**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по учебной практике**

Тема: Поиск маршрута на карте

|  |  |  |
| --- | --- | --- |
| Студенты гр. 2372 | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_  \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Васильев Ю.А.  Мельникова М.А.  Лемешко А.Д. |
| Преподаватель | \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ | Титов Г.С. |

Санкт-Петербург

2024

**ЗАДАНИЕ**

**на УЧЕБНую практику**

|  |  |  |  |
| --- | --- | --- | --- |
| Студенты: | Васильев Ю.А.  Мельникова М.А.  Лемешко А.Д. | | |
| Группа 2372 | | | |
| Тема учебной практики: Поиск маршрута на карте | | | |
| Задание на учебную практику:  Требуется разработать приложение, позволяющее:  создавать карты препятствий;  сохранять карту препятствий в файл;  задавать стартовую и финишную точки маршрута;  прокладывать кратчайший маршрут из стартовой точки в финишную;  отображать построенный маршрут на карте препятствий;  сохранять построенный маршрут в файл. | | | |
| Сроки прохождения практики: 04.03.2024 – 17.05.2024 | | | |
| Дата сдачи отчета: 15.05.2024 | | | |
| Дата защиты отчета: 17.05.2024 | | | |
|  | | | |
| Студенты | |  | Васильев Ю.А.  Мельникова М.А.  Лемешко А.Д. |
| Руководитель | |  | Титов Г.С. |

**Аннотация**

В ходе данной учебной практики необходимо разработать приложение, позволяющее создавать двумерные карты, наносить на них препятствия: полигоны с заданным индексом снижения скорости его пересечения, сохранять карту препятствий и найденный маршрут, открывать сохранённые карты, добавлять две точки на карте: стартовую и финишную и прокладывать между ними самый оптимальный маршрут.

**Summary**

During this training practice, it is necessary to develop an application that allows you to create two-dimensional maps, put obstacles on them: polygons with a given index of decreasing the speed of its intersection, save the obstacle map and the route found, open saved maps, add two points on the map: the starting and finishing points and plot the most optimal route between them.

**содержание**

|  |  |  |
| --- | --- | --- |
|  | Введение | 5 |
| 1. | Постановка задачи | 6 |
| 1.1. | Описание средств решения поставленной задачи | 6 |
| 1.2. | Обоснование выбора языка программирования и объектных библиотек конструкторских классов | 6 |
| 2. | Диаграммы модели приложения | 7 |
| 2.1. | Диаграмма прецедентов | 7 |
| 2.2. | Диаграмма классов | 8 |
| 2.3. | Диаграмма последовательности | 9 |
| 3. | Используемые алгоритмы | 10 |
| 3.1. | Алгоритм Дейкстры, A\* | 10 |
| 3.2. | Алгоритм сглаживания | 13 |
| 4. | Хранение данных | 15 |
| 4.1 | XML-файлы | 15 |
| 5. | Интерфейс приложения | 17 |
| 5.1. | Основное окно | 17 |
| 5.2. | Панель управления и редактирования | 17 |
| 5.3. | Окно графического отображения | 18 |
| 6. | Тестирование | 22 |
|  | Заключение |  |
|  | Список использованных источников |  |

**введение**

Цель: разработать приложение, позволяющее искать кратчайший маршрут между двумя точками, учитывая поставленные препятствия.

Задачи:

1. создавать карты препятствий;
2. сохранять карту препятствий в файл;
3. задавать стартовую и финишную точки маршрута;
4. прокладывать кратчайший маршрут из стартовой точки в финишную;
5. отображать построенный маршрут на карте препятствий;
6. сохранять построенный маршрут в файл.

**1. постановка задачи**

* 1. **Описание средств решения поставленной задачи**

Для реализации поставленной задачи предстоит разработать оконное приложение, содержащее поле для создания, отображения и редактирования двумерной карты. Пользовательский интерфейс, содержащий кнопки: добавление препятствия, удаление препятствия, редактирование препятствия, добавление/редактирование старта, добавление/редактирование финиша. Также необходимо написать алгоритм, учитывающий не только наличие препятствий на пути, но и коэффициент их непроходимости, сравнивающий маршруты, огибающие препятствия и проходящие сквозь них.

**1.2. Обоснование выбора языка программирования и объектных библиотек конструкторских классов**

В ходе выполнения курсовой работы используются рекомендованные для применения язык программирования и объектные библиотеки конструкторских классов: C++ и Qt.

**2. Диаграммы модели приложения**

**2.1. Диаграмма прецедентов**

Для наглядного представления пользовательских сценариев на универсальном языке моделирования UML создана диаграмма прецедентов (рис. 1).

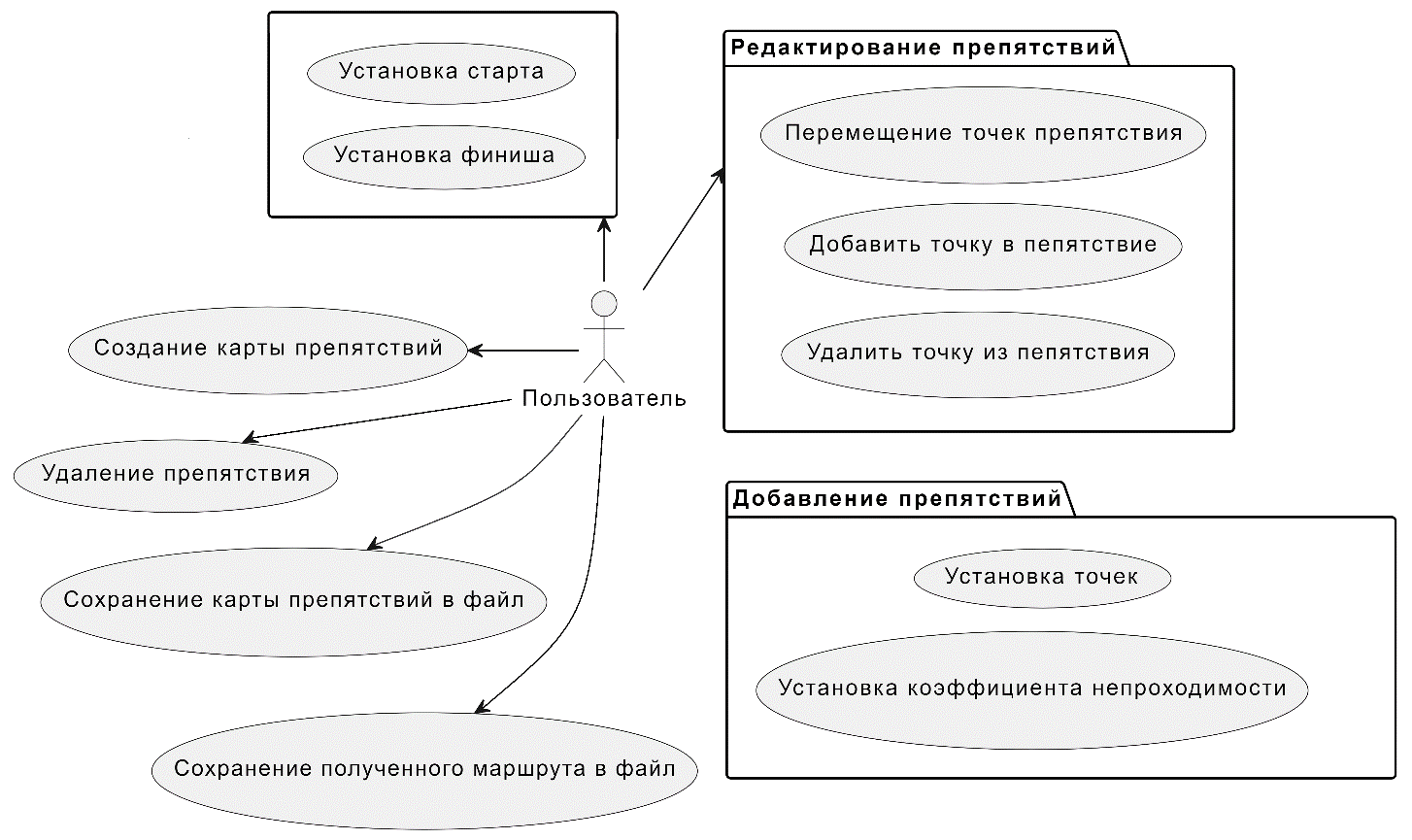


Рисунок 1 – Диаграмма прецедентов

**2.2. Диаграмма классов**

Далее по ходу планирования и разработки создана UML диаграмма классов для схематического отображения их связей (рис. 2).

