Github code repository

Internship Report

Project report for the internship at Healthcare technology innovation center (HTIC), IIT-Madras

Submitted by
Adwait P. Naik
(*Intern, Spine robotics team*)
September 2019 - February 2020

Supervised by
Mr. Manojkumar Lakshmanan
(*Project lead, Spine robotics team*)
(HTIC, IIT-Madras)
&
Mr. Shyam A
(*Robotics Engineer*, *Spine robotics team*)
(HTIC, IIT-Madras)

Github code repository

Contents

Note for readers - Before going through the report it is highly recommended to refer to the presentation that we prepared on motion planning to get the pretext.

[Github code repository]

RRT implementation in rviz

In this task, we implemented the RRT (Rapidly-exploring Random tree) algorithm in rviz in a 3D environment. [code] [video] [RRT]

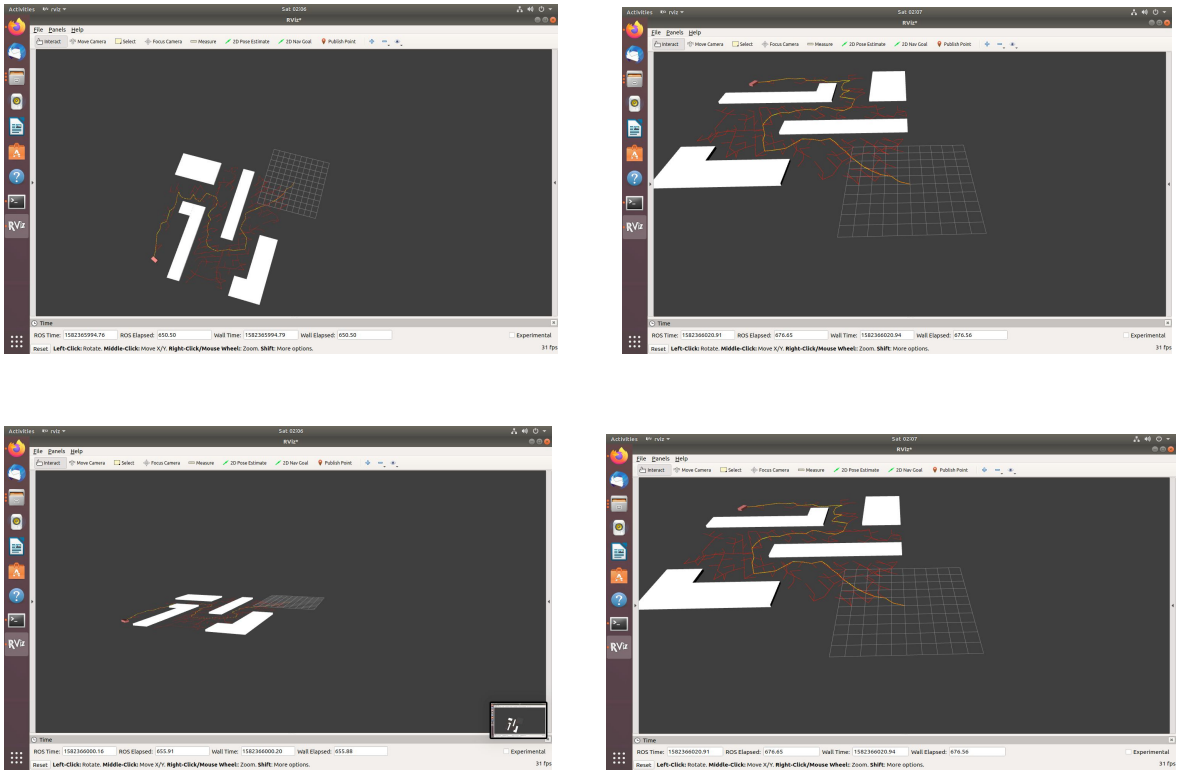

Fig. 1 RRT in Rviz

Our objective was to implement a sampling-based motion planning algorithm like RRT with kinodynamic constraints. We were successfully able to move the robot (*cube-shaped red*) from the **start** node to the **goal** node which is set by the user. We observed that the algorithm took 748 frames* to retrieve the shortest and the most optimal path. It was also observed that the collision avoidance worked perfectly fine as the robot moved without colliding any obstacle or surface.

