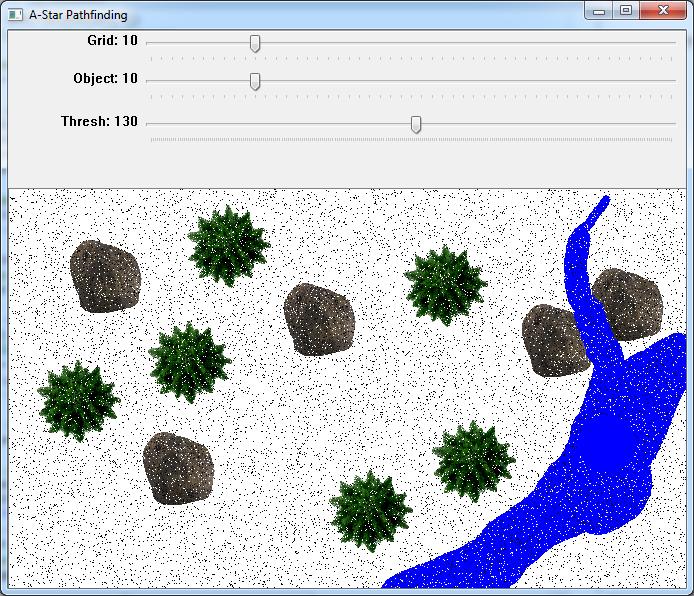
**Поиск кратчайшего пути обхода препятствий агентом**

**Алгоритм предварительной обработки изображений**

В данной программе «снимок местности» генерируется пользователем. Перед анализом снимка местности его в некоторой степени можно привести к реальному снимку, для этого на него накладывается шум типа «соль и перец».

«Соль и перец» - одна из форм шума, который как правило встречается на графических и видео изображениях. Он представляет собой случайно возникающие белые и черные пиксели.

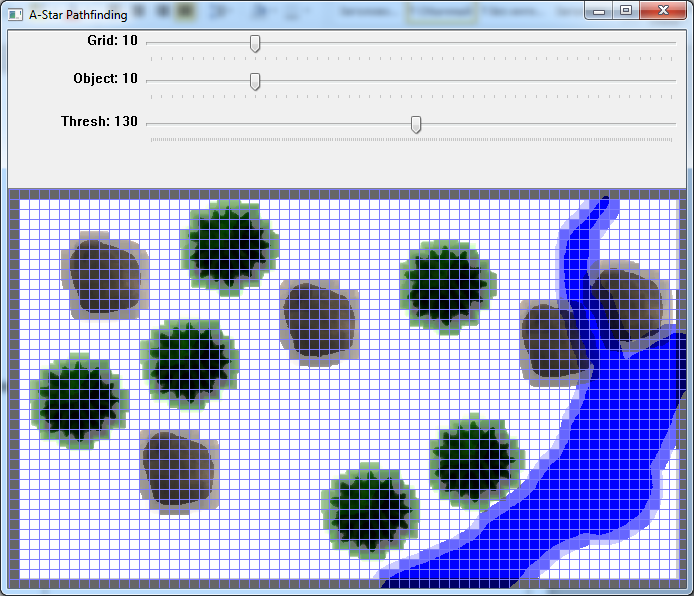


Изображение с наложенным шумом типа «соль, перец», с параметрами: плотность 0.05

Когда изображение готово к анализу, к нему применяется алгоритм медианной фильтрации, после чего шум с изображения практически исчезает.

Апертура своим центром посещает каждый пиксель изображения. Элемент изображения, попавший в центр окна заменяется медианным значением, вычисленным из массива, образованного всеми элементами изображения, попавшими в окно NxN (где N – чётное: 3, 5,7).

Медианой упорядоченной последовательности A(i), i = 1,2,…N является значение:A[(N+1)/2], если N – нечётное; A[A(N/2)+A((N+2)/2)]/2, если N – чётное.



Результат работы медианного фильтра с апертурой 5х5

После удаления шума производится анализ изображения на препятствия.

Изображение преобразовывается в чёрно-белое.