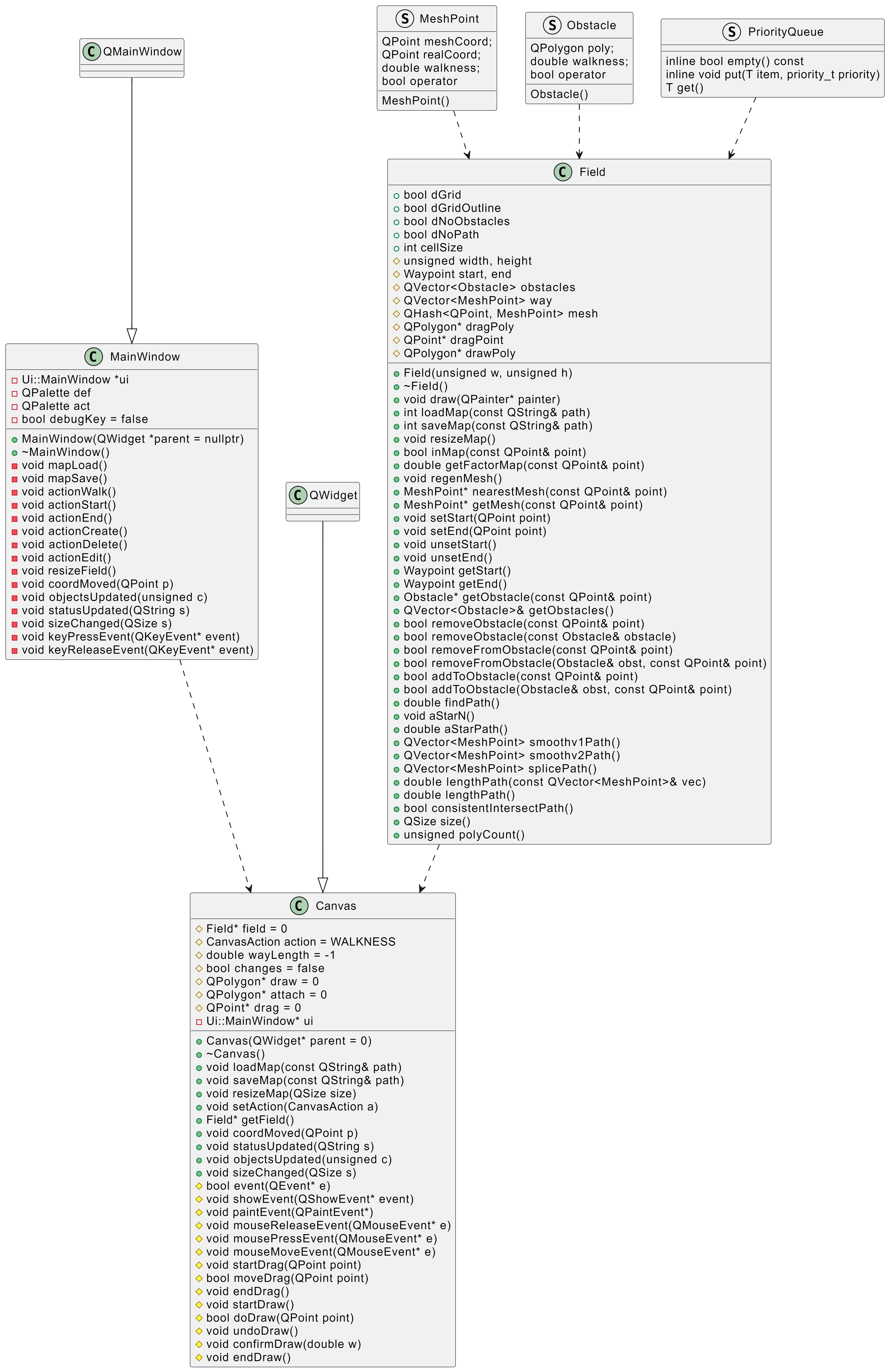
****

Рисунок 2 – Диаграмма классов

**2.3. Диаграмма последовательности**

В процессе разработки и тестирования для удобства понимания последовательности процессов и взаимодействий создана UML диаграмма последовательности (рис. 3).

