CSY3025

ARTIFICIAL INTELLIGENCE TECHNIQUES

ASSIGNMENT I

Exploring the Potential Use of Artificial Intelligence Techniques in Intelligent Transport

Diwas Lamsal
18406547

2019

Contents

ACRONYMS AND ABBREVIATIONS

IoT	Internet of Things
BFS	Breadth-First Search
DFS	Depth-First Search
IDA*	Iterative Deepening A*
V2V	Vehicle to Vehicle
V2X	Vehicle to Everything
VIP	Very Important Person
RBS	Rule-Based System
WHO	World Health Organization
FSM	Finite-State-Machine

1. INTRODUCTION

Artificial Intelligence has grown rapidly over the years. The thought of being able to develop machines that can augment human intelligence and aid humans across numerous aspects is fascinating. While the thoughts of machines being able to think smartly was introduced very long back, the use of term "Artificial Intelligence" was first invented in 1956 by John McCarthy (Smith, et al., 2006). While we head towards the year 2020, the growth since back then in AI has been phenomenal. AI is now used almost everywhere from oil industries such as Abu Dhabi National Oil Company to healthcare; in farming, in manufacturing and many other industries (Kelly, 2019).

This paper discusses the potential use of some Artificial Intelligence techniques in Intelligent Transport. The background section will contain background information and some relevant literature reviews about the techniques that will be used in this project. The background information about each solution will be included in their respective sections. As part of the solution, software will be produced and tested in the solution section of the report. It will also include analytical information such as tests about some techniques. The conclusion section will contain the assessed strengths and weaknesses of each technique utilized or researched as part of this project.

2. BACKGROUND

With the growth in AI, the field of Intelligent Transport has progressed a lot. The evolution of the Internet of Things (IoT), 5G, Computer Vision, Internet and a lot of technological aspects has led to autonomous vehicles requiring little to no human intervention to operate. The thought of Intelligent Transport was introduced in the 1939-40 New York World's Fair where the General Motors provided the visitors with Futurama – a ride to show the world of tomorrow. The ride provided visitors with a view of what the transportation would look like in the 1960s. The ride displayed sets of automated vehicles. It was a display of what the future held, and it set the stage for progress in intelligent transport (Auer, Feese and Lockwood, 2016). Currently, many giants such as Google, Tesla and Mercedes are investing a lot into manufacturing and testing autonomous vehicles.

Figure 1 – Intelligent Transport (Pasqua, 2019)

Although it is out of scope of this paper to discuss about the full history of intelligent transport and to discuss the more advanced topics such as Computer Vision, the possible use cases of techniques such as clustering (which is apparently also used in Computer Vision), rule-based systems and search algorithms will be discussed. The main focus will be on how these techniques can be used as part of an Intelligent Transport system.

2.1 Literature Review

2.1.1 Clustering in Intelligent Transport

Clustering algorithms are used for grouping data that have certain similarities. This is done without providing the system with any knowledge about the data. Clustering techniques are used in many areas such as Computer Vision and Bioinformatics (Kogan, 2007). Clusters of data are generated which can be analyzed to extract patterns and characteristics of the data. There are different kinds of clustering techniques such as the K-Means Algorithm, K-Center Algorithm and Model-Based Clustering. There are also five different categories of clustering algorithms: division-based, hierarchical-based, density-based, grid-based and model-based. It is necessary to choose relevant algorithms. Clustering techniques will allow extracting scientific and reasonable trends from transportation data and interpreting them intelligently (Qiong, Jie and Jinfang, 2011).

Clustering techniques could also be used as potential solutions for data bottlenecks in ad-hoc wireless networks by using the public vehicles as cluster heads. The paper "*Clustering Intelligent Transportation Sensors using Public Transportation"* discusses a mathematical approach to cluster intelligent transportation sensors (Geetla, et al., 2016).

Correspondingly, clustering techniques can also be used to analyze intelligent transport decisions and improve congestion. Intelligent Transport System is the use of scientific information, electronic control and other technology to produce an efficient real-time transportation management system. Data mining (clustering) algorithms can be applied to such Intelligent Transport Systems. This kind of analysis could allow traffic jams and flow predictions and produce effective road and traffic distribution models (Ying, 2017). There could also be other use cases of clustering techniques such as analysis of road accident data to prevent accidents, i.e. predict likely places of accidents or other problems that could arise in a transportation system.

2.1.2 Rule Based Systems in Intelligent Transport

Rule-Based systems are arguably the simplest form of AI. They are mostly implemented in a narrow field in an automated or intelligent system. An RBS uses predefined rules and knowledge for solving a problem. There are basically two approaches or ways how Rule-Based systems work. Forward chaining is where the initial facts are started to head towards the goal or draw new conclusions and backward chaining tends to prove some facts or goal by looking for rules that head towards the provided goal (Grosan and Abraham, 2011).

While they could be a simple form of AI, Rule-Based systems have been used across many intelligent transport systems. Fuzzy rule-based systems have been used in Traffic Signal Control Systems. This can be used to control traffic flow and the flow of emergency vehicles. Traffic signals are important parts of having smooth transportation and is a prerequisite for Intelligent Transport and a fuzzy Rule-Based system can allow controlling these traffic signals without being prone to faulty human decision making at times. Sometimes situations could arise where traffic needs to be diverted for example when there is a VIP movement or a roadblock, in such situations, this kind of rule-based system would be very efficient (Abbas, Sheraz and Noor, 2009).

There are also examples of usage of Rule-Based Systems to transport Hazmat (hazardous) goods. The decision rules could easily represent human language interpretation with if-then scenarios. A Rule-Based inference engine can be used to asses different risks involved while transporting hazardous materials and this can be used to prevent accidents or reduce cost involved (Asadi and Ghatee, 2015).

2.1.3 Search Algorithms

Search algorithms have been discussed during lectures such as BFS, DFS and A* algorithms and their practical use was shown in a pathfinding JavaScript based program. There are also other algorithms that were mentioned but not discussed in depth such as the Dijkstra's algorithm. These search algorithms are mostly used for pathfinding or routing in intelligent transport systems. Some papers have also discussed the use of more complex genetic algorithms to find the distance or even the shortest driving time for vehicles to reach a destination which also refers to Darwin's theory of evolution (Lin, et al., 2008; Kumar et al., 2009). These solutions also take the real-world scenarios such as congestion into account.

2.1.4 Finite-State-Machine (FSM)

Finite state machines are used to describe mathematical models of a system with a limited number of states. These are used to simplify complex problems (Rouse, 2019). There are examples of different papers and practical usages of FSMs to aid transportation and intelligent transportation in particular. Zhang, Li, Girard and Kolmanovsky present in their paper about the use of FSM to devise an automated driving controller. They contend that this work can be used as a baseline for autonomous driving algorithms. They claim that using Finite-State-Machines would make the structure clearer, easy to calibrate and optimize and is thus more reliable. They even simulated their work on a traffic simulation game to assess the performance and concluded that the model based on FSM was very concise and easy to implement (Zhang, Li, Girard and Kolmanovsky, 2017).

There are several other papers discussing different kinds of use cases of FSMs to aid intelligent transport. Another paper discusses the use of FSM in highway surveillance. It is essential to monitor the traffic and roads continually. This paper presents the vehicle detection problem as different states of an FSM with "*Vehicle Moving, Vehicle Departure, Vehicle Arrival*" as the different states. (HyunKim, Kwon, Chen and Choi, 2013).

3. BACKGROUND & SOLUTION OF AI TECHNIQUES

The solutions will tend to address the strengths of mentioned techniques while showing their potential use in intelligent transport. For example: as mentioned that search algorithms are good for pathfinding, the solution will tend to address how these can be used by vehicles or an intelligent transport system to find routes. The solutions will be minimal but highlight the use of AI techniques. There will be different parts and individual small piece of software for each technique.

3.1 Rule-Based-Systems

Rule-Based systems, as the name suggests can allow an Intelligent Transport system to be based on rules. Sets of rules, formulated by experts in the field, can be defined for various purposes. This solution will aim to predict whether an accident is highly likely to occur according to predefined rules. It will not be an expert implementation of RBS but a basic example of how they can be applied. These rules will be implemented in the SWI prolog.

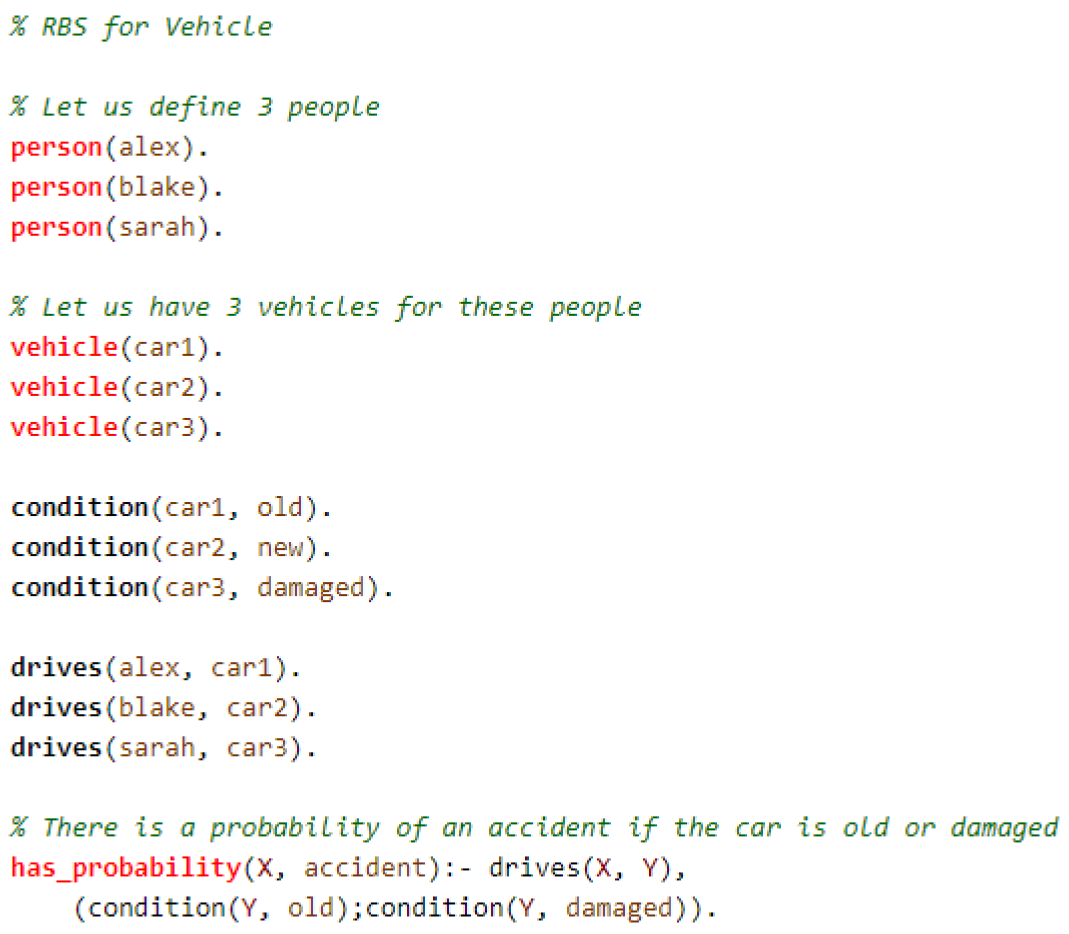
We can examine the main causes of accidents from historical data or the most common or obvious reasons. While there is no sure shot way of predicting an exact location for an accident, we can describe rules for likely conditions for an accident to occur (simple example: drunk driving). There could be different factors involved in road accidents such as weather, flaw in vehicle or human error. Some of these conditions are self-assumed.

