|  |  |  |
| --- | --- | --- |
|  | | |
| Федеральное государственное бюджетное  образовательное учреждение высшего образования  FPMI_ngtu_neti_rgb_polya«Новосибирский государственный технический университет» | | |
|  | | |
| Кафедра теоретической и прикладной информатики | | |
| Индивидуальное задание | | |
| по дисциплине « Методы построения и анализа алгоритмов» | | |
| **Построение пути для мобильного робота**  **в двумерном пространстве** | | |
|  | | |
|  | Вариант 15 |  |
| Группа ПМИ-21 | Комарова анастасия валерьевна |
|
|
|
|
| Преподаватель | Щукин георгий анатольевич |
|

Новосибирск,2023

**Оглавление**

[**Задача** 3](#_Toc152952917)

[**Пространство конфигураций** 3](#_Toc152952918)

[**Алгоритм** 4](#_Toc152952919)

[**Rapidly Exploring Random Trees (RRT)** 4](#_Toc152952920)

[**Optimal Rapidly Exploring Random Trees (RRT\*)** 5](#_Toc152952921)

[**Описание метода** 5](#_Toc152952922)

[**Метод решения** 5](#_Toc152952923)

[**Возможности и интерфейса разработанного GUI-приложения** 6](#_Toc152952924)

[**Описание интерфейса** 6](#_Toc152952925)

[**Возможности интерфейса** 7](#_Toc152952926)

[**Результаты работы приложения в разных сценариях** 7](#_Toc152952927)

[**Литература** 9](#_Toc152952928)

# **Задача**

Дана двумерная плоскость, на которой находятся объекты-препятствия. По плоскости движется робот - материальная точка с координатой (Cx, Cy); робот может перемещаться в любом направлении без ограничений. Заданы начальная и конечная координаты робота Init=(Initx, Inity) и Goal=(Goalx, Goaly).

Считаем, что рабочая область робота ограничена прямоугольной границей, за которую он не может выходить. Препятствия могут перекрываться друг с другом и выходить за границы рабочей области.

Положение робота определяется его координатой в двумерном пространстве (Cx, Cy) ∈ R2.

Путь - последовательность координат (точек) Pi=(Pix, Piy) ∈ R2, i=1..n, P1=Init, Pn=Goal. Координаты в пути соединяются отрезками, т.е. между каждой парой смежных координат робот движется по прямой. Длина пути - сумма длин всех его отрезков.

Требуется: найти кратчайший, по возможности, путь для робота (методом RRT\*) из начальной координаты в конечную, не приводящий к столкновению робота с препятствиями (в виде треугольников), или определить, что такого пути не существует.

# **Пространство конфигураций**

Конфигурация робота - набор параметров, однозначно определяющий положение робота в пространстве. Пространство конфигураций (configuration space, C) - множество всех возможных конфигураций робота. В нашей задаче конфигурация робота - его координата, соответственно имеем двумерное пространство конфигураций.

Пространство конфигураций C можно разбить на два подпространства: Сfree - допустимые конфигурации, т.е. не приводящие к столкновениям с препятствиями, и Cobs - недопустимые конфигурации, т.е. приводящие к столкновениям с препятствиями

В пространстве конфигураций конфигурация робота - **материальная точка**, соответственно задача поиска (кратчайшего) пути сводится к задаче поиска в С (кратчайшей) кривой/ломаной прямой, полностью лежащей в Cfree и соединяющей начальную и конечную конфигурации qInit и qGoal .



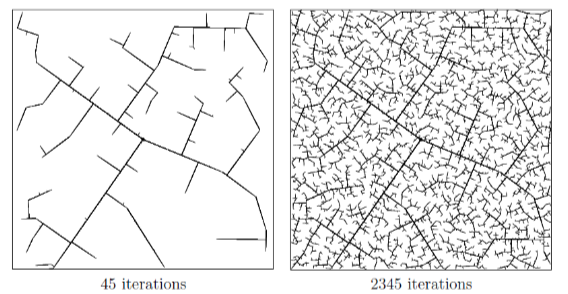
В условиях нашей задачи пространство конфигураций C совпадает с исходным пространством.