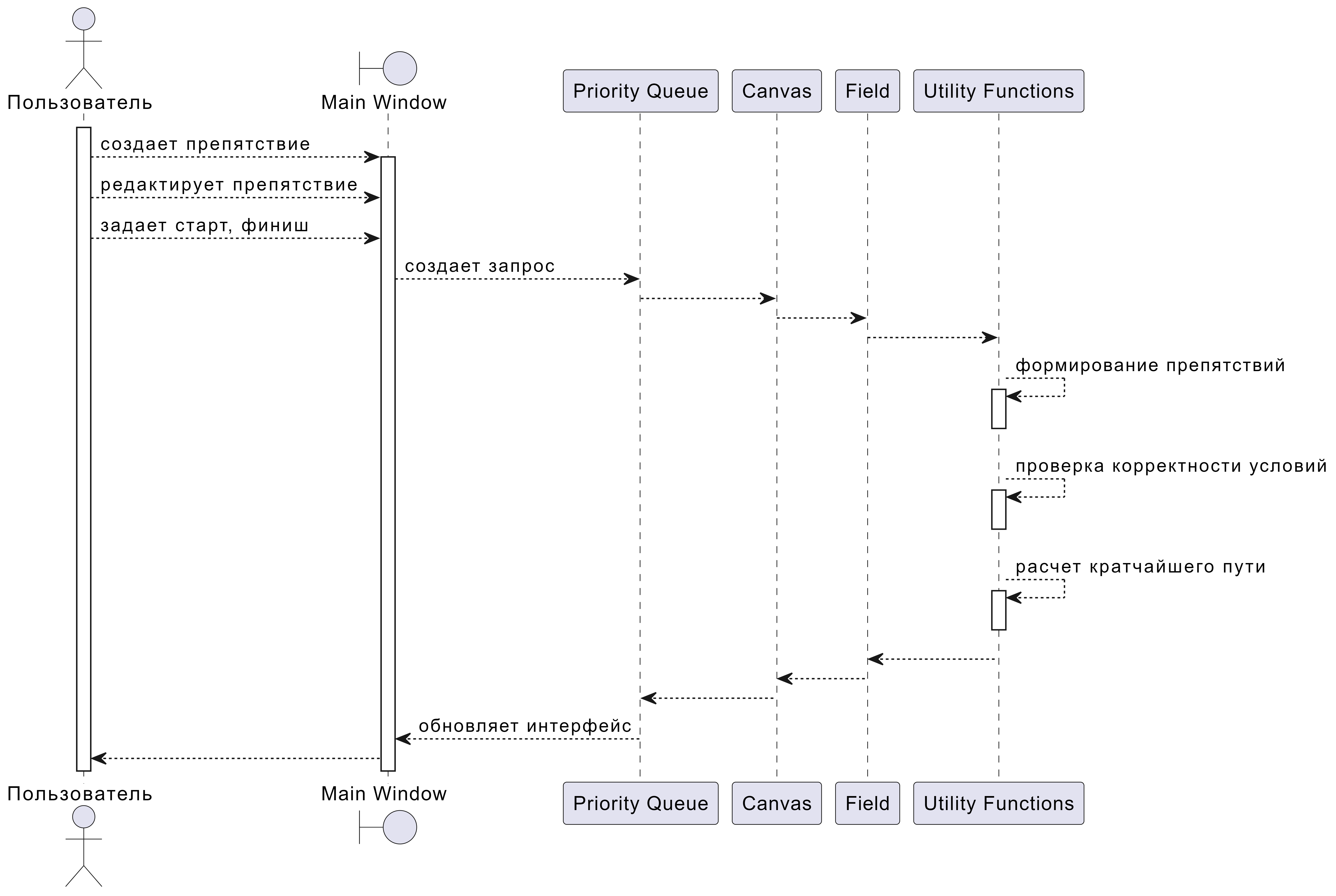


Рисунок 2 – Диаграмма последовательности

**3. используемые Алгоритмы**

**3.1. Алгоритм Дейкстры, A\***

В нашей работе строится меш-сетка на основе препятствий. Каждая клетка сетки содержит координату сетки, координату на холсте и проходимость. Благодаря этому реализуется улучшенная версия алгоритма Дейкстры, A\*. Мы выбрали именно этот алгоритм т.к. он больше всего подходит под нашу задачу поиска кратчайшего пути от начальной точки до единственной конечной.

Для начала разберём алгоритм Дейкстры, который сосредоточивается на поиске пути до нескольких целей:

В начале алгоритма создаётся приоритетная очередь, где каждая точка на нашей "сетке" имеет приоритет типа double, и в неё ставится начальный элемент пути:

PriorityQueue<MeshPoint\*, double> queue;

queue.put(start, 1.);

Приоритеты являются ключевым элементом в алгоритме Дейкстры, А\*, с их помощью алгоритм сначала проходит по областям сетки с наименьшей непроходимостью (самый низкий приоритет). Т.е. чем больше непроходимость, тем выше значение приоритета и тем позже эта точка будет обработана.

Замечание: то, что первый элемент в нашей очереди имеет наименьший приоритет, является различием алгоритма Дейкстры и алгоритма А\*, эта модификация необходима в дальнейших расчётах

Далее мы используем структуры хэш-таблиц из Qt для того, чтобы для каждой точки обозначить кратчайшую "стоимость прохождения пути" от начала до этой точки (хэш-таблица costs) и в какую сторону пойти, чтобы вернуться до начала, так называемая "точка возврата" (хэш-таблица origins)

QHash<QPoint, QPoint> origins;

QHash<QPoint, double> costs;

origins[start->meshCoord] = start->meshCoord;

costs[start->meshCoord] = 1.;

Очевидно, что для стартовой точки "точки возврата" не существует, поэтому устанавливаем ей её же координаты.

while (!queue.empty()) {

MeshPoint\* current = queue.get();

if (current == finish) break;

Здесь мы извлекаем первый элемент с наивысшим приоритетом и кладём его соседей в очередь в определённом порядке в зависимости от позиции:

if ((current->meshCoord.x() + current->meshCoord.y()) % 2 == 0) {

aStarN(queue, origins, costs, current, finish, QPoint(0, 1));

aStarN(queue, origins, costs, current, finish, QPoint(0, -1));

aStarN(queue, origins, costs, current, finish, QPoint(-1, 0));

aStarN(queue, origins, costs, current, finish, QPoint(1, 0));

} else {

aStarN(queue, origins, costs, current, finish, QPoint(1, 0));

aStarN(queue, origins, costs, current, finish, QPoint(-1, 0));

aStarN(queue, origins, costs, current, finish, QPoint(0, -1));

aStarN(queue, origins, costs, current, finish, QPoint(0, 1));

}

}

Если сумма координаты X и Y кратна 2, то мы добавляем "соседей" в очередь в таком порядке: север, юг, запад, восток;

Если же она не равна 2, то добавляем соседей в обратном порядке.

Необходимо понимать, что на "сетке" существует множество различных путей пройти от одной точки до другой, если эти точки лежат на диагонали.

Надо учитывать, что порядок добавления соседей влияет на внешний вид пути, и т.к. пользователю необходимо видеть красивый путь, мы добавляем данные манипуляции для нахождения такого пути, который содержит в себе как можно больше диагоналей.

Теперь рассмотрим алгоритм добавления соседа в очередь (функция aStarN):

if (!mesh.contains(off)) return;

Не добавляем соседа если он лежит вне пределов карты.

MeshPoint\* neighbor = &mesh[off];

if (neighbor->walkness >= 1.) return;

double new\_cost = costs[current->meshCoord] + vectorLength(offset) + neighbor->walkness;

if (!costs.contains(neighbor->meshCoord) || new\_cost < costs[neighbor->meshCoord]) {

Если непроходимость соседа равна 1, то не добавляем его, т.к. 1 считается "стеной".

Далее считаем новую стоимость прохождения от предыдущих клеток до соседа (new\_cost) и, если стоимость не записана вообще (это значит, что через эту клетку не лежит ни один путь) или наша найденная стоимость меньше (мы нашли путь короче), то тогда ведём запись стоимостей и "точек возврата":

costs[neighbor->meshCoord] = new\_cost;

origins[neighbor->meshCoord] = current->meshCoord;

Далее ставим соседа в очередь:

queue.put(neighbor, new\_cost);

Здесь как раз проявляется различие алгоритма Дейкстры и алгоритма А\*. В то время как алгоритм Дейкстры ищет расстояние до нескольких целей, алгоритм А\* фокусируется на одной цели. В связи с этим в приоритет очереди добавляется ещё и "расстояние до конечной точки", т.е. алгоритм будет в первую очередь рассматривать точки, которые лежат ближе всего к концу пути.

queue.put(neighbor, new\_cost + simpleDistance(neighbor->meshCoord, finish->meshCoord));

После обработки всех соседей или после раннего выхода из обработчика соседей мы наконец можем искать путь.

QPoint current = finish->meshCoord;

if (!origins.contains(current)) return 0;

double cost = costs[finish->meshCoord];

Начиная с конца идём по хэш-таблице "точек возврата". Если в этой хэш-таблице нет значения для финиша, то это значит, что программа не смогла найти путь из-за препятствий.

while (current != start->meshCoord) {

MeshPoint meshp = mesh[current];

way.append(meshp);

current = origins[current];

}

После чего добавляем в финальный массив точек путей начальную точку, и переворачиваем массив.

way.append(\*start);

std::reverse(way.begin(), way.end());

**3.1. Алгоритм сглаживания**

В нашей программе для удобства пользователя путь "сглаживается". Из-за того, что поиск был выполнен на мелкой сетке, а нам необходимо найти путь, "логичный для человеческого глаза", в элементах пути содержатся искажения, которые на сетке представляют собой верное решение, а не на сетке представляют собой неточности.

