrag( this );

                QMimeData* mimeData = new QMimeData;
                if(currentItem->type()==BIG_TREE_TYPE)
                {
                    BigTree* tmp = qgraphicsitem_cast<BigTree*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }

                else if(currentItem->type()==MIDSIZE_TREE_TYPE)
                {
                    MidsizeTree* tmp =
qgraphicsitem_cast<MidsizeTree*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==BUSH_TYPE)
                {
                    Bush* tmp = qgraphicsitem_cast<Bush*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );

                }

                else if(currentItem->type()==HOUSE_TYPE)
                {
                    House* tmp = qgraphicsitem_cast<House*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
```

```
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==SAUNA_TYPE)
                {
                    Sauna* tmp = qgraphicsitem_cast<Sauna*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==SHED_TYPE)
                {
                    Shed* tmp = qgraphicsitem_cast<Shed*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==GLASSHOUSE_TYPE)
                {
                    Glasshouse* tmp =
qgraphicsitem_cast<Glasshouse*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==HENHOUSE_TYPE)
                {
                    Henhouse* tmp = qgraphicsitem_cast<Henhouse*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==WATERCLOSET_TYPE)
                {
                    Watercloset* tmp =
qgraphicsitem_cast<Watercloset*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==COMPOST_TYPE)
                {
                    Compost* tmp = qgraphicsitem_cast<Compost*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
```

```
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }
                else if(currentItem->type()==WELL_TYPE)
                {
                    Well* tmp = qgraphicsitem_cast<Well*>(currentItem);
                    drag->setHotSpot(tmp->getPoint());
                    QPixmap pixmap = tmp->getPixmap();
                    QString str = QString::number(currentItem->type());
                    mimeData->setText(str);
                    drag->setMimeData( mimeData );
                    drag->setPixmap( pixmap );
                }

                Qt::DropAction result = drag->exec( Qt::MoveAction );
                qDebug() << "Drop action result: " << result;

            }

    }

    else QGraphicsScene::mouseMoveEvent(event);
}

void DragWidgetScene::mouseReleaseEvent(QGraphicsSceneMouseEvent *event)
{
    QGraphicsScene::mouseReleaseEvent(event);
}


void DragWidgetScene::drawBackground(QPainter *painter, const QRectF &rect)
{
    QBrush brush;
    brush.setStyle(Qt::SolidPattern);
    brush.setColor(QColor(202,252,202,255));
    painter->fillRect(rect,brush);
}
```

Generalplan.cpp

```
/*
Класс главного объекта сцены.
Объект сосотоит из самого участка,дорог,проездов, соседских участков.
*/
#include "generalplan.h"

GeneralPlan::GeneralPlan()
{
    m_size_garden_site = QSize(0,0); //временно

}


QRectF GeneralPlan::boundingRect() const
{
    return QRectF(QPointF(0,0),m_size_garden_site);
}
```

```
void GeneralPlan::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    painter->save();
    QPixmap tmp;
    QBrush brush;
    QPen pen = QPen();
    ////////////////////////////////////////////////
    //Чертим границы участка
    ////////////////////////////////////////////////
    pen.setStyle(Qt::SolidLine);
    pen.setColor(QColor(140,70,20,255));
    pen.setWidth(5);
    painter->setPen(pen);
    painter->drawRect(m_garden_coordinates);
    ////////////////////////////////////////////////
    //Отрисовка соседних объектов
    ////////////////////////////////////////////////
    pixmap.load(":/Textures/Images/asphalt2.jpg");
    tmp = pixmap.scaled(pixmap.size().width()/10,
            pixmap.size().height()/10,Qt::KeepAspectRatio);
    brush.setTexture(tmp);
    pen.setColor(QColor(Qt::white));
    pen.setWidth(4);
    pen.setStyle(Qt::DashLine);
    ////////////////////////////////////////////////
    //Отрисовка дорог
    //Для того, чтобы дорога не "заезжала" на размерную сетку
    //приходится добавлять сдвиги на один пиксель (иначе некрасиво)
    ////////////////////////////////////////////////
    if(m_left_object.getType()==NearestObjectsType::STREET)
    {

        brush.setTexture(tmp);
        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(m_garden_coordinates.x()-c_roadDistance-c_roadWidth+1,
                          1,
                          c_roadWidth-1,
                          m_total_size.height()-1);


    }



    if(m_top_object.getType()==NearestObjectsType::STREET)
    {

        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(1,
                          m_garden_coordinates.y()-c_roadDistance-c_roadWidth+1,
                          m_total_size.width()-1,
                          c_roadWidth-1);

    }

    if(m_right_object.getType()==NearestObjectsType::STREET)
    {

        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
```

```
        painter->drawRect(m_garden_coordinates.x() +
m_garden_coordinates.width() + c_roadDistance+1,
                          1,
                          c_roadWidth-1,
                          m_total_size.height()-1);

    }

    if(m_bottom_object.getType()==NearestObjectsType::STREET)
    {

        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(1,

m_garden_coordinates.y()+m_garden_coordinates.height()+c_roadDistance+1,
                          m_total_size.width()-1,
                          c_roadWidth-1);

    }

    painter->setPen(pen);
    painter->setBrush(Qt::NoBrush);
    ////////////////////////////////////////////////
    //Отрисовка дорожных полос
    ////////////////////////////////////////////////
    if(m_left_object.getType()==NearestObjectsType::STREET)
    {

        painter->drawLine(m_garden_coordinates.x()-c_roadDistance-c_roadWidth/2,
                          1,
                          m_garden_coordinates.x()-c_roadDistance-c_roadWidth/2,
                          m_total_size.height()-1);
    }



    if(m_top_object.getType()==NearestObjectsType::STREET)
    {

        painter->drawLine(1,
                          m_garden_coordinates.y()-c_roadDistance-c_roadWidth/2,
                          m_total_size.width()-1,
                          m_garden_coordinates.y()-c_roadDistance-
c_roadWidth/2);
    }

    if(m_right_object.getType()==NearestObjectsType::STREET)
    {

        painter->drawLine(m_garden_coordinates.x() +
m_garden_coordinates.width()+c_roadDistance+c_roadWidth/2,
                          1,
                          m_garden_coordinates.x() +
m_garden_coordinates.width()+c_roadDistance+c_roadWidth/2,
                          m_total_size.height()-1);
    }

    if(m_bottom_object.getType()==NearestObjectsType::STREET)
    {

        painter->drawLine(1,
```

```
m_garden_coordinates.y()+m_garden_coordinates.height()+c_roadDistance+c_roadWidt
h/2,
                            m_total_size.width()-1,

m_garden_coordinates.y()+m_garden_coordinates.height()+c_roadDistance+c_roadWidt
h/2);
    }

    pixmap.load(":/Textures/Images/stone.jpg");
    tmp = pixmap.scaled(pixmap.size().width()/50,
            pixmap.size().height()/50,Qt::KeepAspectRatio);
    brush.setTexture(tmp);