Human Condition	• Distracted • Drunk • Speeding • Reckless
Vehicle Condition	• Old • Damaged
Weather / Time Condition	• Rainy • Snowy • Nighttime

We will assume that any of the above conditions will make an accident likely to occur.

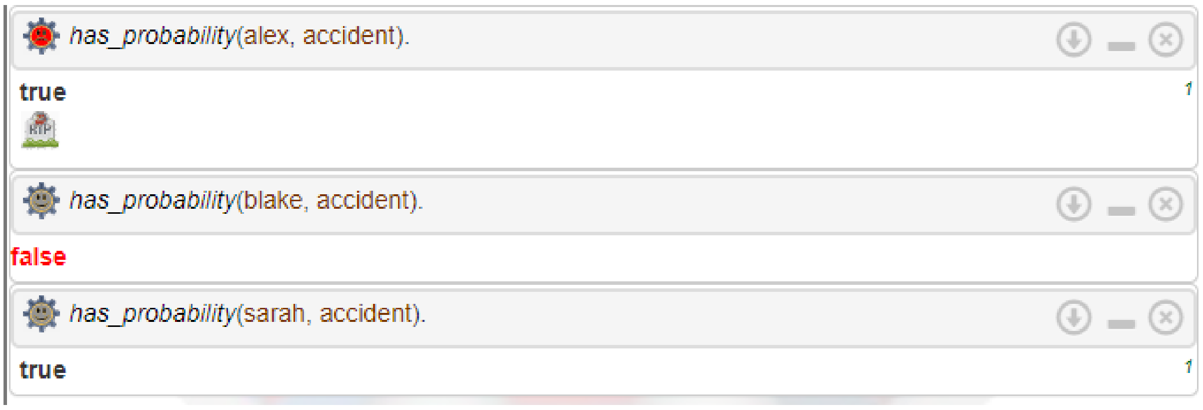
In the first SWI program, we will consider a vehicle's condition to predict whether there is a probability of an accident. We can define three people with three vehicles each. Each vehicle will be either old, new or damaged. We can define a rule that

$$IF\ vehicle = damaged\ or\ old\ THEN\ accidentprobability = TRUE$$



```
% RBS for Vehicle

% Let us define 3 people
person(alex).
person(blake).
person(sarah).

% Let us have 3 vehicles for these people
vehicle(car1).
vehicle(car2).
vehicle(car3).

condition(car1, old).
condition(car2, new).
condition(car3, damaged).

drives(alex, car1).
drives(blake, car2).
drives(sarah, car3).

% There is a probability of an accident if the car is old or damaged
has_probability(X, accident):- drives(X, Y),
    (condition(Y, old);condition(Y, damaged)).
```

Figure 2 – SWI Vehicle RBS



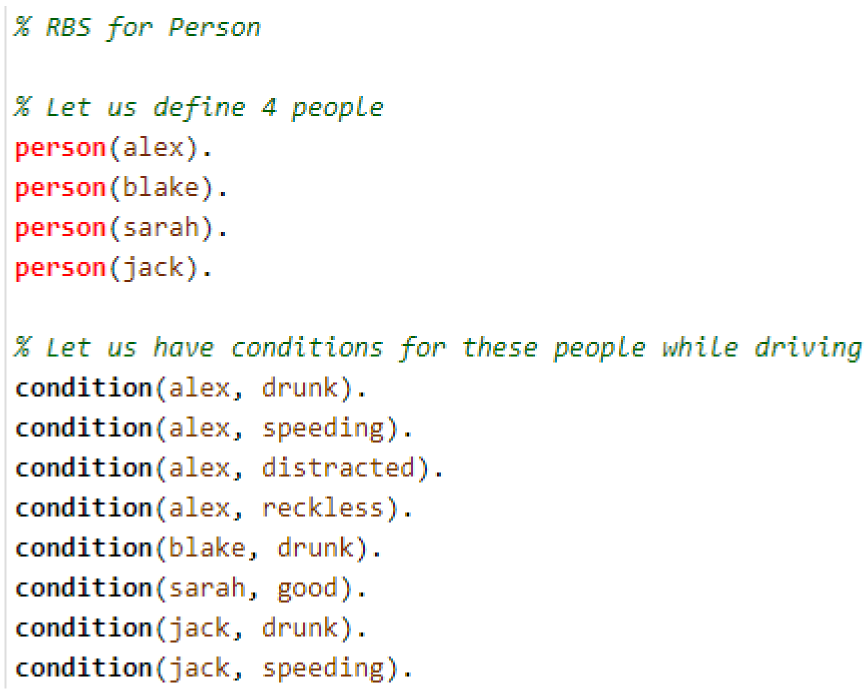
```
has_probability(alex, accident).
true
has_probability(blake, accident).
false
has_probability(sarah, accident).
true
```

Figure 3 – Results SWI Vehicle RBS

Table below provides test results and comments.

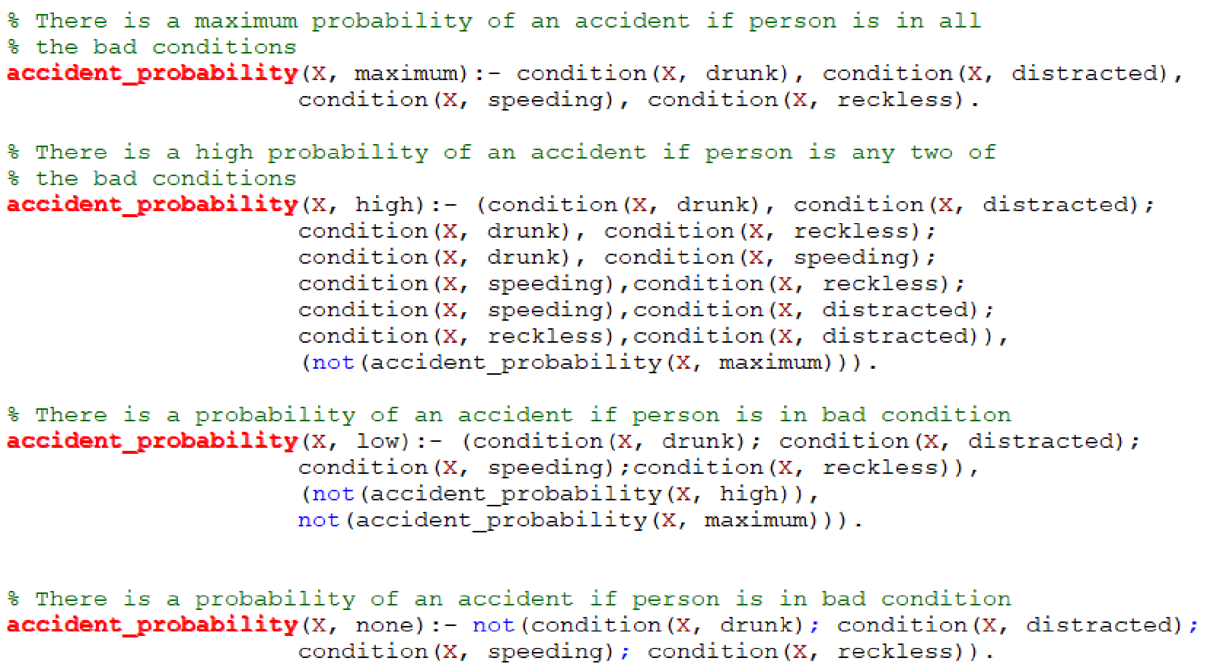
Test	Outcome	Comments
Alex	True	Alex is using an old vehicle which is not in perfect condition and has a probability of accident.
Blake	False	Blake is using a brand-new vehicle which is in perfect condition which will not cause an accident.
Sarah	True	Sarah is using a damaged vehicle which is not in perfect condition and has a probability of accident.

We can have a similar implementation with person's conditions. We can consider four different people with different conditions while driving. Our rules will determine whether there is a maximum, high, low or no probability of accident according to the condition of the person driving. If all the bad conditions exist at once, there is a maximum chance of accident, if two of these conditions exist at once, there is a high probability of an accident, if there is only one there is a low probability and none when there are no bad conditions. Alex has all the four conditions, Blake has a drunk condition, Sarah is in a good condition for driving whereas Jack is speeding and drunk. According to our rules, the system should be able to show the probability of anyone getting in an accident according to their condition.



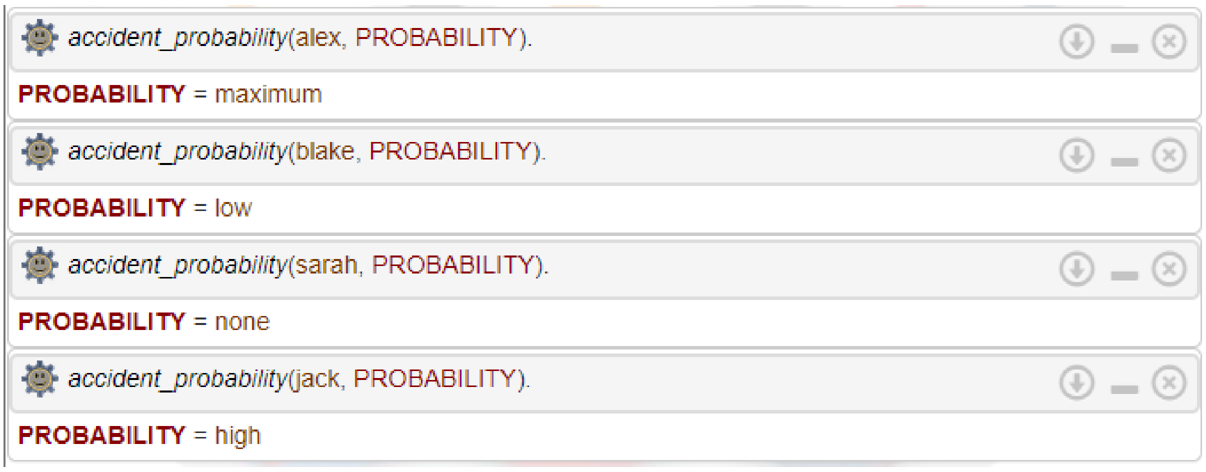
```prolog
% RBS for Person

% Let us define 4 people
person(alex).
person(blake).
person(sarah).
person(jack).

% Let us have conditions for these people while driving
condition(alex, drunk).
condition(alex, speeding).
condition(alex, distracted).
condition(alex, reckless).
condition(blake, drunk).
condition(sarah, good).
condition(jack, drunk).
condition(jack, speeding).
```