Github code repository

**Please refer to the code for actual understanding*

RRT-2D implementation using Pygame

In this task, we implemented the RRT (Rapidly-exploring Random tree) algorithm using pygame in a 2D environment. [paper][code][RRT]

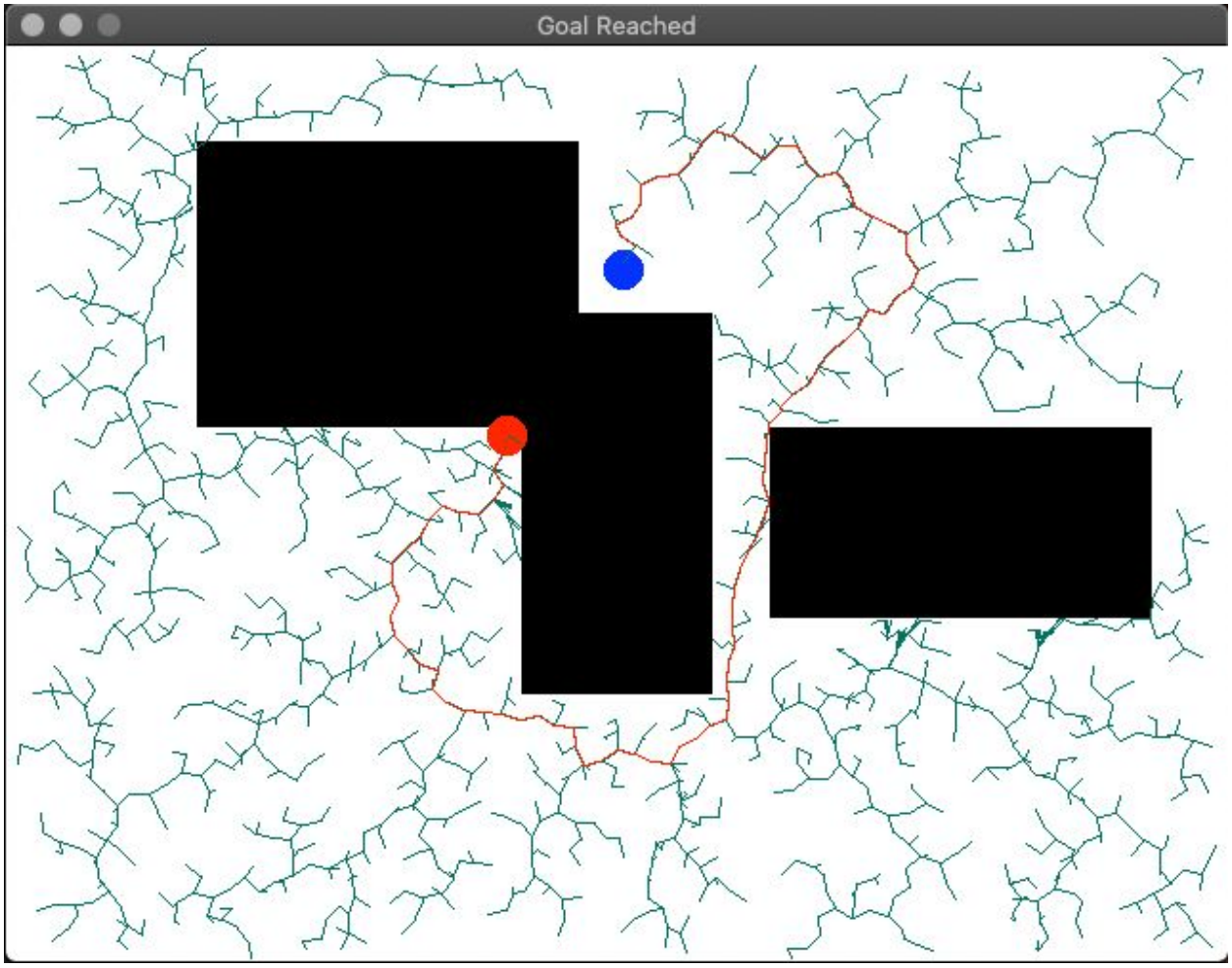

Fig. 2 RRT with obstacles in pygame

In the figure above we have implemented the RRT algorithm using the pygame library to trace the shortest collision-free path in a 2D environment. The red dot is the start node and the blue dot is the goal node. RRT is a single query, sampling-based motion planning algorithm which is probabilistically complete.

Github code repository

The only drawback of this algorithm is the quality of the path, time complexity and memory consumption.