    ////////////////////////////////////////////////
    //Отрисовка проездов
    ////////////////////////////////////////////////
    if(m_left_object.getType()==NearestObjectsType::DRIVEWAY)
    {

        brush.setTexture(tmp);
        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(m_garden_coordinates.left()-c_roadDistance-
c_driveWayWidth+1,
                            m_garden_coordinates.top(),
                            c_driveWayWidth-1,
                            m_garden_coordinates.height());
        if(m_top_object.getType()==NearestObjectsType::STREET)
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance-
c_driveWayWidth+1,
                                m_garden_coordinates.top()-c_roadDistance+1,
                                c_driveWayWidth-1,
                                c_roadDistance-1);
        }
        else
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance-
c_driveWayWidth+1,
                                1,
                                c_driveWayWidth-1,
                                m_top_object.getSize().height()-1);
        }
        if(m_bottom_object.getType()==NearestObjectsType::STREET)
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance-
c_driveWayWidth+1,

m_garden_coordinates.y()+m_garden_coordinates.height()+1,
                                c_driveWayWidth-1,
                                c_roadDistance-1);
        }
        else
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance-
c_driveWayWidth+1,

m_garden_coordinates.y()+m_garden_coordinates.height()+1,
                                c_driveWayWidth-1,
                                m_bottom_object.getSize().height()-1);
        }
    }
```

```
    if(m_top_object.getType()==NearestObjectsType::DRIVEWAY)
    {

        brush.setTexture(tmp);
        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(m_garden_coordinates.left(),
                          m_garden_coordinates.top()-c_roadDistance-
c_driveWayWidth+1,
                          m_garden_coordinates.width(),
                          c_driveWayWidth-1);
        if(m_left_object.getType()==NearestObjectsType::STREET)
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance+1,
                              m_garden_coordinates.top()-c_roadDistance-
c_driveWayWidth+1,
                              c_roadDistance-1,
                              c_driveWayWidth-1);
        }
        else
        {
            painter->drawRect(1,
                              m_garden_coordinates.top()-c_roadDistance-
c_driveWayWidth+1,
                              m_left_object.getSize().width()-1,
                              c_driveWayWidth-1);
        }
        if(m_right_object.getType()==NearestObjectsType::STREET)
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+1,
                              m_garden_coordinates.top()-c_roadDistance-
c_driveWayWidth+1,
                              c_roadDistance-1,
                              c_driveWayWidth-1);
        }
        else
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+1,
                              m_garden_coordinates.top()-c_roadDistance-
c_driveWayWidth+1,
                              m_right_object.getSize().width()-1,
                              c_driveWayWidth-1);
        }
    }

    if(m_right_object.getType()==NearestObjectsType::DRIVEWAY)
    {

        brush.setTexture(tmp);
        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+c_roadDistance+1
,
                          m_garden_coordinates.top(),
                          c_driveWayWidth-1,
                          m_garden_coordinates.height());
        if(m_top_object.getType()==NearestObjectsType::STREET)
        {
```

```
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+c_roadDistance+1
,
                                m_garden_coordinates.top()-c_roadDistance+1,
                                c_driveWayWidth-1,
                                c_roadDistance-1);
        }
        else
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+c_roadDistance+1
,
                                1,
                                c_driveWayWidth-1,
                                m_top_object.getSize().height()-1);
        }
        if(m_bottom_object.getType()==NearestObjectsType::STREET)
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+c_roadDistance+1
,
m_garden_coordinates.y()+m_garden_coordinates.height()+1,
                                c_driveWayWidth-1,
                                c_roadDistance-1);
        }
        else
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+c_roadDistance+1
,
m_garden_coordinates.y()+m_garden_coordinates.height()+1,
                                c_driveWayWidth-1,
                                m_bottom_object.getSize().height()-1);
        }
    }

    if(m_bottom_object.getType()==NearestObjectsType::DRIVEWAY)
    {

        brush.setTexture(tmp);
        painter->setBrush(brush);
        painter->setPen(Qt::NoPen);
        painter->drawRect(m_garden_coordinates.left(),
                          m_garden_coordinates.y()+m_garden_coordinates.height()
+ c_roadDistance +1,
                          m_garden_coordinates.width(),
                          c_driveWayWidth-1);
        if(m_left_object.getType()==NearestObjectsType::STREET)
        {
            painter->drawRect(m_garden_coordinates.left()-c_roadDistance+1,

m_garden_coordinates.y()+m_garden_coordinates.height() + c_roadDistance+1,
                                c_roadDistance-1,
                                c_driveWayWidth-1);
        }
        else
        {
            painter->drawRect(1,

m_garden_coordinates.y()+m_garden_coordinates.height() + c_roadDistance+1,
                                m_left_object.getSize().width()-1,
```

```
                                c_driveWayWidth-1);
        }
        if(m_right_object.getType()==NearestObjectsType::STREET)
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+1,

m_garden_coordinates.y()+m_garden_coordinates.height() + c_roadDistance+1,
                                c_roadDistance-1,
                                c_driveWayWidth-1);
        }
        else
        {
            painter-
>drawRect(m_garden_coordinates.x()+m_garden_coordinates.width()+1,

m_garden_coordinates.y()+m_garden_coordinates.height() + c_roadDistance+1,
                                m_right_object.getSize().width()-1,
                                c_driveWayWidth-1);
        }
    }

