|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**РАСЧЕТНО-ПОЯСНИТЕЛЬНАЯ ЗАПИСКА**

***К ВЫПУСКНОЙ КВАЛИФИКАЦИОННОЙ РАБОТЕ***

***НА ТЕМУ:***

***Алгоритм поиска кратчайшего безопасного пути на электронной карте местности***

Студент \_\_ИУ7-84\_\_\_\_\_\_\_ **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_**Д.И. Монахов**\_\_\_**

(Группа) (Подпись, дата) (И.О.Фамилия)

Руководитель ВКР **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**М. Ю. Барышникова\_

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Консультант **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

Нормоконтролер **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ \_**Ю.В. Строганов**\_\_\_\_\_\_**

(Подпись, дата) (И.О.Фамилия)

*2020 г.*

Оглавление

[Оглавление 2](#_Toc41911997)

[Введение 4](#_Toc41911998)

[Аналитический раздел 6](#_Toc41911999)

[Сеточная карта (дискретное рабочее поле) 7](#_Toc41912000)

[Преобразование в граф 13](#_Toc41912001)

[Поиск в ширину 13](#_Toc41912002)

[Алгоритм HPA\* 20](#_Toc41912003)

[Итог выбора алгоритма 24](#_Toc41912004)

[Конструкторский раздел 27](#_Toc41912005)

[Технология отображения картографических данных 27](#_Toc41912006)

[Формат картографических данных 27](#_Toc41912007)

[Выбор платформы для отображения картографических данных 30](#_Toc41912008)

[Применение алгоритмов поиска кратчайшего пути к электронной карте местности 30](#_Toc41912009)

[Растеризация векторных объектов 32](#_Toc41912010)

[Алгоритм Брезенхема для отображения прямых линий на растровом слое 36](#_Toc41912011)

[Модификация алгоритма HPA\* для исключения опасных маршрутов 38](#_Toc41912012)

[Алгоритм сглаживания пути 47](#_Toc41912013)

[Вывод 50](#_Toc41912014)

[Технологическая часть 51](#_Toc41912015)

[Выбор среды и языка программирования 51](#_Toc41912016)

[Разработка структуры программного комплекса 51](#_Toc41912017)

[Разработка пользовательского интерфейса 55](#_Toc41912018)

[Модульное тестирование 64](#_Toc41912019)

[Заключение 71](#_Toc41912020)

[Список литературы 72](#_Toc41912021)

Введение

Проблема определения маршрута была актуальной еще в древности и является актуальной по сей день. Раньше человеку приходилось использовать карту для поиска оптимального по времени пути, чтобы достичь желаемого пункта назначения. В наши дни мало что изменилось: нам по-прежнему нужно перемещаться и очевидное желание сэкономить время и топливо никуда не исчезло. К тому же начинают появляться роботизированные средства передвижения, такие как беспилотные автомобили, беспилотные летательные аппараты (БПЛА). В военной промышленности также производятся крылатые ракеты с рельефометрическим наведением, которое использует карты для прокладывания маршрута полета таким образом, чтобы не попасть под огонь вражеских средств противоракетной обороны. Согласно проведенным экспериментам, данный тип наведения показывает отличные результаты в преодолении средств защиты и позволяет ракете обладать высокими показателями живучести и помехозащищенности [[1](https://babel.hathitrust.org/cgi/pt?id=mdp.39015013772390&view=1up&seq=349)]. Наведение по рельефу в комплексе с инерциальными средствами навигации позволяет аппарату быть практически невосприимчивым к средствам радиоэлектронной борьбы, т.к. исключает использование радиосвязи в навигации. Автоматизация процесса прокладывания маршрута произошла и в гражданской сфере: практически на каждом мобильном компьютере (смартфоне) имеются программно-аппаратные средства для отображения картографических данных, геопозиционирования пользователя и помощи в прокладывании маршрута. Аналогичные технологии используются в бизнесе. Самый яркий пример – это логистика. Оптимизация пути является основной задачей логистики, от успешности выполнения которой зависит прибыль компании. Стоимость маршрута может зависеть от множества параметров, таких как: время, расстояние, расход топлива и т.д. А сами эти параметры могут зависеть от характеристики ландшафта, по которому прокладывается путь [[2](https://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=82FD69BD60193407E0159A0844367A21?doi=10.1.1.164.8916&rep=rep1&type=pdf)].

Все рассмотренные выше приложения требуют программных средств поиска кратчайшего пути на электронной карте местности, который удовлетворяет некоторым заданным пользователем требованиям. Например, гражданин хочет избежать использования нерегулируемых перекрестков, логистическая компания не готова оплачивать платные дороги, а крылатая ракета не должна попасть в зону покрытия вражеской противоракетной обороны. Исходя из этого, данная работа предлагает рассмотреть применение алгоритмов поиска кратчайшего пути с возможностью исключения неблагоприятных участков пути.

Безопасным путем будет называться такой путь, который не проходит через опасные участки электронной карты местности. Опасные участки задаются пользователем с помощью устройств ввода. Пользователь может предусмотреть такие участки как:

* неблагополучные районы с высоким уровнем преступности;
* регионы с неблагоприятной экологической обстановкой;
* небезопасные участки местности с точки зрения дорожного движения (нерегулируемые перекрестки, дорожные работы и т.д.);
* прочие участки на усмотрение пользователя.

Итак, целью работы является модификация алгоритма поиска кратчайшего пути для решения проблемы прокладывания маршрута на карте местности с учетом требований пользователя по исключению специфических участков.

Для достижения поставленной цели необходимо решить следующие задачи:

* исследовательская (выбор существующего алгоритма поиска кратчайшего пути);
* конструкторская (модификация выбранного алгоритма для решения проблемы);
* технологическая (выбор средства отображения картографических данных, выбор среды и языка программирования, описание основных моментов разработки).

Аналитический раздел

Задача поиска пути заключается в обработке электронной карты местности специальным алгоритмом, входными данными которого, помимо карты, являются начальная и конечная точки, а результатом - последовательность координатных точек на исходной карте, которые представляют путь, удовлетворяющий некоторым требованиям. В случае с алгоритмами поиска кратчайшего пути, результирующий путь должен быть одним из кратчайших среди всех путей, ведущих из начальной точки в конечную.

Алгоритмы поиска кратчайшего пути значительно отличаются друг от друга по скорости работы и точности результата. В данном разделе будет осуществлен выбор лучшего алгоритма на основании следующих критериев:

* Скорость выполнения.
* Точность результата.

Некоторые алгоритмы требуют для своей работы предварительной обработки карты местности. В связи с тем, что реальная местность является практически статичной и меняется крайне медленно, результаты инициализации можно провести один раз при загрузке карты в программу, сохранив результат для повторного использования. Данная особенность проблемы позволяет пренебречь временем инициализации при оценке скорости выполнения.

Алгоритмы поиска кратчайшего пути оперируют ячейками дискретного рабочего пространства, полученного на основе исходной электронной карты местности. Время выполнения внутренних операций этих алгоритмов зависит от конкретной реализации, из чего следует, оно не является надежным критерием сравнения алгоритмов. Поэтому кажется разумным в качестве элементарной операции рассматривать операцию просмотра ячейки. Сравнение алгоритмов будет проводиться в том числе с помощью подсчета количества рассмотренных точек на аналогичных исходных данных. И далее при оценке скорости работы алгоритма будет учитываться количество точек, которые рассмотрел тот или иной алгоритм: чем меньше точек было просмотрено, тем выше скорость.

Кратчайших путей, строго говоря, может быть больше одного. Поэтому алгоритм должен вернуть любой из существующих кратчайших путей. Однако некоторые алгоритмы жертвуют точностью результата ради скорости выполнения. А так как предельная точность в реальных приложениях чаще всего не требуется, то такие алгоритмы также следует рассмотреть наряду с идеально точными. В связи с этим, точность результата должна стать одним из критериев выбора алгоритма.

Рассмотрим существующие алгоритмы поиска кратчайшего пути. Все приведенные ниже алгоритмы основываются на дискретизации электронной карты местности.

Сеточная карта (дискретное рабочее поле)

Для начала следует рассмотреть алгоритмы, работающие с сеточным представлением карты местности.

Карту можно представить в виде сетки (или таблицы), состоящей из проходимых и непроходимых ячеек. Каждая ячейка соответствует некоторой области в реальном мире и имеет две координаты (x, y), совпадающие с положением ячейки в таблице.

Рассмотрим построение такой сетки на примере фрагмента карты. Непроходимыми будем считать те ячейки, которые более чем на 50% заполнены какими-либо объектами реального мира. Эти ячейки закрасим красным цветом, а затем выделим получившуюся таблицу как отдельное изображение (рис 1).





Рисунок 1. Пример создания сеточной карты

Такое представление карты уже достаточно легко анализировать. Очевидно, что в данном примере использовалась слишком низкая частота дискретизации пространства, в разрабатываемом программном продукте точность следует повысить.

**Алгоритм Ли**

Существуют алгоритмы, которые работают с форматом входных данных в виде дискретного рабочего поля. Самый известный такой алгоритм – это алгоритм Ли. Рассмотрим принцип работы данного алгоритма. Он основан на алгоритме поиска в ширину и его сильной стороной является гарантия нахождения кратчайшего пути между двумя точками, если такой путь существует.

Входные данные: дискретное рабочее поле с помеченными непроходимыми ячейками, пара точек (старт и финиш).

Выходные данные: последовательность ячеек, представляющая кратчайший путь из точки старта в точку финиша, если такой путь существует.

Работа алгоритма делится на три этапа: инициализацию, распространение волны и восстановление пути.

1) Инициализация: построение множества ячеек, запоминание стартовой и финишной ячейки.

2) Распространение волны: порождается шаг в соседнюю ячейку с проверкой проходимости и принадлежности к уже рассмотренным ячейкам.

3) Соседняя ячейка помечается как рассмотренная и получает в соответствие число, равное количеству шагов от стартовой ячейки.

4) Из каждой рассмотренной на данном шаге ячейки порождается новая волна, которая трактует данную ячейку как стартовую.

5) При достижении конечной ячейки проводится восстановление пути. Начиная с конечной ячейки ищется соседняя ячейка, которой соответствует число, на единицу меньшее, чем у текущей. Последовательность этих ячеек и становится результатом работы алгоритма (после инверсии составленного списка).

Ниже приведен псевдокод алгоритма:

// Инициализация

Пометить стартовую ячейку

d := 0

// Распространение волны

ЦИКЛ

ДЛЯ каждой ячейки loc, помеченной числом d

пометить все соседние свободные непомеченные ячейки числом d + 1

КЦ

d := d + 1

ПОКА (финишная ячейка не помечена) И (есть возможность распространения волны)

// Восстановление пути

ЕСЛИ финишная ячейка помечена

ТО

перейти в финишную ячейку

ЦИКЛ

выбрать среди соседних ячейку, помеченную числом на 1 меньше числа в текущей ячейке

перейти в выбранную ячейку и добавить её к пути

ПОКА текущая ячейка — не стартовая

ВОЗВРАТ путь найден

ИНАЧЕ

ВОЗВРАТ путь не найден

Следует заметить, что соседство ячеек может быть двух вариантов: в смысле окрестности Мура и окрестности фон Неймана. В первом случае, в отличие от второго, рассматриваются не только ближайшие вертикальные и горизонтальные ячейки, но также диагональные. Принцип работы алгоритма никак не зависит от определения соседства.

Однако у алгоритма Ли есть недостаток - данный алгоритм является очень требовательным ко времени и памяти. Временная сложность и требование к памяти для сетки с размерами M x N составляет O(M x N), т.е. с увеличением размеров исследуемой области требования алгоритма растут в квадратической зависимости.

Способы сократить время выполнения:

1) выбирать в качестве начальной точки ту, которая находится дальше от центра;

2) распространять волну и из начальной точки, и из конечной;

3) исследовать как можно меньшую прямоугольную область, включающую входные точки.

Можно сказать, что данный алгоритм слабо подходит к поставленной задаче. Не смотря на гарантированность результата и его предельную точность, он является слишком медленным, так как стремится рассмотреть все точки рабочего поля. Следует рассмотреть другие алгоритмы поиска кратчайшего пути, которые будут более эффективны по скорости выполнения [[3](http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf)].

Преобразование в граф

Теперь рассмотрим способ превращения карты местности в графовое представление. Реальный мир является не дискретным пространством, что исключает возможность бесконечно точного хранения его реплики в памяти компьютера. Однако для решения задач прокладывания маршрутов это не понадобится. Достаточно разбить карту на равномерную сетку, где каждая проходимая ячейка представляется вершиной графа, а непроходимые участки (например, стены) удаляют соответствующие рёбра между вершинами (см. рис. 1). Данный подход является расширением дискретного рабочего поля, который был описан выше.



Рисунок 2. Графовое представление пространства

Такое представление пространства позволит применить классические алгоритмы обхода графов, которые и будут рассмотрены ниже.

Поиск в ширину

Поиск в ширину – один из простейших алгоритмов обхода графа. Он является базой для многих продвинутых алгоритмов поиска пути. Например, алгоритм Прима, алгоритм Дейкстры и алгоритм Ли используют те же принципы, что и поиск в ширину.

Получив на вход граф и начальный узел, поиск в ширину систематично обойдет грани графа, чтобы найти все узлы, достижимые из начального. Алгоритм вычисляет расстояние (минимальное количество граней) из начального узла в каждый достижимый узел и создает дерево, содержащее все достижимые узлы.