Алгоритм сглаживания на вход принимает вектор const QVector<MeshPoint>& vec, состоящий из всех точек кратчайшего пути, полученного алгоритмом A\*. Далее идет проверка: если вектор пуст - сглаживать нечего, функция возвращает этот же пустой вектор. Если вектор не пуст, то:

1) Создается новый пустой вектор QVector <MeshPoint> finalVec, в который мы будем добавлять необходимые точки для сглаживания пути

2) Создаем переменную int curr, равную нулю. Эта переменная будет обозначать начало каждой линии нашего кратчайшего пути

3) Добавляем в finalvec первую точку из vec ( finalVec.append(vec[curr]) )

4) цикл по всем i от 1 до длины vec ( i < vec.lehgth, т.к. i будет индексом второй точки линии, которая берётся из vec)

5) Создаем линию, где первая точка - реальные координаты vec[curr], а вторая - реальные координаты vec[i]

6) цикл по всем j от 0 до количества препятствий на карте - 1 ( j < obstacles.length(), т.к. j - индексы)

7) С помощью функции consistentIntersectPath(line, obstacles[j]) проверяем, пересекает ли линия какое либо препятствие, или точки находятся в нем:

7.1) Если не пересекает, то продолжаем цикл

7.2) Если пересекает, то добавляем в finalVec точку с индексом i-1, текущей точке curr присваиваем значение i-1 (индекс последней точки, которая не пересекалась), а также завершаем цикл по j.

Выше мы применили функцию bool Field::consistentIntersectPath(const QLine& line, const Obstacle& obst). Она помогает сглаживать путь точек, находящихся внутри полигонов препятствий. Рассмотрим алгоритм её работы:

1) На вход принимает линию const QLine& line и препятствие const Obstacle& obst

2) В double w1 и double w2 записываются проходимости точек линии

3) Если обе точки вне препятствия, то в return записывается значение функции lineIntersectsPolygon(line, obst.poly), в которой проверяется, пересекает ли линия полигон препятствия

4) Если обе точки в препятствии, то возвращается false

5) Иначе возвращается true, т.к. линия очевидно пересекает препятствие (одна точка вне препятствия, другая в нем)

**4. хранение данных**

**4.1. XML-файлы**

Результатом работы пользователя в программе может быть карта препятствий или найденный маршрут. Для сохранения результатов работы в файл нами был разработан соответствующий формат XML- файла. Структура XML-файла содержит в себе описание размеров карты (map), в ней описание списка препятствий (polygons), каждое препятствие (polygon) содержит в себе параметр непроходимости (walkness) и описание всех точки полигона point (и их координаты x, y)  
Также содержит описание сгенерированного сглаженного пути (path), где путь описан точками point с координатами x, y. Каждое значение проверяется в программе и не может выходить за границы дозволенного диапазона (например, ширина/ высота между 100 и 2000), и т.д.

Вот пример XML-файла для пустой карты

<?xml version="1.0" encoding="UTF-8"?>

<map width="800" height="500">

<polygons/>

<path/>

</map>

Пример XML-файла для карты с двумя препятствиями, точками старта и финиша

<?xml version="1.0" encoding="UTF-8"?>

<map width="800" height="500">

<polygons>

<poly walkness="0.5">

<point x="112" y="35"/>

<point x="59" y="125"/>

<point x="80" y="275"/>

<point x="124" y="370"/>

<point x="156" y="89"/>

</poly>

<poly walkness="0.7">

<point x="319" y="258"/>

<point x="310" y="172"/>

<point x="755" y="177"/>

<point x="765" y="273"/>

</poly>

</polygons>

<path>

<point x="360" y="316"/>

<point x="132" y="316"/>

<point x="130" y="316"/>

<point x="100" y="316"/>

<point x="98" y="316"/>

<point x="36" y="316"/>

</path>

<start x="37" y="316"/>

<end x="360" y="316"/>

</map>

**5. ИНТЕРФЕЙС ПРИЛОЖЕНИЯ**

**5.1. Основное окно**

Приложение имеет одно окно, в котором реализованы все необходимые функции. При открытии мы видим кнопки, с помощью которых осуществляется взаимодействие (1), пустое поле карты (2), подсказки (3), размер поля (4), информацию о количестве объектов и текущих координатах (5, 6), статус (7).

Для удобства на каждый элемент интерфейса можно навести курсор и появятся подсказки с указанием того, что этот элемент делает.

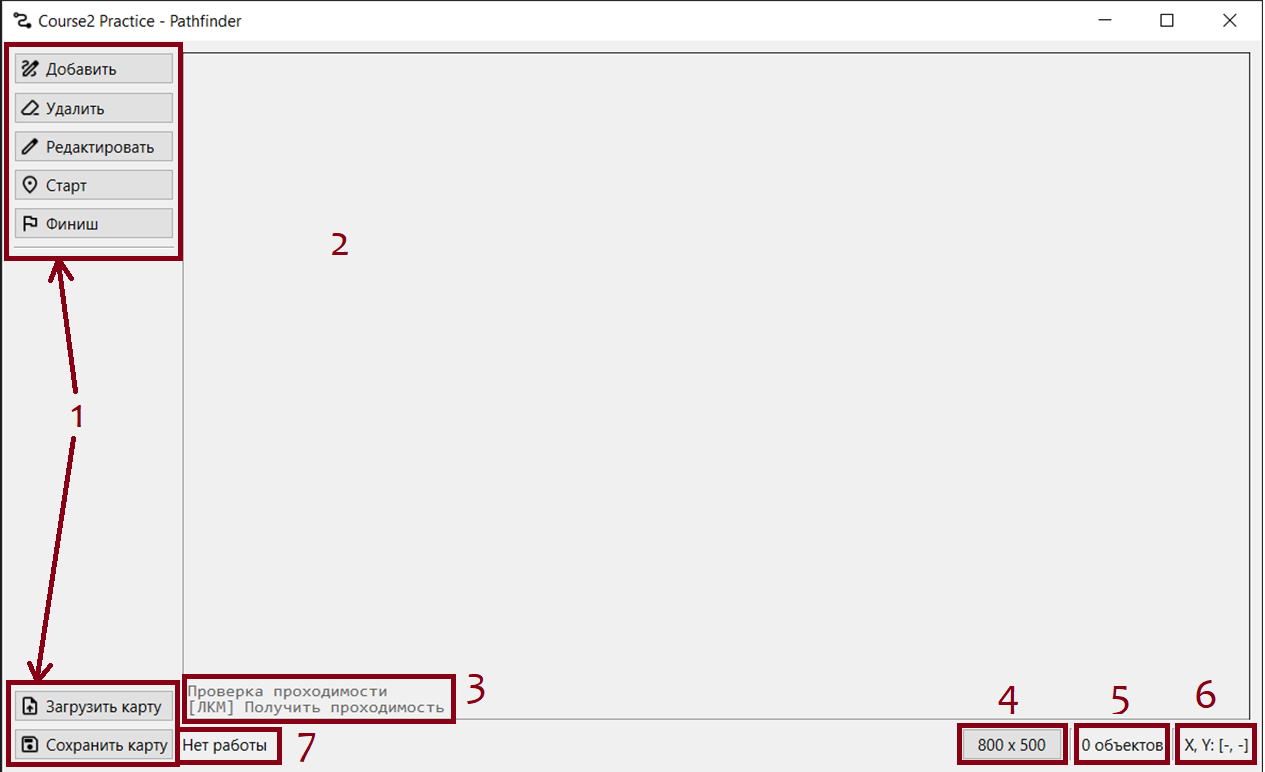


Рисунок 3 – Окно приложения

**5.2. Панель управления и редактирования**

Кнопками контроля управления (рис. 4) можно взаимодействовать с объектами на карте. Также есть возможность горячих клавиш от 1 до 6 для удобства редактирования карты:

|  |  |
| --- | --- |
| 1 – Проверка проходимости  2 – Создание препятствия  3 – Удаление препятствия  4 – Изменение препятствия  5 – Установка старта  6 – Установка финиша | Рисунок 4 |

При работе с инструментами управления на главном экране в левом нижнем углу будет отображаться подсказка с текущим инструментом и его управлением (рис. 7).



Рисунок 5 – Подсказка к текущему пункту меню

А в левом верхнем углу будет отображаться длина найденного пути.