На изображения вокруг всех не белых объектов делается отступ размером с половину радиуса робота (устанавливается в интерфейсе), для того, чтобы робота можно было рассматривать как материальную точку.

После чего изображение преобразовывается в бинарное.

Затем изображение разбивается на квадратные сегменты заданного размера. Данный размер можно изменять из интерфейса программы.

Далее каждый сегмент изображения анализируется на проходимость.

Для этого вычисляется «средний» цвет сегмента изображения и если он выше порога (который, также можно установить из интерфейса), тогда сегмент считается непроходимым. Если выше – робот может быть расположен в данном сегменте.

Результатом вышеописанного алгоритма является список непроходимых сегментов изображения.

На данном этапе предварительная обработка изображений заканчивается.

По её результатам мы имеем матрицу-сетку, в координатах которой можно расположить робота и перемещаться. Данный метод позволяет снизить вычислительную нагрузку, в сравнении с перемещением по «пикселям» карты.

**Планирование траектории**

После анализа изображения и получения необходимых данных относительно них производится планирование траектории. Для её вычисления используется алгоритм нахождения кратчайшего пути А\*.

А\* - это модификация алгоритма Дейкстры, оптимизированная для единственной конечной точки. Алгоритм Дейкстры может находить пути ко всем точкам, A\* находит путь к одной точке. Он отдаёт приоритет путям, которые ведут ближе к цели.

Ключевая идея алгоритма заключается в том, что мы отслеживаем состояние расширяющегося кольца, которое называется границей. В сетке этот процесс иногда называется заливкой. Для этого повторяем следующие шаги, пока граница не окажется пустой:

- Выбираем и удаляем точку из границы.

- Помечаем точку как посещённую, чтобы знать, что не нужно обрабатывать её повторно.

- Расширяем границу, глядя на её соседей. Всех соседей, которых мы ещё не видели, добавляем к границе. Мы можем прекратить расширять границу, как только найдём нашу цель.

Также нам необходимо отслеживать откуда мы пришлидля каждой посещённой точки. Этого нам достаточно для воссоздания целого пути. Для этого следуем обратно от цели к началу.

В некоторых случаях поиска путей у разных типов движения есть разная стоимость. В нашем случае, из каждого сегмента изображения мы можем отправиться в 8 сторон. Для перемещения по диагонали введём стоимость 14, т.к. этот путь длиннее, чем перемещение по осям а для перемещения по осям введём стоимость 10.

Теперь необходимо, чтобы поиск пути учитывал эту стоимость. Для этого нам нужен алгоритм Дейкстры. Нам нужно отслеживать стоимость движения, поэтому мы будем следить за общей стоимостью движения с начальной точки. При оценке точек нам нужно учитывать стоимость передвижения. Вместо добавления точки к границе в случае, когда точку ни разу не посещали, мы добавляем её, если новый путь к точке лучше, чем наилучший предыдущий путь.

Но всё же граница расширяется во всех направлениях. Это логичный выбор, если вы ищете путь ко всем точкам или ко множеству точек. Т.к. поиск выполняется только для одной точки сделаем так, чтобы граница расширялась к цели больше, чем в других направлениях. Во-первых, определим эвристическую функцию, сообщающую нам, насколько мы близки к цели: abs(a.x - b.x) + abs(a.y - b.y). Также используем очередь с приоритетами для расширения границ в правильном направлении. Направление будем определять используя оцененное расстояние до цели. Точка, ближайшая к цели, будет исследована первой.

Таким образом у нас будет найден кратчайший путь по сетке.

Хотя с оценки кратчайшего расстояния данный алгоритм оптимален, его результатов есть 2 существенных недостатка. В поиске не учитываются повороты робота, а из-за них увеличивается время прохождения пути. А также представление карты в виде сетки сильно ограничивает перемещения робота по настоящей карте.

Есть 2 пути решения данной проблемы: изменить представление карты или спрямить полученный путь. Наиболее оптимальным и правильным решением будет изменить представление карты. Но т.к. мне интереснее попробовать второй вариант, выбираем его.