# **Алгоритм**

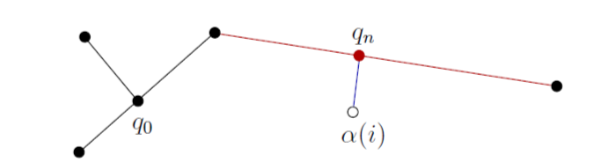
## **Rapidly Exploring Random Trees (RRT)**

Так как RRT\* является модификацией RRT следует сначала объяснить алгоритм работы метода **Rapidly Exploring Random Trees (RRT)**

В данном методе из начальной конфигурации qInit строится граф (дерево), покрывающий Cfree

****

Построение графа происходит следующим образом: производится некоторое заданное число итераций, на каждой итерации берется случайная конфигурация (сэмпл) из Cfree, которая становится вершиной графа и связывается ребром (путем по прямой) с другой ближайшей вершиной из графа; вес ребра - длина пути. В случае, если новая вершина близка не к вершине графа, а к ребру (т.е. ближайшая к ней конфигурация лежит на ребре), ребро делится на два путем добавления дополнительной вершины в этом месте, и новая вершина соединяется уже с этой дополнительной вершиной. Во всех случаях, если путь из новой вершины по новому ребру приводит к коллизии, на этом ребре ищется другая конфигурация, ближайшая к новой, не приводящая к коллизии (например, путем исследования ребра с некоторым шагом), и в граф добавляется уже эта обновленная вершина с соответствующим ребром.





После построения графа (дерева) в него добавляется qGoal и производится поиск кратчайшего пути из начальной конфигурации в конечную. Данный метод ориентирован на поиск ответа на один запрос с конкретными qInit и qGoal.

Метод RRT зависим от числа сэмплов и может не найти путь, даже если он существует. Большое число сэмплов повышает вероятность нахождения пути, но не гарантирует, что найденный путь будет оптимальным (кратчайшим).

## **Optimal Rapidly Exploring Random Trees (RRT\*)**

### **Описание метода**

Базовый алгоритм RRT не всегда находит оптимальный (кратчайший) пункт, даже при большом числе сэмплов. Модификация алгоритма под названием RRT\* решает эту проблему путем перестройки графа (дерева) при добавлении новых сэмплов

Перестройка графа/дерева происходит следующим образом: каждая вершина в графе имеет стоимость (Cost), обозначающую текущую длину кратчайшего пути в графе из стартовой вершины Qinit (корня дерева) до этой вершины. После добавления новой вершины рассматривается некоторое заданное число ближайших к ней вершин (или все вершины, лежащие в пределах заданного радиуса). Для каждой такой ближайшей вершины, если ее стоимость может быть уменьшена путем замены ее текущей родительской вершины на новую вершину, такая замена производится путем удаления и добавления соответствующих ребер, для сохранения структуры дерева.

Как и метод RRT, RRT\* зависит от числа сэмплов и может также не найти путь при малом числе сэмплов, но в случае RRT\* с увеличением числа сэмплов найденный путь стремится к  оптимальному.

### **Метод решения**

Описание вспомогательных функций:

RandomSample() - возвращает новую случайную конфигурацию из Cfree.

CollisionFree(X, Y) - проверяет, что путь по прямой из конфигурации X в Y не приводит к коллизиям, т.е. отрезок XY не пересекается с препятствиями из Cobs.

Nearest(G, X) - возвращает вершину (конфигурацию) из графа G, ближайшую к конфигурации X. Если оказывается, что X близка не к одной вершине, а к ребру (т.е. что ближайшая к X конфигурация лежит на ребре графа), то в этом месте ребро делится на две части и в граф добавляется новая вершина, которая затем возвращается.

Steer(X, Y) - возвращает новую вершину (конфигурацию) Z из Cfree, лежащую на отрезке XY и ближайшую к Y, такую что CollisionFree(X, Z).

Near(G, X, k) - возвращает k вершин из графа G, ближайших к конфигурации X. Альтернативная версия функции может вместо числа вершин k использовать расстояние dist: тогда возвращаются все вершины графа ближе чем dist к конфигурации X.