RRT_star -2D implementation using Pygame

In this task, we implemented the RRT -star (Rapidly-exploring Random tree) algorithm using pygame in a 2D environment. [paper][code][RRT-star]

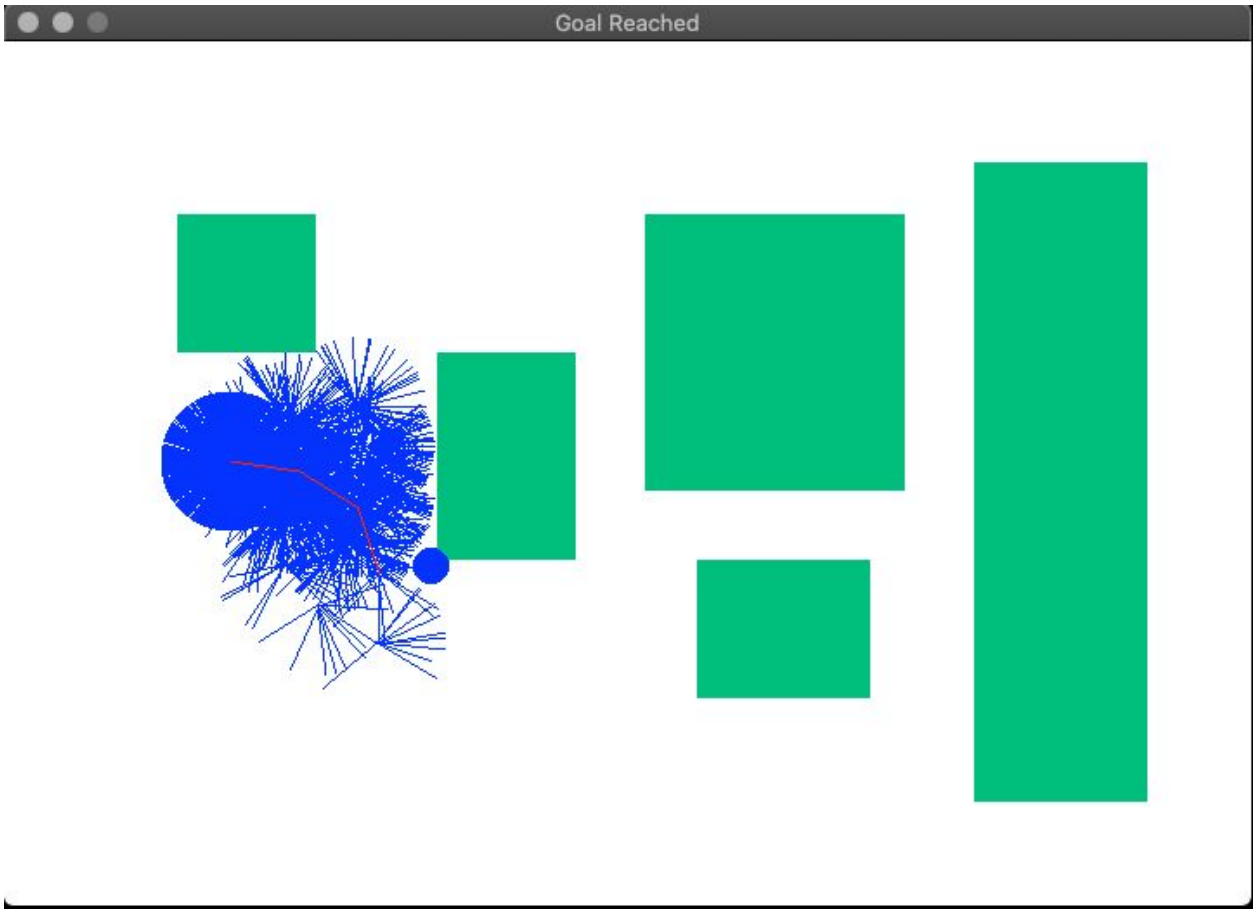

Fig. 3 RRT-star implementation using pygame

In the figure above we have implemented RRT-star which is an optimized version of the RRT algorithm. It traces the shortest collision-free path in lesser time as compared to the RRT, the reason being the change in sampling strategy to find the

Github code repository

nearest node which produces a less jaggy path. Thus reducing the time complexity by a great margin.