    ////////////////////////////////////////////////
    //Отрисовка границ соседнего участка
    ////////////////////////////////////////////////

    pen.setStyle(Qt::DashLine);
    pen.setColor(QColor(140,70,20,255));
    pen.setWidth(3);
    painter->setPen(pen);

    if(m_left_object.getType()==NearestObjectsType::NEIGHBOUR )
    {
        painter->drawLine(m_garden_coordinates.x(),m_garden_coordinates.y(),
                          0,m_garden_coordinates.y());
        painter-
>drawLine(m_garden_coordinates.x(),m_garden_coordinates.y()+m_left_object.getSiz
e().height(),

0,m_garden_coordinates.y()+m_garden_coordinates.height());
    }

    if(m_top_object.getType()==NearestObjectsType::NEIGHBOUR )
    {
        painter->drawLine(m_garden_coordinates.x(),m_garden_coordinates.y(),
                          m_garden_coordinates.x(),0);
        painter-
>drawLine(m_garden_coordinates.x()+m_garden_coordinates.width(),m_garden_coordin
ates.y(),

m_garden_coordinates.x()+m_garden_coordinates.width(),0);
    }
    if(m_right_object.getType()==NearestObjectsType::NEIGHBOUR )
    {
        painter-
>drawLine(m_garden_coordinates.x()+m_garden_coordinates.width(),m_garden_coordin
ates.y(),

m_garden_coordinates.x()+m_garden_coordinates.width()+m_right_object.getSize().w
idth(),m_garden_coordinates.y());
        painter-
>drawLine(m_garden_coordinates.x()+m_garden_coordinates.width(),m_garden_coordin
ates.y()+m_garden_coordinates.height(),
```

```
m_garden_coordinates.x()+m_garden_coordinates.width()+m_right_object.getSize().w
idth(),m_garden_coordinates.y()+m_garden_coordinates.height());
    }

    if(m_bottom_object.getType()==NearestObjectsType::NEIGHBOUR )
    {
        painter-
>drawLine(m_garden_coordinates.x(),m_garden_coordinates.y()+m_garden_coordinates
.height(),

m_garden_coordinates.x(),m_garden_coordinates.y()+m_garden_coordinates.height()+
m_bottom_object.getSize().height());
        painter-
>drawLine(m_garden_coordinates.x()+m_garden_coordinates.width(),m_garden_coordin
ates.y()+m_garden_coordinates.height(),

m_garden_coordinates.x()+m_garden_coordinates.width(),m_garden_coordinates.y()+m
_garden_coordinates.height()+m_bottom_object.getSize().height());
    }




    painter->restore();
}

void GeneralPlan::adjustSize()
{
    m_total_size =
QSize(m_left_object.getSize().width()+m_size_garden_site.width()
                          +m_right_object.getSize().width(),

m_top_object.getSize().height()+m_size_garden_site.height()
                          +m_bottom_object.getSize().height());
}

NearestObjects & GeneralPlan::getLeftObject()
{
    return m_left_object;
}

NearestObjects & GeneralPlan::getRightObject()
{
    return m_right_object;
}

NearestObjects & GeneralPlan::getTopObject()
{
    return m_top_object;
}

NearestObjects & GeneralPlan::getBottomObject()
{
    return m_bottom_object;
}

void GeneralPlan::setLeftObjectData(uint distance, uint type, uint sideSize)
{
    m_left_object.setObjectData(distance,type,sideSize);
}
```

```
void GeneralPlan::setRightObjectData(uint distance, uint type, uint sideSize)
{
    m_right_object.setObjectData(distance,type,sideSize);
}

void GeneralPlan::setTopObjectData(uint distance, uint type, uint sideSize)
{
    m_top_object.setObjectData(distance,type,sideSize);
}

void GeneralPlan::setBottomObjectData(uint distance, uint type, uint sideSize)
{
    m_bottom_object.setObjectData(distance,type,sideSize);
}

void GeneralPlan::setSizeGardenSite(QSize size)
{
    m_size_garden_site = size;
}

QSize GeneralPlan::getTotalSize()
{
    return m_total_size;
}


void GeneralPlan::setGardenCoordinates()
{
    m_garden_coordinates =
QRect(m_left_object.getSize().width(),m_top_object.getSize().height(),

m_size_garden_site.width(),m_size_garden_site.height());
    qDebug()<<m_garden_coordinates;
}

QRect GeneralPlan::getGardenCoordinates()
{
    return m_garden_coordinates;
}

QLine GeneralPlan::getLeftSideLine()
{
    return QLine(m_garden_coordinates.x()-m_left_object.getDistance(),
                 m_garden_coordinates.y(),
                 m_garden_coordinates.x()-m_left_object.getDistance(),
                 m_garden_coordinates.y()+m_garden_coordinates.height());
}

QLine GeneralPlan::getRightSideLine()
{
    return QLine(m_garden_coordinates.x()+m_garden_coordinates.width() +
m_right_object.getDistance(),
                 m_garden_coordinates.y(),
                 m_garden_coordinates.x()+m_garden_coordinates.width() +
m_right_object.getDistance(),
                 m_garden_coordinates.y()+m_garden_coordinates.height());
}


QLine GeneralPlan::getTopSideLine()
{
    return QLine(m_garden_coordinates.x(),
                 m_garden_coordinates.y()-m_top_object.getDistance(),
```

```
                m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y()-m_top_object.getDistance());
}

QLine GeneralPlan::getBottomSideLine()
{
    return QLine(m_garden_coordinates.x(),
                m_garden_coordinates.y()+m_garden_coordinates.height() +
m_bottom_object.getDistance(),
                m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y()+m_garden_coordinates.height() +
m_bottom_object.getDistance());
}

QLine GeneralPlan::getLeftSideFence()
{
    return QLine(m_garden_coordinates.x(),
                m_garden_coordinates.y(),
                m_garden_coordinates.x(),
                m_garden_coordinates.y()+m_garden_coordinates.height());
}

QLine GeneralPlan::getRightSideFence()
{
    return QLine(m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y(),
                m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y()+m_garden_coordinates.height());
}

QLine GeneralPlan::getTopSideFence()
{
    return QLine(m_garden_coordinates.x(),
                m_garden_coordinates.y(),
                m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y());
}

QLine GeneralPlan::getBottomSideFence()
{
    return QLine(m_garden_coordinates.x(),
                m_garden_coordinates.y()+m_garden_coordinates.height(),
                m_garden_coordinates.x()+m_garden_coordinates.width(),
                m_garden_coordinates.y()+m_garden_coordinates.height());
}
```

Nearestobjects.cpp