Рисунок 6

В панели статуса можно увидеть строку статуса (слева), где будут показываться результаты действий управления (например, "успешно" или "точка не найдена").



Рисунок 7

Справа есть текст-кнопка (рис. 3, элемент 4), одновременно позволяющий поменять размеры карты и показывающий текущие размеры карты. Границы ширины и высоты ограниченны значениями от 100 до 2000.

Также есть индикатор текущих координат, который отображает координаты курсора мыши на карте, и индикатор количества препятствий (рис. 3, элементы 5, 6).

**5.3. Окно графического отображения**

Рассмотрим добавление препятствия на карту.  
*Левый клик мыши* на карте ставит/добавляет точку, формируя препятствие-многоугольник (полигон).  
Можно отменить установку последней точки *нажатием левой кнопки мыши с зажатым Shift*.  
*Правый клик мыши* завершает создание препятствия и предлагает пользователю указать непроходимость в открывшемся окне.  
*Правый клик мыши с зажатым Shift* отменяет создание препятствия и переходит в режим проверки проходимости.

Препятствие не может пересекаться с другими и обязано иметь индекс непроходимости от 1% до 100%, где препятствие со 100% непроходимостью считается "стеной". При постановке менее чем трёх точек препятствие не будет создано.

Рассмотрим удаление препятствия. Необходимо войти в режим удаления препятствий с карты.  
*Левый клик мыши* по любому препятствию удаляет его.  
*Правый клик мыши* выходит из режима удаления препятствия и переходит в режим проверки проходимости.

Изменение препятствий: начать редактирование существующих препятствий на карте.  
*Зажатие левого клика мыши* на карте по точке многоугольника берёт эту точку в захват, позволяя перемещать её по карте. Перемещение точки не допускает создания пересечений на карте.  
Можно добавить точку нажатием *левой кнопки мыши с зажатым Shift* в многоугольнике. Это действие разделит ближайшее ребро на две части с новой точкой.  
*Правый клик мыши* завершает редактирование и переходит в режим проверки проходимости.  
*Правый клик мыши с зажатым Shift* удаляет точку рядом с курсором. Курсор должен лежать в многоугольнике той точки, которую вы собираетесь удалить.

Установить точку старта/финиша на карте.  
*Левый клик мыши* устанавливает точку старта/финиша, либо заменяет её, если она уже существует.  
*Правый клик мыши с зажатым Shift* убирает точку старта/финиша.  
*Правый клик мыши* завершает редактирование и переходит в режим проверки проходимости.

После установки старта и финиша ищется кратчайший путь между этими двумя точками.  
Длина пути будет отображена в верхнем левом углу главного экрана.

Путь не может быть найден, если:

* Точка старта/финиша лежит в препятствии со 100% непроходимостью
* Точка старта и точка финиша разделены препятствием со 100% непроходимостью В таких случаях в верхнем левом углу появится надпись "путь не найден" (рис. 8).