RRT_star -2D implementation using Pygame with manhattan distance

In this task, we implemented the RRT -star (Rapidly-exploring Random tree) algorithm using pygame in a 2D environment. In this algorithm, we changed the distance metric to Manhattan distance which is based on the **Taxicab geometry principle.** [code]

Why Manhattan distance?
Generally, Euclidean distance is the metric used in motion planning algorithms to calculate the distance between two points. We observed that when the nodes are sampled at a distance larger than the usual the algorithm yielded a jaggy path that deviated by a large margin. In order to eliminate it, we used a different distance metric based on taxicab geometry.

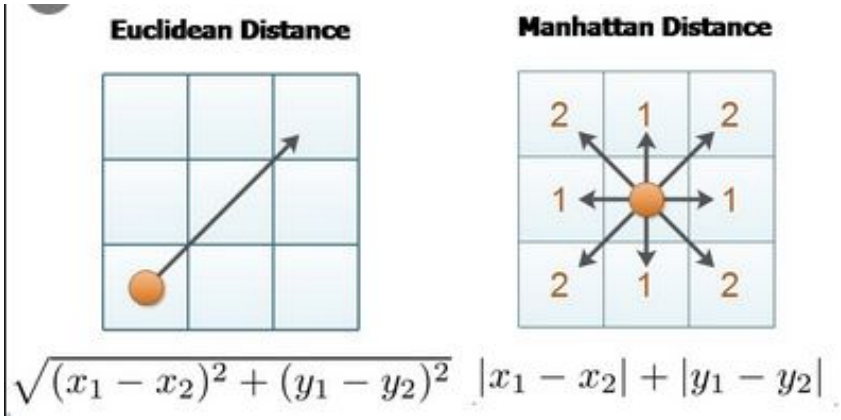

Fig. 4 Euclidean vs Manhattan

Unlike the euclidean geometry, the manhattan distance calculates the distance between the nodes at the right angles. In other words, the algorithm is able to reach the nearest node in minimum time.

Github code repository

RRT connect -2D implementation using Pygame

In this task, we implemented the RRT connect (Rapidly-exploring Random tree) algorithm using pygame in a 2D environment. In this algorithm, we changed the distance metric to Manhattan distance which is based on the **Taxicab geometry principle.** [code] [paper]

What is RRT connect?

RRT connect is the variant of the RRT algorithm that grows the tree from the source node and from the goal node in a particular direction until both the trees meet.

Fig. 5 RRT connect implementation using pygame

Github code repository

In the figure shown above, we observe that the tree is made to spread from the start node (*green*) and the goal node (*blue*) until it meets at a point. The idea behind using this strategy is to reduce the time to trace the path.

RRT B-spline 2D implementation using Pygame

In this task, we implemented the RRT B-spline (Rapidly-exploring Random tree) algorithm using pygame in a 2D environment. [code]