Length(X, Y) - возвращает длину отрезка XY.

Parent(G, V) - возвращает родителя вершины V в графе G.

Cost(G, V) - возвращает стоимость вершины V в графе G, Cost(G, V) = Cost(G, Parent(V)) + Length(Parent(V), V).

Параметры алгоритма RRT\_Star:

N - число сэмплов (чем больше тем точнее будет результат)

k - число ближайших вершин к рассмотрению (альтернативно - максимальное расстояние dist для поиска ближайших вершин)

Qinit, Qgoal - начальная и конечная конфигурации

ChooseParent(G, Qs, Qn, Qnear):

Qmin = Qn

Cmin = Cost(G, Qn) + Length(Qn, Qs)

Для каждой вершины Q из Qnear:

Если CollisionFree(Q, Qs) и

Cost(G, Q) + Length(Q, Qs) < Cmin:

Qmin = Q

Cmin = Cost(G, Q) + Length(Q, Qs)

Вернуть Qmin

Update(G, Qs, Qnear):

Для каждой вершины Q из Qnear:

Если CollisionFree(Qs, Q) и

Cost(G, Qs) + Length(Qs, Q) < Cost(G, Q):

Убрать ребро (Parent(G, Q), Q) из графа G

Добавить ребро (Qs, Q) в граф G

RRT\_Star(N, Qinit, Qgoal):

G - пустой граф

Добавить вершину Qinit в G

Для i = 1..N:

Qrand = RandomSample()

Qn = Nearest(G, Qrand)

Qs = Steer(Qn, Qrand)

Qnear = Near(G, Qs, k)

Добавить вершину Qs в граф G

Qp = ChooseParent(G, Qs, Qn, Qnear)

Добавить ребро (Qp, Qs) в граф G

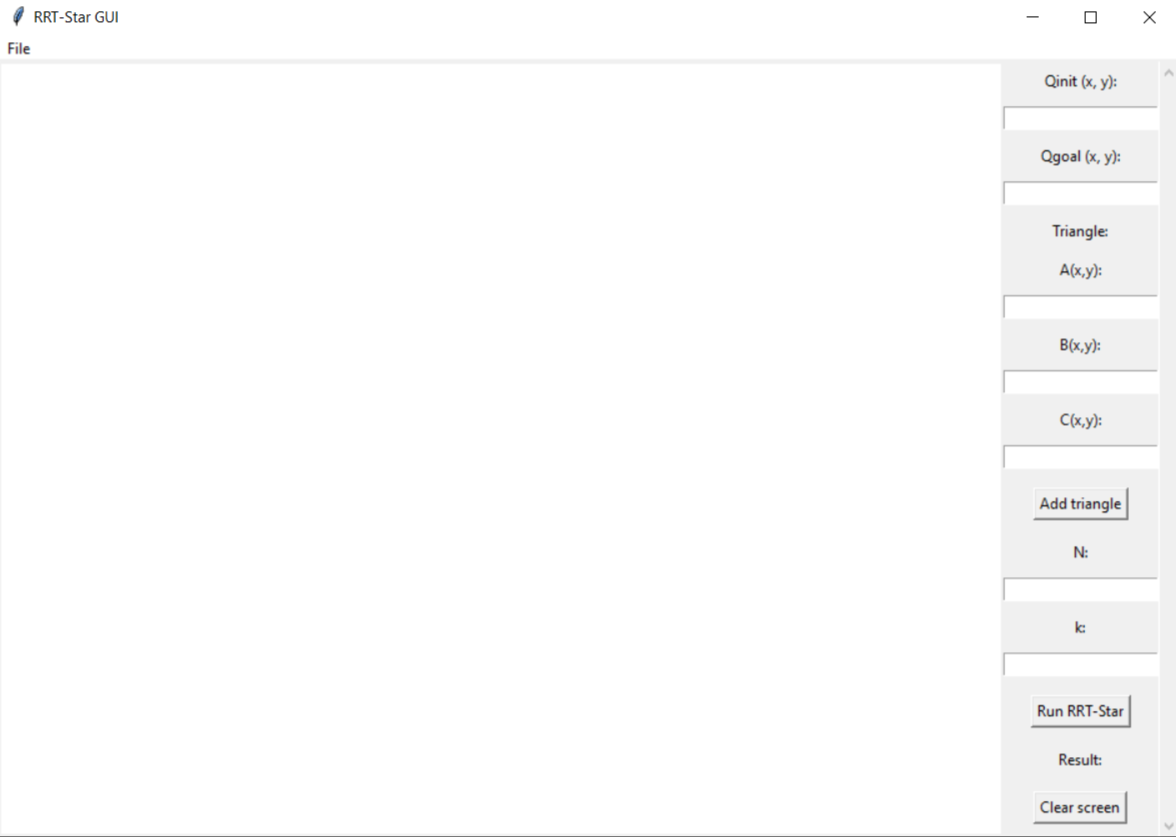
Update(G, Qs, Qnear)