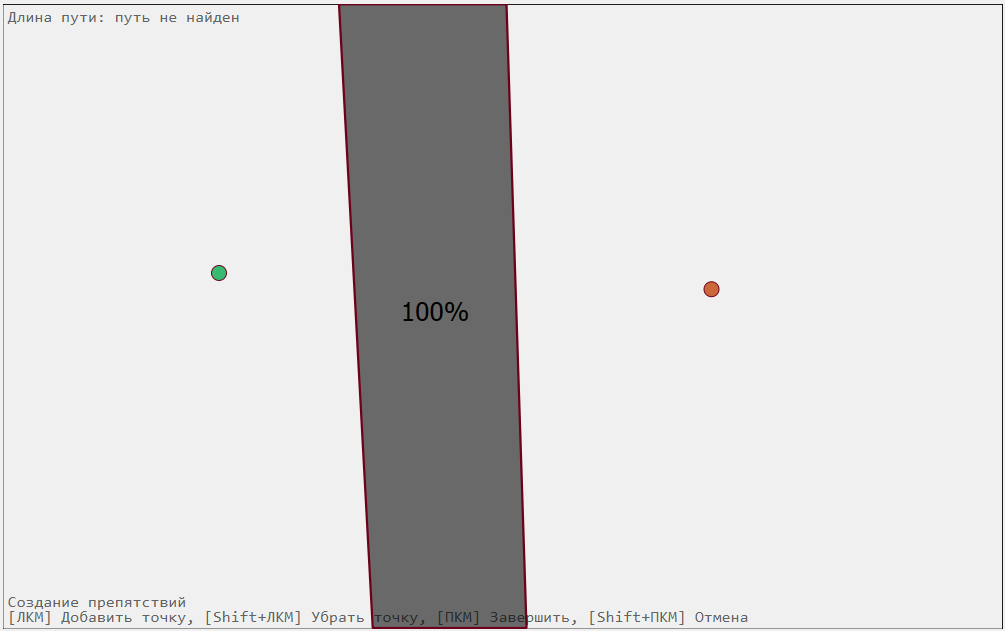


Рисунок 9 – Пример ситуации, когда маршрут не найден

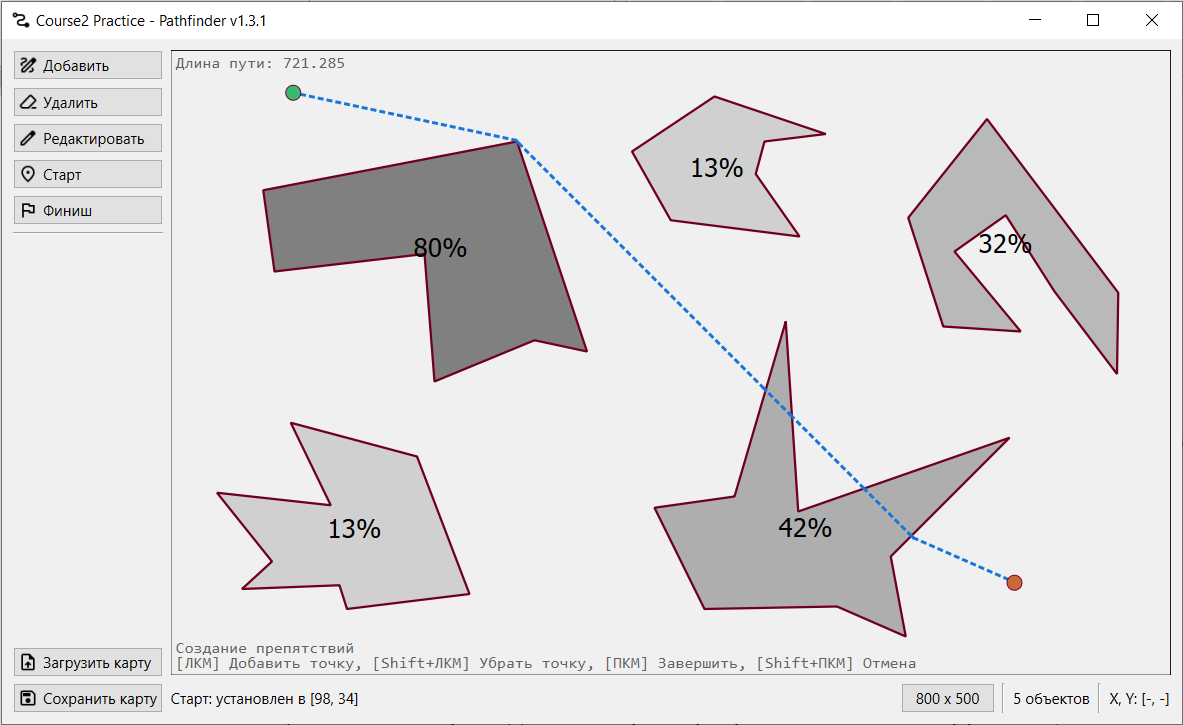


Рисунок 9 – Пример маршрута

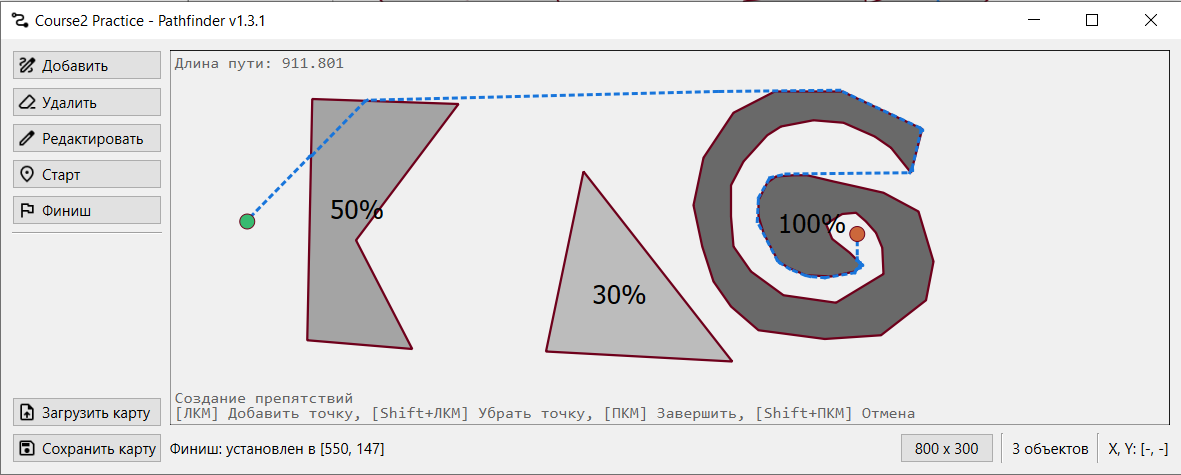


Рисунок 10 – Пример маршрута

В целях разработки были добавлены сочетания клавиш, позволяющие отладить программу (кнопка отладки - D):

* D + G Enable/disable mesh grid
* D + Shift + G Enable/disable mesh grid outline
* D + O Enable/disable obstacle drawing
* D + P Enable/disable path drawing
* D + Arrow Up Raise cell size
* D + Shift + Arrow Up Raise cell size without mesh generation
* D + Arrow Down Lower cell size
* D + Shift + Arrow Down Lower cell size without mesh generation
* D + M Regenerate mesh

**6. ТЕСТИРОВАНИЕ**

Рассмотрим контрольный пример.

В новом пустом поле установим несколько препятствий разной проходимости.

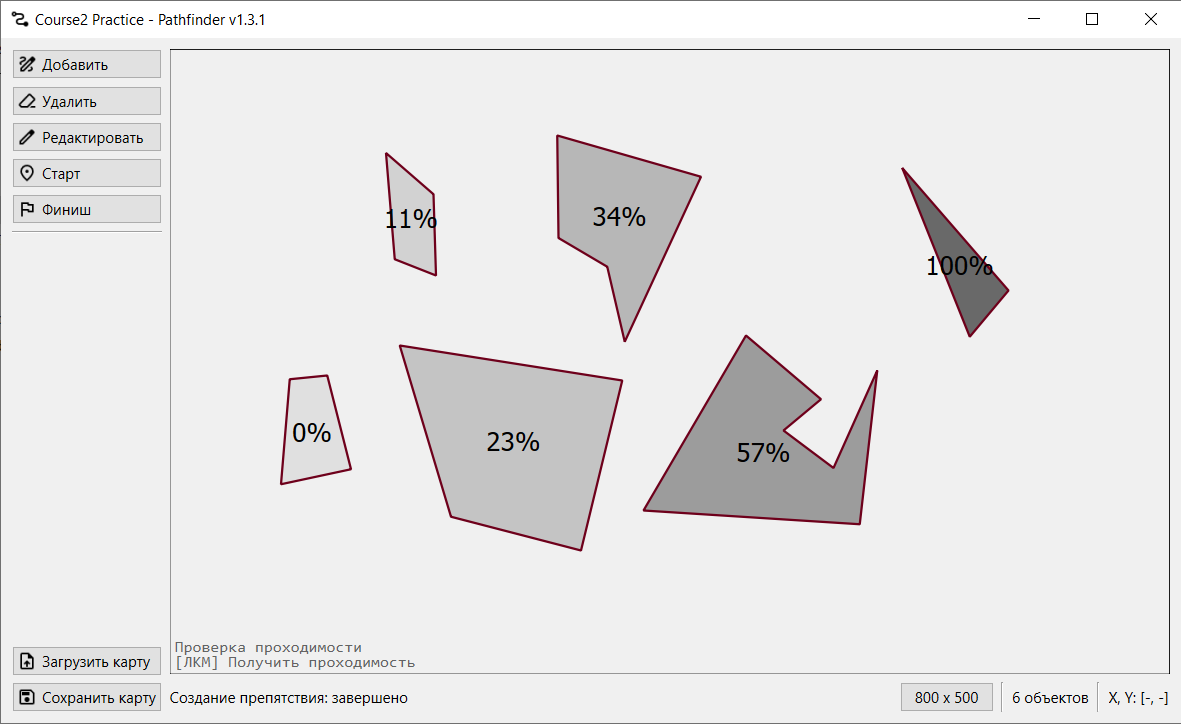


Рисунок 11

Препятствия с любой проходимостью от 0 до 100% устанавливаются. Препятствия с отрицательным коэффициентом непроходимости или с коэффициентом больше 1 задать нельзя.

Попробуем построить маршрут, где на линии старт-финиш будут препятствия с очень маленьким коэффициентом непроходимости и с достаточно большим.

Результат – небольшие препятствия с большой непроходимостью огибаются, препятствия с небольшой непроходимостью проходятся насквозь независимо от размера.