Для спрямления траектории используем следующий алгоритм.

Выбираем i позицию пути.

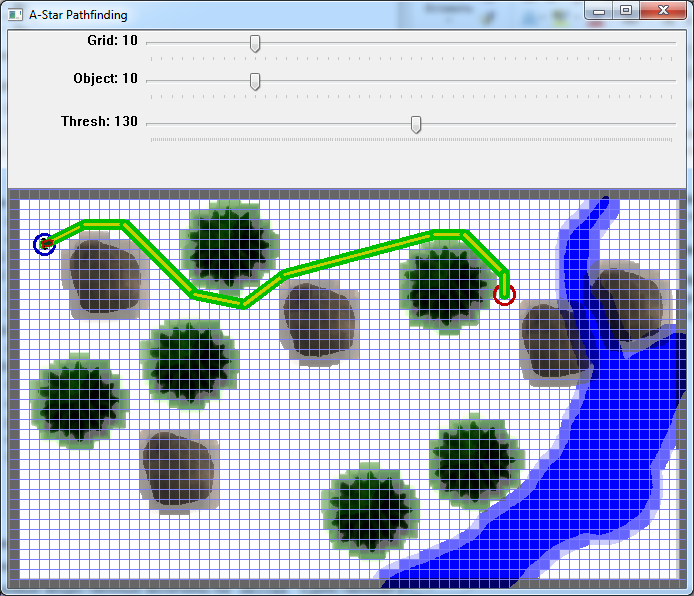
Проверяем можно ли из i позиции добраться напрямую до позиции i+2.

Если да, то удаляем из пути позицию i+1.

Если нет, то выбираем i равное i+2, после чего повторяем действия сначала до тех пор, пока i не станет равно «количеству позиций в пути минус 2».

Для проверки возможности прохождения между позициями используем алгоритм Брезенхема. Это алгоритм, определяющий, какие точки двумерного растра нужно закрасить, чтобы получить близкое приближение прямой линии между двумя заданными точками.

В ходе алгоритма определяются ближайшие пиксели к прямой линии между позициями и, т.к. робот на карте предтавлен материальной точкой, анализируя их, мы можем сделать вывод о возможности прохождении между ними.



Результат работы алгоритма поиска пути

**Составление команд**

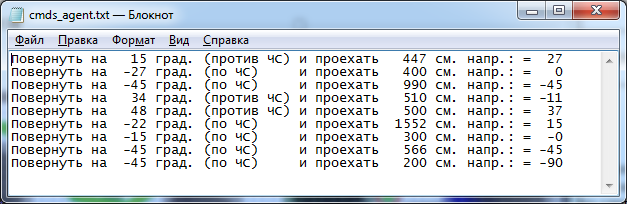
После того как мы получили координаты на карте, которые робот должен посетить для прохождения до цели, мы должны сформировать список команд, выполняя которые он может это сделать.

Команды буду формироваться для 2 уровней: робота и двигателей.

На уровне робота команды будут выглядеть более «человечнее» и удобочитаемыми, они нужны для того, чтобы было проще оценить правильность их составления. Они будут выдавать угол на который необходимо повернуть и расстояние, которое необходимо проехать.

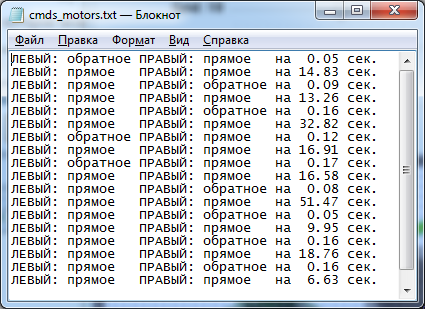
На уровне двигателей команды будут представлять время на которое буде запускаться каждый двигатель и направление в котором он будет вращаться.

Для формирования команд уровня робота, на данном этапе, мы имеем достаточно данных.