Соединить Qgoal с графом G (добавить вершину и ребро)

Вернуть кратчайший путь в графе G из Qinit в Qgoal

# **Возможности и интерфейса разработанного GUI-приложения**

## **Описание интерфейса**



**Интерфейс приложения**

Интерфейс приложения содержит следующие части:

1. “Меню” приложения, находящееся в левом верхнем углу. При нажатии на “File” становятся доступными варианты “Save scene” и “Load scene” (соответственно сохранение и загрузка сцены)
2. Canvas полотно размером 800x600 пикселей, на котором будет наглядно представляться результаты программы.
3. Набор кнопок, задающих условия для работы алгоритма RRT\*, а именно:

a) Добавление начальной и конечной точек маршрута ( Qinit и Qgoal ) в формате x-координата, y-координата.

б) Добавление препятствий, посредством указания координат трёх вершин (формат x-координата, y-координата)

в)Указание количества итераций (N)

г)Указание параметра k, отвечающего за число ближайших вершин к рассмотрению

4) Кнопка запуска алгоритма

5) Окно вывода результата. Если путь найден, то в этом окне пишется его длина, иначе – вид ошибки или сообщение о том, что путь не найден

6) Кнопка очистки экрана

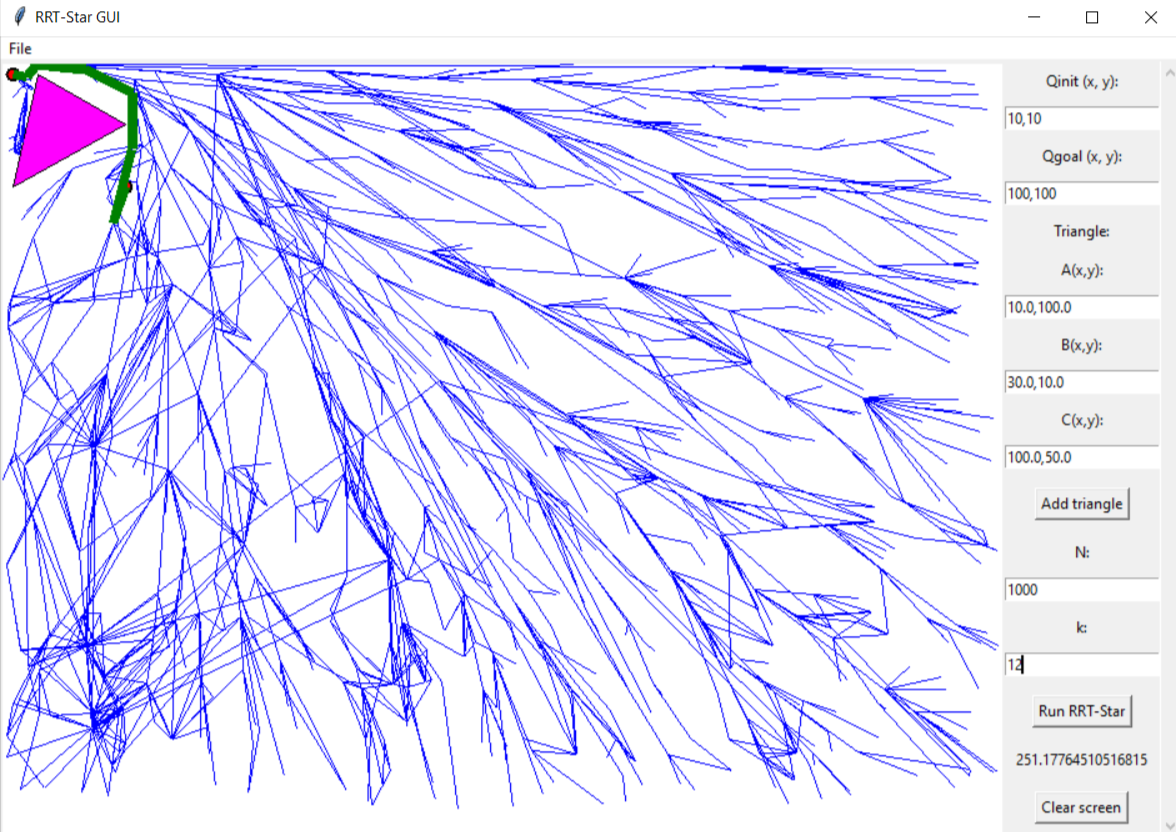
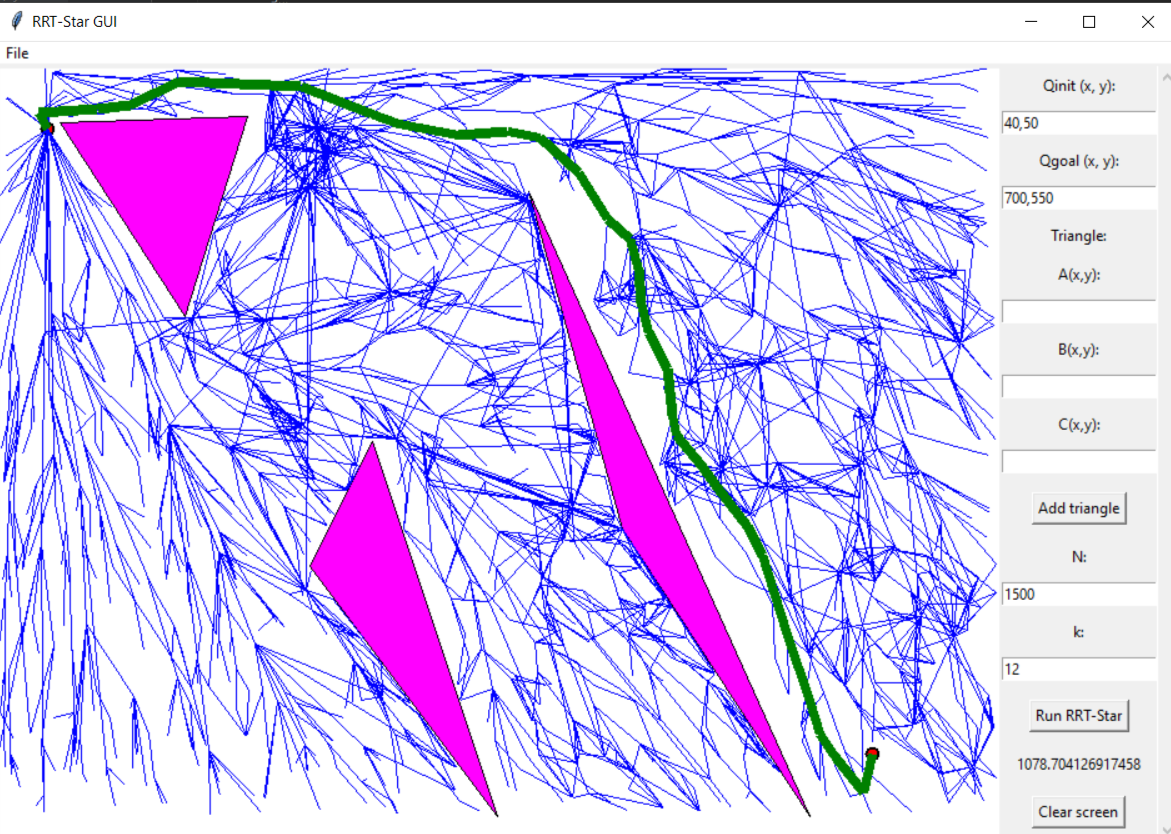
## **Возможности интерфейса**