Figure 4 – SWI Person RBS Declaration



```prolog
% There is a maximum probability of an accident if person is in all
% the bad conditions
accident_probability(X, maximum):- condition(X, drunk), condition(X, distracted),
                    condition(X, speeding), condition(X, reckless).

% There is a high probability of an accident if person is any two of
% the bad conditions
accident_probability(X, high):- (condition(X, drunk), condition(X, distracted);
                    condition(X, drunk), condition(X, reckless);
                    condition(X, drunk), condition(X, speeding);
                    condition(X, speeding),condition(X, reckless);
                    condition(X, speeding),condition(X, distracted);
                    condition(X, reckless),condition(X, distracted)),
                    (not(accident_probability(X, maximum))).

% There is a probability of an accident if person is in bad condition
accident_probability(X, low):- (condition(X, drunk); condition(X, distracted);
                    condition(X, speeding);condition(X, reckless)),
                    (not(accident_probability(X, high)),
                    not(accident_probability(X, maximum))).


% There is a probability of an accident if person is in bad condition
accident_probability(X, none):- not(condition(X, drunk); condition(X, distracted);
                    condition(X, speeding); condition(X, reckless)).
```

Figure 5 – SWI Person RBS Rules



```
accident_probability(alex, PROBABILITY).
PROBABILITY = maximum
accident_probability(blake, PROBABILITY).
PROBABILITY = low
accident_probability(sarah, PROBABILITY).
PROBABILITY = none
accident_probability(jack, PROBABILITY).
PROBABILITY = high
```

Figure 6 – Results SWI RBS Person

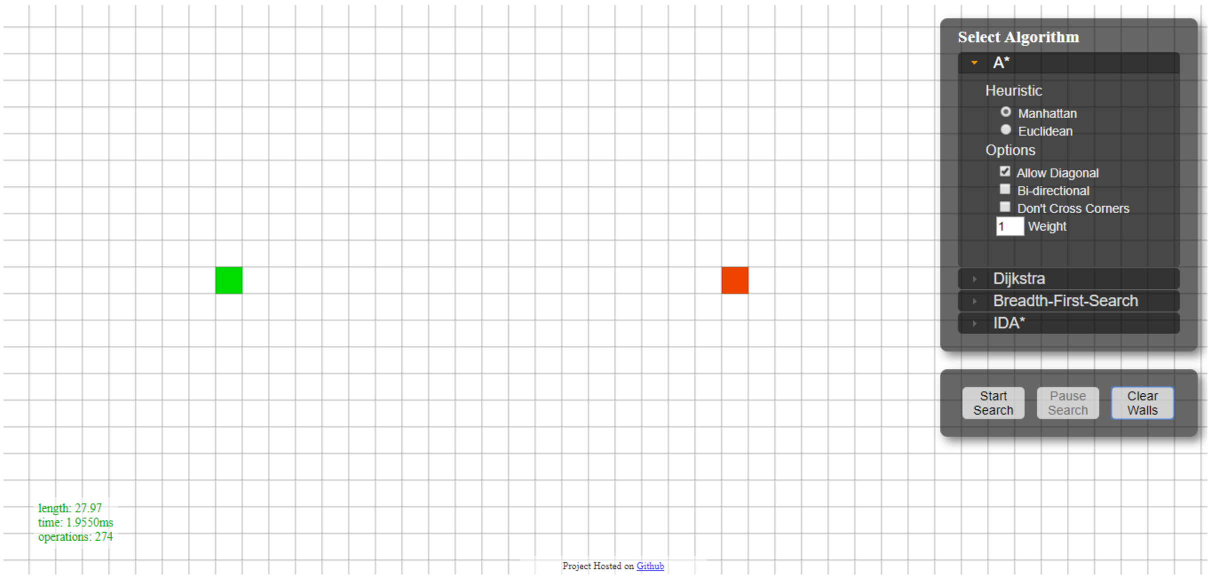
Table below provides test results and comments.

Test	Outcome	Comments
Alex	Maximum	Alex is drunk, distracted, reckless and speeding. As a result, his probability of accident is maximum.
Blake	Low	Blake is drunk but has no other conditions. This makes the probability of accident existent but low.
Sarah	None	Sarah is in good condition to drive and has no probability of an accident.
Jack	High	Jack is drunk and speeding and there is a high chance of an accident occurring.

A similar program can be produced for weather conditions. We can also use a combination of different measures to predict accidents according to rules. For example, we can take vehicle as well as weather conditions together to predict chances of an accident. This was a basic implementation of rule-based systems in the SWI prolog. Reality requires more complex rules and conditions for better precision.

3.2 Search Algorithms and Pathfinding

Search algorithms are mostly used for finding paths from a point to the other. The different search techniques could be better in their own aspects and provide better results depending on the situation. The search techniques that will be focused here will be BFS(Breadth First Search), A* algorithm, Dijkstra's algorithm and IDA* (Iterative Deepening A*) algorithm. The solution system is taken from an online source at GitHub which is an open source pathfinding JavaScript library (Xu, 2017). The author has included an intuitive visual representation of how these techniques work in finding a path.


Select Algorithm
A*
Heuristic
[✓ Manhattan]
Euclidean
Options
[✓ Allow Diagonal]
Bi-directional
Don't Cross Corners
1 Weight

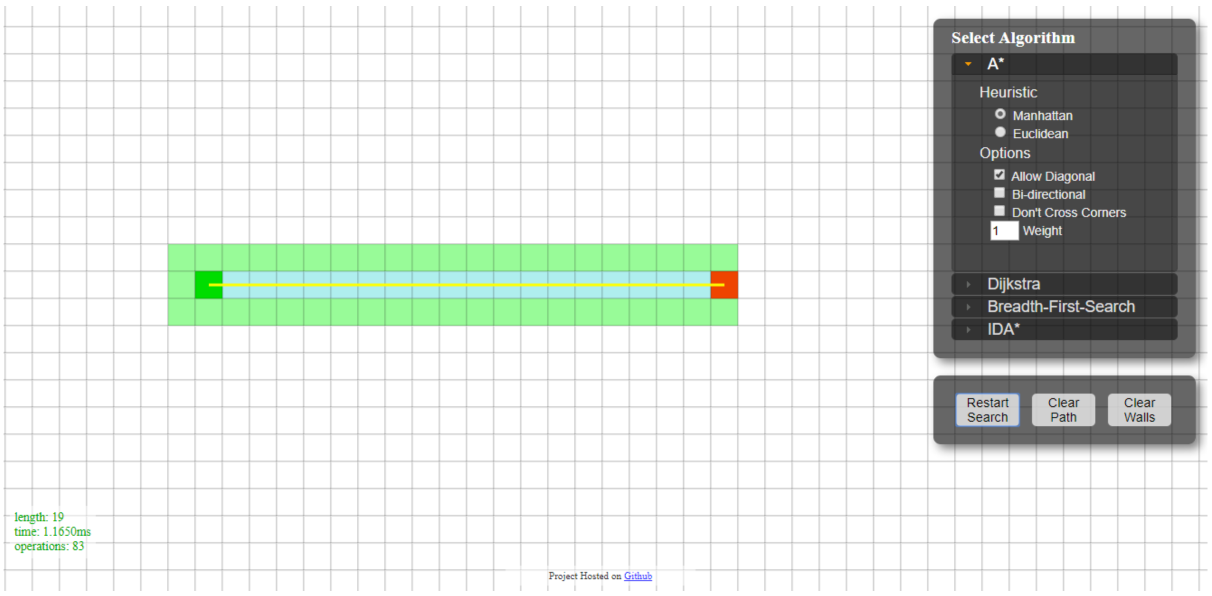
```
length: 27.97
time: 1.9550ms
operations: 274
```

Figure 7 – Pathfinding Index Page

Obstacles in the form of walls can be added to observe how different techniques work with these conditions which could be interpreted as an actual road. This can provide reflection of which technique is better at provided conditions. There could be several paths to reach the same destination, search techniques should provide the optimum path with minimal processing power or operations involved. These are the key things that separate search algorithms, i.e. the better search algorithm should provide the best path and should do so faster with less steps which will be key in real-world situations. It is even more important to do so where intelligent or self-driving vehicles are involved as instantaneous decision making can be a need at times.

3.2.1 A* Algorithm

The main aim of A* algorithm is to find an approximation of the shortest path between any two nodes. This takes obstacles into account and is very commonly used in maps and games for navigation. It is an extension of the Dijkstra's algorithm which will also be talked about in one of the next topics. The key difference from blind search techniques such as Breadth-First-Search and Depth-First-Search is that A* also measures cost of nodes and is a heuristic search technique. There could be several heuristic approaches among which some are faster (Manhattan) while some could be more accurate while being slower (Euclidean).



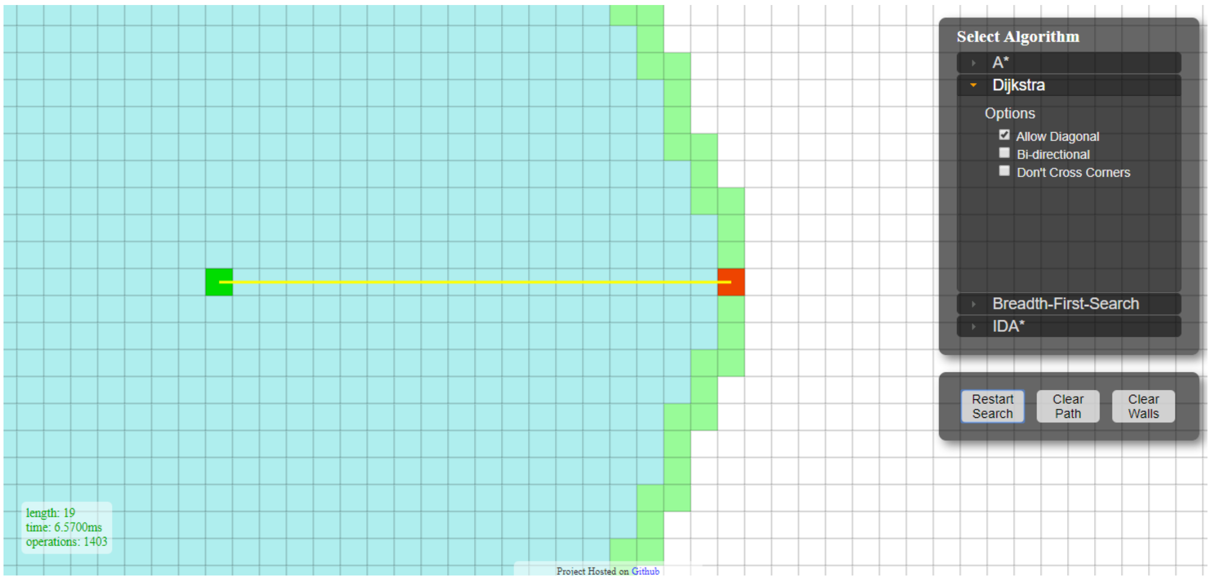
Heuristic
[✓ Manhattan]
Euclidean
Options
[✓ Allow Diagonal]
Bi-directional
Don't Cross Corners
1 Weight

```
length: 19
time: 1.1650ms
operations: 83
```

Figure 8 – A Manhattan Between Two Nodes*

3.2.2 Dijkstra's Algorithm

Like A*, Dijkstra's algorithm also works to find the shortest path between nodes. This algorithm can be applied on a weighted graph. The weight of the nodes can simulate effects such as traffic congestion. For example: there is a shorter road with huge traffic compared to a longer road with very less traffic, the weight of such paths could still be same, or the longer road could possibly even have a better weight.