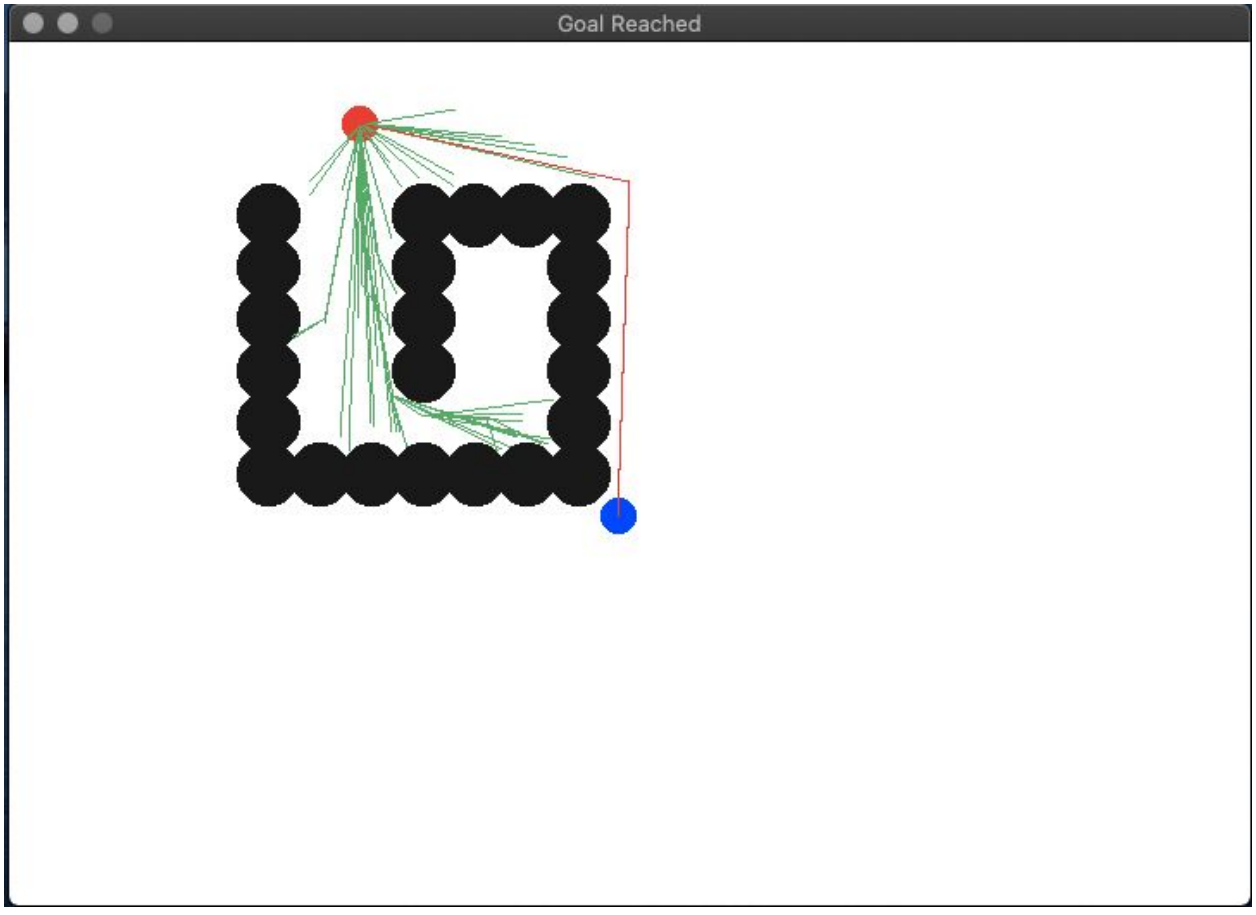

Fig. 6 RRT B-spline with pruning

Github code repository

The basic idea behind using the pruning concept is a uniform sampling of the nodes so that the algorithm is able to find the narrower paths. It successfully reduces the number of nodes in the path thereby removing the nodes which are not part of the curve. For mathematics behind the technique please visit the link.