Пусть задан невзвешенный ориентированный граф *G*=(*V*,*E*), в котором выделена исходная вершина. Требуется найти длину кратчайшего пути (если таковой имеется) от одной заданной вершины до другой. Частным случаем указанного графа является невзвешенный неориентированный граф, т.е. граф, в котором для каждого ребра найдется обратное, соединяющее те же вершины в другом направлении.

Для алгоритма нам потребуются очередь и множество посещенных вершин *was*, которые изначально содержат одну вершину *s*. На каждом шагу алгоритм берет из начала очереди вершину *v* и добавляет все не посещённые смежные с *v* вершины в *was* и в конец очереди. Если очередь пуста, то алгоритм завершает работу.

Временная сложность алгоритма может быть выражена как O(|E| + |V|), где |E| - количество граней, а |V| - количество узлов графа. При этом значение O(|E| + |V|) может варьироваться между O(|V|) и O(|V|2) в зависимости от строения графа [[4](https://www.researchgate.net/publication/282488307_Pathfinding_Algorithm_Efficiency_Analysis_in_2D_Grid)].

Ниже приведен псевдокод алгоритма

1. Поместить исходную вершину в очередь

2. Извлечь вершину x из очереди и пометить как посещенный

а. Если узел x является целевым, то завершить поиск как успешный

б. Иначе, добавить в очередь все неотмеченные вершины, которые связаны с x и не находятся в очереди

3. Если очередь пуста, то завершить поиск как неудачный

4. Перейти в п. 2

Аналогично алгоритму Ли, поиск в ширину стремится рассмотреть все точки исходного рабочего пространства, не учитывая взаимного положения начальной и конечной точки. В связи с этим, алгоритм является неэффективным по скорости выполнения.

**Алгоритм Дейкстры**

Алгоритм Дейкстры работает с проблемами поиска кратчайшего пути в направленных взвешенных графах G = (V, E) с неотрицательными весами граней. Алгоритм создает массив узлов S, чьи веса кратчайшего пути из начального узла уже были определены. Алгоритм последовательно выбирает узлы с наименьшим ожидаемым расстоянием, не входящие в S и «релаксирует» все исходящие из этих узлов грани. Алгоритм Дейкстры называют «жадным» алгоритмом, т.к. он всегда выбирает ближайший узел, который следует добавить в S. Худшее время работы алгоритма оценивается как O(|E| + |V|log|V|)|), где |E| - количество граней, а |V| - количество узлов графа.

Данный алгоритм не очень удобно использовать при рассматриваемом способе представления пространства. Все проходимые ячейки пространства должны соединяться гранями с одинаковым весом, т.к. располагаются на одинаковом расстоянии. В случае, если все веса графа равны, то алгоритм Дейкстры перестает чем-либо отличаться от алгоритма поиска в ширину.

**Выводы по алгоритмам семейства поиска в ширину**

Данные алгоритмы являются неэффективными по количеству рассматриваемых точек для использования в решении поставленной задачи. Основным недостатком алгоритмов семейства поиска в ширину является стремление обойти все узлы графа, не учитывая взаимного расположения начального и конечного узлов. Теоретическая оценка сложности показывает квадратическую зависимость от размеров сетки пространства. Рассмотрим пример выполнения экспериментальной программы.

Для эксперимента использовалась модельная карта местности с одной горизонтальной преградой между начальной и конечной точкой.



Рисунок 3. Пример работы алгоритмов семейства поиска в ширину

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен алгоритм Дейкстры, на правой – алгоритм Ли. Оба алгоритма нашли кратчайший путь (выделен красным). Однако видно, что алгоритмы действовали явно не оптимально – в обоих случаях подавляющее большинство ячеек было посещено. Можно сделать вывод, что на реальных данных алгоритмы будут показывать неудовлетворительную производительность.

**Поиск по первому наилучшему совпадению**

Поиск «лучший - первый» - алгоритм поиска, исследующий граф путём расширения наиболее перспективных узлов, выбираемых в соответствии с указанным правилом. Данное семейство алгоритмов вводит новое понятие – эвристическая функция оценки. Эта функция представляет собой правило, по которому оценивается перспективность очередного узла n. Чаще всего данной оценкой является расстояние в декартовой системе координат, т.е. наиболее перспективным является наиболее близкий к цели узел.

Рассмотрим простейшую реализацию алгоритма «лучший – первый».

1) Создать список открытых узлов OPEN.

2) Добавить в OPEN начальный узел.

3) Если очередь пуста, то перейти к 9.

4) Удалить лучший узел N из OPEN.

5) Если N является конечным, то произвести восстановление пути и вернуть путь.

6) Создать список соседних узлов для N.

7) Оценить каждый соседний узел, добавить их в OPEN и пометить N как родительский.

8) Перейти к 3.

9) Закончить.

Эта реализация алгоритма имеет недостаток – она может не найти путь между двумя узлами, даже если он существует. Например, алгоритм может зациклиться, если зайдет в тупик (ситуация, когда потомок узла является его родителем).

Добавление списка уже посещенных узлов позволит избежать зацикливания в тупиках. [5]

Проведем эксперимент и сравним жадную версию алгоритма «лучший – первый» с алгоритмом, основанным на поиске в ширину. Для эксперимента использовалась модельная карта местности с одной горизонтальной преградой между начальной и конечной точкой.



Рисунок 4. Сравнение жадного алгоритма «лучший-первый» и алгоритма Ли

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен жадный алгоритм «лучший-первый», на правой – алгоритм Ли. Оба алгоритма нашли кратчайший путь (выделен красным). Видно, что алгоритм на левой стороне посетил значительно меньшее количество ячеек. Это связано с тем, что данный алгоритм использует дополнительную информацию о положении цели и учитывает ее с помощью эвристической функции.

Рассмотрим карту другого вида. Добавим на местность так называемую «ловушку» - область, в виде незамкнутой выпуклой фигуры. Ориентируем ее так, чтобы отверстие в стене находилось ближе к начальной точке. Результаты приведены на рисунке 5.



Рисунок 5. Ловушка для жадного алгоритма «лучший-первый»

Видно, что алгоритм Ли (справа) нашел кратчайший путь, а алгоритм «лучший-первый» дал неверный результат. Данный путь не является кратчайшим. Такой результат связан с тем, что алгоритм никак не учитывает фактическую длину пути, а полагается лишь на эвристическую функцию. Последняя заводит алгоритм в ловушку, из которой нужно искать выход.

Данный недостаток делает жадный алгоритм «лучший-первый» неподходящим для решения задачи кратчайшего пути. Несмотря на значительно более высокую эффективность по сравнению с поиском в ширину, низкая точность результата вынуждает отказаться от его применения.

**Алгоритм A\***

Ранее был рассмотрен алгоритм, использующий эвристическую функцию для предсказания наиболее перспективного направления обхода графа. Однако алгоритм не давал гарантии того, что результирующий путь будет кратчайшим. Алгоритм A\* призван устранить этот недостаток. Он основывается на том же принципе эвристики, комбинируя сильные стороны алгоритма «лучший-первый» и алгоритма Дейкстры.

Отличия от алгоритма «лучший-первый» заключается в специальной эвристической функции, которая учитывает не только приблизительное расстояние до цели, но и количество шагов до начальной точки.

Общий вид эвристической функции алгоритма A\*:

Здесь f(n) – эвристическая функция, g(n) – количество шагов от старта до узла n, h(n) – оценочное расстояние от узла n до целевого узла.

Алгоритм A\* гарантирует нахождение кратчайшего пути, если таковой существует. Данное свойство делает его подходящим для решения поставленной задачи. Для оценки эффективности проведем эксперимент.



Рисунок 6. Сравнение алгоритма A\* и алгоритма Ли

Были запущены два алгоритма на одинаковых моделях карты. Зеленым цветом изображены непроходимые ячейки. Числа в ячейках показывают количество шагов из начальной точки в эту ячейку. Если в ячейке нет числа, значит алгоритм не посещал эту ячейку. На левой копии карты был запущен алгоритм A\*, на правой – алгоритм Ли. Оба алгоритма нашли кратчайший путь (выделен красным). Видно, что алгоритм A\* на левой стороне посетил значительно меньшее количество ячеек. Количество рассматриваемых точек является менее впечатляющим, чем у алгоритма «лучший-первый», но такова цена гарантии нахождения кратчайшего пути.

Алгоритм HPA\*

Проблема поиска кратчайшего пути должна решаться в кратчайшее время (а в некоторых приложениях в реальном времени) в условиях ограниченной памяти и процессорного времени. Рассмотренный алгоритм A\* хоть и избегает рассматривания всех ячеек электронной карты, но все равно является достаточно медленным в случае его использования на больших картах.

Алгоритм HPA\* (Hierarchical Path-Finding A\*) призван решить эту проблему. Он основывается на иерархическом подходе к разбиению пространства и дальнейшем применении обычного A\* не к изначальной проблеме и карте, а к небольшим участкам. Исходное дискретное пространство разбивается на взаимосвязанные кластеры ячеек. До первого запуска поиска пути вычисляются и запоминаются оптимальные пути внутри каждого кластера. Далее путь ищется по абстрактному графу, составленному из кластеров. После этого путь уточняется с помощью рассмотрения путей внутри задействованных кластеров.

Поиск пути состоит из трех этапов. Во-первых, алгоритм находит путь от начальной точки до границы кластера, который ее содержит. Во-вторых, по абстрактному графу ищется путь от начального кластера к конечному. В-третьих, путь завершается нахождением пути от границы конечного кластера к конечной точке. Абстрактный граф строится на основе информации, полученной из электронной карты местности.

Рассмотрим этап первоначальной обработки карты, который производится до первого запуска поиска пути. Карта разбивается на прямоугольные группы ячеек, называемые кластерами (рис. 7). Черные клетки соответствуют непроходимым участкам карты, белые, соответственно, являются проходимыми. Кластеры разделены красными линиями. Разбиение осуществляется без учета характеристик самого пространства, достаточно только задать желаемый размер кластера.

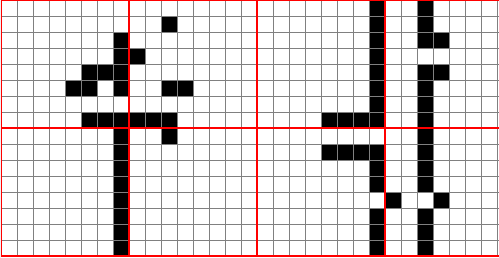


Рисунок 7. Разбиение на кластеры

Для каждой границы между кластерами находится массив входов, которые их соединяют. Под входом понимается наибольший свободный от препятствий сегмент, располагающийся вдоль общей границы двух соседних кластеров c1 и c2. Пусть имеются два смежных отрезка l1 и l2, принадлежащих кластерам c1 и с2 соответственно, и определяющих смежные грани кластеров. Для каждой клетки определим клетку symm(t) как клетку, симметричную клетке t по отношению к границе между кластерами. t и symm(t) прилегают друг к другу и никогда не принадлежат одному и тому же кластеру. Тогда вход e – это массив ячеек, удовлетворяющий следующим условиям:

* вход не выходит за границы кластеров: ;
* симметрия: ;
* вход не содержит непроходимых ячеек;
* вход является максимально возможным.

Результат работы этого процесса изображен на рисунке 8. Границы входов представлены синими кругами. Вход может быть представлен одной или двумя точками, в зависимости от длины отрезка.

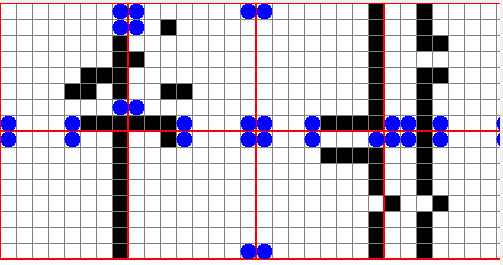


Рисунок 8. Границы входов

Далее строится взвешенный граф, узлами которого являются границы входов. Все грани узлов, представляющих симметричные входы, всегда имеют вес, равный 1. Для граней, соединяющих узлы одного кластера, веса вычисляются с помощью нахождения длины кратчайшего пути между ними. Эти пути можно сохранить для дальнейшего использования, однако в таком случае потребуется значительно больший объем памяти. Остальные узлы между собой не соединяются. Результат этого этапа можно увидеть на рис. 9. Зелеными линиями представлены грани полученного графа. Черными числами показаны веса граней.

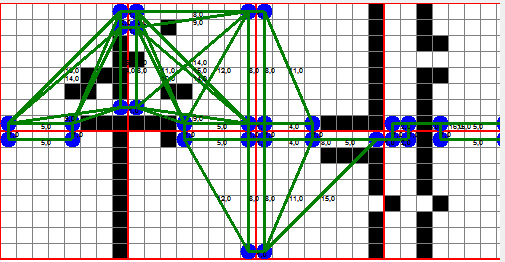


Рисунок 9. Построение графа

После того, как граф был построен и были посчитаны веса связей внутри кластеров, электронная карта готова к использованию алгоритмом HPA\*. Эта информация может быть вычислена заранее, сохранена вместе с файлами геометрии карты и загружена при запуске программы. Для динамически меняющихся карт, заранее полученная информация должна модифицироваться.