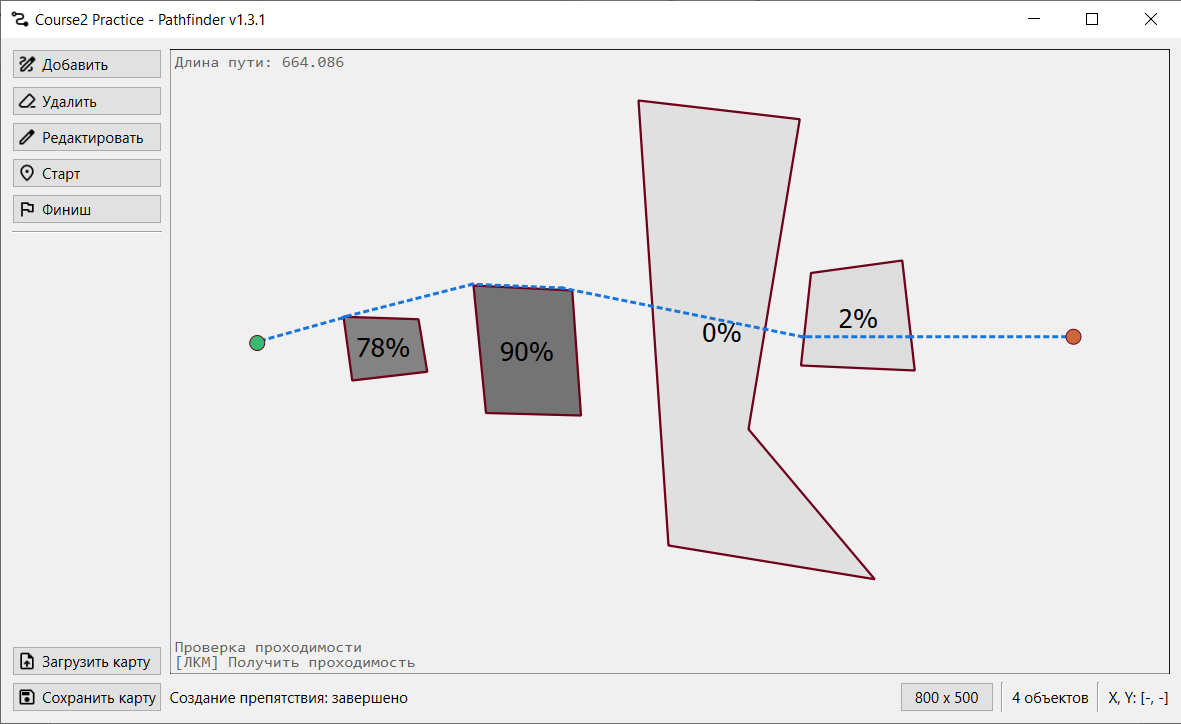


Рисунок 12

Теперь создадим непроницаемое препятствие. Поставим одну из точек маршрута внутрь, а другую снаружи. Видим, что для двух изолированных друг от друга точек программа выдает сообщение «путь не найден». Похожая ситуация показана на рисунке 9. Как видим, результат работы приложения совпадает с ожидаемым.

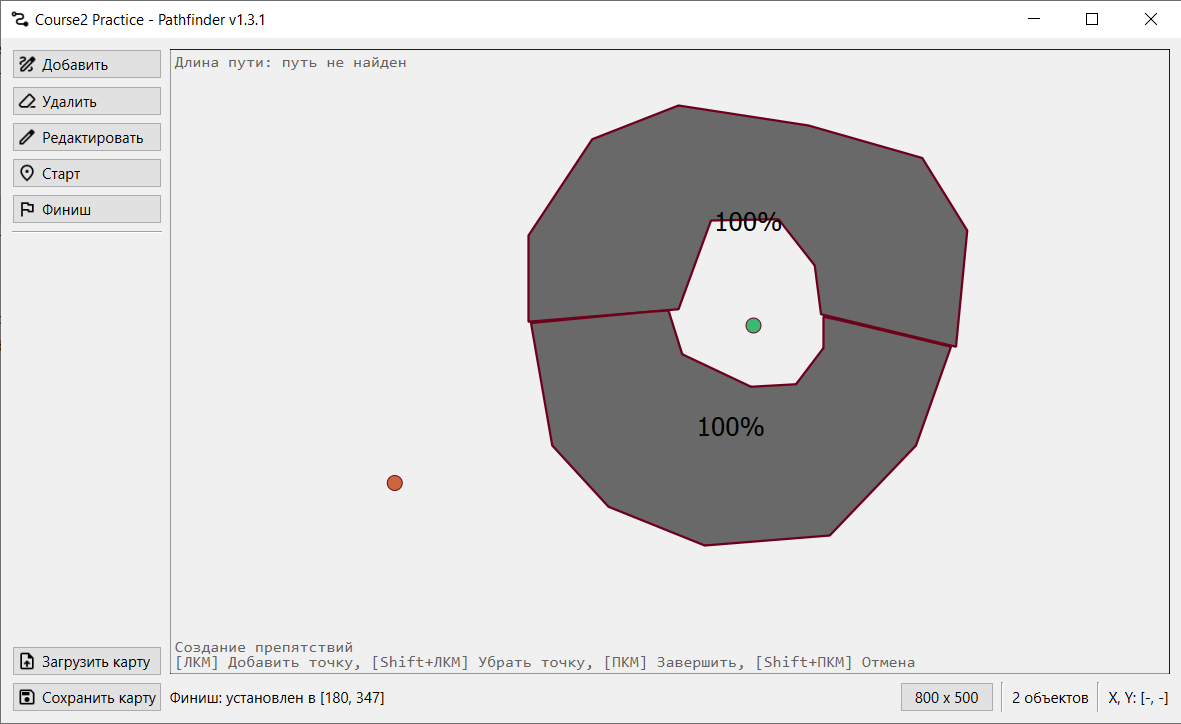


Рисунок 13

Попробуем изменить размер карты. Он меняется в заявленном диапазоне от 100 до 2000, только на целочисленные значения. Если мы захотим изменить масштаб карты, на которую уже нанесены препятствия или точки маршрута, то все данные сбросятся и поле с измененным масштабом будет чистым. Создадим маршрут. Сохраним его.

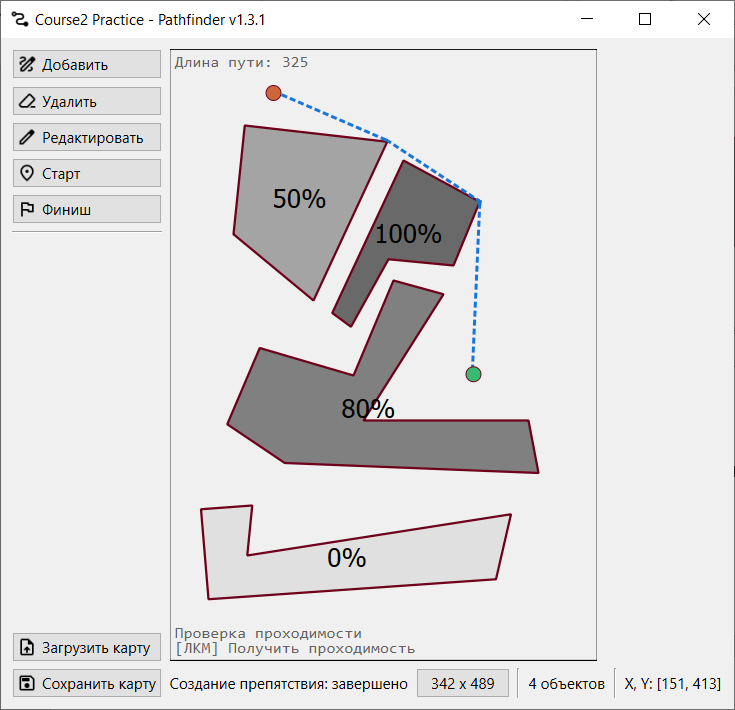


Рисунок 14

Попробуем открыть сохраненный файл.