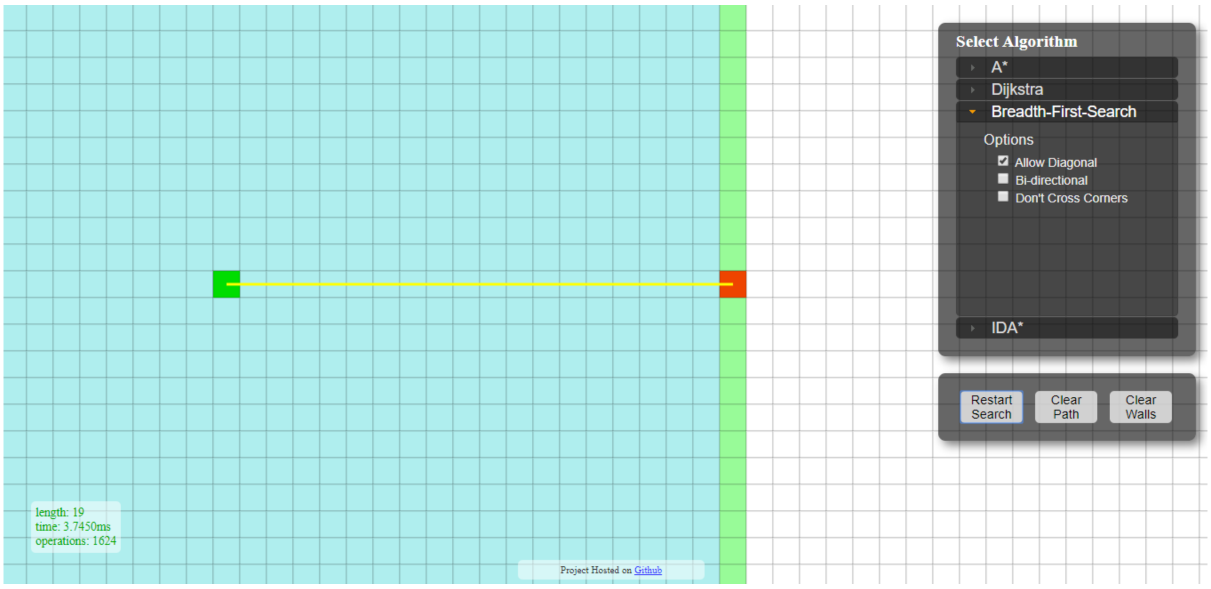
☑ Allow Diagonal
☐ Bi-directional
☐ Don't Cross Corners

```
length: 19
time: 6.5700ms
operations: 1403
```

Figure 9 – Dijkstra Between Two Nodes

3.2.3 Breadth First Search

Breadth First Search also uses a similar strategy as Dijkstra's algorithm but works on unweighted graphs. Both of these algorithms can be used to find a path between two points. Breadth First Search provides a path regardless of it being the shortest one whereas if there are multiple paths, Dijkstra's algorithm provides the shortest path in terms of weight. All the neighborhood nodes are explored until the goal node is reached.



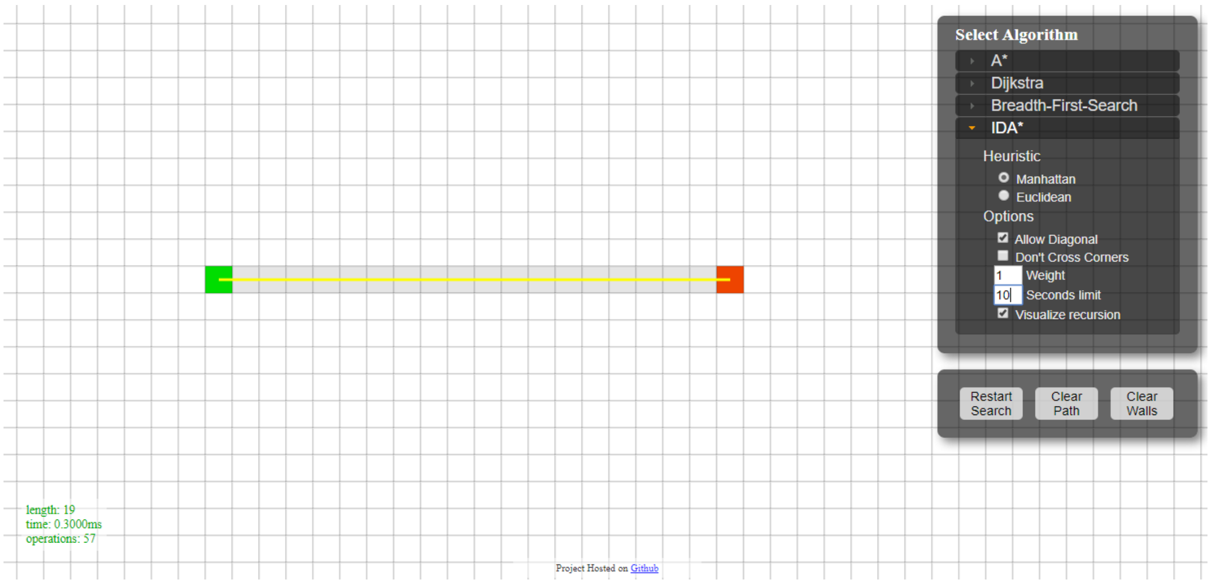
☑ Allow Diagonal
☐ Bi-directional
☐ Don't Cross Corners

```
length: 19
time: 3.7450ms
operations: 1624
```

Figure 10 – BFS Between Two Nodes

3.2.4 IDA* Algorithm

IDA* algorithm is a depth-first-search approach and often provides results faster than A* (Bu and Korf, 2019). Although providing lower memory consumption, IDA* could at times explore same visited nodes many times. The main case where IDA* could be more appropriate is in a large search space requiring large memory where A* could become very impractical.



☑ Allow Diagonal
☐ Don't Cross Corners
1 Weight
10 Seconds limit
☑ Visualize recursion

```
length: 19
time: 0.3000ms
operations: 57
```

Figure 11 – IDA Between Two Nodes*

The comparison between these search techniques via the solution system is included in the appendix.

3.3 Finite-State-Machine

Finite-State-Machines can be used to define different states within a system. The examples have been discussed in the literature review section. The Finite-State-Machines can also be used for traffic signal control and in other cases where fixed sum of states can be labelled. The FSM that will be produced as part of this solution will be a small one that could aid in parking as part of Intelligent Transport. The system will detect and notify different states in a parking lot to notify upcoming vehicles about parking spaces. Different states in a particular parking space can be recorded similar to the highway surveillance discussed earlier, for example: "vacant, occupied, vehicle arriving, vehicle departing and partially empty" to record whether the parking space is available for any incoming vehicle. The time can be calculated between vehicle arriving and vehicle departing states to calculate parking fee. The partially empty state can notify when a vehicle is not parked correctly and is occupying two parking spaces at once.

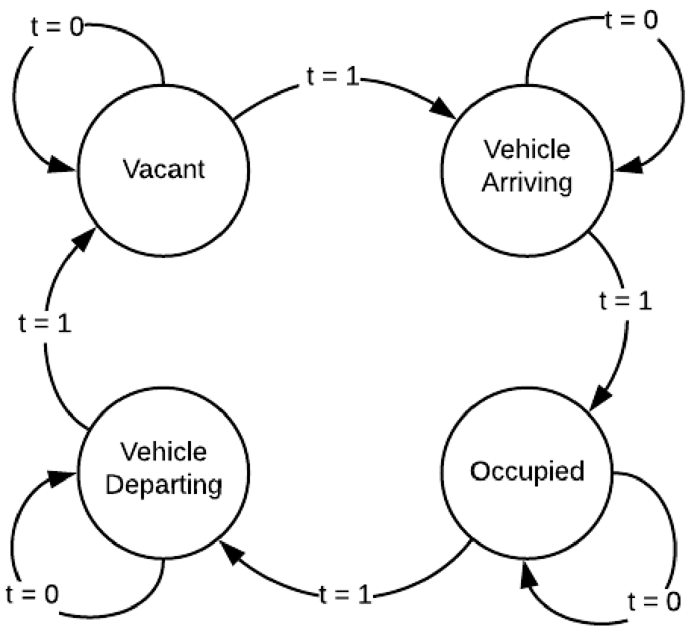


Figure 12 – Moore Finite State Machine for Intelligent Parking Space

The illustration is a simple FSM that consists of four different states. The Vacant state is the initial state of the machine where the parking space is empty. With an input of 1, the state of the machine changes to Vehicle Arriving which defines the state where a vehicle is entering the parking space and is moving. When the vehicle is fully parked, the state is transitioned to Occupied. When the vehicle starts to move out of the parking space, the state is switched to

Vehicle Departing which ultimately leads to the initial state of Vacant. Using sensors and actuators, combined with this FSM, a parking lot can be made intelligent and informative. Using another digital circuit, the time of one full cycle can be calculated to deduce parking costs by noting the time interval between transition from Vacant to Vehicle Arriving and Vehicle Departing to Vacant.

Timer(t)	Current State	Next State	Output
0	Vacant	Vacant	00
1	Vacant	Vehicle Arriving	00
0	Vehicle Arriving	Vehicle Arriving	01
1	Vehicle Arriving	Occupied	01
0	Occupied	Occupied	10
1	Occupied	Vehicle Departing	10
0	Vehicle Departing	Vehicle Departing	11
1	Vehicle Departing	Vacant	11

Vacant:	00
Vehicle Arriving:	01
Occupied:	10
Vehicle Departing:	11

This was a simpler FSM based on Moore Finite State Machine. Depending on the purpose of the FSM, Mealy Finite State Machine can be used for example to find possibility of pollution or accident in a transportation system. The FSM below will be designed for detecting whether an emergency vehicle is stuck in a traffic jam.