```
#include "nearestobjects.h"

NearestObjects::NearestObjects()
{

}

NearestObjects::NearestObjects(Position object_position)
{
    m_distance = 0;
    m_size = QSize(0,0);
    m_type = NearestObjectsType::EMPTY;
    m_object_position = object_position;
```

```
    if(object_position == Position::LEFT)
    {
        m_forbidden_area_direction = Direction::RIGHT;
    }
    if(object_position == Position::TOP)
    {
        m_forbidden_area_direction = Direction::BOTTOM;
    }
    if(object_position == Position::RIGHT)
    {
        m_forbidden_area_direction = Direction::LEFT;
    }
    if(object_position == Position::BOTTOM)
    {
        m_forbidden_area_direction = Direction::TOP;
    }

}

void NearestObjects::setObjectData(uint distance,uint type,uint sideSize)
{
    m_distance = distance;
    m_type = static_cast<NearestObjectsType>(type);

    if(m_object_position == Position::RIGHT || m_object_position ==
Position::LEFT)
    {

        if(m_type==NearestObjectsType::EMPTY)
            {
                m_size = QSize(c_emptyObjectSize,sideSize);
            }
            if(m_type==NearestObjectsType::STREET)
            {
                m_size = QSize(c_roadDistance+c_streetObjectSize,sideSize);
            }
            if(m_type==NearestObjectsType::DRIVEWAY)
            {
                m_size = QSize(c_roadDistance+c_driveWayObjectSize,sideSize);
            }
            if(m_type==NearestObjectsType::NEIGHBOUR)
            {
                m_size = QSize(c_neighbourObjectSize,sideSize);
            }
    }
    else
    {
        qDebug()<<"А теперь тут";
        if(m_type==NearestObjectsType::EMPTY)
            {
                m_size = QSize(sideSize,c_emptyObjectSize);
            }
            if(m_type==NearestObjectsType::STREET)
            {
                m_size = QSize(sideSize,c_roadDistance+c_streetObjectSize);
            }
            if(m_type==NearestObjectsType::DRIVEWAY)
            {
                m_size = QSize(sideSize,c_roadDistance+c_driveWayObjectSize);
            }
            if(m_type==NearestObjectsType::NEIGHBOUR)
            {
                m_size = QSize(sideSize,c_neighbourObjectSize);
```

```
            }
        }
}

uint NearestObjects::getDistance()
{
    return m_distance;
}

QSize NearestObjects::getSize()
{
    return m_size;
}

NearestObjectsType NearestObjects::getType()
{
    return m_type;
}

Direction NearestObjects::getForbiddenAreaDirection()
{
    return m_forbidden_area_direction;
}
```

Restrictedarea.cpp

```
#include "restrictedarea.h"

RestrictedArea::RestrictedArea()
{

}

RestrictedArea::RestrictedArea(QRect rect, GardenFlags::Option flag)
{
    m_rect_Area = rect;
    m_flag = flag;
    m_mode = displayMode::EMPTY;

}

RestrictedArea::RestrictedArea(QRect rect, GardenFlags::Option flag, int radius)
{
    m_rect_Area = rect;
    m_flag = flag;
    m_Radius = radius;
    m_mode = displayMode::EMPTY;
}

void RestrictedArea::setMode(displayMode dm)
{
    m_mode = dm;
}

GardenFlags::Option RestrictedArea::getFlag()
{
    return m_flag;
}

displayMode RestrictedArea::getMode()
{
```

```
    return m_mode;
}

QRectF RestrictedArea::boundingRect() const
{
    return m_rect_Area;
}

QPainterPath RestrictedArea::shape() const
{
    if(m_flag == GardenFlags::MyHouse8 ||
       m_flag == GardenFlags::Sauna8 ||
       m_flag == GardenFlags::WaterCloset8 ||
       m_flag == GardenFlags::Compost8)
    {
        QPainterPath path;
        path.addRoundedRect(boundingRect(), m_Radius*c_cellSize,
m_Radius*c_cellSize);
        return path;
    }
    else if(m_flag == GardenFlags::Well8)
    {
        QPainterPath path;
        path.addEllipse(m_rect_Area);
        return path;
    }
    else
    {
        QPainterPath path;
        path.addRect(m_rect_Area);
        return path;
    }
}

void RestrictedArea::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    if(m_mode == displayMode::EMPTY)
    {
        QPen pen;
        pen.setStyle(Qt::NoPen);
        painter->setPen(pen);
        painter->drawPath(this->shape());
    }

    else if(m_mode == displayMode::BORDER)
    {
        QPen pen;
        QBrush brush;
        pen.setStyle(Qt::SolidLine);
        pen.setColor(QColor(Qt::red));
        brush.setStyle(Qt::Dense6Pattern);
        brush.setColor(Qt::red);
        painter->setPen(pen);
        painter->setBrush(brush);
        painter->drawPath(this->shape());
    }
    else if(m_mode == displayMode::FILLED)
    {
        QPen pen;
        QBrush brush;
        brush.setStyle(Qt::DiagCrossPattern);
        brush.setColor(Qt::red);
```

```
        pen.setStyle(Qt::SolidLine);
        pen.setColor(QColor(Qt::red));
        painter->setPen(pen);
        painter->setBrush(brush);
        painter->drawPath(this->shape());
    }

}
```

Dialoggeneralplan.cpp

```
#include "dialoggeneralplan.h"
#include "ui_dialoggeneralplan.h"

