ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ
ВЫСШЕГО ПРОФЕССИОНАЛЬНОГО ОБРАЗОВАНИЯ
«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
“ВЫСШАЯ ШКОЛА ЭКОНОМИКИ”»

Московский институт электроники и математики
Департамент компьютерной инженерии
Вдовкин Василий Алексеевич

Разработка программы для поиска кратчайшего пути на карте с учетом препятствий

Междисциплинарная курсовая работа по направлению подготовки
09.03.01 «Информатика и вычислительная техника»
студент группы № ИВТ-11
(образовательная программа «Информатика и вычислительная техника»).

Научный руководитель
Ассистент Романов Александр Юрьевич

_____

Москва 2015

Содержание

1 Аннотация **2**
1.1 Русский . . . 2
1.1.1 Основная цель . . . 2
1.1.2 Задачи . . . 2
1.1.3 Результаты . . . 2
1.1.4 Рекомендации . . . 2
1.2 English (WIP) . . . 3
1.2.1 The main objective . . . 3
1.2.2 Tasks . . . 3
1.2.3 Results . . . 3
1.2.4 Recomendations . . . 3

2 Введение **4**

3 Теоретическая часть **5**
3.1 Алгоритмы . . . 5
3.1.1 Алгоритм Дейкстры . . . 5
3.1.2 Алгоритм A* . . . 7

4 Реализация **9**
4.1 Список используемых технологий . . . 9
4.2 Язык программирования и библиотеки . . . 9
4.3 Реализации алгоритмов . . . 10
4.3.1 Структура карты . . . 10
4.3.2 Структура поиска пути . . . 11
4.3.3 Эвристика . . . 12
4.4 Реализация интерфейса . . . 13
4.4.1 Внешняя спецификация . . . 13
4.4.2 Внутренняя спецификация . . . 15

5 Заключение **17**

Список литературы **18**

1 Аннотация

1.1 Русский

1.1.1 Основная цель

Исследование алгоритмов, позволяющих определить кратчайший путь между двумя вершинами в графе. Их практическое применение на примере карты с препятствиями.

1.1.2 Задачи

Реализация программы на любом языке программирования, визуализирующей работу рассматриваемых алгоритмов поиска кратчайшего пути.

1.1.3 Результаты

Реализована программа на языке JavaScript, позволяющая:

- Найти кратчайший путь между двумя точками на карте, если он существует.
- Оценить время работы алгоритма, количество обработанных вершин, существование оптимального пути и его длину.
- Изучить и сравнить алгоритм A* и алгоритм Дейкстры на основе визуализации их работы.
- Изменять размер карты и её топологию.
- Выбирать начальную и конечную точку, если они не являются препятствиями.

1.1.4 Рекомендации

- Имеется начальная карта с размещёнными препятствиями, начальной точкой и конечной точкой пути.
- Начальная и конечная точки должны находятся внутри карты.

- Конечная точка должна быть пустой.

1.2 English (WIP)

1.2.1 The main objective

Исследование алгоритмов, позволяющих определить оптимальный путь между двумя вершинами в графе.

1.2.2 Tasks

Реализация программы на любом языке программирования, визуализирующей работу рассматриваемых алгоритмов поиска оптимального пути.

1.2.3 Results

Реализована программа на языке JavaScript, позволяющая:

- Найти кратчайший путь между двумя точками на карте, если он существует.
- Оценить время работы алгоритма, количество обработанных вершин, существование оптимального пути и его длину.
- Изучить и сравнить алгоритм A* и алгоритм Дейкстры на основе визуализации их работы.
- Изменять размер карты и её топологию.
- Выбирать начальную и конечную точку, если они не являются препятствиями.

1.2.4 Recomendations

- Имеется начальная карта с размещёнными препятствиями, начальной точкой и конечной точкой пути.
- Начальная и конечная точки должны находятся внутри карты.
- Конечная точка должна быть пустой.

2 Введение

Поиск пути является одной из важнейших задач в теории графов. Решение данной задачи имеет широкое практическое применение в современных технологиях. В любой сфере разработки, в которой рабочее пространство можно представить в виде графа, реализация поиска пути является одним из основных аспектов.