Далее проводится поиск пути по полученному графу. Первым этапом этого процесса является добавление стартового узла к границе кластера, который его содержит. Аналогичное действие производится с конечным узлом и конечным кластером. После добавления узлов ищется кратчайший путь по графу с использованием алгоритма A\*. Результатом работы алгоритма является последовательность узлов графа, составленного из проходов между кластерами. Последним этапом является восстановление кратчайших путей между узлами графа с использованием A\* внутри кластеров.

HPA\* позволяет добавлять новые уровни абстракции, объединяя кластеры в более крупные группы. Таким образом алгоритм становится масштабируемым на все более крупные электронные карты местности.

За счет значительного сокращения пространства поиска, алгоритм показывает временную эффективность, до 10 раз превосходящую A\*, давая результат с погрешностью всего лишь в 1%. На рисунке 10 приведены результаты сравнения алгоритмов A\* и HPA\*, проведенного исследователями University of Alberta. По оси абсцисс отложена длина кратчайшего пути, а по оси ординат – количество рассмотренных алгоритмами узлов и время выполнения [[6](file:///D:\Загрузки\6.%09http:\webdocs.cs.ualberta.ca\~kulchits\Jonathan_Testing\publications\ai_publications\jogd.pdf)].

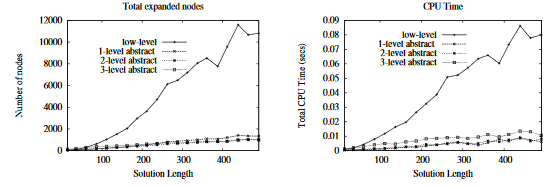


Рисунок . Сравнение HPA\* и A\*

По графикам видно, что HPA\* обыгрывает A\* по количеству рассматриваемых точек.

Итог выбора алгоритма

Ранее в разделе были теоретически обоснованы характеристики различных алгоритмов, и приведены приблизительные оценки скорости их выполнения на основании подсчета количества посещенных точек электронной карты местности. Теперь рассмотрим результаты экспериментального сравнения этих алгоритмов, приведенные в работе исследователей Rezekne Academy of Technologies [4]. Данное исследование приводит данные по количеству посещенных точек различными алгоритмами (см. табл. 1). Данные по алгоритму лучший-первый отсутствовали в исследовании, число получено с помощью разработанной в рамках работы программы.

Таблица 1. Сравнение алгоритмов по количеству рассматриваемых точек

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Алгоритм** | **Размер дискретного рабочего пространства** | | | | |
| **64 x 64** | **128 x 128** | **256 x 256** | **512 на 512** | **1024 x 1024** |
| Поиск в ширину | 3155 | 12887 | 52367 | 213648 | 1159255 |
| Алгоритм Дейкстры | 3173 | 13058 | 52068 | 209251 | 836977 |
| A\* | 623 | 1576 | 8071 | 40333 | 104109 |
| HPA\* | 454 | 1334 | 3551 | 10629 | 41491 |
| Лучший-первый |  |  |  |  | 1786 |

В таблице не представлен алгоритм Ли, т.к. фактически он аналогичен поиску в ширину за тем исключением, что работает на необработанной сеточной карте, в то время как поиску в ширину требуется графовое представление. Как был разъяснено ранее, время инициализации в данной работе не учитывается.

В этой работе размеры карты являются большими (порядка квадрата со стороной 1000 ячеек и более), поэтому для сравнения алгоритмов по скорости будет рассматриваться только данные крайнего столбца (1024\*1024).

Составим сводную таблицу исследуемых алгоритмов с учетом критерия точности результата.

Таблица 2. Сравнение алгоритмов по выбранным критериям

|  |  |  |
| --- | --- | --- |
| **Алгоритм** | Количество рассматриваемых точек | Точность результата |
| Поиск в ширину | 1159255 | Точный |
| Алгоритм Дейкстры | 836977 | Точный |
| A\* | 104109 | Точный |
| HPA\* | 41491 | Погрешность 1% |
| Лучший-первый | 1786 | Произвольный путь |

Видно, что алгоритм «Лучший-первый» является наиболее быстрым из всех рассмотренных алгоритмов. Однако данный алгоритм возвращает не кратчайший путь, а любой существующий. Поэтому данный алгоритм исключается из выбора.

Алгоритмы поиска в ширину (включая алгоритм Ли) являются слишком неэффективными по количеству рассматриваемых точек. Следовательно, остается выбрать между A\* и его наследником HPA\*. HPA\* дает очень значительную прибавку к скорости за счет снижения точности на 1%. Данная погрешность не является критической для поставленной задачи, поэтому выбор в рамках данной работы будет использоваться алгоритм HPA\*.

Конструкторский раздел

Технология отображения картографических данных

Для реализации отображения электронной карты местности необходимо использовать специальный набор программных средств, который называется ГИС. ГИС (географическая информационная система) - информационная система, обеспечивающая сбор, хранение, обработку, доступ, отображение и распространение пространственно-координированных данных (пространственных данных). ГИС содержит данные о пространственных объектах в форме их цифровых представлений (векторных, растровых, и иных) [[7](http://www.gisa.ru/13058.html)]. Включение такой системы в программный комплекс позволит работать с электронными картами местности, которые представляют собой модель реальной местности.

Формат картографических данных

В картографических системах данные чаще всего представляются в векторной форме. Это связано с тем, что при схематическом изображении пространства реальный внешний вид объекта не играет никакой роли, а значит можно хранить информацию только о его очертаниях. Также электронная карта должна быть легко масштабируема. Это условие накладывает ряд серьезных ограничений на растровую графику. В связи с этим растровые карты (например, снимки со спутника) должны иметь очень высокое разрешение, или набор слоев разного разрешения под разный масштаб. В связи с тем, что конечный продукт будет выполняться на персональных компьютерах, требования к производительности рендеринга достаточно высокие. Следовательно, предпочтительным является векторный формат карт.

Популярным форматом векторных карт является формат Shapefile. Этот формат описывает векторные элементы: точки, ломаные линии, многоугольники, которые в свою очередь могут представлять водоемы, здания, дороги и прочие объекты реального мира. Также Shapefile предоставляет возможность сопоставлять объектам различные значения (названия, температуру, ограничения скорости, количество населения и т.д.). Карты в формате Shapefile состоят из трёх обязательных файлов:

1) .shp – данные о геометрии объектов;

2) .shx – данные индексации

3) .dbf – атрибуты объектов в формате dBase IV.

Помимо этих файлов карте могут принадлежать несколько необязательных дополнительных файлов:

1) .prj – описание проекции;

2) .sbn и .sbx – пространственные индексы объектов;

3) .fbn и .fbx - пространственные индексы объектов только для чтения;

4) .ain и .aih – индексные файлы атрибутов таблиц;

5) .ixs - индекс геокодирования наборов данных для чтения и записи;

6) .mxs - индекс геокодирования наборов данных для чтения и записи (формат ODB);

7) .atx – индекс атрибута для .dbf файла;

8) .shp.xml – метаданные в XML формате;

9) .cpg – кодовая страница для определения кодировки символов .dbf файла;

10) .qix - альтернативный пространственный индекс дерева квадрантов, используемый программами MapServer и GDAL / OGR.

В файлах .shp, .shx и .dbf объекты в каждом файле соответствуют по позиции записи в файле, т.е. объекту из первой записи в .shp файле соответствуют атрибуты из первой запись в .dbf и т.д.

Рассмотрим формат векторной геометрии в shapefile. Двоичный файл .shp состоит из заголовка фиксированной длины, за которым следуют записи переменной длины. Каждая запись состоит из заголовка записи и содержимого. Упорядочивание двухмерных координат точек предполагает декартову систему координат с использованием порядка (X, Y) или (Easting Northing). Этот порядок согласуется с географическими координатами, в которых порядок аналогичен (долгота широта). Геометрические объекты также могут поддерживать трёхмерные и четырехмерные координаты.

Заголовок файла занимает 100 Байт и содержит 17 полей:

1) код– всегда равен 0x0000270a;

2) длина файла в 16-битных словах;

3) версия;

4) типы геометрических объектов;

5) минимальная прямоугольная граница;

6) диапазон Z-координат;

7) диапазон M-координат.

Заголовок записи занимает 8 байт:

1) номер записи;

2) длина записи в 16-битных словах.

Содержимое записи:

1) тип геометрического объекта;

2) список точек объекта.

Shapefile поддерживает следующие основные геометрические примитивы:

1) точка – точка с координатами X, Y;

2) поли-линия – ломаная линия, состоящая из набора отрезков;

3) многоугольник;

4) мульти-точка – объект, состоящий из нескольких точек;

5) трехмерная точка – точка с координатами X, Y, Z;

6) трехмерная поли-линия;

7) трехмерный многоугольник;

8) трехмерная мульти-точка.

Существует ряд других поддерживаемых примитивов, но они не представляют интереса для данного проекта.

Индексный файл .shx содержит такой же 100-байтовый заголовок, что и .shp файл. Содержимое индексного файла состоит из последовательности 8-байтовых записей из двух полей:

1) смещение записи в 16-битных словах;

2) длина записи в 16-битных словах.

Используя индексный файл, можно легко искать записи, двигаясь не по самому файлу с геометрией, а по набору записей фиксированного размера. Таким образом, поиск возможно проводить с конца файла.

Файл атрибутов .dbf содержит атрибуты объектов. Он использует формат dBase IV. Данные представляются в виде таблицы, где заголовки столбцов содержат имена атрибутов, а номер строки соответствует номеру записи объекта в .shp файле [[8](https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf)].

Выбор платформы для отображения картографических данных

Наиболее удобной формой использования этого набора является подключаемая библиотека, поэтому были рассмотрены наиболее популярные ГИС-библиотеки. Требованием к библиотеке была поддержка платформы .NET Framework и языка программирования C#.

Выбор пал на библиотеку MapAround. Это платформа для решения прикладных задач обработки, хранения и отображения пространственных данных. Поддерживает формат Shapefile и построение интерфейса WindowsForms [[9](http://www.maparound.ru/)]. Платформа и ее исходный код распространяются под лицензией, совместимой с GNU GPL License V3, что позволяет использовать ее в приложениях с открытым исходным кодом [[10](http://www.maparound.ru/Download.aspx)].

Применение алгоритмов поиска кратчайшего пути к электронной карте местности

Как было сказано выше, электронная карта местности представляет из себя набор геометрических примитивов, хранящихся в векторном формате. Это очень удобно для хранения огромного количества объектов в связи с тем, что объект хранится в виде лишь нескольких координатных точек.

Однако данный вид хранения не подходит для работы алгоритмов поиска пути. Рассмотренные алгоритмы требуют на вход либо дискретное рабочее пространство, либо граф. В аналитическом разделе была описана концепция конвертации дискретного рабочего пространства в граф, здесь же будет детально изложен алгоритм, её реализующий.

Но начать нужно с получения дискретного рабочего пространства на основе списка векторных объектов. Преобразовывать всю карту не имеет смысла – скорее всего путь существует внутри минимального прямоугольника, в который входят начальная и конечная точки. Однако бывают редкие случаи, когда такого пути не существует, поэтому программа должна позволять пользователю выбирать участок поиска пути вручную.

Рассмотрим получившуюся последовательность преобразований:

* Выбрать фрагмент электронной карты местности.
* Выбрать начальную и конечную точки.
* Преобразовать карту в формат дискретного рабочего пространства.
* Запустить выполнение алгоритма.

В данной последовательности действий умышленно опущены этапы, не связанные с преобразованием рабочего пространства. Указание опасных участков, вывод результатов на экран и прочие моменты будут рассмотрены позднее.

Наглядное представление преобразования карты можно увидеть на рисунке 11. Слева схематически изображены исходные данные программы: начальная и конечная точки (цветные круги) и непроходимые препятствия (например, здания) в векторном формате (изображен спроецированный участок карты). На правом фрагменте рисунка представлена растеризация векторной карты и определение начальной и конечной ячеек.

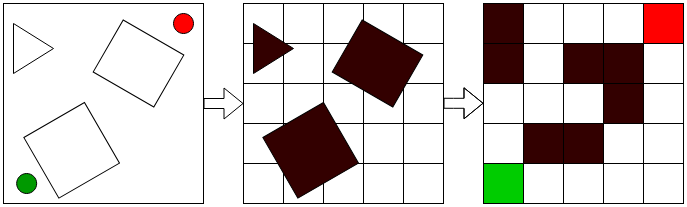


Рисунок 11. Преобразования электронной карты

Данное преобразование можно выполнить, используя алгоритмы, аналогичные алгоритмам растеризации из области компьютерной графики. Отличие лишь в том, что необходимо выполнить проецирование мировых координат в координаты экрана пользователя, а затем выполнить растеризацию полученных векторных объектов не на экран, а в специальный буфер, представляющий дискретное рабочее пространство.

Растеризация векторных объектов

Для растеризации был выбран алгоритм построчного сканирования. Его особенностью является то, что он позволяет заполнять как выпуклые, так и невыпуклые многоугольники любой сложности. Входными данными для него является набор точек многоугольника, полученный из электронной карты. Результаты своей работы он записывает в матрицу дискретного рабочего пространства.

Этот алгоритм основывается на том, что соседние пикселы в строке скорее всего одинаковы и меняются только в месте пересечения строки сканирования с ребрами. Это свойство называется когерентностью растровых строк. Достаточно лишь определить X-координаты пересечений строк сканирования с ребрами. Пары отсортированных точек пересечения задают интервалы заливки.