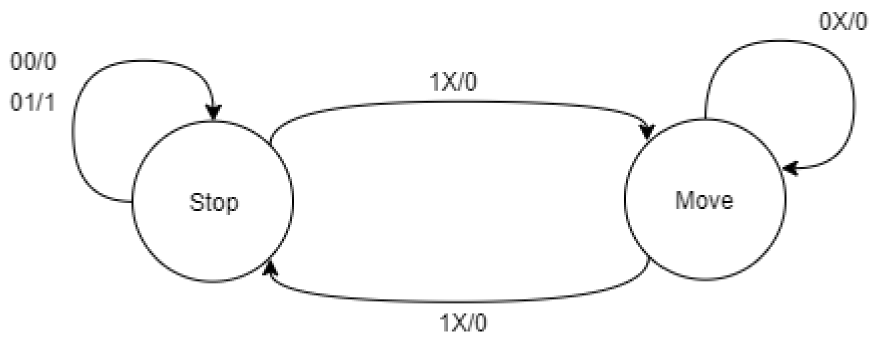


Figure 13 – Mealy Finite State Machine for Detecting Emergency Vehicles in Traffic Jam

Timer:	t
Emergency Vehicle:	ev

This is a slightly different approach as the output depends not only on the states but also on what the input is. We can display the results in a truth table:

Timer(t)	Emergency Vehicle(ev)	Current State	Next State	Output
0	0	Stop	Stop	0
0	0	Move	Move	0
0	1	Stop	Stop	1
0	1	Move	Move	0
1	0	Stop	Move	0
1	0	Move	Stop	0
1	1	Stop	Move	0
1	1	Move	Stop	0

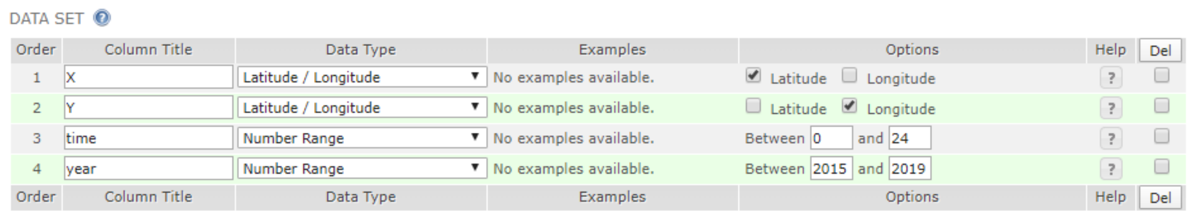
This provides the output as 1 or true when sensors detect an emergency vehicle and the vehicle is stuck in a traffic jam. A similar example is shown for pollution detection produced from trucks by Abelardo Pardo in his lesson on Finite State Machines (Abelardo Pardo, 2013).

3.4 Clustering

As mentioned, clustering is used to group similar types of data without providing any labels. We can use this technique to identify areas where accidents occur frequently, group accidents or associate different attributes with accidents such as time or weather. Microsoft provides ML.NET to train and use machine learning models which can run image classification, deep learning, clustering and many other algorithms. A console C# application will be written to implement K-Means Clustering which will calculate the value of a centroid after grouping items into clusters. Farragher explains an easy way to implement clustering using ML.NET on the Iris Flower Dataset (Farragher, 2019) which is also included in the Microsoft ML.NET documentation. A similar program will be implemented to cluster accident records and predict under which cluster a new accident is categorized.

3.4.1 The Dataset

The first step is to design or refer to an existing dataset. This solution will attempt to produce imaginary data in order to observe the clusters. Selecting correct attributes is important. The attributes that will be used to train the model will be XY-Coordinates as latitude and longitude, Time (in hours) and Year between 2015 and 2019.

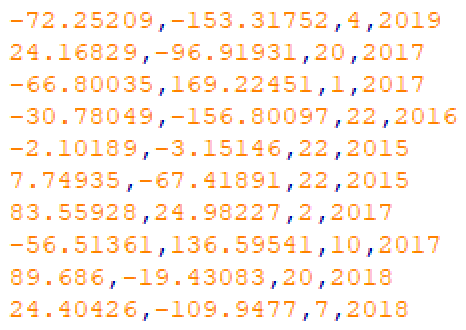


DATA SET

Order	Column Title	Data Type	Examples	Options	Help	Del
1	X	Latitude / Longitude	No examples available.	✓ Latitude ☐ Longitude	?	☐
2	Y	Latitude / Longitude	No examples available.	☐ Latitude ✓ Longitude	?	☐
3	time	Number Range	No examples available.	Between 0 and 24	?	☐
4	year	Number Range	No examples available.	Between 2015 and 2019	?	☐
Order	Column Title	Data Type	Examples	Options	Help	Del

Figure 14 – Creating the Dataset from Generatedata.com

100 rows of data was generated. This will be used to train the model and group the data into clusters.

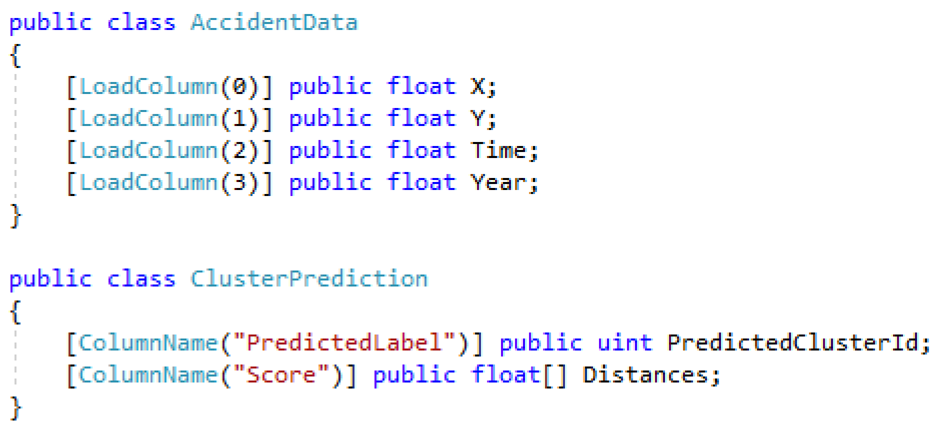


```
-72.25209,-153.31752,4,2019
24.16829,-96.91931,20,2017
-66.80035,169.22451,1,2017
-30.78049,-156.80097,22,2016
-2.10189,-3.15146,22,2015
7.74935,-67.41891,22,2015
83.55928,24.98227,2,2017
-56.51361,136.59541,10,2017
89.686,-19.43083,20,2018
24.40426,-109.9477,7,2018
```

Figure 15 – Generated Data

3.4.2 The Solution Program

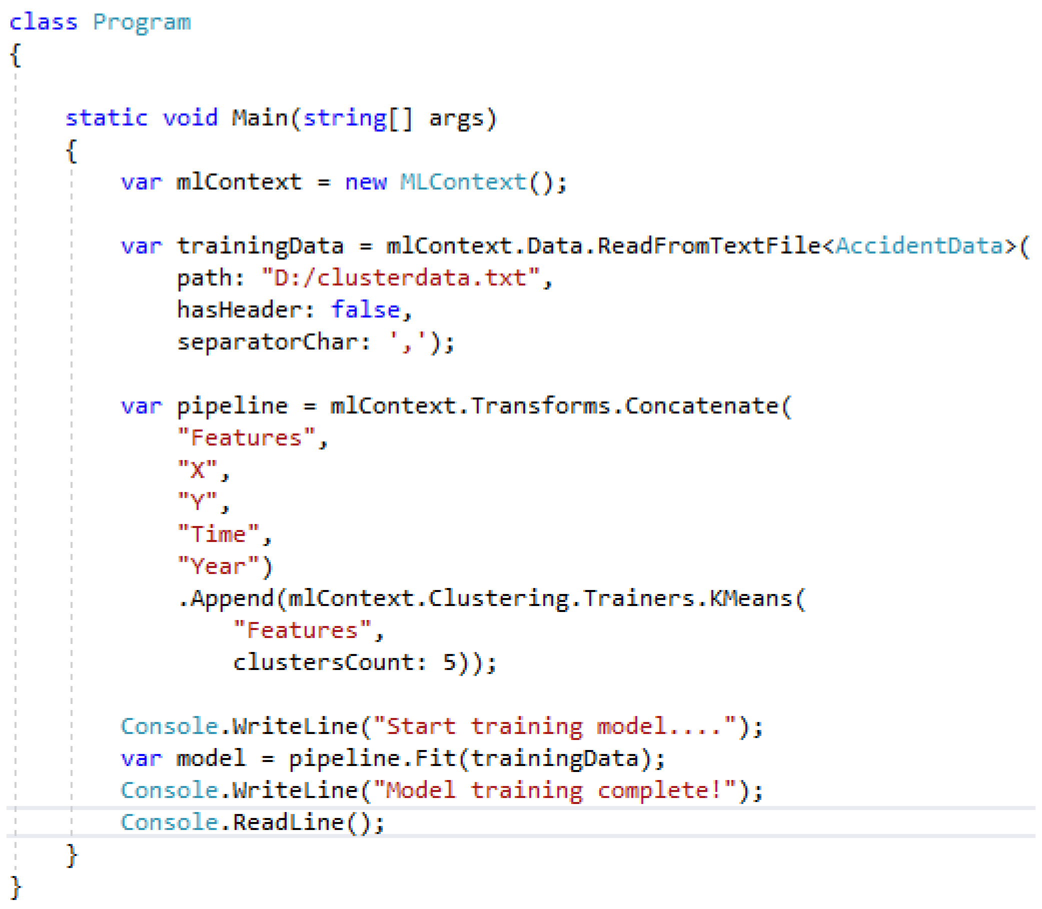
The second step is coding the solution. As provided in the referenced source, the use of clustering on petals is replaced to use it on accident data.



```
public class AccidentData
{
    [LoadColumn(0)] public float X;
    [LoadColumn(1)] public float Y;
    [LoadColumn(2)] public float Time;
    [LoadColumn(3)] public float Year;
}

public class ClusterPrediction
{
    [ColumnName("PredictedLabel")] public uint PredictedClusterId;
    [ColumnName("Score")] public float[] Distances;
}
```

Figure 16 – Accident Data and Reporting

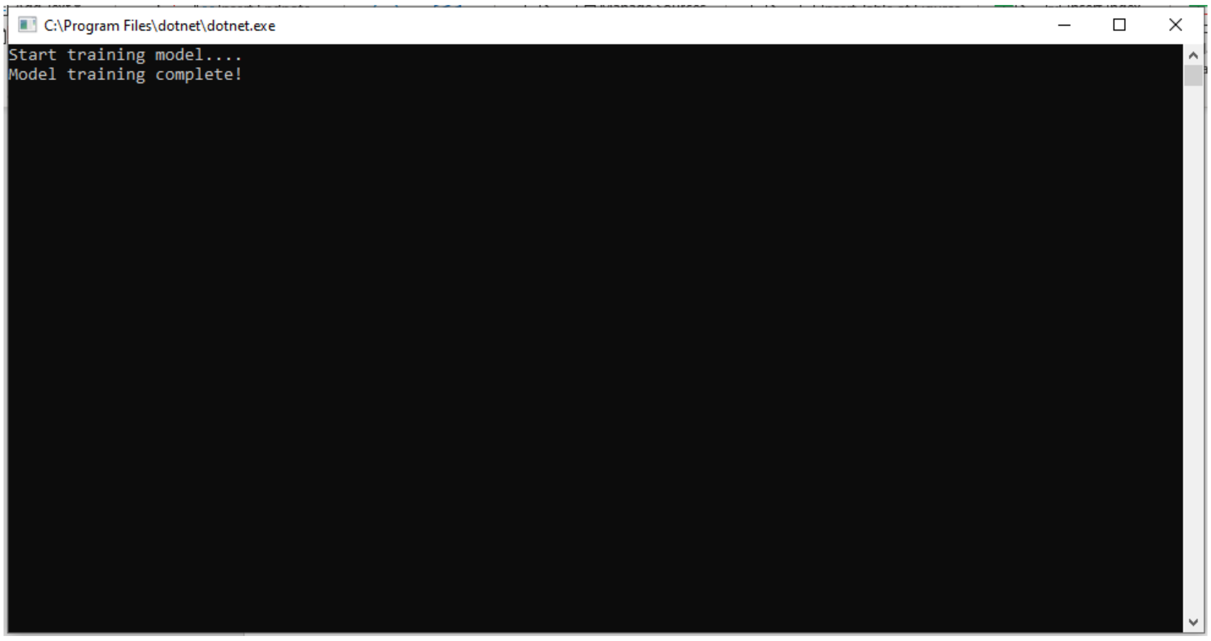
The two classes here will hold the accident data and report of the cluster prediction.



```
class Program
{

    static void Main(string[] args)
    {
        var mlContext = new MLContext();

        var trainingData = mlContext.Data.ReadFromTextFile<AccidentData>(
            path: "D:/clusterdata.txt",
            hasHeader: false,
            separatorChar: ',');

        var pipeline = mlContext.Transforms.Concatenate(
            "Features",
            "X",
            "Y",
            "Time",
            "Year")
            .Append(mlContext.Clustering.Trainers.KMeans(
                "Features",
                clustersCount: 5));

        Console.WriteLine("Start training model....");
        var model = pipeline.Fit(trainingData);
        Console.WriteLine("Model training complete!");
        Console.ReadLine();
    }
}
```

Figure 17 – The Program Class and Main Method

The Program class contains the main method where the file will be loaded, and the model will be trained.