Представление сети дорог ориентированным графом с положительными весами и возможностью изменения веса ребер позволяет разработчикам картографических сервисов решить данную задачу с учетом расположения физических объектов, длины дорог, их типа, проходимости, наличия пробок. Алгоритмы маршрутизации, с помощью которых информация находит свой путь от одного устройства к другому, также основываются на теории графов и задаче поиска пути. Создание искусственного интеллекта во многих играх не обходится без поиска пути между объектами на игровой карте.

Первые алгоритмы, позволяющие оптимально решить данную задачу, начали появляться в конце 50-ых — начале 60-ых годов XX века. Один из самых известных, алгоритм Дейкстры, был описан в 1959 и стал основой для многих последующих алгоритмов поиска пути. В 1968 году появилось его улучшение: алгоритм A*, который также приобрёл широкую популярность за счёт более быстрой работы в условиях определённости конечной точки.

Необходимость ускорения и оптимизации работы алгоритмов поиска пути вызвало появление огромного количество их реализаций с различными структурами хранения вершин.

3 Теоретическая часть

3.1 Алгоритмы

На сегодняшний день существует большое количество алгоритмов поиска пути, в этой работе будут рассматриваться алгоритм Дейкстры (англ. Dijkstra's algorithm) и A* (англ. A star). Алгоритмы применимы во взвешенном графе, все рёбра которого неотрицательны. Примером такого графа является сеть дорог или сетка (англ. Grid).

3.1.1 Алгоритм Дейкстры

Данный алгоритм находит кратчайший путь от стартовой вершины до любой другой в графе. Определённость конечной вершины во время работы алгоритма не требуется. Введём обозначения:

- $G = (V, E)$ — взвешенный ориентированный граф, где V — множество вершин, а E — множество рёбер.
- s — исходная вершина, v — текущая вершина, для которой кратчайший путь уже расчитан, u — вершина, имеющая общее ребро с текущей.
- $\pi[v]$ — родительская вершина.
- S — множество уже обработанных вершин.
- $w(u, v)$ — весовая функция, возвращает вес ребра uv.
- $g[u]$ — максимальная оценка кратчайшего пути из s в u.
- $g[v]$ — длина кратчайшего пути из s в v.

Входные данные: $G = (V, E), s$.

Выходные данные: $G = (V, E), \forall v \in V : g[v], \pi[v]$.

Опишем алгоритм с помощью псевдокода [1].

Путь от стартовой клетки до самой себя равен нулю, поэтому в первой строчке мы приравниваем $g[s]$ к нулю, родителя $\pi[s]$ у данной клетки нет.

Алгоритм 1: Псевдокод алгоритма Дейкстры

```
1  g[s] ← 0, π[s] ← nil
2  for u ∈ V, u ≠ s do
3      g[u] ← ∞
4      π[u] ← nil
5  Q ← {s}
6  S ← ∅
7  while Q ≠ ∅ do
8      v ← Extract − Min(Q, g[v])
9      S ← S ∪ {v}
10     for v ∉ S, uv ∈ E do
11         if g[u] > g[v] + w(u, v) then
12             g[u] ← d[v] + w(u, v)
13             π[u] ← v
14             Q ← Q ∪ {v}
```

В строчках 2-4, происходит инициализация параметров всех вершин, кроме стартовой. Для удобства считаем $g[v]$ бесконечно большим числом.

Далее, вводятся два множества Q и S. Первое множество содержит обрабатываемые вершины, а второе те, для которых кратчайший путь уже расчитан.

Пока множество Q не пустое, из него будет переноситься в множество S вершина с наименьшим $g[v]$.

Далее, в строках 12-14, для каждой смежной с ней вершиной u, не входящей в множество S, применяется основной метод алгоритма — релаксация. Эта техника заключается в хранении для каждой вершины u максимальной оценки кратчайшего пути $g[u]$ из s в u. Если из вершины u кратчайший путь $g[u]$ больше при проходе через её текущего родителя, то производится пересчёт кратчайшего пути в u через вершину-родителя v, то есть значение $g[u]$ уменьшается до $g[v] + w(u, v)$. Вершина u добавляется в множество Q. Если Q является очередью, то повторно добавлять u в очередь нельзя.