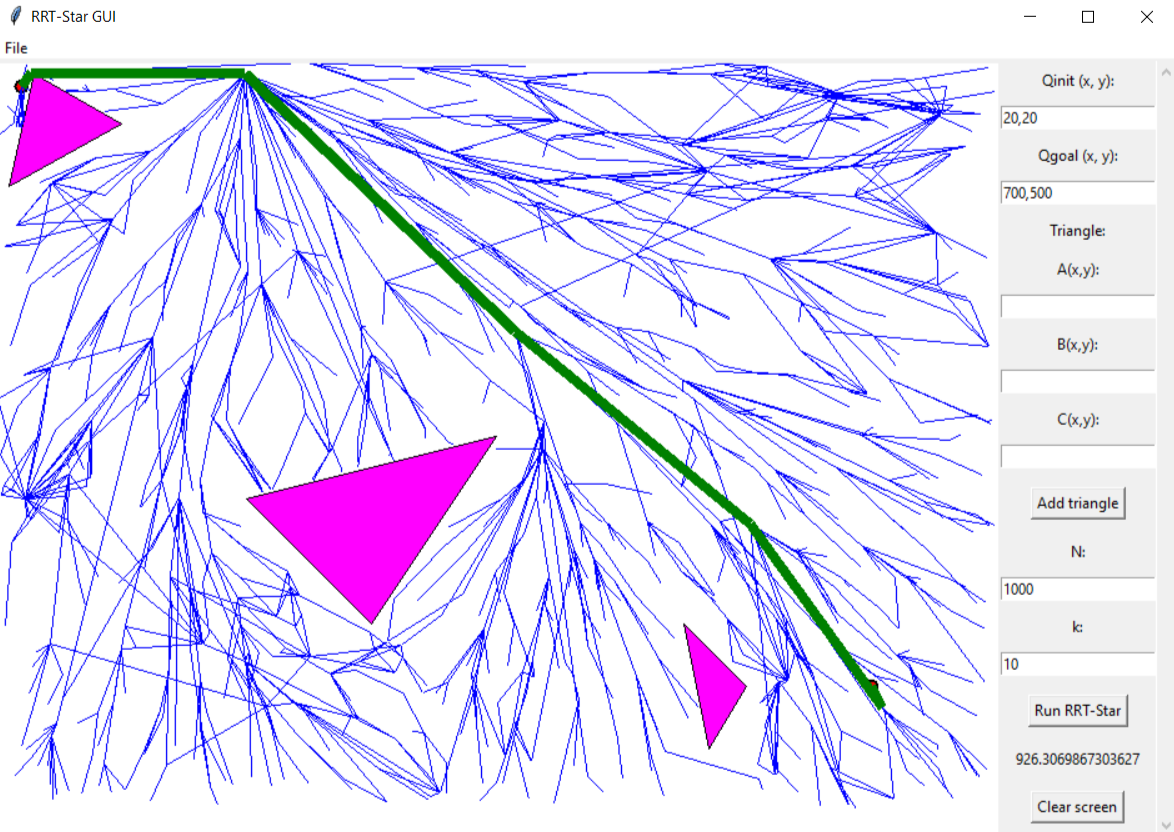
Основные возможности, которые поддерживает GUI-приложение:

* интерактивное задание и изменение сцены: препятствий и координат начальной, конечной точек пути
* отображение сцены: препятствий, начальной, конечной точек пути
* возможность сохранения сцены в файл и загрузки ее из файла, с выбором файлов для сохранения и загрузки
* отображение построенного пути или сообщения о том, что путь не удалось построить
* отображение вспомогательных структур, использовавшихся при построении пути: дерева
* задание параметров алгоритма в интерфейсе (N,k)