C:\Program Files\dotnet\dotnet.exe

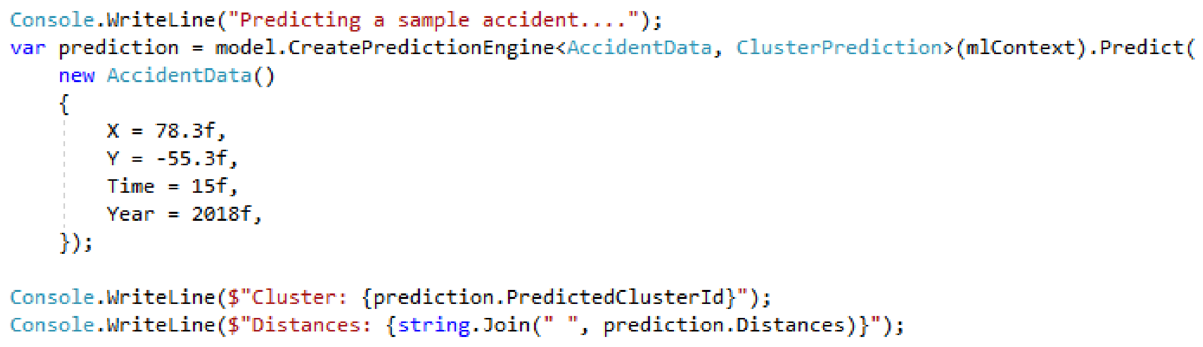
```
Start training model....
Model training complete!
```

Figure 18 – Training Model Console Results

Running the application now will train the model and display the messages in the console.

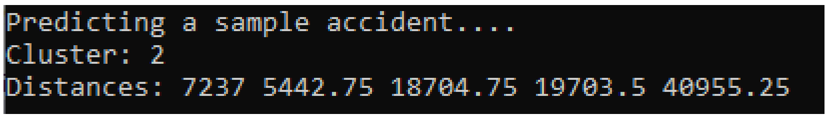
3.4.3 Analyzing the Results

We can predict under which cluster a new accident is categorized. This random data is used to predict the cluster and distances. The centroid is calculated for every cluster which can be referred to as the exact type of accident that is occurring in that cluster. The distance shows how much the predicted accident deviates from the centroid.



```
Console.WriteLine("Predicting a sample accident....");
var prediction = model.CreatePredictionEngine<AccidentData, ClusterPrediction>(mlContext).Predict(
    new AccidentData()
    {
        X = 78.3f,
        Y = -55.3f,
        Time = 15f,
        Year = 2018f,
    });

Console.WriteLine($"Cluster: {prediction.PredictedClusterId}");
Console.WriteLine($"Distances: {string.Join(" ", prediction.Distances)}");
```

Figure 19 – Predicting Accident Cluster



```
Predicting a sample accident....
Cluster: 2
Distances: 7237 5442.75 18704.75 19703.5 40955.25
```

Figure 20 – Cluster Prediction Results

The accident matches and inclines towards the second cluster but deviates with the provided distances. The distances are squared Euclidean distances from the centroid. The 5 values mean that there are 5 clusters in total and the least deviation is from the second cluster which has the least value at 5442.75.

Similarly, different other accident records can be predicted under these clusters. This was only a basic implementation of clustering on self-produced accident data. These results can be improved drastically with the use of correct and a larger number of data and grouping them into suitable number of clusters. With each accident, the type of accident can be easily identified with clustering.

4. CONCLUSIONS & FURTHER WORK

4.1 Analysis of Different Techniques

Technique	Description
Rule Based Systems	The solution in this project included programs in the SWI prolog. It comprised of two programs which discussed about using predefined rules and facts to predict probability of an accident. The solution was basic and was intended to provide some level of evidence of how Rule-Based systems can be used in the real world. For a better program, the facts included could be merged within a single application, the rules could be refined with the help of experts and use of other facts that model the real world correctly (example: damaged or poorly built roads, bridges, disasters, pedestrians, etc.). A more advanced application, for example: with the use of knowledge-base and inference engine would simplify the process of predicting the likelihood of an accident. Basically, Rule-Based systems can be very effective at automating different situations. It was noted how Fuzzy Rule-Based Systems are used to control traffic flow along with movement of VIP or emergency vehicles from the review of literature. Another literature discussed the use in transporting Hazmat goods such as chemicals. However, good expertise and very accurate information for the facts and rules is crucial to produce an effective application. There are several good as well as poor aspects of Rule-Based Systems.

Strengths	Weaknesses
Good representation of human language and reasoning with if-then conditions.	Difficult to implement in a large-scale application.
Even after building, new rules can be added as improvements.	In order to stay updated, the rules might need to be updated frequently.

The facts and rules are mostly easy to articulate and program as they are readily available through statistics and expert human knowledge.	There could however be certain conditions where these rules and facts are difficult to implement.

	We can evidentially conclude through the experimentation with self-derived solution as well as literatures that Rule-Based Systems can be very effective in real-world situations. These systems can be used as substitutes or to augment human expertise where repetitive workloads requiring accuracy and expert knowledge is important. Long and repetitive tasks could be prone to human error; use of machines that can reason and are prepared with the knowledge of same human workers would be very efficient and effective.
Search Algorithms (BFS, Dijkstra, A*)	The solution was taken from an online GitHub source which was provided as an open source pathfinding JavaScript library. The appendix contains test results of different situations. There could be cases where some search algorithms are better than other. Most search techniques are derived and improved from one another such as Dijkstra's Algorithm being an improvised Breadth-First Search that takes weight of paths into account. Several other search algorithms also exist which could lead to better results depending on the situation. As mentioned, the main area where these search techniques could be implemented for an intelligent transport system would be for navigation. It could be for providing an intelligent or self-driving vehicle with a path to its destination or simply provide road navigation based on GPS (Global Positioning System) or to provide the driver with a suitable path (usually best and shortest) between points which is used in many mapping applications such as Google maps.

Finite-State-Machines	An FSM can be a good way to represent a system with limited number of states. Similar implementations have also been used in the early Software Analysis Techniques (Modern Structured Analysis) as State Transition Diagrams. Through research, it is known that FSM can produce clear structures and simplify complex problems. An example paper discussed the use of FSM for highway surveillance. The solution presented in this paper attempts to show practical evidences of how FSM can be used to model real-world situations in the transportation industry via the included two types of FSMs (Moore and Mealy). The FSMs were used to define different states of a parking lot and to check the presence of an emergency vehicle stuck in a traffic jam. Using FSMs, together with digital circuits, sensors and actuators, efficient autonomous systems can be produced to make life easier. It can be concluded that FSMs are very effective at modelling situations with finite number of states. For a larger application with a very large number of states however, other techniques could be better. FSM can still be used to represent states at smaller levels (breaking them down).
Clustering	Clustering technique has been produced as a solution in this project to produce experimental evidences, along with the information obtained through research and literature review which can be critically interpreted. The solution system tended to identify and group accidents into clusters. It attempted to find patterns from accident data. It was a basic implementation of clustering to group accidents into clusters. Some authors have also provided separate clustering algorithms for identifying black spots (Szenasi, 2014). This can help look up for the root causes and solve the issue in faults such as engineering or road conditions. Authors have also investigated patterns of accidents using clustering in real-world

	situations such as by grouping and visualizing accident frequencies of Riyadh, Saudi Arabia (Almjewail et al., 2018). As mentioned earlier, clustering techniques are used to group similar data. If the data is grouped according to labels, it is called classification. Clustering is done without feeding any knowledge such as labels and falls under unsupervised learning. The literatures mentioned about different types of clustering techniques and how some of them could be used to aid intelligent transport. The main benefit of clustering will be discovery of patterns or trends that we would otherwise not recognize. For example: based on previous statistical evidences of previous accidents available from reliable sources such as WHO (World Health Organization), patterns could be discovered which would pinpoint the root cause of accidents. The major advantage is that clustering techniques will provide original information which is not prone to human error. One of the literatures also mentioned about the use of clustering to avoid congestion or traffic jams. Clustering techniques can have several good as well as bad aspects. Clustering techniques are highly scalable; the more the data is available for clustering, the better the results will be understood by the AI and will provide better outcomes. It also allows to find patterns or trends that would otherwise not be recognized by humans. The significant disadvantage of clustering however is that it requires a lot of data to produce good results that make sense. Another important factor that needs to be considered with using clustering is bias. There have been bad examples of bias in classification and clustering such as sexism (relating picture of kitchen or shopping with women more often) or racism (Simonite, 2017). The data that used to train the AI needs to be relevant and selected carefully and possible bias should be considered while using clustering techniques.

4.2 Future of AI in Intelligent Transport

The future of AI, as in many other sectors, is very bright in the field of Transportation. Many attempts are being made to automate driving as the roads are prone to accidents because of human factors. These are not yet implemented practically because of failed tests such as the Uber incident (BBC News, 2018). Many experts believe that AI will make Transportation safe and efficient. This will in turn increase the growth of other industries. Vehicles being able to communicate with other smart technology (V2X) and with each other (V2V) will make them smarter. The development of transportation industry will not just be limited to roads but will also happen in waterways and air transportation through Artificial Intelligence.

Figure 21 – Smart Transportation System (E-SPIN Group, 2019)

REFERENCES

History of Artificial Intelligence (Lecture)

Smith, C., et al. (2006). *The History of Artificial Intelligence*. [ebook] lecture notes, University of Washington. Available from: https://courses.cs.washington.edu/courses/csep590/06au/projects/history-ai.pdf [Accessed 12/12/2019].

Applications of AI (Course)

Kelly, J. (2019). *Some Applications of AI*, Introduction to Artificial Intelligence(AI), lecture notes, Rav Ahuja, IBM, Coursera. Available from: https://www.coursera.org/learn/introduction-to-ai/lecture/6zl2S/some-applications-of-ai [Accessed 12/12/2019].