После окончания работы алгоритма у каждой вершины, из которой возможен путь к стартовой, будет определён родитель $\pi[v]$ и длина кратчайшего пути $g[v]$. Чтобы пройти по вершинам, составляющим кратчайший путь необходимо двигаться от дочерней вершины к родительской, пока она не станет равна стартовой. Если же $\pi[v]$ неопределена, то найти кратчайший путь из вершины s в v нельзя. Такое возможно только в случае несвязанного графа.

Так как вершина после попадания в множество S больше не обрабатывается, то количество итерации цикла while будет равно V в случае связанного графа и меньше V в случае несвязанного графа.

3.1.2 Алгоритм A*

Данный алгоритм находит кратчайший путь от стартовой вершины s до заданной a.

Алгоритм эквивалентен алгоритму Дейкстры с добавлением эвристики и раннего выхода при достижении заданной вершины. Допустимая эвристическая оценка $h[u]$ — примерное расстояние от вершины u до заданной a, непереоценивающая реальное расстояние, тогда $f[v] = g[v] + h[v]$ — примерное расстояние от s до a.

Отличия A* от алгоритма Дейкстры показаны на псевдокоде [2]. Так же показана работа функции pathTo, которая позволяет получить список вершин, через которые проходит кратчайший путь.

Входные данные: $G = (V, E), s, a$.

Выходные данные: $G = (V, E), g[a], \pi[a]$.

Теперь текущая вершина v выбирается по минимальному значению $f[v]$, вычисляемому в 22 строчке для всех клеток, кроме стартовой, в множестве Q. Очевидно, что при $h[v] = 0$ A* превращается в алгоритм Дейкстры.

В строчках 16-17 реализован ранний выход.

Алгоритм 2: Псевдокод алгоритма A*

```
1  function pathTo(v)
2      path ← ∅
3      while ∃π[v] do
4          path ← path ∪ {v}
5          v ← pi[v]
6      return reverse(path)
7  g[s] ← 0, π[s] ← nil, f[s] = 0 + h[s]
8  for u ∈ V, u ≠ s do
9      g[u] ← ∞
10     π[u] ← nil
11 Q ← {s}
12 S ← ∅
13 while Q ≠ ∅ do
14     v ← Extract-Min(Q,f[v])
15     S ← S ∪ {v}
16     if v = a then
17         exit pathTo(v)
18     for v ∉ S, uv ∈ E do
19         if g[u] > g[v] + w(u, v) then
20             g[u] ← d[v] + w(u, v)
21             π[u] ← v
22             f[v] ← g[v] + h[v]
23             Q ← Q ∪ {v}
```

4 Реализация

4.1 Список используемых технологий

1. JavaScript — интерпретируемый браузером язык программирования.
2. HTML5, CSS — языки разметки для создания интерфейса.
3. Bootstrap, jQuery — подключённые библиотеки.
4. LaTeX — система компьютерной верстки, значительно облегчающая создание технической литературы. Используется для создания отчёта.
5. VCS Git — система контроля версий.

4.2 Язык программирования и библиотеки

Для реализации поставленных задач был выбран язык JavaScript. Его программой-интерпретатором является браузер (ссылка). Язык прост в освоении, имеет похожий на язык C синтаксис, динамическую типизацию переменных, элементы объектно-ориентированного программирования позволяет работать с обработчиками событий(сслыка).

JavaScript поддерживается всеми современными браузерами, что обеспечивает реализованной программе кроссплатформенность и возможность запуска даже на мобильных устройствах.

По соображениям безопасности, на язык наложены некоторые ограничения. Он не имеет прямого доступа к операционной системе. Например, возможность чтения и записи файлов сильно ограничена.

Язык имеет полную интеграцию с языком разметки HTML и CSS. Первый позволяет создавать прототип интерфейса, его скелет, а второй определяет внешний вид объектов интерфейса.

Вместе тройка данных языков предоставляет невероятные возможности для построения интерфейса и визуализации, что хорошо подходит для осуществления поставленных задач.

Для упрощения работы подключены библиотеки jQuery и Bootstrap. jQuery используется для быстрого доступа к содержимому HTML и необ-

ходим для работы Bootstrap. Bootstrap предоставляет готовые решения для элементов интерфейса, выполненные профессиональными дизайнерами.

Совокупность перечисленных языков и библиотек позволяет сосредоточится на алгоритмах и правильной работе коде, а не тратить время на изучение огромного количества документации к интерфейсу, как в многих решениях для компилируемых языков программирования.