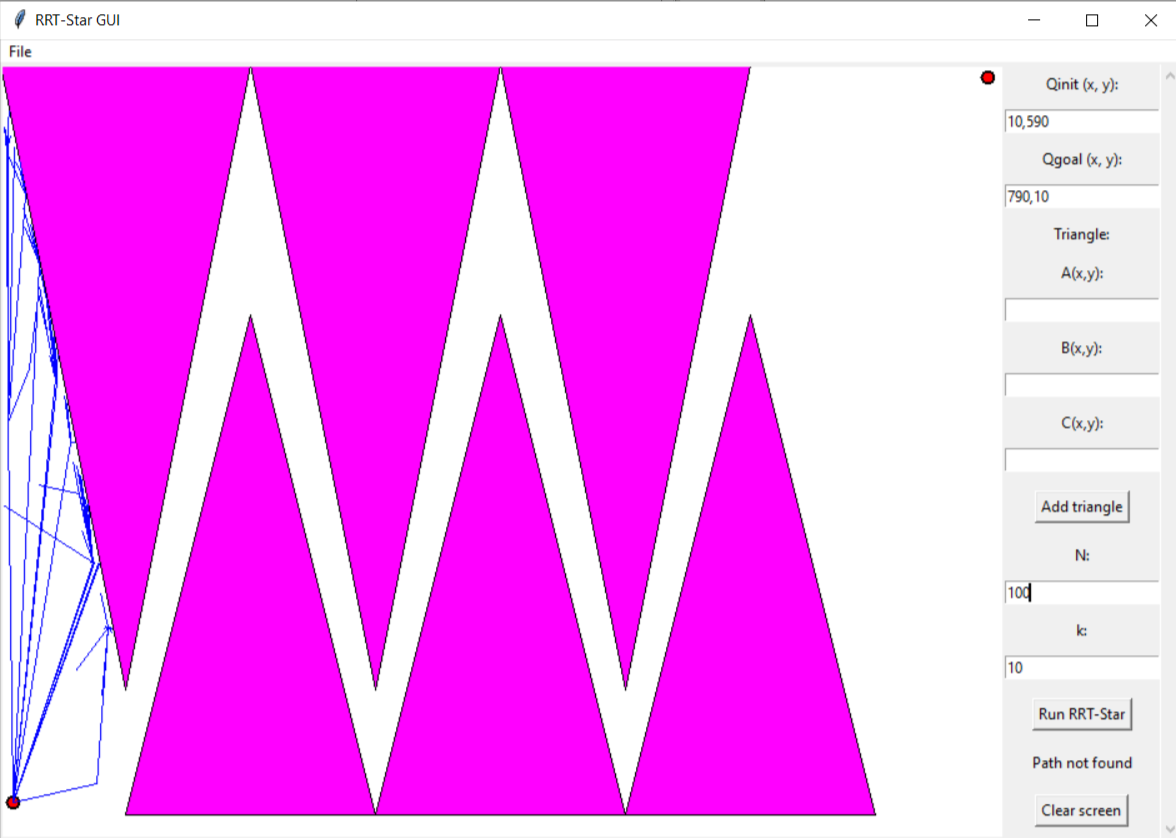
# **Результаты работы приложения в разных сценариях**