Intelligent Transport (Image)

Pasqua, E. (2019). *The combination of 5G and AI would allow driver assistance and traffic monitoring systems to reach their full potential*. [image] Available at: https://iot-analytics.com/wp/wp-content/uploads/2019/02/Intelligent-Connectivity-Combination-of-5G-and-AI-Driver-Assistance-Traffic-Monitoring-Potential-Intel.jpg [Accessed 12 Dec. 2019].

History of Intelligent Transport (Book)

Auer, A., Feese, S. and Lockwood, S. (2016). *History of Intelligent Transportation Systems*. 1200 New Jersey Ave SE: United States Department of Transportation. Intelligent Transportation Systems Joint Program Office, pp.1-31.

Clustering Definition (Book)

Kogan, J. (2007). *Introduction to clustering large and high-dimensional data*. New York: Cambridge University Press, pp. xiii.

Clustering in Intelligent Transport (Journal)

Qiong, L., Jie, Y., & Jinfang, Z. (2011). *Application of Clustering Algorithm in Intelligent Transportation Data Analysis*. Information and Management Engineering, pp.467–473. Available from DOI: 10.1007/978-3-642-24097-3_70 [Accessed: 12/12/2019]

Clustering Transportation Sensors (Journal)

Geetla, T., Batta, R., Blatt, A., Flanigan, M. and Majka, K. (2016). *Clustering intelligent transportation sensors using public transportation*. TOP, 24(**3**), pp.594-611. Available from DOI: 10.1007/s11750-016-0410-7 [Accessed: 15/12/2019]

Intelligent Transport Decision Analysis Based on Big Data Mining (Journal)

Ying, C. (2017). *Intelligent Transport Decision Analysis System Based on Big Data Mining*. 7th International Conference on Education, Management, Information and Computer Science (ICEMC 2017). Atlantis Press. Available from DOI: 10.2991/icemc-17.2017.246 [Accessed: 15/12/2019]

Rule-Based Systems (Journal)

Grosan, C., & Abraham, A. (2011). *Rule-Based Expert Systems*. Intelligent Systems, pp.149–185. Available from DOI: 10.1007/978-3-642-21004-4_7 [Accessed: 15/12/2019]

Fuzzy Rule-Based Traffic Control System (Journal)

Sheraz, S. M., Abbas, S. A., & Noor, H. (2009). *Fuzzy Rule Based Traffic Signal Control System for Oversaturated Intersections*. 2009 International Conference on Computational Intelligence and Natural Computing. Available from DOI:10.1109/cinc.2009.245 [Accessed: 16/12/2019]

Hazmat Transport (Journal)

Asadi, R., & Ghatee, M. (2015). *A Rule-Based Decision Support System in Intelligent Hazmat Transportation System*. IEEE Transactions on Intelligent Transportation Systems, 16(**5**), 2756–2764. Available from DOI:10.1109/tits.2015.2420993 [Accessed: 16/12/2019]

Finite-State-Machines Definition (Website)

Rouse, M. (2019). *What is a Finite State Machine?* [online] WhatIs.com. Available at: https://whatis.techtarget.com/definition/finite-state-machine [Accessed 19 Dec. 2019].

Finite-State-Machine Base Driving Controller (Journal)

Zhang, M., Li, N., Girard, A., & Kolmanovsky, I. (2017). *A Finite State Machine Based Automated Driving Controller and its Stochastic Optimization*. Volume 2: Mechatronics; Estimation and Identification; Uncertain Systems and Robustness; Path Planning and Motion

Control; Tracking Control Systems; Multi-Agent and Networked Systems; Manufacturing; Intelligent Transportation and Vehicles; Sensors and Actuators; Diagnostics and Detection; Unmanned, Ground and Surface Robotics; Motion and Vibration Control Applications. Available from DOI:10.1115/dscc2017-5209 [Accessed 19 Dec. 2019].

Finite-State-Machine Based Surveillance System (Journal)
HyunKim, D., Kwon, J., Chen, K., & Choi, K. (2013). *Finite state machine for vehicle detection in highway surveillance systems*. The 19th Korea-Japan Joint Workshop on Frontiers of Computer Vision. Available from DOI:10.1109/fcv.2013.6485465 [Accessed 19 Dec. 2019].

Shortest Pathfinding Genetic Algorithms (Journal)
Lin, C.-H., Yu, J.-L., Liu, J.-C., & Lee, C.-J. (2008). *Genetic Algorithm for Shortest Driving Time in Intelligent Transportation Systems*. 2008 International Conference on Multimedia and Ubiquitous Engineering (mue 2008). Available from DOI:10.1109/mue.2008.16 [Accessed 19 Dec. 2019].

Kumar, A. & Jeyaraj, Arunadevi & Mohan, V. (2009). *Intelligent Transport Route Planning Using Genetic Algorithms in Path Computation Algorithms*. European Journal of Scientific Research. 25(**3**), pp.463-468.

Finite State Machines Lesson (YouTube)
Abelardo Pardo (2013). *Finite State Machines explained*. [video] Available at: https://www.youtube.com/watch?v=hJIST1cEf6A [Accessed 20 Dec. 2019].

Pathfinding JavaScript Library (GitHub)
Xu, X. (2017). *qiao/PathFinding.js*. [online] GitHub. Available at: https://github.com/qiao/PathFinding.js/ [Accessed 2 Jan. 2020].

Machine Bias (Online News)
Simonite, T. (2017). *Machines Taught by Photos Learn a Sexist View of Women*. [online] Wired. Available at: https://www.wired.com/story/machines-taught-by-photos-learn-a-sexist-view-of-women/ [Accessed 3 Jan. 2020].

IDA* Algorithm (Journal)

Bu, Z. and Korf, R. (2019). A*+IDA*: A Simple Hybrid Search Algorithm. *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence*. Available from DOI: 10.24963/ijcai.2019/168 [Accessed 3 Jan. 2020].

Uber Crash BBC News (Online News)

BBC News. (2018). *Uber self-driving crash: Footage shows moment before impact*. [online] Available at: https://www.bbc.com/news/world-us-canada-43497364 [Accessed 4 Jan. 2020].

Intelligent Transport System (Image)

E-SPIN Group. (2019). *Intelligent Public Transportation Systems (IPTS) | E-SPIN Group*. [image] Available at: https://www.e-spincorp.com/intelligent-public-transportation-systems-ipts/ [Accessed 4 Jan. 2020].

Using ML.NET Clustering (Website)

Farragher, M. (2019). *Easy K-Means Clustering with C# and ML.NET*. [online] Medium. Available at: https://medium.com/machinelearningadvantage/easy-k-means-clustering-with-c-and-ml-net-7b154ccd219e [Accessed 5 Jan. 2020].

Using Clustering to Identify Black Spots (Journal)

Szenasi, S. (2014). *CLUSTERING ALGORITHM IN ORDER TO FIND ACCIDENT BLACK SPOTS IDENTIFIED BY GPS COORDIANTES*. 14th SGEM GeoConference on INFORMATICS, GEOINFORMATICS AND REMOTE SENSING. Available from DOI: 10.5593/SGEM2014/B21/S8.063 [Accessed 5 Jan. 2020].

Clustering Accident Data (Journal)

Almjewail, A., Almjewail, A., Alsenaydi, S., ALSudairy, H., & Al-Turaiki, I. (2018). Analysis of Traffic Accident in Riyadh Using Clustering Algorithms. 5th International Symposium on Data Mining Applications, 12–25. Available from DOI:10.1007/978-3-319-78753-4_2 [Accessed 5 Jan. 2020].

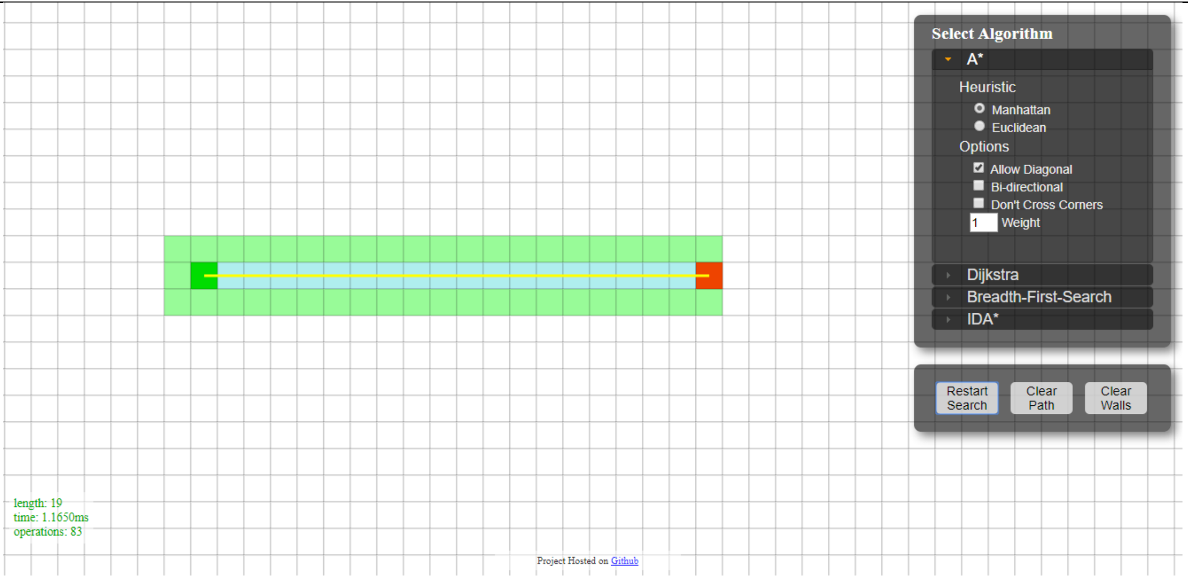
APPENDIX - TESTING SEARCH TECHNIQUES

The testing section will include the results of each search technique that was used and compare the results visually and numerically. The results from RBS and FSM are already included in the solution part and are thus not included here.