4.3 Реализации алгоритмов

4.3.1 Структура карты

Граф представлен сеткой (англ. Grid) — массив клеток, который условно можно назвать картой. Каждая клетка является вершиной, имеющей 4 или 8 соседних вершин в зависимости от разрешённости диагонального движения. Координаты клетки определяются её положением в двухмерном массиве карты. Чтобы не загружать браузер вычислениями квадратного корня, вес ребра равен вертикального и горизонтального ребра равны 10, а диагонального 14.

Рассмотрим клетку как прототип объекта. Определены поля f, h, g, $parent$, $closed$, $visited$, $type$, i, j. Значения первых трёх полей понятно из описания алгоритмов, приведённого выше. Поле $closed$ позволяет алгоритму понять окончательно обработана клетка или нет и избавиться от хранения объектов в закртом списке (эквивалент множества S) и постоянной проверке наличия в нём элементов. Поле $visited$ необходимо для избежания повторного добавления клетки в открытый список $openData$ (эквивалент множества Q). i, j — координаты. $type$ — тип клетки, он может принимать 6 разных значений, но для алгоритма важна только проходимость клетки. Методы у клетки однотипны и завязаны на проверке клетки на значение поля $type$ или его изменения, возвращают логический тип.

При генерации клеток в зависимости от коэффициента рандомно определяется является ли клетка препятствием или нет. Пользователь может изменять топологию карты: убирать старые и создавать новые препятствия.

4.3.2 Структура поиска пути

Для удобного использования и интеграции с интерфейсом поиск пути реализован в объекте, состоящем из методов:

- $ClearParams$ — входные параметры: map — двумерный массив клеток. Изменяются поля каждой клетки, использующиеся в процессе работы алгоритма, до стандартных значений на случай, если поиск вызывается повторно на одной и той же карте.

- $pathTo$ — входные параметры: $cell$ — объект-клетка. Проходит по родительским клеткам, пока они существуют. Возвращает спиок клеток $path$ — кратчайший путь.

- $getG$ — входные параметры: клетки $cell, parent$. Возвращает коэффициент $g[cell]$ при проходе через новую клеку-родителя $parent$.

- $relax$ —входные параметры: $cell, newparent, G$. Производит релаксацию. Значение $g[cell]$ уменьшается до G и $cell$ присваивается новый родитель $newparent$.

- $heuristic$ — входные параметры: $pos0, pos1$ — массивы координат. Расчитывает точную эвристическую оценку при движении из $pos0$ в $pos1$.

- $getNeighbours$ — входные параметры: карта map, клетка $cell$. Возвращает массив клеток, имеющих общее ребро с $cell$.

- $extractMin$ — входные параметры: $openData$ список. Извлекает из данного списка минимальный по f элемент.

- $isWallCorner$ — входные параметры: $cell, parent, map$. При диагональном движении из $parent$ в $cell$ определяет есть ли на пути угол препятствия. Данный метод не используется в описании алгоритма, но появляется из-за особенности grid-карт.

Основным методом является $search$. Он вызывается из интерфейса и имеет параметры: $map, start, end, options$, где map — grid-карта, $start, end$

стартовая и конечная клетка. Так как в этом методе одновременно реализован и алгоритм Дейкстры и A*, массив $options$ необходим для конфигурации поиска пути. Первый элемент отвечает за наличие диагонального движения, а второй за то, какой именно алгоритм используется. Ранний выход при достижении конечной клетки работает не зависимо от алгоритма. Метод возвращает массив из элементов: массив клеток, составляющих кратчайший путь, время работы и длину кратчайшего пути. При включённой отладке[], клетки, извлекаемые из открытого списка, помещаются в глобальный список, который после нахождения пути обрабатывается скриптом интерфейса для наглядной визуализации работы.

4.3.3 Эвристика

Важным элементом в реализации выступает эвристика. Мы не будет считать простую дистанцию между координатами текущей и конечной клеток, так как необходимо соблюдать единость масштаба коэффициентов g и h. Для grid-карт есть два способа посчитать точную эвристику, то есть кратчайший путь без учёта препятствий на пути.