DialogGeneralPlan::DialogGeneralPlan(QWidget *parent) :
    QDialog(parent),
    ui(new Ui::DialogGeneralPlan)
{
    m_width = 20;
    m_height = 50;
    m_left_object_type = 0;
    m_left_object_distance = 0;
    m_top_object_type = 0;
    m_top_object_distance = 0;
    m_right_object_type = 0;
    m_right_object_distance = 0;
    m_bottom_object_type = 0;
    m_bottom_object_distance = 0;

    ui->setupUi(this);
    ui->CB_Left_Neighbour->addItem("Пустой участок", 0);
    ui->CB_Left_Neighbour->addItem("Улица", 1);
    ui->CB_Left_Neighbour->addItem("Проезд", 2);
    ui->CB_Left_Neighbour->addItem("Соседский участок", 3);

    ui->CB_Top_Neighbour->addItem("Пустой участок", 0);
    ui->CB_Top_Neighbour->addItem("Улица", 1);
    ui->CB_Top_Neighbour->addItem("Проезд", 2);
    ui->CB_Top_Neighbour->addItem("Соседский участок", 3);

    ui->CB_Right_Neighbour->addItem("Пустой участок", 0);
    ui->CB_Right_Neighbour->addItem("Улица", 1);
    ui->CB_Right_Neighbour->addItem("Проезд", 2);
    ui->CB_Right_Neighbour->addItem("Соседский участок", 3);

    ui->CB_Bottom_Neighbour->addItem("Пустой участок", 0);
    ui->CB_Bottom_Neighbour->addItem("Улица", 1);
    ui->CB_Bottom_Neighbour->addItem("Проезд", 2);
    ui->CB_Bottom_Neighbour->addItem("Соседский участок", 3);

    connect(ui-
>SB_Width,SIGNAL(valueChanged(int)),this,SLOT(slotSetWidth(int)));
    connect(ui-
>SB_Height,SIGNAL(valueChanged(int)),this,SLOT(slotSetHeight(int)));
    connect(ui-
>CB_Left_Neighbour,SIGNAL(currentIndexChanged(int)),this,SLOT(slotSetLeftObjectT
ype(int)));
    connect(ui-
>SB_RedLineDistanceLeft,SIGNAL(valueChanged(int)),this,SLOT(slotSetLeftObjectDis
tance(int)));
```

```
    connect(ui->CB_Top_Neighbour,SIGNAL(currentIndexChanged(int)),this,SLOT(slotSetTopObjectType(int)));
    connect(ui->SB_RedLineDistanceTop,SIGNAL(valueChanged(int)),this,SLOT(slotSetTopObjectDistance(int)));
    connect(ui->CB_Right_Neighbour,SIGNAL(currentIndexChanged(int)),this,SLOT(slotSetRightObjectType(int)));
    connect(ui->SB_RedLineDistanceRight,SIGNAL(valueChanged(int)),this,SLOT(slotSetRightObjectDistance(int)));
    connect(ui->CB_Bottom_Neighbour,SIGNAL(currentIndexChanged(int)),this,SLOT(slotSetBottomObjectType(int)));
    connect(ui->SB_RedLineDistanceBottom,SIGNAL(valueChanged(int)),this,SLOT(slotSetBottomObjectDistance(int)));
}

DialogGeneralPlan::~DialogGeneralPlan()
{
    delete ui;
}

void DialogGeneralPlan::slotSetWidth(int width)
{
    m_width = width;
}

void DialogGeneralPlan::slotSetHeight(int height)
{
    m_height = height;
}

void DialogGeneralPlan::slotSetLeftObjectType(int type)
{
    m_left_object_type = type;
}

void DialogGeneralPlan::slotSetLeftObjectDistance(int distance)
{
    m_left_object_distance = distance;
}

void DialogGeneralPlan::slotSetTopObjectType(int type)
{
    m_top_object_type = type;
}

void DialogGeneralPlan::slotSetTopObjectDistance(int distance)
{
    m_top_object_distance = distance;
}

void DialogGeneralPlan::slotSetRightObjectType(int type)
{
    m_right_object_type = type;
}

void DialogGeneralPlan::slotSetRightObjectDistance(int distance)
{
    m_right_object_distance = distance;
```

```
}

void DialogGeneralPlan::slotSetBottomObjectType(int type)
{
    m_bottom_object_type = type;
}

void DialogGeneralPlan::slotSetBottomObjectDistance(int distance)
{
    m_bottom_object_distance = distance;
}

int DialogGeneralPlan::getWidth()
{
    return m_width;
}

int DialogGeneralPlan::getHeight()
{
    return m_height;
}

int DialogGeneralPlan::getLeftObjectType()
{
    return m_left_object_type;
}

int DialogGeneralPlan::getLeftObjectDistance()
{
    return m_left_object_distance;
}

int DialogGeneralPlan::getTopObjectType()
{
    return m_top_object_type;
}

int DialogGeneralPlan::getTopObjectDistance()
{
    return m_top_object_distance;
}

int DialogGeneralPlan::getRightObjectType()
{
    return m_right_object_type;
}

int DialogGeneralPlan::getRightObjectDistance()
{
    return m_right_object_distance;
}

int DialogGeneralPlan::getBottomObjectType()
{
    return m_bottom_object_type;
}

int DialogGeneralPlan::getBottomObjectDistance()
{
    return m_bottom_object_distance;
}
```

Drivewayredline.cpp

```
#include "drivewayredline.h"

DriveWayRedLine::DriveWayRedLine()
{

}

DriveWayRedLine::DriveWayRedLine(QLine line, Direction direction)
{
    m_line = line;
    if(direction == Direction::LEFT)
    {
        m_rect5 = QRect(m_line.x1()-
5*c_cellSize,m_line.y1(),5*c_cellSize,m_line.y2()-m_line.y1());
        m_rect3 = QRect(m_line.x1()-
3*c_cellSize,m_line.y1(),3*c_cellSize,m_line.y2()-m_line.y1());
    }
    if(direction == Direction::TOP)
    {
        m_rect5 = QRect(m_line.x1(),m_line.y1()-5*c_cellSize,m_line.x2()-
m_line.x1(),5*c_cellSize);
        m_rect3 = QRect(m_line.x1(),m_line.y1()-3*c_cellSize,m_line.x2()-
m_line.x1(),3*c_cellSize);
    }
    if(direction == Direction::RIGHT)
    {
        m_rect5 = QRect(m_line.x1(),m_line.y1(),5*c_cellSize,m_line.y2()-
m_line.y1());
        m_rect3 = QRect(m_line.x1(),m_line.y1(),3*c_cellSize,m_line.y2()-
m_line.y1());
    }
    if(direction == Direction::BOTTOM)
    {
        m_rect5 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),5*c_cellSize);
        m_rect3 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),3*c_cellSize);
    }

    m_restrictedArea5 = new
RestrictedArea(m_rect5,GardenFlags::DriveWayRedLine5);
    m_restrictedArea3 = new
RestrictedArea(m_rect3,GardenFlags::DriveWayRedLine3);