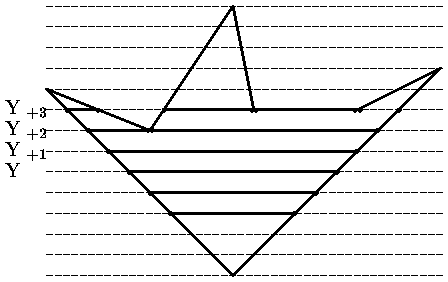


Рисунок 12. Построчное заполнение

Если какие-то ребра пересекались i-ой строкой, то скорее всего будут пересекаться также и строкой i + 1 (строки Yi и Yi+1 на рис. 12). Это называется когерентностью ребер. При переходе к новой строке легко вычислить новую X-координату точки пересечения ребра, используя X-координату старой точки пересечения и тангенс угла наклона ребра:

где k = dy/dx – тангенс угла наклона ребра. dy = 1, следователь 1/k = dx.

Смена количества интервалов заливки происходит только тогда, когда в строке сканирования появляется вершина.

Для каждой строки сканирования рассматриваются только те ребра, которые пересекают строку. Они задаются списком активных ребер (САР). При переходе к следующей строке для пересекаемых ребер вычисляются новые X-координаты пересечений. При появлении в строке сканирования вершин производится перестройка САР. Ребра, которые перестали пересекаться, удаляются из САР, а все новые ребра, пересекаемые строкой, заносятся в него.

Последовательность действий алгоритма:

1. Подготовить массивы Y-координат вершин.
2. Отсортировать Y-координаты по возрастанию.
3. Определить пределы закраски Ymin и Ymax.
4. Установить текущую строку сканирования на YMin.
5. Определить число вершин на строке сканирования.
6. Для каждой вершины дополнить список актвных рёбер, используя информацию о соседних вершинах.
7. По списку активных рёбер определяется следующая опорная Y-координата Ynext ближайшей вершины. До ее достижения список активных ребер не меняется.
8. От текущей строки сканирования до Ynext:

8.1 Выбрать из списка активных ребер и отсортировать X-координаты пересечений активных ребер со строкой сканирования;

8.2 Определить интервалы и выполнить закраску;

8.3 Вычислить новые координаты пересечений для следующей строки сканирования.

9. Проверить достижение предела Ymax. Если достигнута, то заполнение окончено.

10. Очистить список активных ребер, закончившихся в строке Ynext и перейти к пункту 5 [[11](http://algolist.ru/graphics/fill.php)].

Рассмотрим пример использования алгоритма растеризации. На рис. 13 приведен снимок экрана тестовой программы. В данный момент она отображает электронную карту города. Растеризацию проводит компонент платформы MapAround. Эту информацию хоть и можно использовать, но лучше провести растеризацию отдельно: экран программы имеет слишком большое разрешение, а также может отображать проходимые объекты.



Рисунок 13. Пример электронной карты

Выберем на карте фрагмент, представляющий интерес для воображаемого пользователя (рис. 14).

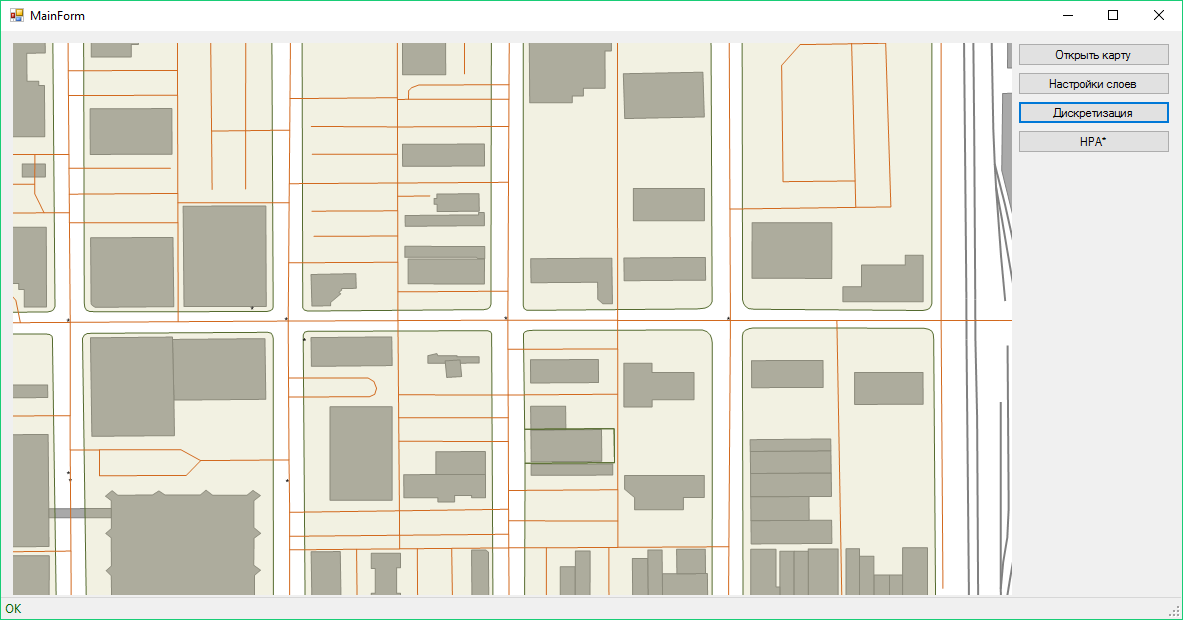


Рисунок 14. Фрагмент векторной карты

Проведем растеризацию только непроходимых объектов (темные многоугольники на рис. 14) и визуализируем на специальном экране (рис. 15).

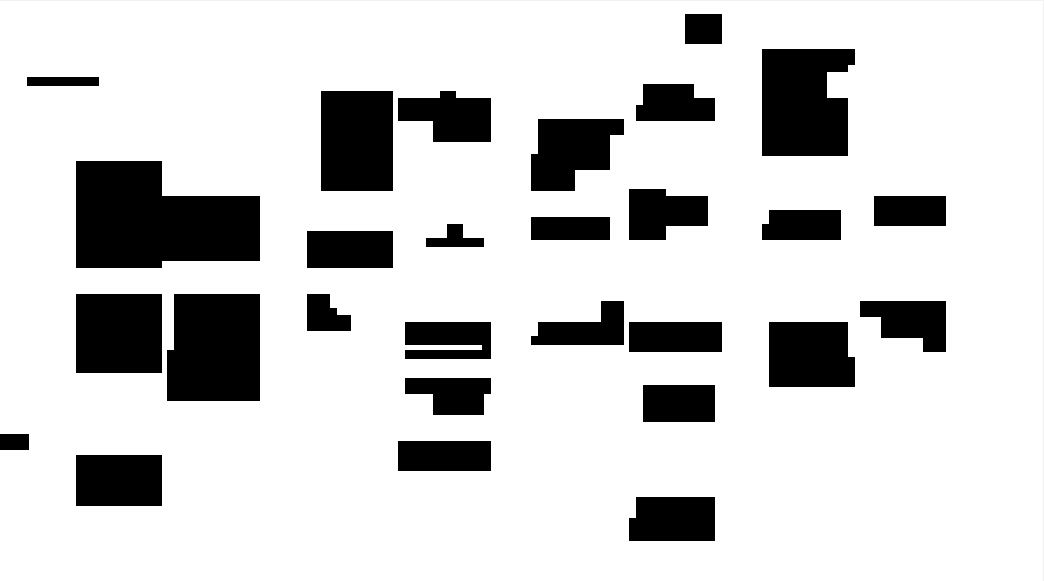


Рисунок 15. Результат растеризации

Рисунок в целом повторяет очертания объектов из исходной карты. Однако это визуализация результата, на самом деле результат работы алгоритмы хранится не в графическом формате, а в виде матрицы проходимости. Данная матрица имеет меньшее разрешение по сравнению с растеризацией для графики. Это необходимо, т.к., как было показано в аналитическом разделе, чем больше рабочее пространство, тем большей получается вычислительная сложность любого алгоритма поиска пути. В реальных картах, здания чаще всего являются достаточно простыми фигурами, а значит потери на снижении разрешения будут несущественными.

Таким образом и получается дискретное рабочее пространство, пригодное для дальнейших преобразований.

Алгоритм Брезенхема для отображения прямых линий на растровом слое

Для исключения опасных участков требуется реализовать удобный для пользователя способ ввода данных. В данной работе был выбран ввод с помощью мыши. Пользователь указывает вершины многоугольника, который затем добавляется в список исключенных участков карты. Чтобы пользователю было удобно осуществлять ввод, используется отображение промежуточных результатов ввода в виде ломаной линии, представляющей незавершенный многоугольник. Линии, составляющие ломаную, отображаются средствами растрового рендеринга MapAround (рис. 16).

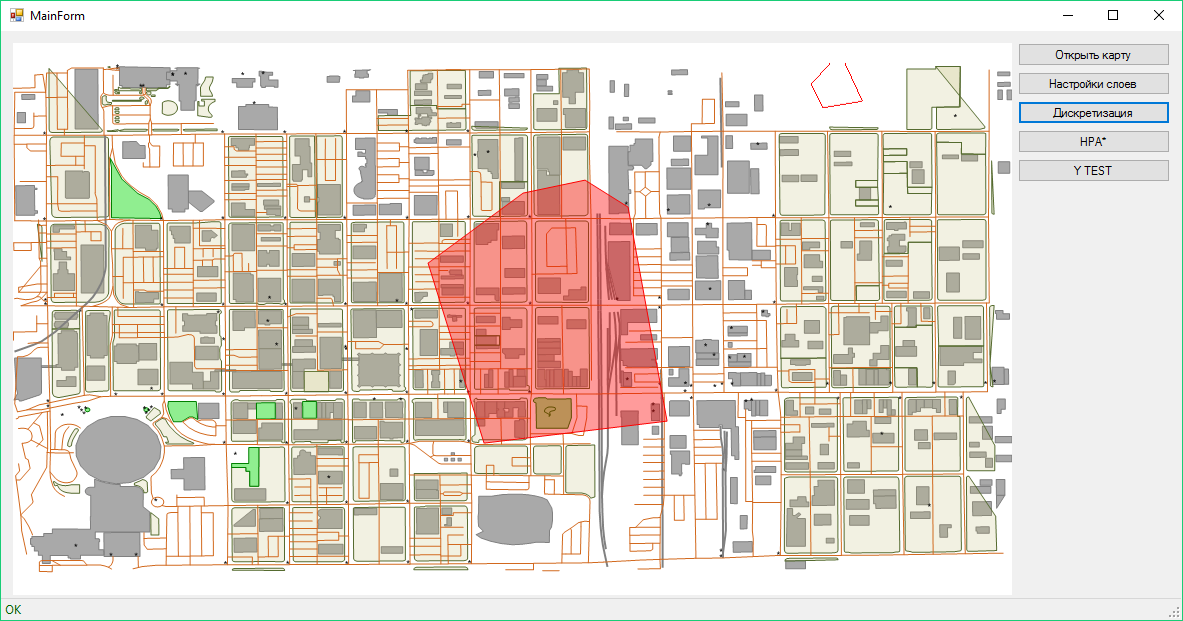


Рисунок 16. Демонстрация работы алгоритма Брезенхема (в правом верхнем углу)

Для построения отрезка было решено использовать алгоритм Брезенхема. Он эффективнее чем алгоритм DDA-линии (использует только операции вычитания и сложения целых чисел) и обеспечивает достаточную скорость выполнения.

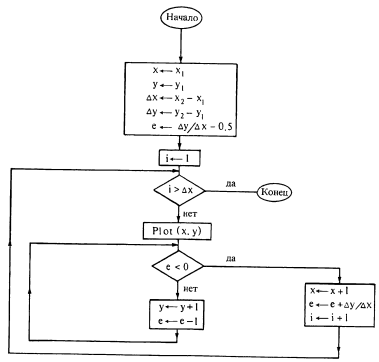


Рисунок 17. Блок-схема алгоритма Брезенхема

Алгоритм Брезенхема выбирает оптимальные растровые координаты для представления отрезка. В процессе работы одна из координат – либо x, либо y (в зависимости от углового коэффициента) – изменяется на единицу. Изменение другой координаты (либо на ноль, либо на единицу) зависит от расстояния между действительным положением отрезка и ближайшими координатами сетки. Такое расстояние называют ошибкой. Блок-схема алгоритма представлена на рисунке 17 [12]. Для отображения линий в любом октанте используется способ замены осей: оси X и Y меняются местами.

Модификация алгоритма HPA\* для исключения опасных маршрутов

Существует два способа исключения участков электронной карты местности. Оба этих способа основываются на добавлении новых «непроходимых» объектов в исходную карту. Эти объекты можно добавить так, что они не будут отображаться пользователю в виде препятствий, но будут влиять на принятие решения со стороны алгоритма поиска кратчайшего пути. Таким образом, принцип работы алгоритма остается прежним, т.к. модификации подвергаются данные на начальном этапе работы.

Рассмотрим первый вариант модификации – установка дополнительных непроходимых ячеек в дискретном рабочем пространстве. После проведения растеризации по описанному в предыдущем разделе методу, программа получит список опасных для пользователя участков карты и передаст алгоритму поиска пути модифицированное дискретное рабочее пространство. Данный метод проиллюстрирован на рисунке 18.