Первый способ — Манхэттенское расстояние, которое считает количество клеток по диагонали и вертикали, оставшихся до конечной клетки. Она очень хорошо подходит, когда диагональное движение запрещено и расчитывается по формуле:

$$D * (|cell.x - start.x| + |cell.y - end.y|)$$

где D — это вес ребра. В нашем случае это 10.

Второй способ — диагональное расстояние. Оно используется, когда имеется возможность передвигаться по диагоналям, причём стоимость диагональных ребёр отличается от вертикальных или горизонтальных. Расчитывается по формуле:

$$D * (dx + dy) + (D2 - 2 * D) * min(dx, dy)$$

где $dx = |cell.x - start.x|$, $dy = |cell.y - end.y|$, а $D2$ равен весу дианонального ребра. В нашем случае это 14.

Наглядно увидеть два данных вида точной эвристики можно увидеть на рис. 1 и 2.

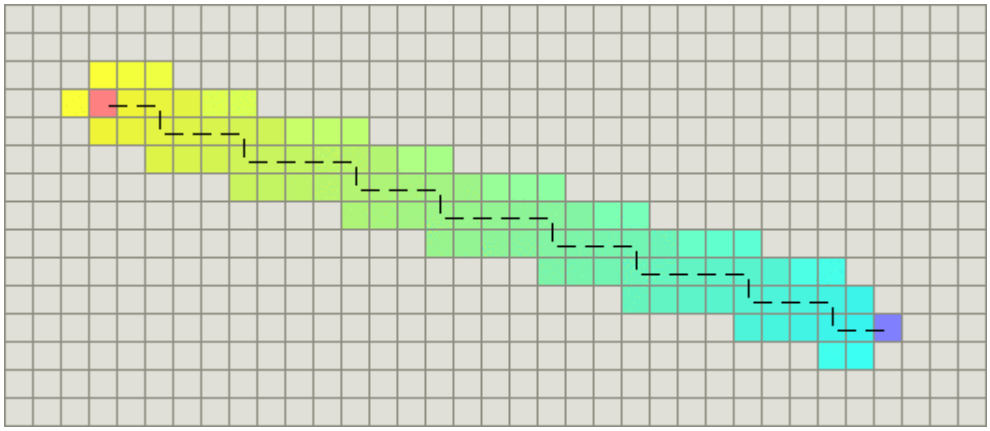

Рис. 1: Манхэттенское расстояние

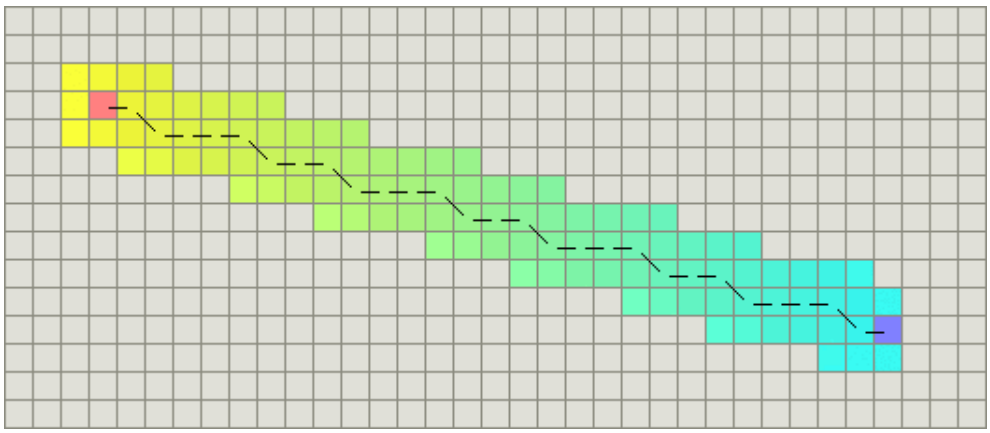

Рис. 2: Диагональное расстояние

4.4 Реализация интерфейса

4.4.1 Внешняя спецификация

Интерфейс программы представляет из себя 4 основных элемента: карта, панель управления, информационное окно, окно дополнительных настроек.