}

QVector<RestrictedArea *> DriveWayRedLine::getRestrictedAreas()
{
    restricted_areas.push_back(m_restrictedArea5);
    restricted_areas.push_back(m_restrictedArea3);
    return restricted_areas;
}


QRectF DriveWayRedLine::boundingRect() const
{
    return
QRectF(QPointF(m_line.x1(),m_line.y1()),QPointF(m_line.x2(),m_line.y2()));
}
```

```
void DriveWayRedLine::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    m_color = Qt::red;
    m_penWidth = 2;
    pen.setColor(m_color);
    pen.setWidth(m_penWidth);
    painter->setPen(pen);
    painter->drawLine(m_line);
}
```

Fence.cpp

```
#include "fence.h"

Fence::Fence()
{

}

Fence::Fence(QLine line, Direction direction)
{
    m_line = line;
    if(direction == Direction::LEFT)
    {
        m_rect2 = QRect(m_line.x1()-
2*c_cellSize,m_line.y1(),2*c_cellSize,m_line.y2()-m_line.y1());
    }
    if(direction == Direction::TOP)
    {
        m_rect2 = QRect(m_line.x1(),m_line.y1()-2*c_cellSize,m_line.x2()-
m_line.x1(),2*c_cellSize);
    }
    if(direction == Direction::RIGHT)
    {
        m_rect2 = QRect(m_line.x1(),m_line.y1(),2*c_cellSize,m_line.y2()-
m_line.y1());
    }
    if(direction == Direction::BOTTOM)
    {
        m_rect2 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),2*c_cellSize);
    }

    m_restrictedArea2 = new RestrictedArea(m_rect2,GardenFlags::Fence2);
}

QVector<RestrictedArea *> Fence::getRestrictedAreas()
{
    restricted_areas.push_back(m_restrictedArea2);
    return restricted_areas;
}


QRectF Fence::boundingRect() const
{
    return
QRectF(QPointF(m_line.x1(),m_line.y1()),QPointF(m_line.x2(),m_line.y2()));
}
```

```
void Fence::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    pen.setStyle(Qt::NoPen);
    painter->setPen(pen);
    painter->drawLine(m_line);
}
```

Neighbourborder.cpp

```
#include "neighbourborder.h"

NeighbourBorder::NeighbourBorder()
{

}

NeighbourBorder::NeighbourBorder(QLine line, Direction direction)
{
    m_line = line;
    if(direction == Direction::LEFT)
    {
        m_rect1 = QRect(m_line.x1()-
1*c_cellSize,m_line.y1(),1*c_cellSize,m_line.y2()-m_line.y1());
        m_rect2 = QRect(m_line.x1()-
2*c_cellSize,m_line.y1(),2*c_cellSize,m_line.y2()-m_line.y1());
        m_rect3 = QRect(m_line.x1()-
3*c_cellSize,m_line.y1(),3*c_cellSize,m_line.y2()-m_line.y1());
        m_rect4 = QRect(m_line.x1()-
4*c_cellSize,m_line.y1(),4*c_cellSize,m_line.y2()-m_line.y1());
    }
    if(direction == Direction::TOP)
    {
        m_rect1 = QRect(m_line.x1(),m_line.y1()-1*c_cellSize,m_line.x2()-
m_line.x1(),1*c_cellSize);
        m_rect2 = QRect(m_line.x1(),m_line.y1()-2*c_cellSize,m_line.x2()-
m_line.x1(),2*c_cellSize);
        m_rect3 = QRect(m_line.x1(),m_line.y1()-3*c_cellSize,m_line.x2()-
m_line.x1(),3*c_cellSize);
        m_rect4 = QRect(m_line.x1(),m_line.y1()-4*c_cellSize,m_line.x2()-
m_line.x1(),4*c_cellSize);
    }
    if(direction == Direction::RIGHT)
    {
        m_rect1 = QRect(m_line.x1(),m_line.y1(),1*c_cellSize,m_line.y2()-
m_line.y1());
        m_rect2 = QRect(m_line.x1(),m_line.y1(),2*c_cellSize,m_line.y2()-
m_line.y1());
        m_rect3 = QRect(m_line.x1(),m_line.y1(),3*c_cellSize,m_line.y2()-
m_line.y1());
        m_rect4 = QRect(m_line.x1(),m_line.y1(),4*c_cellSize,m_line.y2()-
m_line.y1());
    }
    if(direction == Direction::BOTTOM)
    {
        m_rect1 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),1*c_cellSize);
        m_rect2 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),2*c_cellSize);
        m_rect3 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),3*c_cellSize);
```

```
        m_rect4 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),4*c_cellSize);
    }

    m_restrictedArea1 = new
RestrictedArea(m_rect1,GardenFlags::NeighbourBorder1);
    m_restrictedArea2 = new
RestrictedArea(m_rect2,GardenFlags::NeighbourBorder2);
    m_restrictedArea3 = new
RestrictedArea(m_rect3,GardenFlags::NeighbourBorder3);
    m_restrictedArea4 = new
RestrictedArea(m_rect4,GardenFlags::NeighbourBorder4);
}

QVector<RestrictedArea *> NeighbourBorder::getRestrictedAreas()
{
    restricted_areas.push_back(m_restrictedArea1);
    restricted_areas.push_back(m_restrictedArea2);
    restricted_areas.push_back(m_restrictedArea3);
    restricted_areas.push_back(m_restrictedArea4);
    return restricted_areas;
}

void NeighbourBorder::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    pen.setStyle(Qt::NoPen);
    painter->setPen(pen);
    painter->drawLine(m_line);
}



QRectF NeighbourBorder::boundingRect() const
{
    return
QRectF(QPointF(m_line.x1(),m_line.y1()),QPointF(m_line.x2(),m_line.y2()));
}
```

Streetredline.cpp

```
#include "streetredline.h"

StreetRedLine::StreetRedLine()
{

}

StreetRedLine::StreetRedLine(QLine line, Direction direction)
{
    m_line = line;
    if(direction == Direction::LEFT)
    {
        m_rect5 = QRect(m_line.x1()-
5*c_cellSize,m_line.y1(),5*c_cellSize,m_line.y2()-m_line.y1());
    }
    if(direction == Direction::TOP)
    {
```