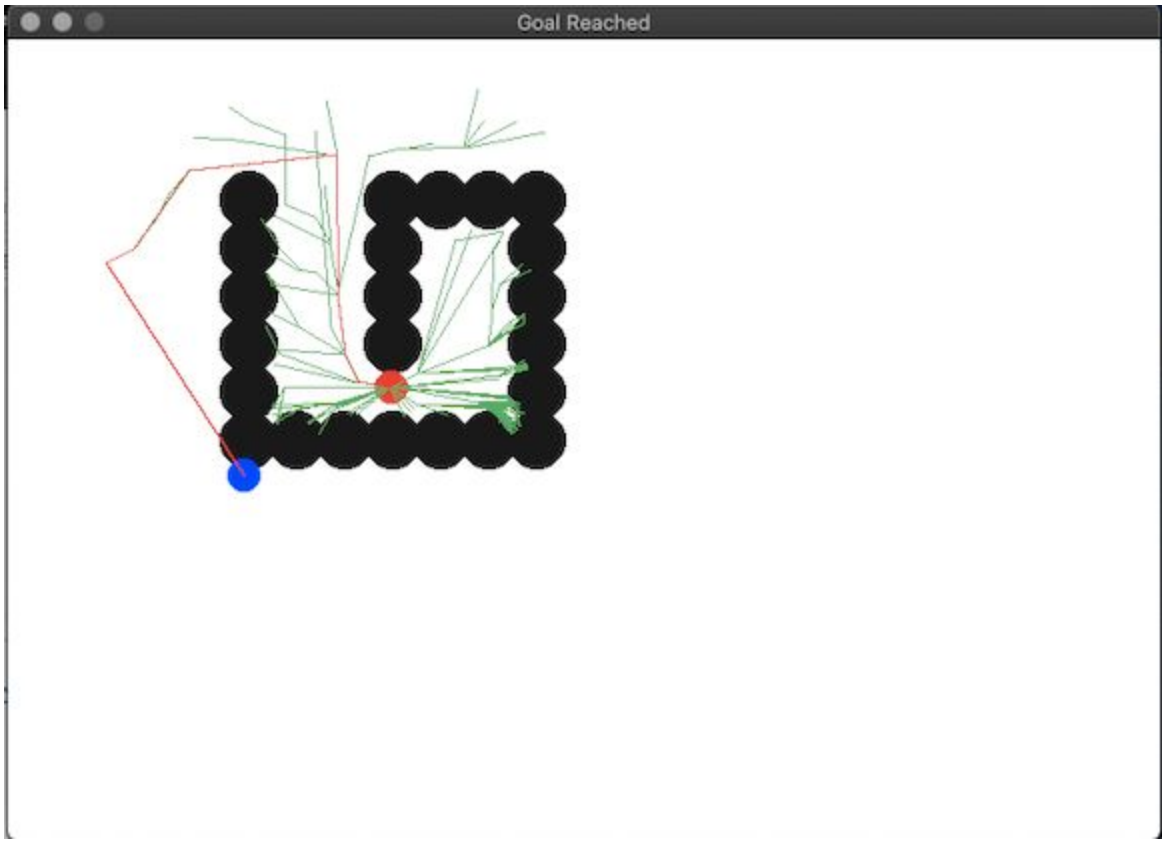

Fig. 7 RRT B-spline with pruning

In the figure, we attempted replacing the distance metric with the euclidean distance which resulted in the deviation of the path as compared to the path generated in *Figure. 6.*

Github code repository

A-star algorithm

In this task, we implemented the A-star algorithm which is based on the heuristic approach. This algorithm is similar to the famous Dijkstra's algorithm that is used extensively in networking to locate and eliminate the faulty node. [code][A-star][paper]
It is based on the concept of graph traversal in which it stores all the nodes in its memory that increase the space complexity making it inefficient in many ways.