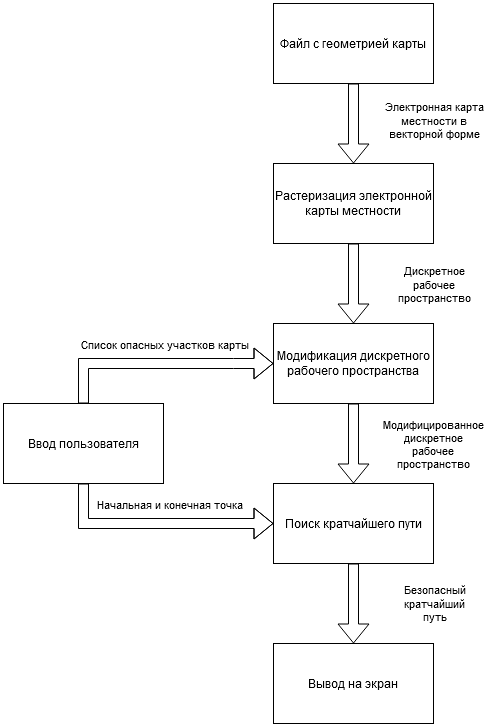


Рисунок 18. Модификация дискретного рабочего пространства

Данный метод исключения опасных участков хорош тем, что он универсален и подходит для работы с любым из рассмотренных алгоритмов поиска кратчайшего пути. Особенно высока его эффективность в случае применения алгоритмов, которые наподобие алгоритма Ли пропускают этап преобразования дискретного рабочего пространства в граф.

В качестве алгоритма поиска кратчайшего пути в данной работе было решено применить HPA\*. Как было показано выше, особенность работы алгоритма заключается в том, что HPA\* проводит начальные преобразования дискретного рабочего пространства, которые могут быть сохранены для дальнейшего использования. HPA\* выполняет анализ исходного рабочего пространства, следовательно, любая модификация этого пространства потребует повторных затрат на анализ. Тем не менее, рассмотрение данного метода исключения рабочего пространства не является бессмысленным в связи с высокой универсальностью.

Попробуем изменить метод так, чтобы алгоритму HPA\* не приходилось повторять анализ исходного рабочего пространства. Для этого рассмотрим схему работы алгоритма HPA\* (рис. 19)

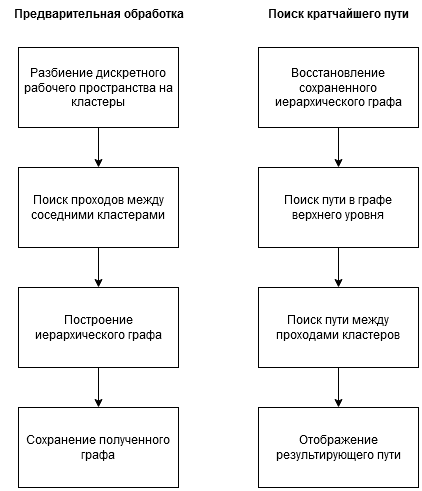


Рисунок 19. Схема алгоритма HPA\*

Для увеличения эффективности, следует сохранять результаты предварительной обработки. Следовательно, правильнее всего будет вмешаться во вторую часть работы алгоритма, которая осуществляет поиск пути на основе введенных пользователем данных. Данный метод предполагает изменение графов нижнего и верхнего уровней для внесения информации об опасных участках маршрута. В таком случае не потребуется повторно выполнять длительный анализ дискретного рабочего пространства.

Работа программы по поиску пути будет разделена на две части: предварительная обработка, которая выполняется только один раз при запуске программы (рис. 20), и собственно поиск пути, учитывающий опасные участки (рис. 21).

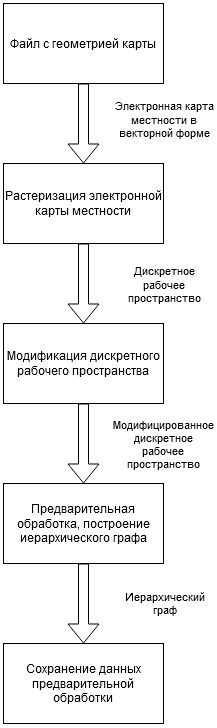


Рисунок 20. Предварительная обработка электронной карты

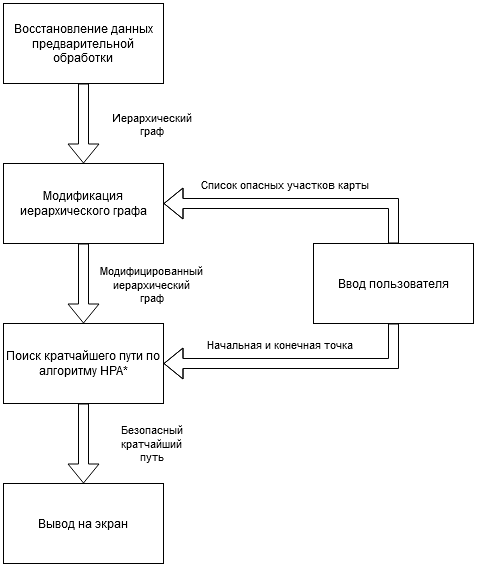


Рисунок 21. Поиск пути с помощью HPA\*

Такой вариант модификации алгоритма для поиска безопасного пути является достаточно эффективным по времени выполнения.

Рассмотрим модификацию иерархического графа. Как было сказано выше, HPA\* разбивает карту на кластеры, ищет «проходы» между соседними кластерами, а также вычисляет длины кратчайших путей между проходами внутри каждого кластера. Для учёта опасных участков карты необходимо наложить растрированную опасную область на дискретное рабочее пространство, определить измененные кластеры, заново найти проходы между кластерами-соседями и пересчитать длины путей между проходами. Блок-схема модификации приведена на рисунке 22.



Рисунок 22. Блок-схема модификации алгоритма HPA\*

Визуализация работы данной модификации изображена на рисунках 23 - 25. В середину исходного дискретного рабочего пространства (рис. 23) будет добавлено новое препятствие. Непроходимые ячейки изображены черным цветом.

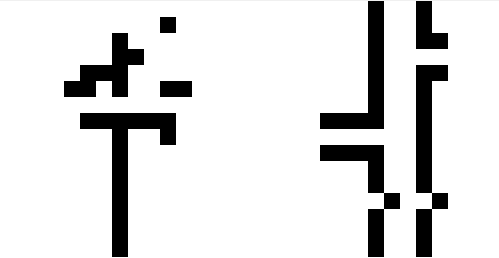


Рисунок 23. Исходное дискретное рабочее пространство

Это препятствие попадет в периметры сразу нескольких кластеров, поэтому произойдут значительные изменения в иерархическом графе. На рисунке 24 изображено исходное состояние верхнего уровня иерархического графа. Синими кругами показаны его узлы, зелеными – грани. Нижний уровень не изображен. Красные линии разграничивают кластеры.

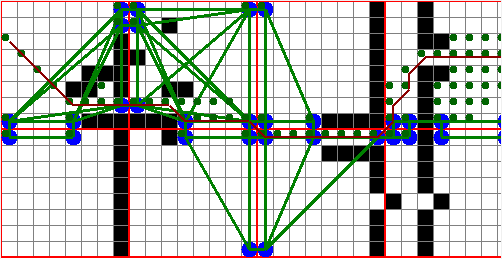


Рисунок 24. Исходное состояние иерархического графа

Добавим в дискретное рабочее пространство слой с препятствием (рис. 25).

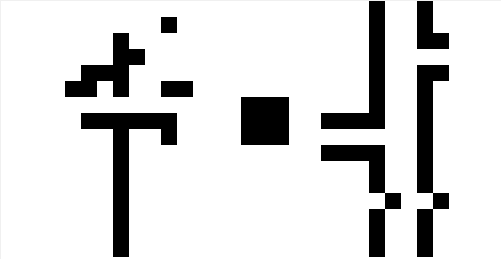


Рисунок 25. Добавлено новое препятствие

Результат работы алгоритма модификации иерархического графа представлен на рисунке 26. Для кластеров, содержащих ячейки нового препятствия были заново найдены проходы и пересчитаны длины внутренних путей.

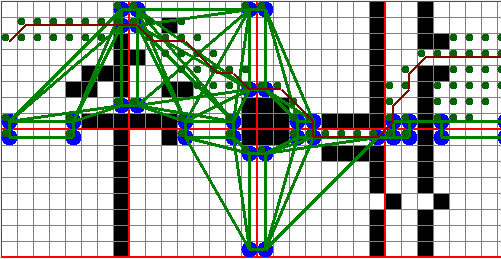


Рисунок 26. Модификация иерархического графа

Алгоритм сглаживания пути

Алгоритмы поиска кратчайшего пути выдают последовательность ячеек дискретного рабочего пространства, которая представляет кратчайший путь из начальной ячейки в конечную. Этот путь является кратчайшим, если элементарной составляющей пути является ячейка дискретного рабочего пространства. В реальности же объекты не движутся по ячейкам. Данная проблема проиллюстрирована на рисунке 27. Зеленой линией изображен путь, найденный алгоритмом HPA\*. Синими стрелками изображен путь, который является кратчайшим для человека. Из-за того, что HPA\* работает на дискретном поле с целочисленными координатами, с его точки зрения путь действительно является кратчайшим. Однако для пользователя это не так.

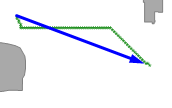


Рисунок 27. Проблема рабочего пространства

Чтобы избежать этой ситуации был разработан алгоритм сглаживания пути. Результат его работы на аналогичных исходных данных представлен на рисунке 28.

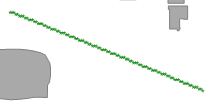


Рисунок 28. Сглаживание пути

Блок схема алгоритма сглаживания пути представлена на рисунке 29.

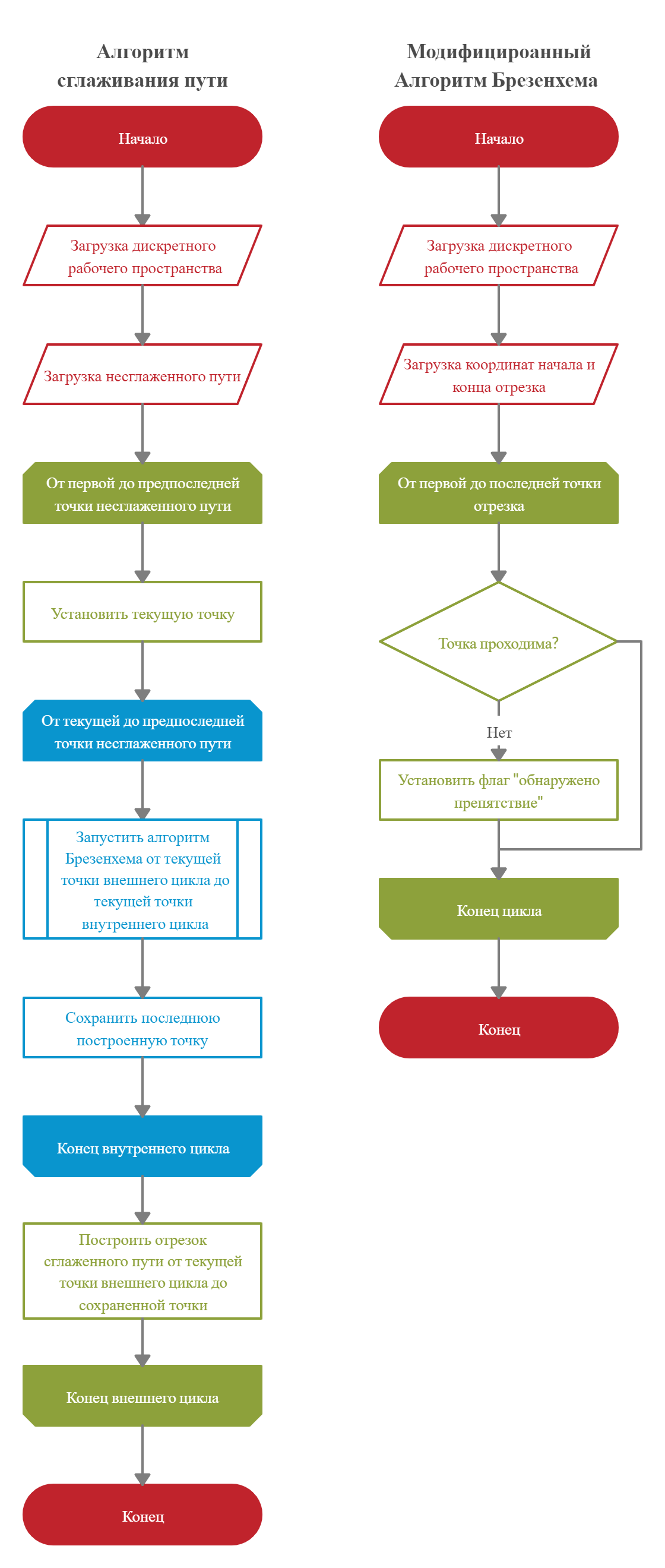


Рисунок 29. Алгоритм сглаживания пути

На рисунке 30 изображен пример работы алгоритма на реальных картографических данных. Видно, что путь представляет собой набор отрезков с различными углами наклона.