```
        m_rect5 = QRect(m_line.x1(),m_line.y1()-5*c_cellSize,m_line.x2()-
m_line.x1(),5*c_cellSize);
    }
    if(direction == Direction::RIGHT)
    {
        m_rect5 = QRect(m_line.x1(),m_line.y1(),5*c_cellSize,m_line.y2()-
m_line.y1());
    }
    if(direction == Direction::BOTTOM)
    {
        m_rect5 = QRect(m_line.x1(),m_line.y1(),m_line.x2()-
m_line.x1(),5*c_cellSize);
    }

    m_restrictedArea5 = new RestrictedArea(m_rect5,GardenFlags::StreetRedLine5);
}

QVector<RestrictedArea*> StreetRedLine::getRestrictedAreas()
{
    restricted_areas.push_back(m_restrictedArea5);
    return restricted_areas;

}



void StreetRedLine::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    m_color = Qt::red;
    m_penWidth = 2;
    pen.setColor(m_color);
    pen.setWidth(m_penWidth);
    painter->setPen(pen);
    painter->drawLine(m_line);
}



QRectF StreetRedLine::boundingRect() const
{
    return
QRectF(QPointF(m_line.x1(),m_line.y1()),QPointF(m_line.x2(),m_line.y2()));
}
```

Bigtree.cpp

```
#include "bigtree.h"

BigTree::BigTree()
{
    m_radius = 2;
    m_rect = QRect(0,0,m_radius*c_cellSize*2,m_radius*c_cellSize*2);
    m_center_Point = QPoint(m_radius*c_cellSize,m_radius*c_cellSize);
    pixmap.load("tree.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
}
```

```
QPixmap BigTree::getPixmap()
{
    return pixmap;
}


QPoint BigTree::getPoint()
{
    return m_center_Point;
}



QRectF BigTree::boundingRect() const
{
    return m_rect;
}

void BigTree::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{

    painter->drawPixmap(m_rect,pixmap);
}

QPainterPath BigTree::shape() const
{
    QPainterPath path;
    path.addEllipse(m_rect);
    return path;
}
```

Bush.cpp

```
#include "bush.h"

Bush::Bush()
{
    m_radius = 2;
    m_rect = QRect(0,0,m_radius*c_cellSize*2,m_radius*c_cellSize*2);
    m_center_Point = QPoint(m_radius*c_cellSize,m_radius*c_cellSize);
    pixmap.load("Bush.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
}

QPixmap Bush::getPixmap()
{
    return pixmap;
}

QPoint Bush::getPoint()
{
    return m_center_Point;
}


QRectF Bush::boundingRect() const
{
    return m_rect;
```

```
}

QPainterPath Bush::shape() const
{
    QPainterPath path;
    path.addEllipse(m_rect);
    return path;
}

void Bush::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}
```

Compost.cpp

```
#include "compost.h"

Compost::Compost()
{
    m_width = 2 * c_cellSize;
    m_height = 2 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("compost.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
    QRect rA = QRect(m_rect.x()-8*c_cellSize,m_rect.y()-
8*c_cellSize,m_width+8*2*c_cellSize,m_height+8*2*c_cellSize);
    m_compostArea8 = new RestrictedArea(rA,GardenFlags::WaterCloset8,8);
}

QVector<RestrictedArea *> Compost::getRestrictedAreas()
{
    restricted_areas.push_back(m_compostArea8);
    return restricted_areas;
}


QRectF Compost::boundingRect() const
{
    return m_rect;
}

void Compost::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}

QPixmap Compost::getPixmap()
{
    return pixmap;
}

QPoint Compost::getPoint()
{
    return m_center_Point;
}
```

Glasshouse.cpp

```
#include "glasshouse.h"

Glasshouse::Glasshouse()
{
    m_width = 4 * c_cellSize;
    m_height = 2 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("glasshouse.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
}


QRectF Glasshouse::boundingRect() const
{
    return m_rect;
}

void Glasshouse::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}


QPixmap Glasshouse::getPixmap()
{
    return pixmap;
}

QPoint Glasshouse::getPoint()
{
    return m_center_Point;
}
```

Henhouse.cpp

```
#include "henhouse.h"

Henhouse::Henhouse()
{
    m_width = 3 * c_cellSize;
    m_height = 3 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("henhouse.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
}


QRectF Henhouse::boundingRect() const
{
    return m_rect;
}
```

```
void Henhouse::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}

QPixmap Henhouse::getPixmap()
{
    return pixmap;
}

QPoint Henhouse::getPoint()
{
    return m_center_Point;
}
```

House.cpp

```
#include "house.h"

House::House()
{
    m_width = 8 * c_cellSize;
    m_height = 6 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("roof.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
    QRect rA = QRect(m_rect.x()-8*c_cellSize,m_rect.y()-
8*c_cellSize,m_width+8*2*c_cellSize,m_height+8*2*c_cellSize);
    m_houseArea8 = new RestrictedArea(rA,GardenFlags::MyHouse8,8);
}

QPixmap House::getPixmap()
{
    return pixmap;
}

QPoint House::getPoint()
{
    return m_center_Point;
}

QVector<RestrictedArea *> House::getRestrictedAreas()
{
    restricted_areas.push_back(m_houseArea8);
    return restricted_areas;
}

QRectF House::boundingRect() const
{
    return m_rect;
}

void House::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}
```

Midsizetree.cpp

```
#include "midsizetree.h"

MidsizeTree::MidsizeTree()
{
    m_radius = 1.5;
    m_rect = QRect(0,0,m_radius*c_cellSize*2,m_radius*c_cellSize*2);
    m_center_Point = QPoint(m_radius*c_cellSize,m_radius*c_cellSize);
    pixmap.load("fir.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
}



QPixmap MidsizeTree::getPixmap()
{
    return pixmap;
}

QPoint MidsizeTree::getPoint()
{
    return m_center_Point;
}


QRectF MidsizeTree::boundingRect() const
{
    return m_rect;
}

QPainterPath MidsizeTree::shape() const
{
    QPainterPath path;
    path.addEllipse(m_rect);
    return path;
}

void MidsizeTree::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}
```

Sauna.cpp

```
#include "sauna.h"

Sauna::Sauna()
{
    m_width = 4 * c_cellSize;
    m_height = 3 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("sauna.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
```