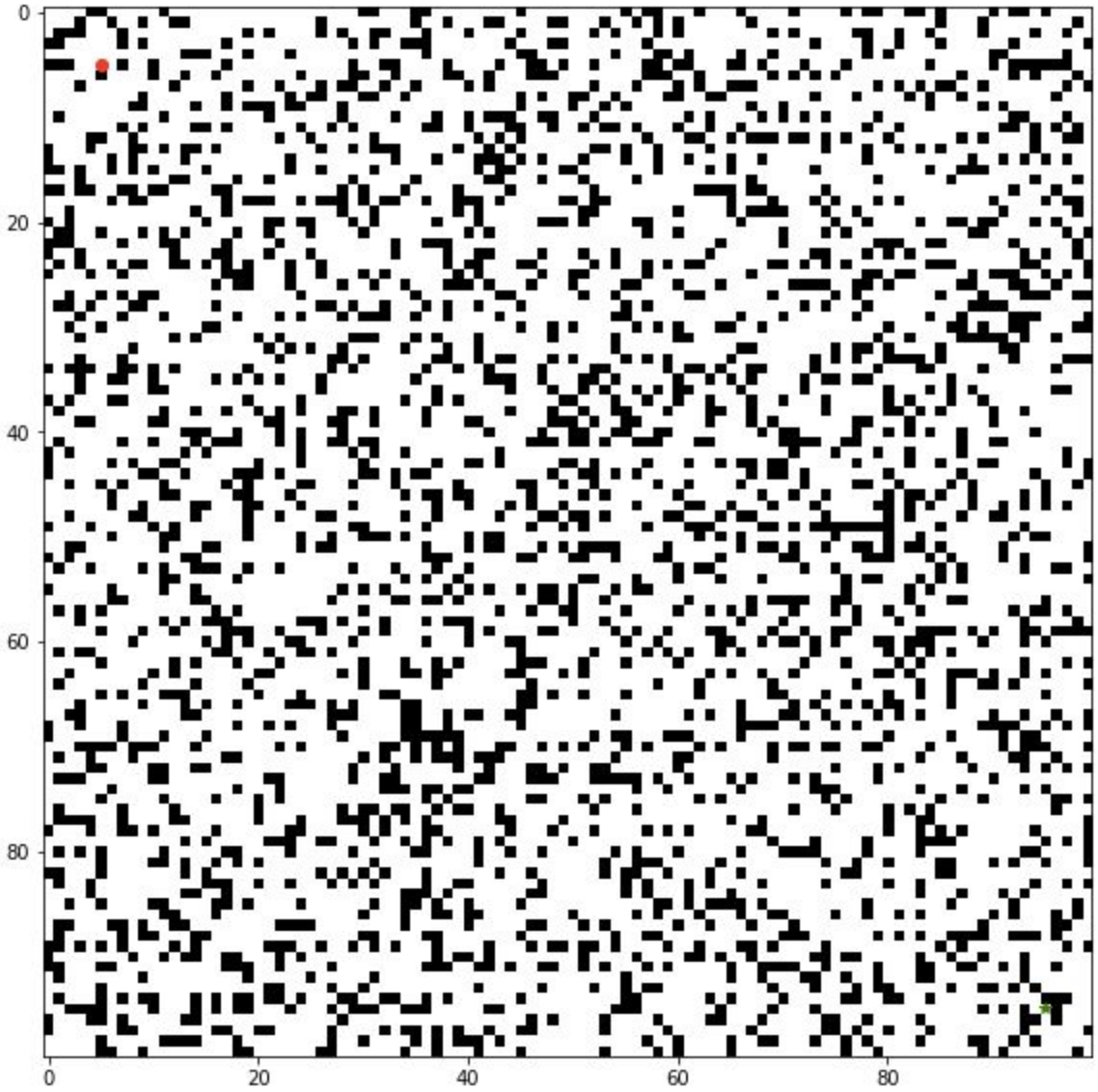

Fig. 7 A star search algorithm

[Github code repository]

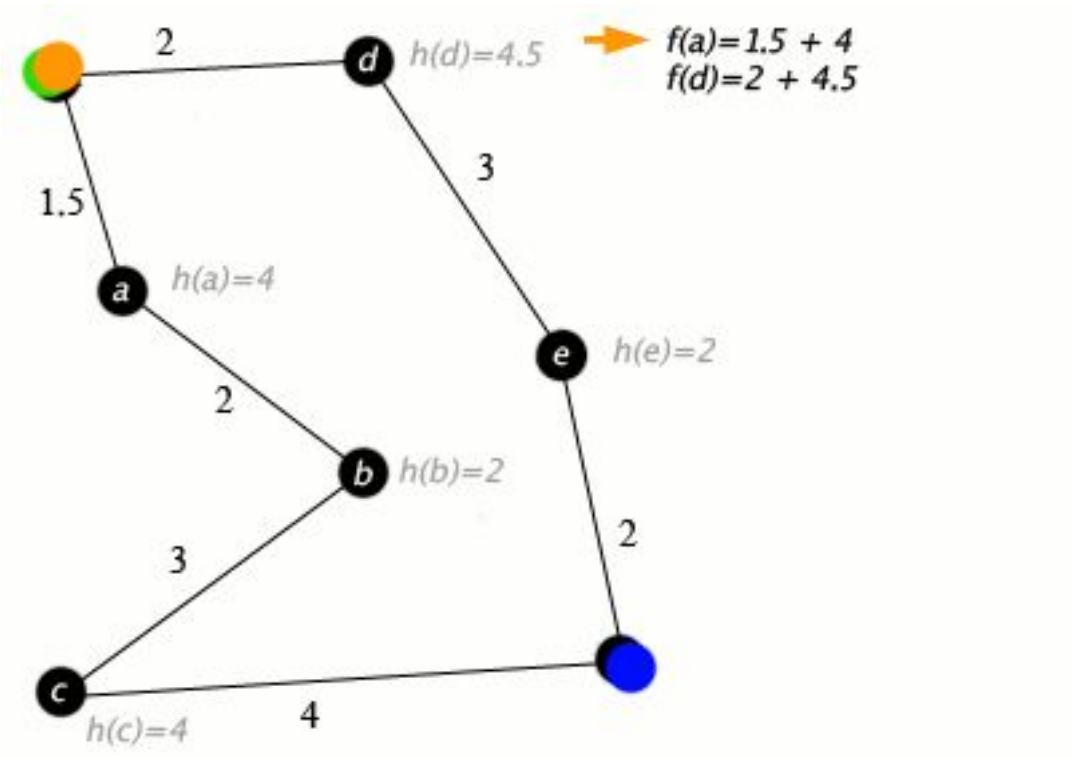

Fig 8. A-star in action (image courtesy - Wikipedia)

Applications

Commonly used in Gaming and path-finding.

Github code repository