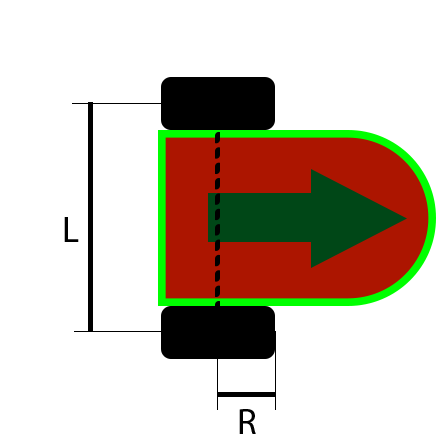


Результат генерации команд роботу

Для формирования команд двигателям необходимо знать некоторые параметры робота, а именно: обороты колеса в минуту, радиус колеса, ширина колёсной оси. Данные параметры вводятся в интерфейс сразу при запуске программы. Проведя с ними необходимые расчёты можно получить список команд для двигателей.



Результат генерации команд двигателям



План робота с необходимыми параметрами (габаритным размером робота считается расстояние от оси до самой дальней передней точки платформы)

**Описание программы**

Управление программой:

ЛКМ - стартовая точка, ПКМ - целевая точка.

Для построения пути точки можно ставить только на светлые (проходимые) поля.

m - смена режима программы (поиск пути / рисование карты препятствий).

Опции режима рисования:

ЛКМ - рисование;

c – очистка поля для рисования;

- – уменьшение размера кисти для рисования на 1 px;

= – увеличение размера кисти для рисования на 1 px;

w – установка белого цвета кисти;

b – установка черного цвета кисти;

t – выбор дерева для отрисовки;

s – выбор камня для отрисовки;

n – наложение шума типа "соль, перец"

Tab – команды двигателям.

Space – команды агенту.

Esc – завершение программы.

В верхней части окна можно изменять параметры анализа изображения.

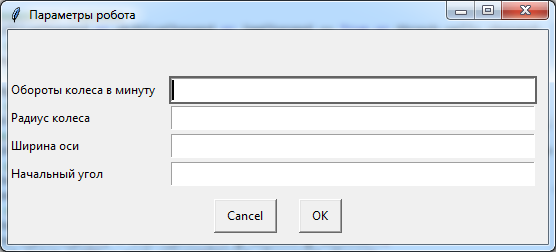
Grid: регулирует размер сетки в px (5 ~ 50).

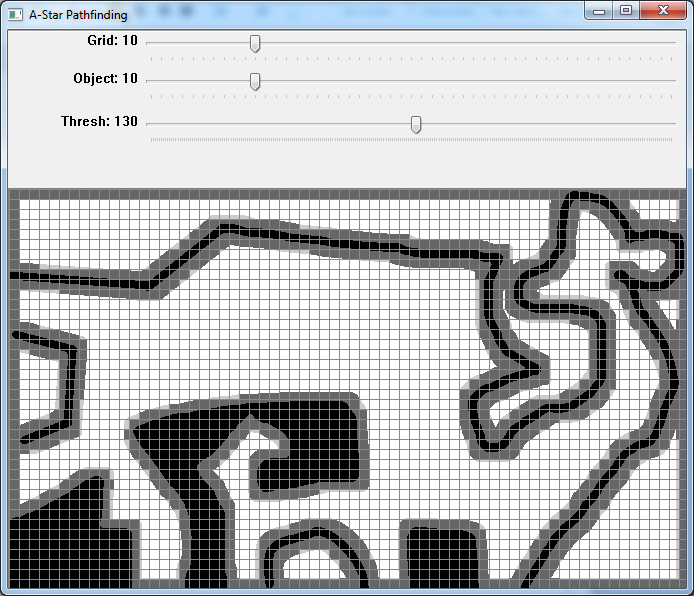
Object: размер объекта в px (0 ~ 50).

Thresh: порог для анализа препятствий ("средний" цвет ячейки, 0 ~ 255).

В качестве карты по умолчанию используется изображение "maze.png". Если оно отсутствует, создаётся изображение белого цвета размером 800 х 400 px.

Перед началом работы программа запрашивает у пользователя параметры робота, которые необходимо ввести.





Интерфейс программы