Карта (рис. 3) является главным элементом всего интерфейса, на ней выполняется демострация алгоритмов. По умолчанию размер каждой клетки равен 30 пикселям[], а количество клеток задаётся автоматически на основе

размера экрана устройства. Выбор стартовой или конечной точки производится по клику. Устанавливать или убирать препятствия можно не только по клику, но и при движении мыши с зажатой кнопкой. Если на клетке нет препятствия, то оно установится, в противном случае уберётся. После выполнения поиска путь нарисуется на карте линией. Если включен режим отладки (кнопка Debug), то обработанные клетки покроются голубым цветом

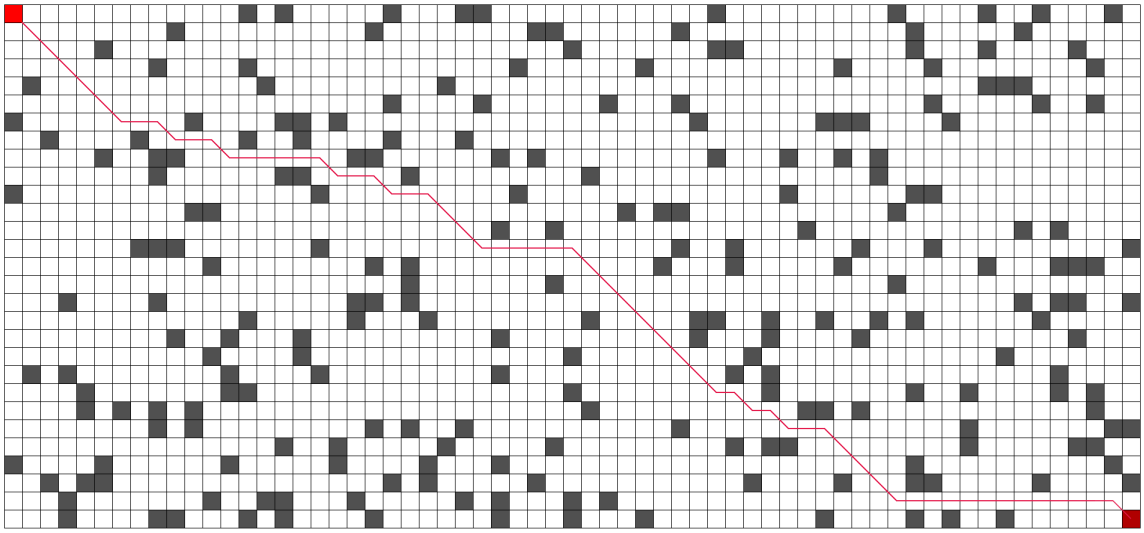

Рис. 3: Карта

В панели управления (рис. 4) можно изменить размеры карты, коэффициент случайной генерации препятствий (при 0 препятствий нет), включить режим отладки для просмотра обрабатываемых алгоритмом клеток, переключатся между режимом выбора конечной и стартовой точки и режимом установки препятствий, скрыть или показать окно информации, открыть окно дополнительных настроек.

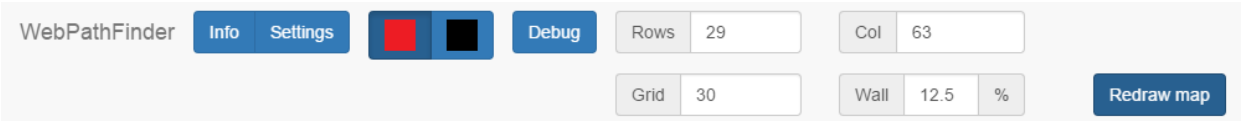

Рис. 4: Панель управления

Информационное окно (рис. 5) привязано к кнопке Info и отображает информацию о последнем поиске пути: использованный алгоритм, время поиска, длину кратчайшего пути и количество обработанных клеток (при включенном режиме отладки). Если путь не удалось найти, то окно уведомит об этом.

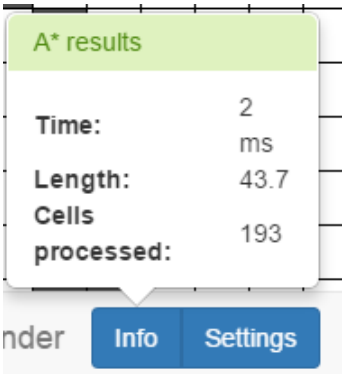

Рис. 5: Информационное окно

В окне дополнительных настроек (рис. 6) можно сменить алгоритм поиска, настроить режим отладки (WIP), отключить отрисовку границ клеток (необходимо при задании больших карт с клетками в 1-2 пикселя), отключить диагональное движение.