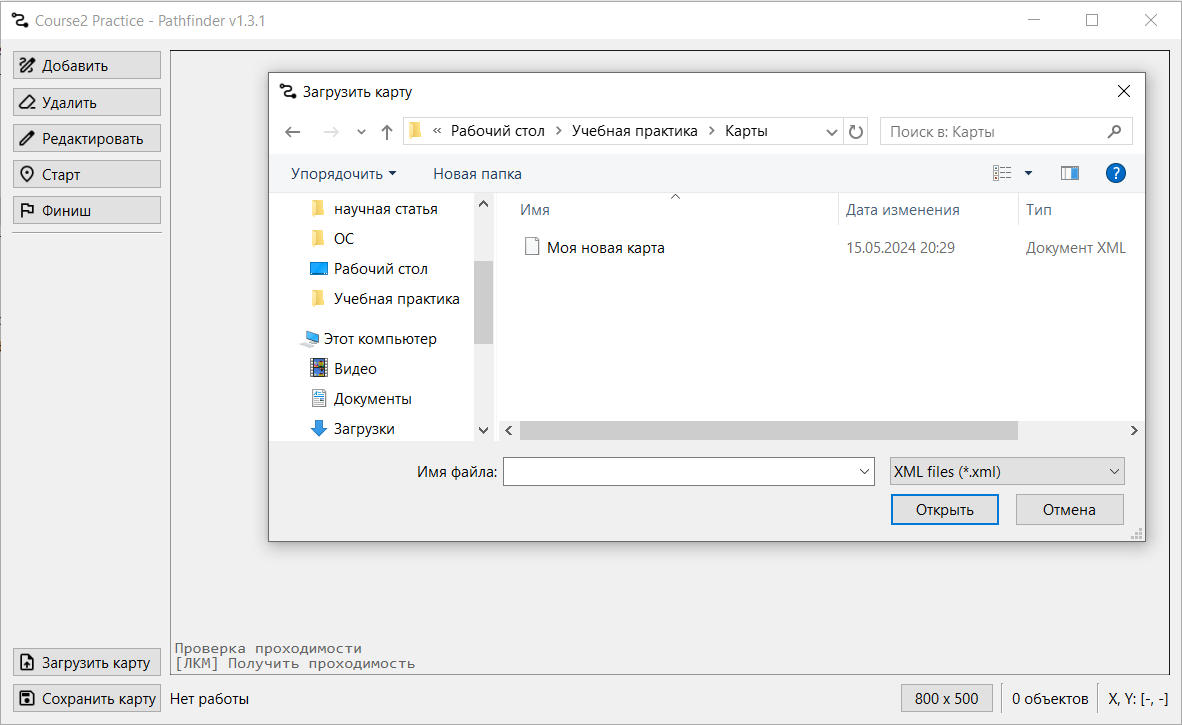


Рисунок 15

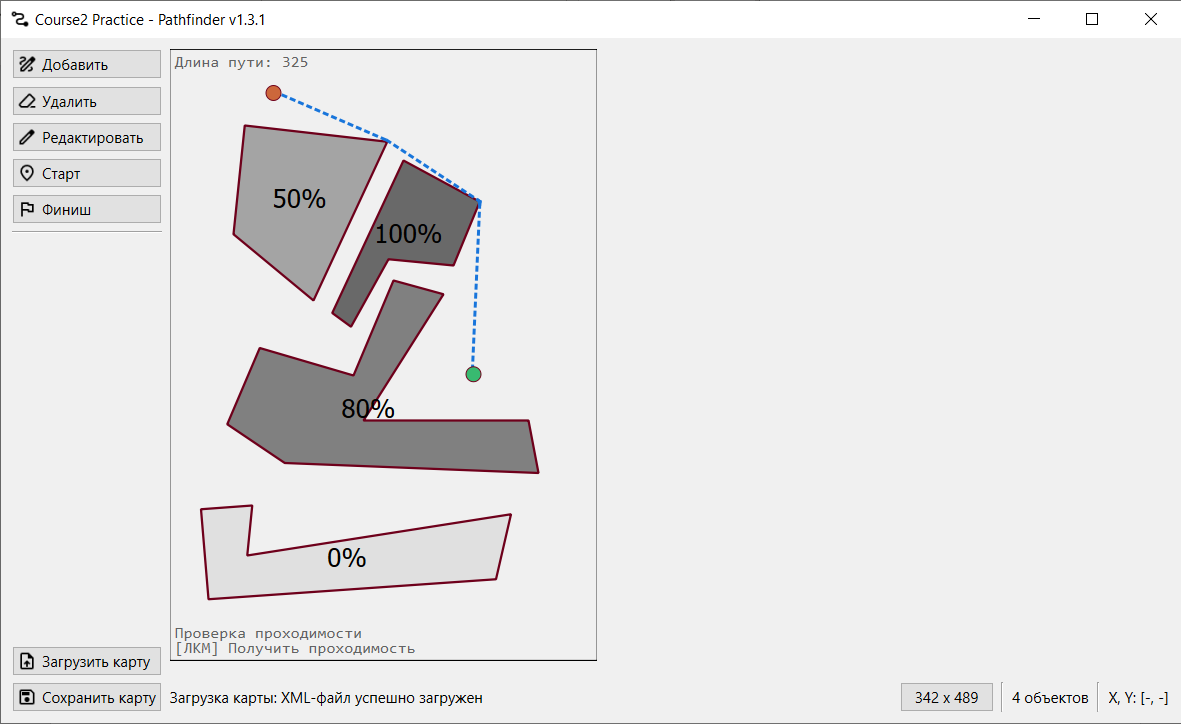


Рисунок 16

Файл был успешно открыт. И в дальнейшем поддаётся последующему редактированию.

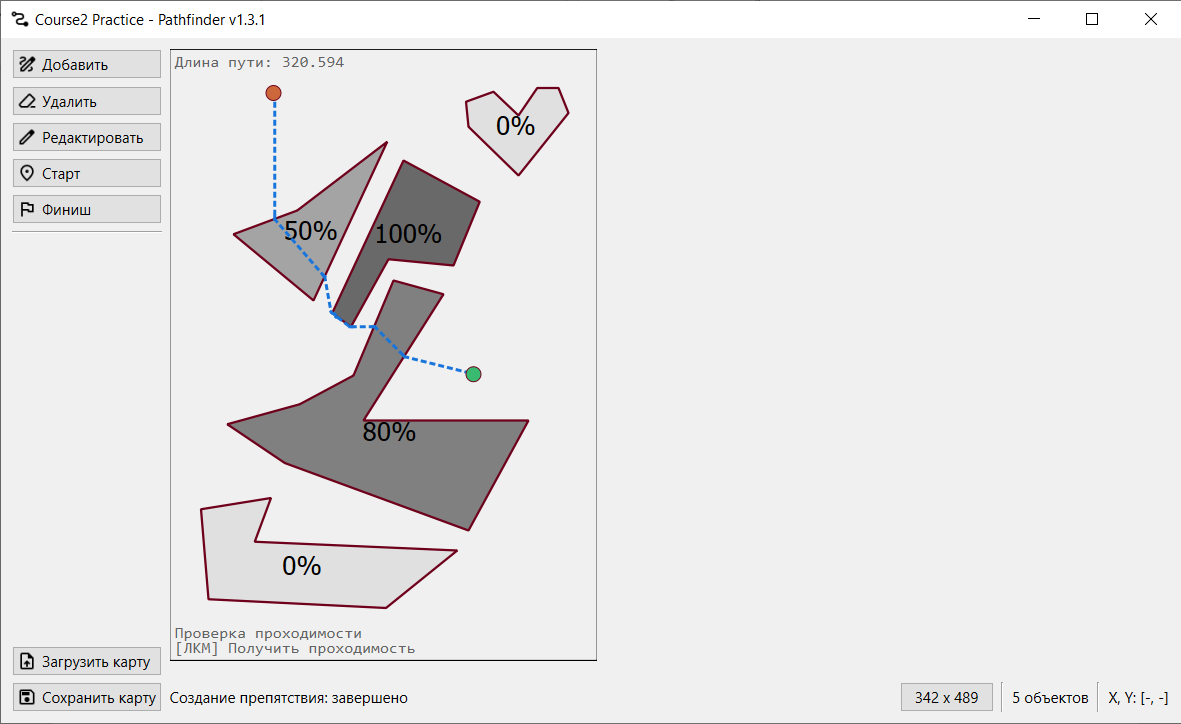


Рисунок 17

**заключение**

В ходе выполнения учебной практики было разработано приложение, выполняющее поставленную задачу. Разработан алгоритм нахождения кратчайшего пути, учитывающего препятствия. Продумана и построена на универсальном языке моделирования UML диаграмма прецедентов, диаграмма классов и диаграмма последовательности. Была проведена отладка программы.

**список использованных источников**

1. Статья о поиске графов // redblobgames.com URL: <https://www.redblobgames.com/pathfinding/a-star/implementation.html#troubleshooting-ugly-path>
2. Руководство Qt // doc.qt.io URL: <https://doc.qt.io/qtcreator/>
3. Справочное руководство по языку PlantUML // plantuml.com URL: <https://plantuml.com/ru/guide>