1)Сценарий №1: есть путь

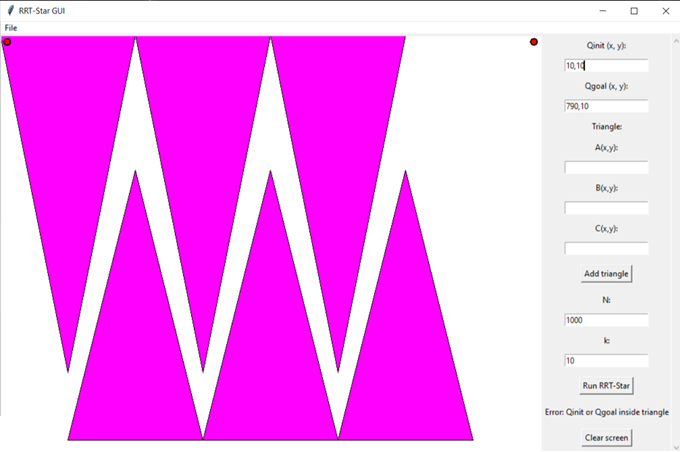




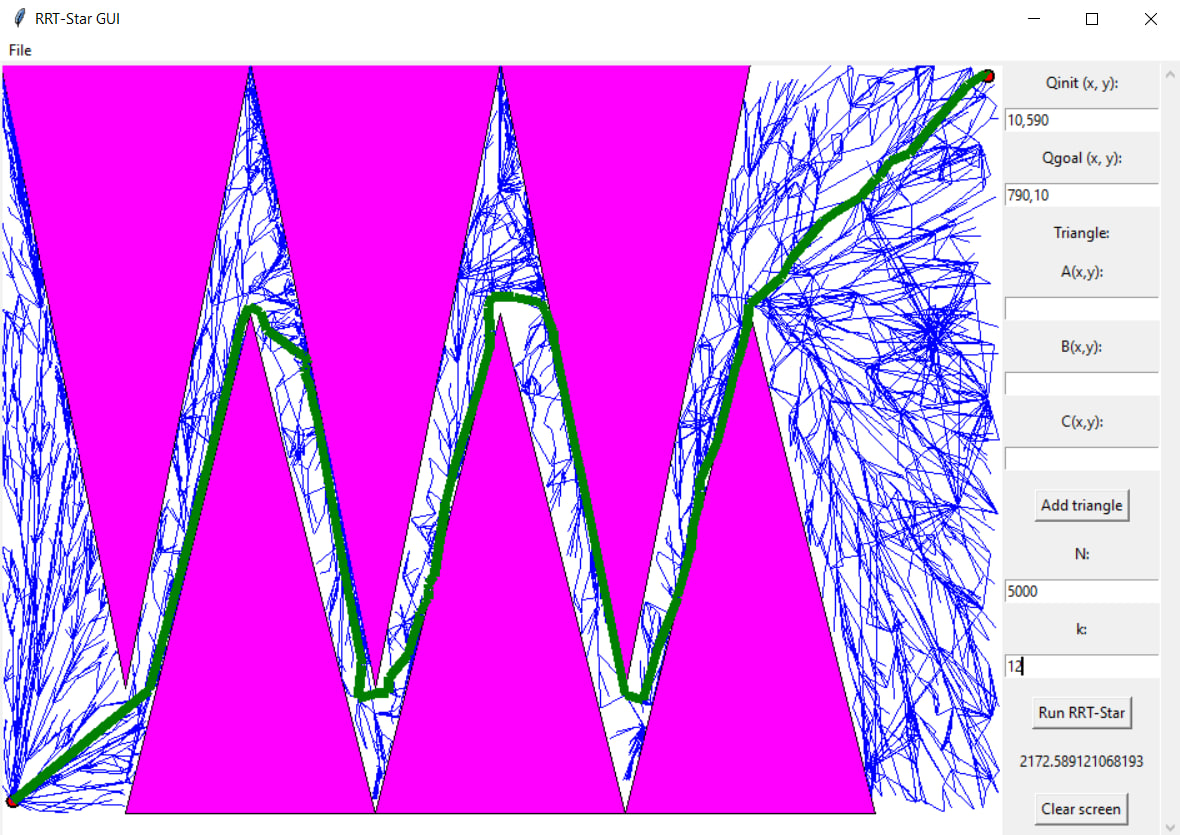
2)Сценарий №2: нет пути



3)Сценарий №3: начальная и конечная точка в препятствии



4)Сценарий №4: лабиринт (сложное препятствие, большое количество итераций)



# **Литература**

1. Steven M. LaValle, Planning algorithms, Cambridge University Press, 2006.
2. Howie Choset и др., Principles of robot motion: theory, algorithms and implementation, The MIT Press, 2005.
3. Jean-Claude Latombe, Robot motion planning, Springer US, 1991.
4. Sertac Karaman, Emilio Frazzoli, Sampling-based algorithms for optimal motion planning, 2011.