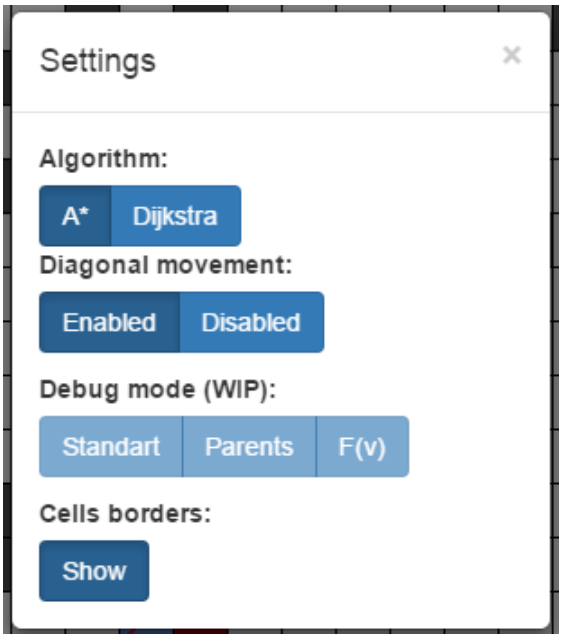

Рис. 6: Дополнительные настройки

4.4.2 Внутренняя спецификация

Интерфейс построен на обработчиках событий (англ. Event Handlers). При происхождении указанного события с указанным элементом интерфейса происходит вызов функции. Используются следующие события:

- Mousedown — срабатывает при нажатии кнопки мышки, наложен на слои

карты (всего слоёв два: основной и слой для анимации). Определяет установку стартовой и конечной точки или стены.

- Mousemove — срабатывает при движении мыши, наложен на основной слой карты. Определяет клетку, на которой находится курсор, и установку/удаление препятствий при движении мыши.

- Click — срабатывает при клике на элемент. Наложен на кнопки. («Redraw map», «Settings»)

- Change — срабатывает при кликах на checkbox и radiobox. Определяет изменение практически всех параметров для поиска.

- Keydown — срабатывает при нажатии кнопки на клавиатуре. Наложен на все поля ввода и вызывает перерисовку карты при нажатии кнопки Enter.

Для отрисовки карты используется Canvas — элемент HTML5 предназначенный для создания изображения при помощи встроенных методов на JavaScript.

Анимации созданы с помощью методов Canvas и функций setTimeout и setInterval, которые выполняют код функции через указанное время. Язык JavaScript не имеет команды sleep. Анимации выполняются после окончания поиска пути и не отражают реальное время работы алгоритма. Искусственное замедление сделано для более понятной визуализации.

Поиск пути на больших картах занимает значительное время, в этот момент браузер полностью занят выполнением функции поиска и не обрабатывает интерфейс, что вызывает его зависание на время работы алгоритма. На маленьких картах этого не заметно.

5 Заключение

В работе были рассмотрены два распространённых алгоритма поиска кратчайшего пути: A* и алгоритм Дейкстры.

В данной реализации использовался простой граф с одинаковой стоимостью рёбер. На практике же чаще всего идеальных условий не бывает. В сети дорог существуют разные типы дорог, поэтому необходимо установить разный вес рёбер: например, для асфальтированных дорог меньше, для просёлочных больше. Можно динамически редактировать вес, например, чтобы учитывать пробки: чем больше пробка, тем больше вес. Главное, соблюдать одинаковую размерность весов.

Чаще всего требуется быстрая работа алгоритма в условиях огромных карт. В реализованной программе для хранения данных используется список, что замедляет работу алгоритма, так как каждый раз приходится искать вершину с наименьшим коэффициентом и удалять её сдвигом. Для увеличения скорости можно использовать, например, бинарную кучу, тогда необходимо просто будет извлекать первый элемент.

На практике поиск кратчайшего пути за оптимальное время очень важен не только в устройствах навигации, но и в коммуникационной сфере и даже в искусственом интеллекте.

Список литературы

[1] Кормен, Т., Лейзерсон, Ч., Ривест, Р. Алгоритмы: построение и анализ = Introduction to Algorithms. — 1-е. — М.: МЦНМО, 2000. — 960 с. — ISBN 5-900916-37-5.