```
    QRect rA = QRect(m_rect.x()-8*c_cellSize,m_rect.y()-
8*c_cellSize,m_width+8*2*c_cellSize,m_height+8*2*c_cellSize);
    m_saunaArea8 = new RestrictedArea(rA,GardenFlags::Sauna8,8);
}

QPixmap Sauna::getPixmap()
{
    return pixmap;
}

QPoint Sauna::getPoint()
{
    return m_center_Point;
}

QVector<RestrictedArea *> Sauna::getRestrictedAreas()
{
    restricted_areas.push_back(m_saunaArea8);
    return restricted_areas;
}


QRectF Sauna::boundingRect() const
{
    return m_rect;
}

void Sauna::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}
```

Shed.cpp

```
#include "shed.h"

Shed::Shed()
{
    m_width = 4 * c_cellSize;
    m_height = 2 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("shed.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());

}


QRectF Shed::boundingRect() const
{
    return m_rect;
}

void Shed::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}


QPixmap Shed::getPixmap()
```

```
{
    return pixmap;
}

QPoint Shed::getPoint()
{
    return m_center_Point;
}
```

Watercloset.cpp

```
#include "watercloset.h"

Watercloset::Watercloset()
{
    m_width = 2 * c_cellSize;
    m_height = 2 * c_cellSize;
    m_rect = QRect(0,0,m_width,m_height);
    m_center_Point = QPoint(m_rect.center());
    pixmap.load("watercloset.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
    QRect rA = QRect(m_rect.x()-8*c_cellSize,m_rect.y()-
8*c_cellSize,m_width+8*2*c_cellSize,m_height+8*2*c_cellSize);
    m_waterclosetArea8 = new RestrictedArea(rA,GardenFlags::WaterCloset8,8);
}

QVector<RestrictedArea *> Watercloset::getRestrictedAreas()
{
    restricted_areas.push_back(m_waterclosetArea8);
    return restricted_areas;
}


QRectF Watercloset::boundingRect() const
{
    return m_rect;
}

void Watercloset::paint(QPainter *painter, const QStyleOptionGraphicsItem
*option, QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}


QPixmap Watercloset::getPixmap()
{
    return pixmap;
}

QPoint Watercloset::getPoint()
{
    return m_center_Point;
}
```

Well.cpp

```
#include "well.h"

Well::Well()
{
    m_radius = 1;
    m_rect = QRect(0,0,m_radius*c_cellSize*2,m_radius*c_cellSize*2);
    m_center_Point = QPoint(m_radius*c_cellSize,m_radius*c_cellSize);
    pixmap.load("well.png");
    pixmap = pixmap.scaled(m_rect.width(),m_rect.height());
    QRect rA = QRect(m_rect.x()-8*c_cellSize,m_rect.y()-
8*c_cellSize,m_radius*2*c_cellSize+8*2*c_cellSize,m_radius*2*c_cellSize+8*2*c_ce
llSize);
    m_wellArea8 = new RestrictedArea(rA,GardenFlags::Well8);
}

QVector<RestrictedArea *> Well::getRestrictedAreas()
{
    restricted_areas.push_back(m_wellArea8);
    return restricted_areas;
}


QRectF Well::boundingRect() const
{
    return m_rect;
}

QPainterPath Well::shape() const
{
    QPainterPath path;
    path.addEllipse(m_rect);
    return path;
}

void Well::paint(QPainter *painter, const QStyleOptionGraphicsItem *option,
QWidget *widget)
{
    painter->drawPixmap(m_rect,pixmap);
}

QPixmap Well::getPixmap()
{
    return pixmap;
}

QPoint Well::getPoint()
{
    return m_center_Point;
}
```

Garden_plan.pro

```
QT       += core gui

greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
```

```
CONFIG += c++11

# The following define makes your compiler emit warnings if you use
# any Qt feature that has been marked deprecated (the exact warnings
# depend on your compiler). Please consult the documentation of the
# deprecated API in order to know how to port your code away from it.
DEFINES += QT_DEPRECATED_WARNINGS

# You can also make your code fail to compile if it uses deprecated APIs.
# In order to do so, uncomment the following line.
# You can also select to disable deprecated APIs only up to a certain version of
Qt.
#DEFINES += QT_DISABLE_DEPRECATED_BEFORE=0x060000    # disables all the APIs
deprecated before Qt 6.0.0

SOURCES += \
    FixedObjects/fence.cpp \
    FixedObjects/neighbourborder.cpp \
    Objects/bush.cpp \
    Objects/compost.cpp \
    Objects/glasshouse.cpp \
    Objects/henhouse.cpp \
    Objects/house.cpp \
    Objects/bigtree.cpp \
    Objects/midsizetree.cpp \
    Objects/sauna.cpp \
    Objects/shed.cpp \
    Objects/watercloset.cpp \
    Objects/well.cpp \
    dialoggeneralplan.cpp \
    dragwidgetscene.cpp \
    FixedObjects/drivewayredline.cpp \
    gardenflags.cpp \
    gardenscene.cpp \
    generalplan.cpp \
    main.cpp \
    mainwindow.cpp \
    nearestobjects.cpp \
    restrictedarea.cpp \
    FixedObjects/streetredline.cpp \


HEADERS += \
    Constants.h \
    FixedObjects/fence.h \
    FixedObjects/neighbourborder.h \
    Objects/bush.h \
    Objects/compost.h \
    Objects/gardenobjects.h \
    Objects/glasshouse.h \
    Objects/henhouse.h \
    Objects/house.h \
    Objects/bigtree.h \
    Objects/midsizetree.h \
    Objects/sauna.h \
    Objects/shed.h \
    Objects/watercloset.h \
    Objects/well.h \
    dialoggeneralplan.h \
    dragwidgetscene.h \
    FixedObjects/drivewayredline.h \
    gardenflags.h \
```

```
    gardenscene.h \
    generalplan.h \
    mainwindow.h \
    nearestobjects.h \
    restrictedarea.h \
    FixedObjects/streetredline.h \


FORMS += \
    dialoggeneralplan.ui \
    mainwindow.ui

# Default rules for deployment.
qnx: target.path = /tmp/$${TARGET}/bin
else: unix:!android: target.path = /opt/$${TARGET}/bin
!isEmpty(target.path): INSTALLS += target

RESOURCES += \
    resources.qrc
```