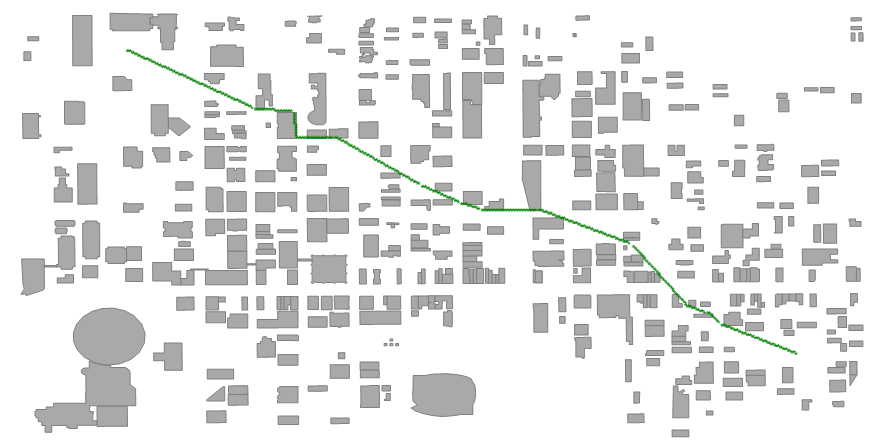


Рисунок 30. Сглаженный путь на реальной карте местности

Вывод

В конструкторском разделе были описаны существующие системы отображения картографической информации и был произведен выбор такой системы. Также описан формат хранения геометрии карты. Представлены описания использовавшихся алгоритмов растеризации, описан принцип преобразования электронной карты местности в дискретное рабочее пространство, пригодное для работы алгоритмов поиска кратчайшего пути. Были сформулированы требования к модификации алгоритма HPA\*, которая позволит пользователю исключать опасные фрагменты карты. В итоге была разработана модификация алгоритма HPA\* для поиска безопасного кратчайшего пути.

Технологическая часть

Выбор среды и языка программирования

В качестве платформы для разработки программного продукта была выбрана “.NET Framework” и язык C#. Выбор обоснован несколькими факторами:

1. платформа .NET Framework снабжена мощными ресурсами документации, которые позволяют найти информацию и рассмотреть тонкие моменты практически каждой детали системы;
2. язык C# значительно безопаснее языка C++ с точки зрения управления памятью и уровней доступа. В C# умышленно удалены многие опасные с архитектурной точки зрения возможности, такие как множественное наследование;
3. благодаря технологии JIT-компиляции, программы, написанные на этом языке, не имеют значительных потерь в производительности по отношению к компилируемым языкам, таким как C++, при этом позволяя системе контролировать безопасность кода и не допускать выполнения критических инструкций;
4. платформа (и язык C#) позволяют комбинировать управляемый код с библиотечными вызовами кода компилируемых языков, таких как C и C++;
5. язык C# позволяет разработчику сконцентрироваться на основных задачах проекта, не отвлекаясь на мелочи (благодаря ресурсам платформы .NET Framework).

Разработка структуры программного комплекса

Рассмотрим Use-Case диаграмму разрабатываемого программного продукта (рис 31).

Первым обязательным шагом является выбор файла, содержащего электронную карту местности. После этого пользователь может выполнить любое из следующих действий:

1. выбрать центр области просмотра;
2. выбрать масштаб карты;
3. указать опасные участки карты;
4. запросить поиск кратчайшего пути.

Все эти действия могут быть пропущены, т.к. программа имеет значения по умолчанию:

1. после загрузки карты центр устанавливается в геометрический центр электронной карты;
2. после загрузки карты устанавливается минимальный масштаб, достаточный для того, чтобы все объекты карты уместились в экране программы;
3. после загрузки карты список опасных участков является пустым.

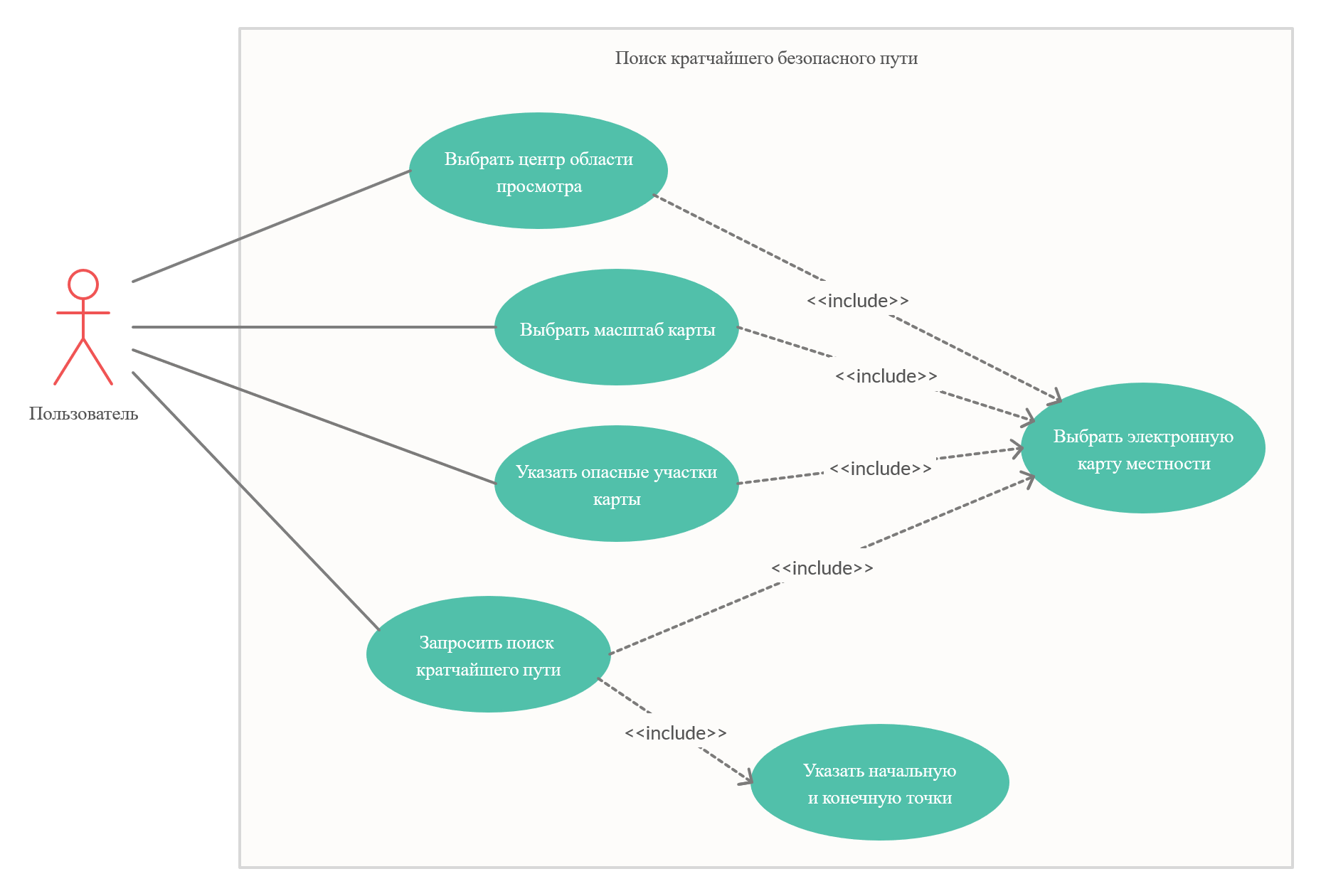


Рисунок 31. Use-Case

С учетом Use-Case диаграммы рассмотрим структуру программного проекта. На рисунке 32 изображена схема уровней абстракции.

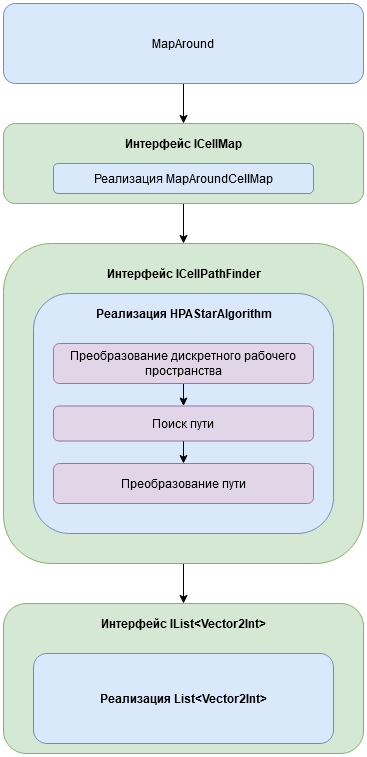


Рисунок 32. Уровни абстракции

Точкой входа приложения является форма MainForm, которая обрабатывает события ввода и занимается рендерингом электронной карты местности. Она является своеобразным «мостом» между пользователем и программой.

Следующим уровнем является программный интерфейс ICellMap. Данный интерфейс представляет собой абстрактное дискретное рабочее пространство. Интерфейс предоставляет информацию о размерах дискретного рабочего пространства и о проходимости ячеек. Для алгоритмов поиска пути внутренняя структура дискретного рабочего поля не имеет значения, следовательно, реализаций интерфейса ICellMap может быть сколь угодно много. При этом поведение алгоритмов поиска пути не изменится.

У интерфейса ICellMap имеется производный тип ICellFragmen (на рисунке 32 отсутствует). Он содержит дополнительную информацию – смещение относительно точки (0, 0). С помощью этого типа можно сообщать алгоритму поиска пути об указанных пользователем опасных фрагментах карты. Каждый ICellFragment является ICellMap (по определению интерфейса), поэтому появляется возможность строить многослойные дискретные рабочие пространства. Это свойство используется в модификации алгоритма HPA\*.

Далее идет уровень абстракции ICellPathFinder. Этот интерфейс представляет алгоритм поиска кратчайшего пути. Для программы не имеет значения принцип работы алгоритма, поэтому все реализованные алгоритмы поиска пути скрываются в воображаемом «черном ящике». По умолчанию, в программе используется реализация HPAStarAlgorithm, которая работает на основе модифицированного алгоритма HPA\* (см. аналитический раздел).

Последним уровнем является представление результирующего пути в виде списка точек. Список точек упакован в интерфейс IList<Vector2Int>. Каждый алгоритм может возвращать различные реализации IList, но обычно за оберткой скрывается встроенный тип List.

За счет использования такой структуры, программа является легко модифицируемой. Также это позволяет быстро и без изменений интерфейсов изменять или тестировать используемый алгоритм.

На рисунке 33 представлена упрощенная диаграмма классов разработанного программного продукта.

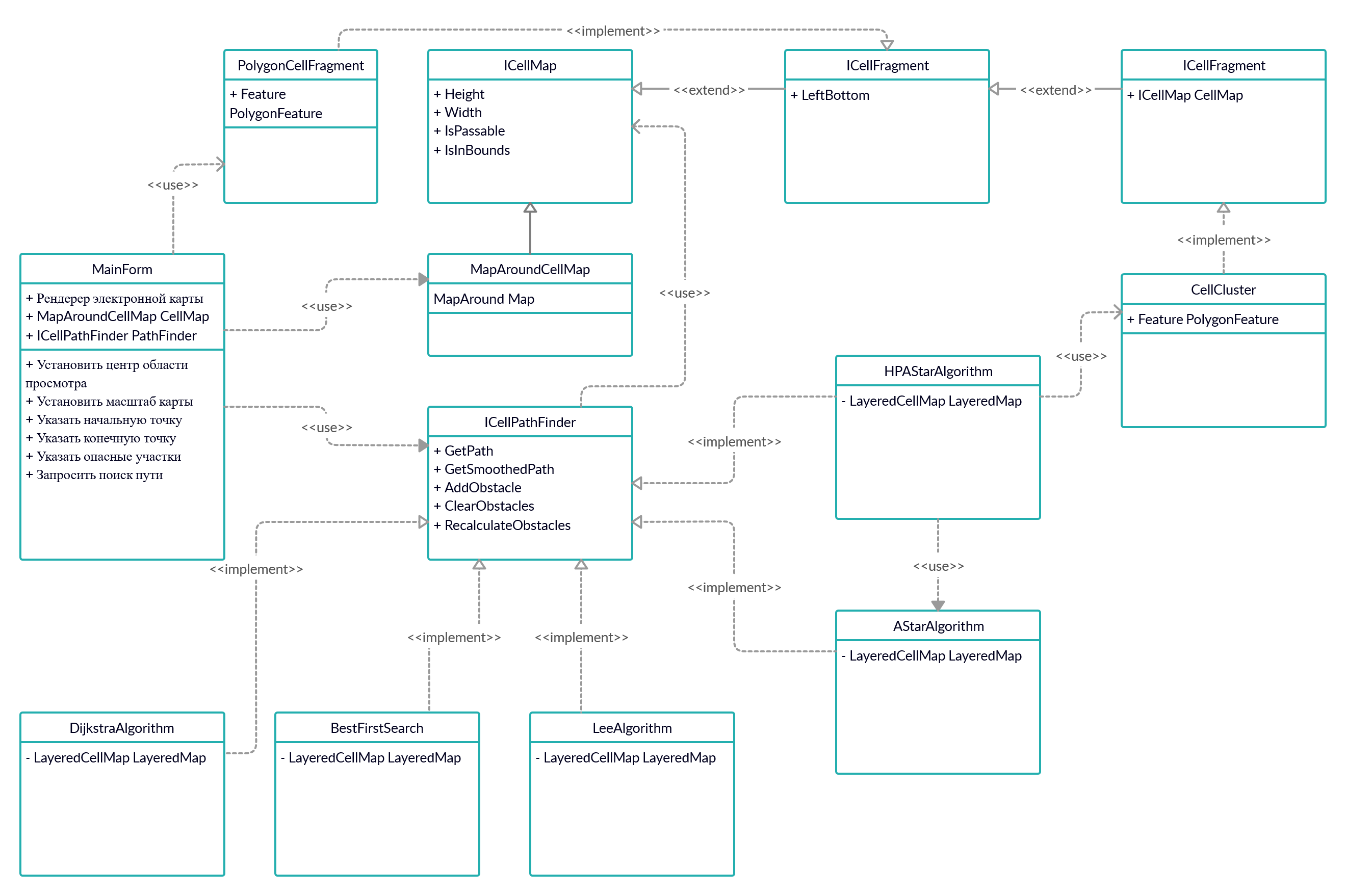


Рисунок 33. Диаграмма классов

Разработка пользовательского интерфейса

Основное окно называется «Поиск кратчайшего безопасного пути» (рис. 34).

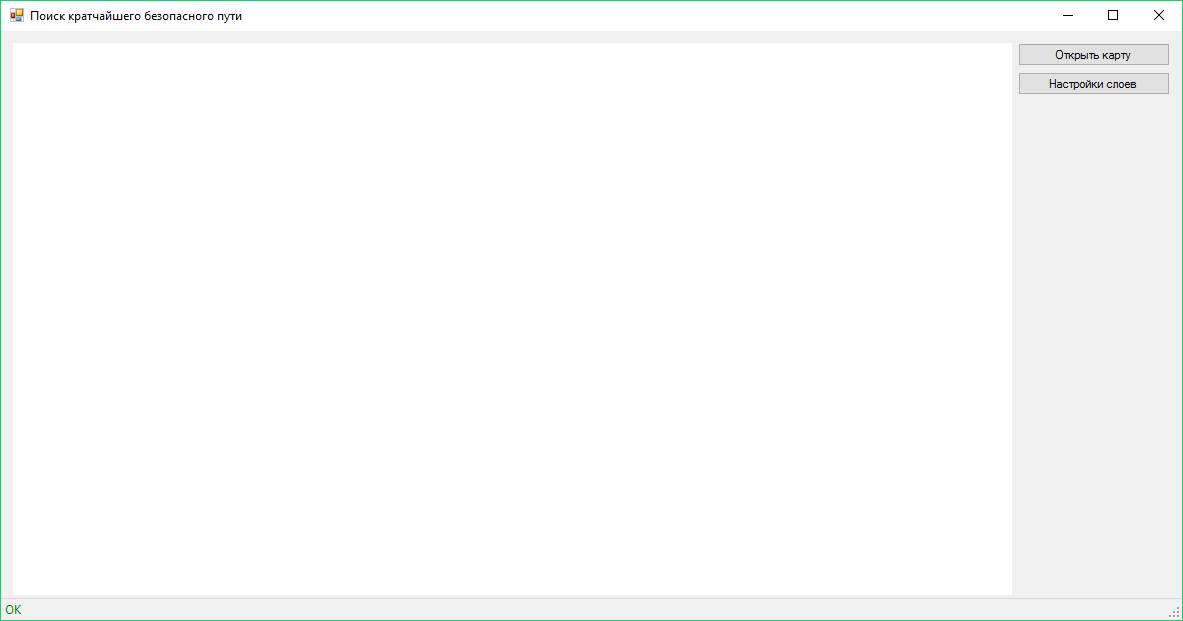


Рисунок 34. Главное окно (сразу после запуска)

Для начала работы пользователю необходимо нажать кнопку «открыть карту». Будет вызвано диалоговое окно «обзор папок» (рис. 35), с помощью которого необходимо указать на папку, содержащую файлы электронной карты местности (см. конструкторский раздел).

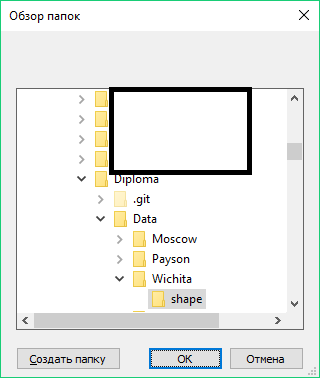


Рисунок 35. Обзор папок

Нажатие кнопки «ОК» приведет к загрузке выбранной карты и инициализации алгоритма HPA\*. Загрузка может занять некоторое время, в зависимости от размеров карты. Программа сразу же настроит область обзора и масштаб так, чтобы на экране вывелись все объекты карты. Во время инициализации кнопки управления отключаются для того, чтобы избежать ошибок и конфликтов параллельных операций (рис. 36).



Рисунок 36. Загруженная карта

После завершения инициализации станут доступны кнопки «открыть карту», которая позволит загрузить другую электронную карту, и кнопка «настройка слоев». Нажатие на кнопку «настройка слоев» вызывает диалоговое окно, в котором можно изменить видимость определенных слоев с помощью флажков. Справа от флажка указано имя слоя, которое соответствует названию файла «.shp», из которого данный слой был загружен. Исключения составляют только слои «User region layer» и «Path», которые создаются во время выполнения (рис. 37). Нажатие на флажок или на название слоя мгновенно изменяет состояние флажка и видимость слоя. Нажатие кнопки «ОК» или кнопки «Крестик» закрывает окно.

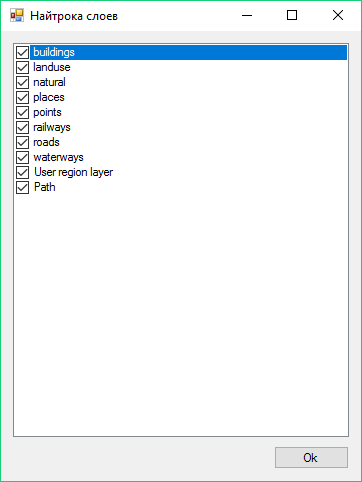


Рисунок 37. Настройка слоев

Рассмотрим окно приложения, оставив включенным только слои «buildings», «User region layer» и «Path» (рис. 38).



Рисунок 38. Несущественные слои выключены

Слой «buildings» используется для формирования дискретного рабочего пространства. User region layer хранит информацию об исключенных пользователем участках карты. Слой Path хранит точки результирующего пути, если такой путь существует.

Изменение центра окна просмотра производится с помощью перемещения курсора мыши по окну рендеринга с нажатой левой клавишей. Чтобы изменить масштаб, необходимо воспользоваться поворотом ролика мыши. На рисунке 39 представлен пример окна программы с измененным окном просмотра.



Рисунок 39. Изменение масштаба и центра карты

Для выделения опасных участков карты используются нажатия правой клавиши мыши на окно рендеринга. Каждое нажатие добавляет новую точку к многоугольнику. Для завершения многоугольника необходимо нажать на область, окружающую первую выбранную точку. Размер области – 10 пикселей. Незаконченный многоугольник изображается в виде ломаной красной линии (рис. 40).

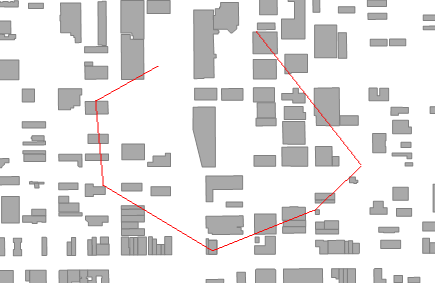


Рисунок 40. Незавершенный многоугольник

После завершения этот многоугольник добавляется на слой «User region layer» и может быть использован для построения безопасного пути (рис. 41).



Рисунок 41. Завершенный опасный участок карты

Для указания начальной и конечной точки используются нажатия на правую клавишу мыши с зажатой клавишей Ctrl. Первой указывается начальная точка. Сразу же после указания конечной точки запускается поиск кратчайшего безопасного пути. Во время работы алгоритма пользовательский интерфейс переходит в неактивное состояние. Найденный кратчайший путь отображается с помощью последовательности зеленых точек, соединяющих начальную и конечную точки (рис. 42).

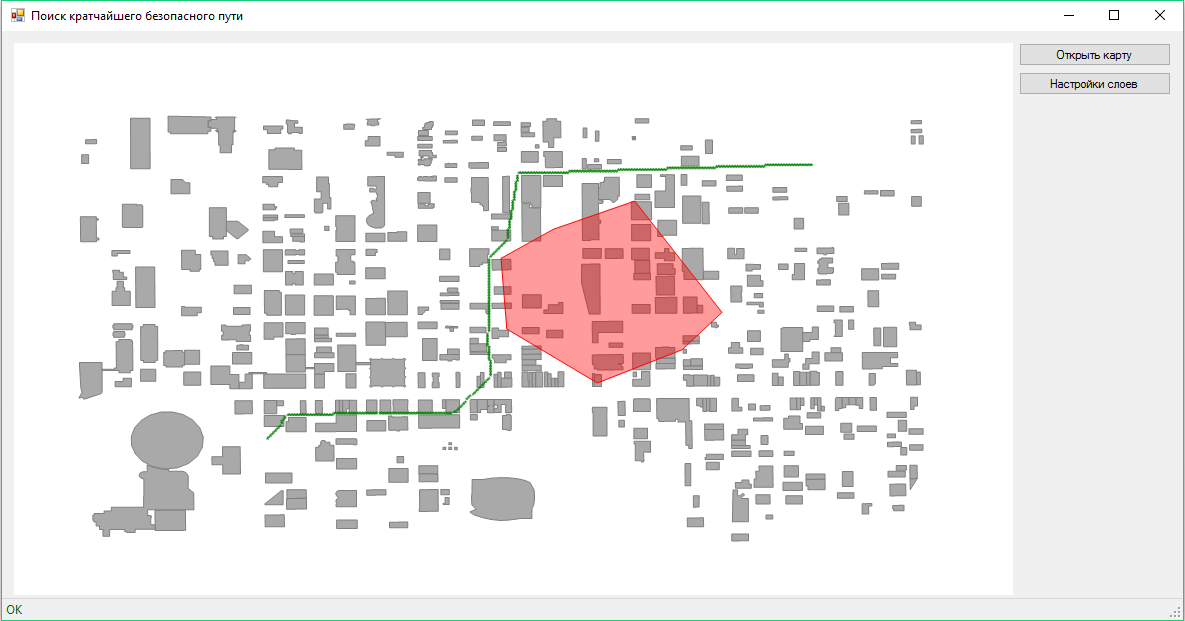


Рисунок 42. Кратчайший безопасный путь

Этот путь будет сохраняться в окне рендеринга до тех пор, пока не будут указаны новые начальная и конечная точки.

В ходе работы программы может возникнуть ряд ошибок. В случае, если пользователь укажет в качестве начальной или конечной точки такую точку, которая выходит за пределы загруженной электронной карты, то будет выведено соответствующее сообщение об ошибке (рис. 43).

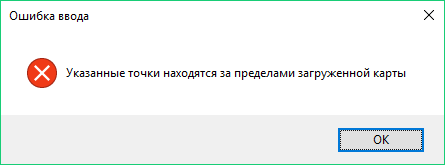


Рисунок 43. Сообщение об ошибке выхода за пределы карты

В том случае, если пути не существует, приложение выдаст сообщение «Путь не найден» (рис. 44).

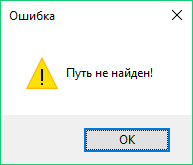


Рисунок 44. Сообщение "Путь не найден"

Модульное тестирование

Для выполнения модульного тестирования использовались средства, встроенные в Visual Studio 2019. Данная IDE позволяет значительно упростить рутинные операции, выполняемые во время тестирования.

Первым объектом тестирования является алгоритм вставки элемента в отсортированный список. Этот алгоритм применяется в алгоритме A\* для ускорения операции поиска предпочтительного узла графа. Наборы входных данных представлены в таблице 3.

Таблица 3. Тестирование алгоритма вставки в отсортированный список

|  |  |  |  |
| --- | --- | --- | --- |
| **Класс входных данных** | **Входные данные (список)** | **Входные данные (элемент)** | **Корректный результат (сортировка по убыванию)** |
| Пустой список | Пустой список | 7 | {7} |
| Один элемент, меньший вставляемого | {6} | 7 | {7, 6} |
| Один элемент, больший вставляемого | {8} | 7 | {8, 7} |
| Вставка в начало списка | { 6, 5, 4, 3 } | 7 | { 7, 6, 5, 4, 3 } |
| Вставка в конец списка | { 10, 9, 8 } | 7 | { 10, 9, 8, 7 } |
| Вставка в середину списка | { 10, 9, 8, 6, 5} | 7 | { 10, 9, 8, 7, 6, 5 } |

Следующим объектом тестирования является преобразование дискретного рабочего пространства во взвешенный граф. Проверяется количество созданных узлов и корректность связей между ними. На рисунке представлена блок-схема алгоритма тестирования генератора взвешенного графа. Под словом «соединен» подразумевается существование связи, имеющей не бесконечный вес. Конкретные значения весов не проверяются, так как они зависят от выбранного способа определения расстояния между ячейками.

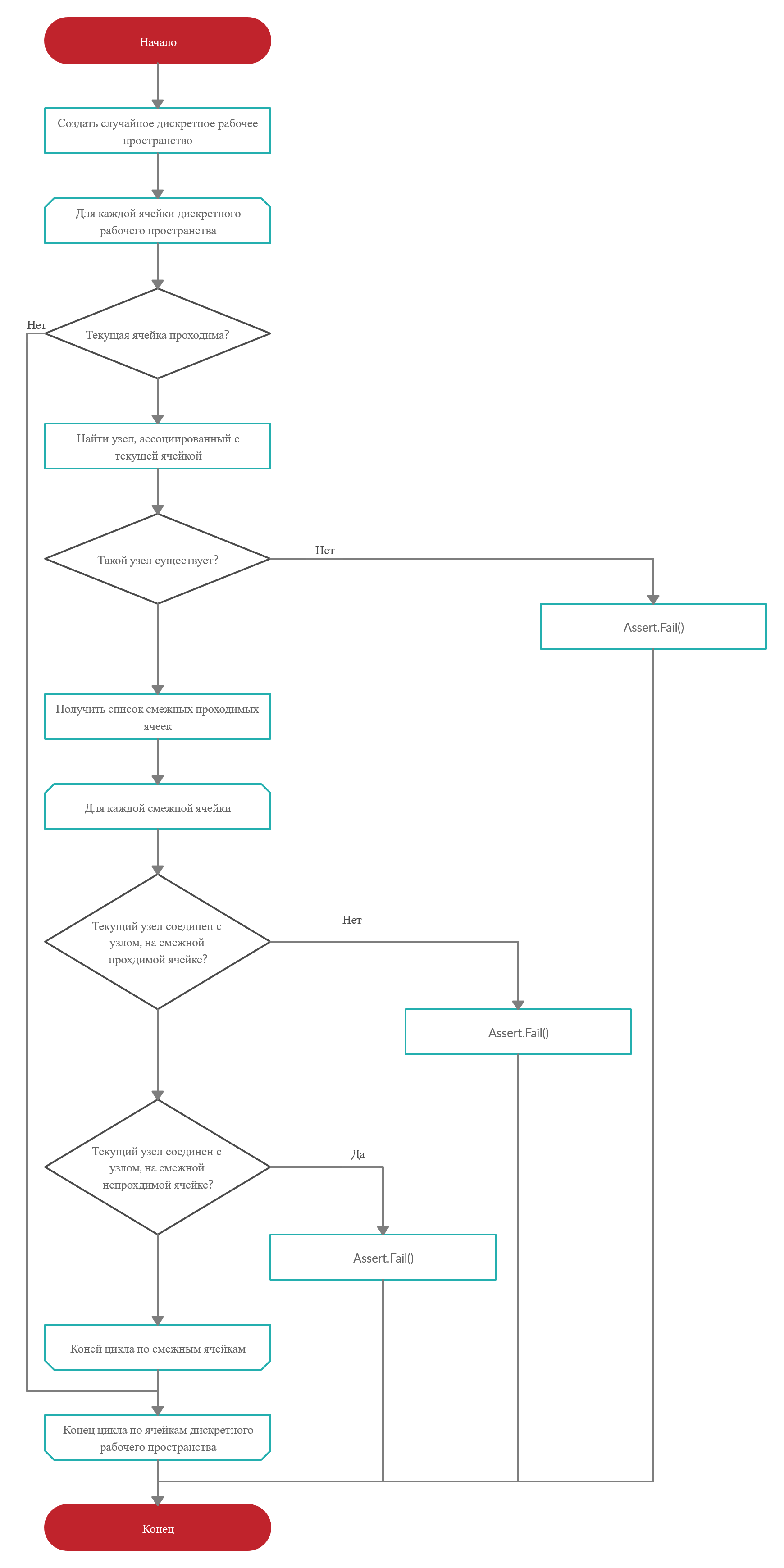


Рисунок 45. Тестирование генератора графов

Для тестирования алгоритмов поиска пути применялось ручное визуальное тестирование. Автоматизировать данный процесс довольно сложно в связи с неоднозначностью результата (во многих ситуациях может быть несколько правильных результатов). Для тестирования работоспособности простых алгоритмов (Алгоритм Ли, A\*) использовалось специально разработанное приложение, визуализирующее работу алгоритмов на небольшом дискретном рабочем пространстве. Тестовые карты задавались с помощью текстовых файлов особого формата:

XXXXXXXXXXXXXXXX

X....X.........X

X.XXXXXXXXXXXX.X

X..............X

X.XXXX.XXXXXXX.X

X....X.X.......X

XXXX.X.X.XXXXX.S

E..X.X.X....X..X

XX.X.X.XXXXXXX.X

XX.X.X.X.......X

X..X..X..XX.XXXX

X.XXX.X.X......X

X.XXX.X.X.XX.X.X

X.....X.X.X..X.X

XXXXXXXXXXXXXXXX

Данный файл с помощью символьных кодов формировал небольшое дискретное рабочее пространство, а также указывал начальную и конечную точки. Приложение позволяет отобразить ход выполнения двух разных алгоритмов на идентичных входных данных, что позволят сравнить результаты их работы (рис. 46). Зеленым цветом обозначаются непроходимые участки, красным – результат работы. В ячейках, которые «посетил» алгоритм, пишется расстояние от текущей ячейки до начальной. С помощью этой информации можно определить длину пути, а также оценить эффективность алгоритма по количеству «посещенных ячеек».

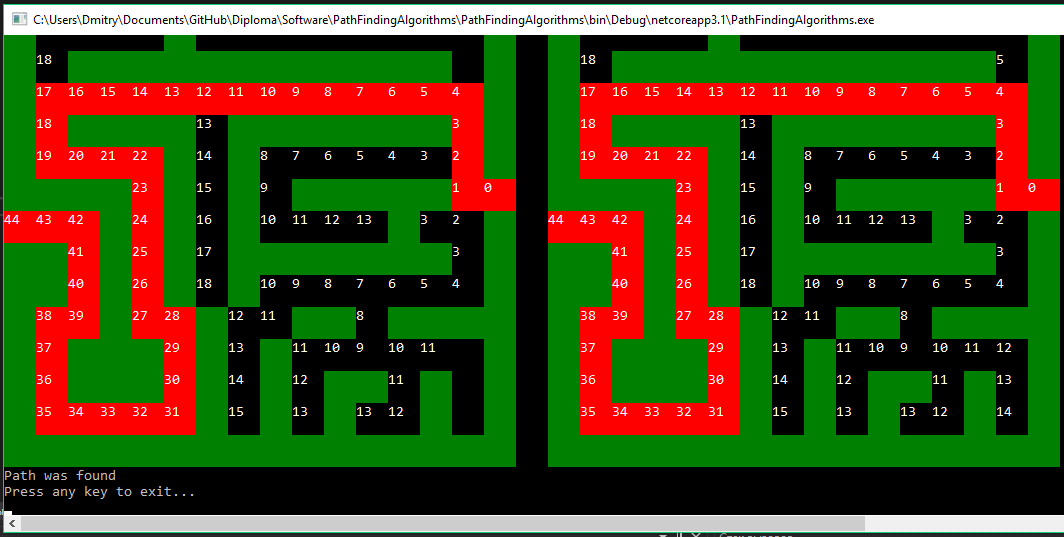


Рисунок 46. Тестирование алгоритмов

Для тестирования алгоритма HPA\* требуется уже более сложная графика. Поэтому для него был разработана программа визуализации, которая отображает внутренние данные HPA\* (рис. 47): структуру кластеров (красные линии), ячейки дискретного рабочего пространства (серые линии, разделяющие белые – проходимые и черные – непроходимые ячейки), посещенные ячейки (зеленые круги) и структуру иерархического графа (синие круги для узлов, зеленые линии для граней верхнего уровня). Эта же программа позволяет протестировать работу алгоритма модификации иерархического графа. Исходное дискретное рабочее пространство задается с помощью текстовых файлов, аналогичных тем, которые использовались для тестирования простых алгоритмов.

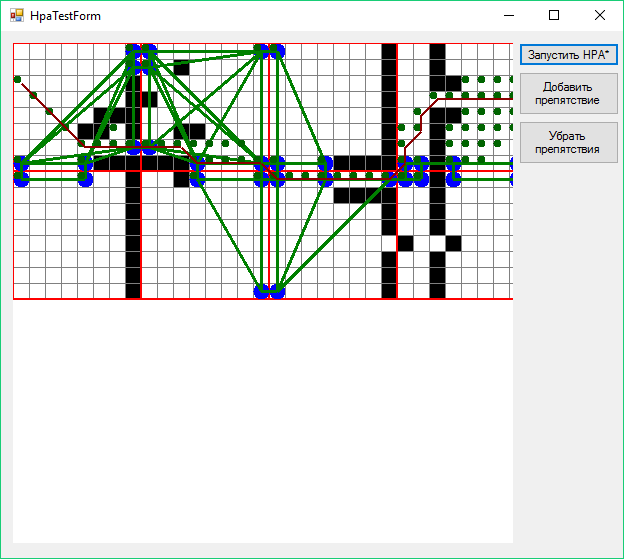


Рисунок 47. Тестирование алгоритма HPA\*

Для тестирования алгоритма построчного заполнения также была разработана программа-визуализатор (рис. 48). Аналогичным образом проводилось тестирование алгоритма создания дискретного рабочего пространства (рис. 49).

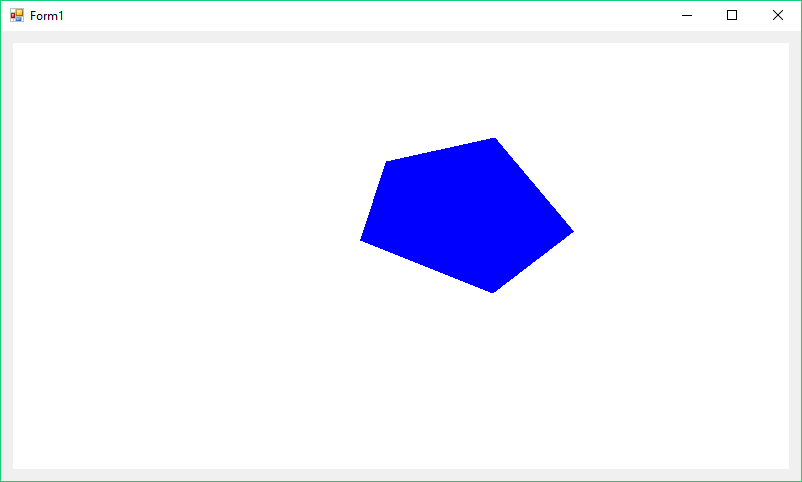


Рисунок 48. Тестирование алгоритма построчного заполнения



Рисунок 49. Тестирование генератора дискретного рабочего пространства

Заключение

В ходе работы были рассмотрены существующие алгоритмы поиска кратчайшего пути, изучены области их применения, проанализированы сильные и слабые стороны наиболее известных алгоритмов. Были полностью выполнены поставленные задачи: осуществлен выбор алгоритма поиска кратчайшего пути, проведена его модификация и реализован программный продукт с его использованием. Цель работы достигнута в полной мере, программный продукт решает проблему поиска кратчайшего пути на карте местности с учетом требований пользователя по исключению опасных участков.

Из данной работы можно сделать вывод, что проблема поиска кратчайшего пути является как никогда актуальной, а идеального средства для его решения так и не найдено. Все существующие алгоритмы имеют свои преимущества и недостатки, в связи с чем невозможно уверенно выбрать универсальный алгоритм, подходящий для решения любой задачи из этой области. Алгоритм HPA\*, который был рассмотрен и модифицирован в данной работе, является наиболее подходящим для поставленной в этой работе задачи.

Список литературы

1. Prepared Statement of Capt. Walter M. Locke, U.S. Navy, Tomahawk Cruise Missile Project Manager. Hearings before the Subcommittee on Research and Development of the Committee on Armed Services, United States Senate, March 31, 1977. — Washington, D.C.: U.S. Government Printing Office, 1977. — 6454 p.
2. Engineering Route Planning Algorithms. Daniel Delling, Peter Sanders, Dominik Schultes, and Dorothea Wagner. Universit ̈at Karlsruhe (TH), 76128 Karlsruhe, Germany
3. <http://users.eecs.northwestern.edu/~haizhou/357/lec6.pdf>
4. Zarembo, Imants & Kodors, Sergejs. (2015). Pathfinding Algorithm Efficiency Analysis in 2D Grid. Environment. Technology. Resources. Proceedings of the International Scientific and Practical Conference. 2. 46. 10.17770/etr2013vol2.868. Страница 47
5. *Pearl J.* Heuristics: Intelligent Search Strategies for Computer Problem Solving. — Addison-Wesley, 1984. — С. 48.
6. http://webdocs.cs.ualberta.ca/~kulchits/Jonathan\_Testing/publications/ai\_publications/jogd.pdf
7. <http://www.gisa.ru/13058.html>
8. <https://www.esri.com/library/whitepapers/pdfs/shapefile.pdf>
9. http://www.maparound.ru
10. <http://www.maparound.ru/Download.aspx>
11. <http://algolist.ru/graphics/fill.php>
12. *Роджерс Д.* Алгоритмические основы машинной графики. — М.: Мир, 1989. — С. 54-63. — ISBN 5-03-000476-9.