1. No obstacles, Straight Path

A* Algorithm

Number of operations: 83



Select Algorithm

A*

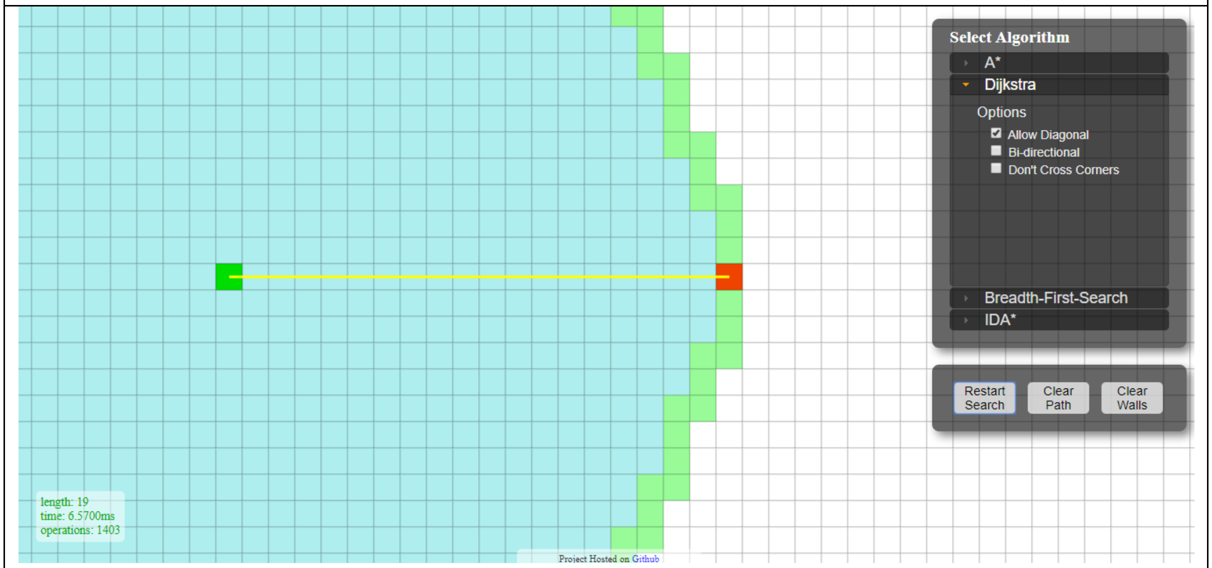
Heuristic
- Manhattan
- Euclidean

Options
- [✓] Allow Diagonal
- [ ] Bi-directional
- [ ] Don't Cross Corners
- 1 Weight

```
length: 19
time: 1.1650ms
operations: 83
```

Dijkstra's Algorithm

Number of operations: 1403



Select Algorithm

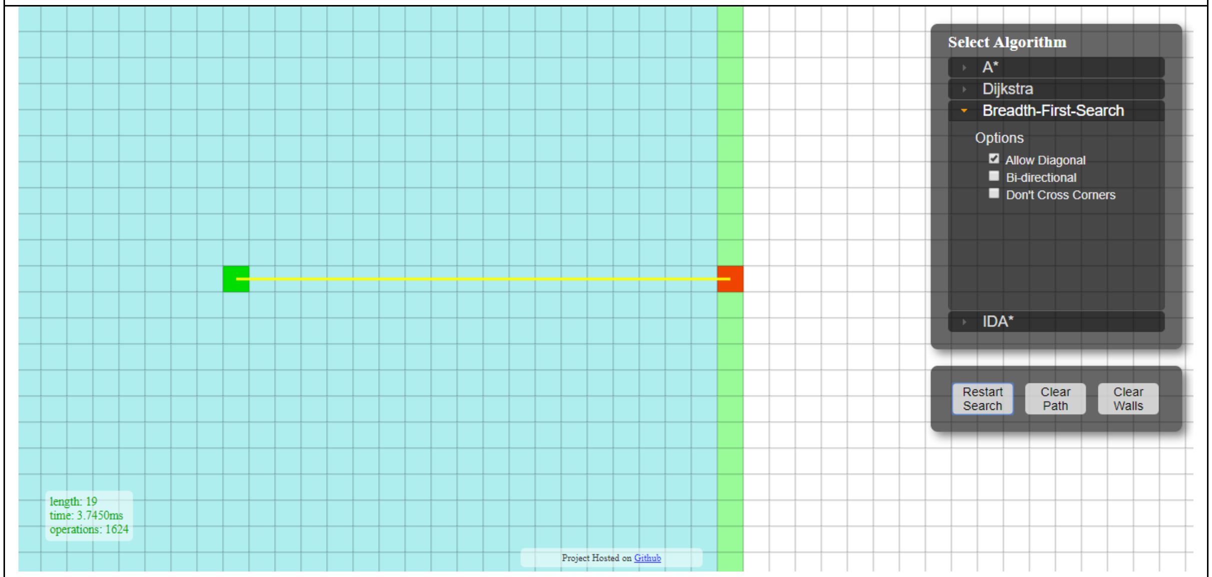
Dijkstra

Options
- [✓] Allow Diagonal
- [ ] Bi-directional
- [ ] Don't Cross Corners

```
length: 19
time: 6.5700ms
operations: 1403
```

Breadth-First Search Algorithm

Number of operations: 1624

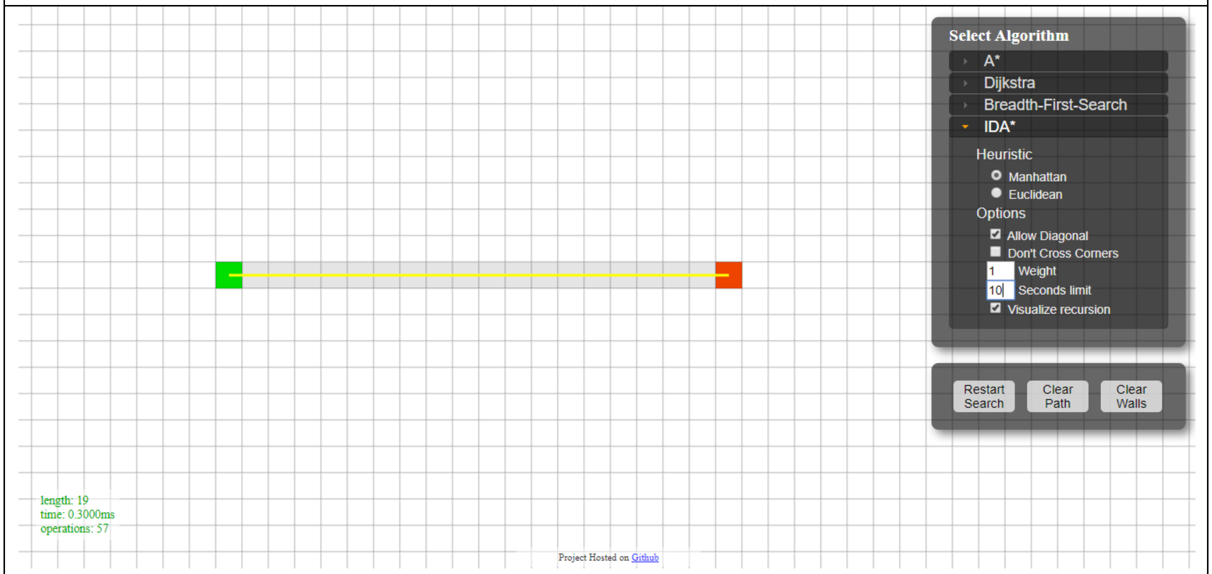


Select Algorithm
- A*
- Dijkstra
- Breadth-First-Search
  - Options
    - [✓] Allow Diagonal
    - [ ] Bi-directional
    - [ ] Don't Cross Corners
- IDA*

length: 19
time: 3.7450ms
operations: 1624

IDA* Algorithm

Number of operations: 57

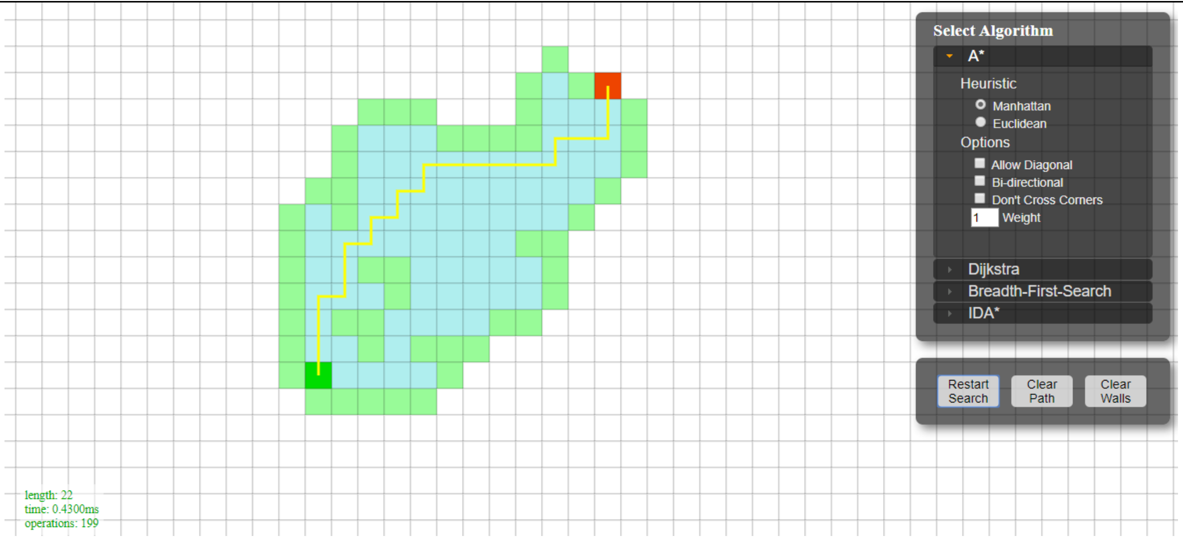
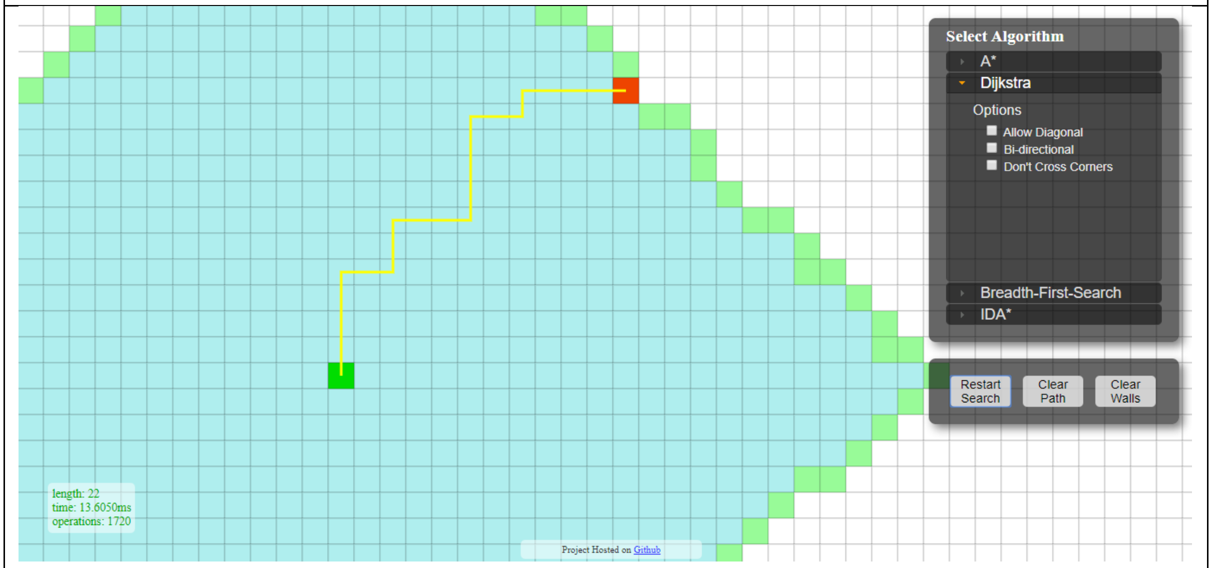


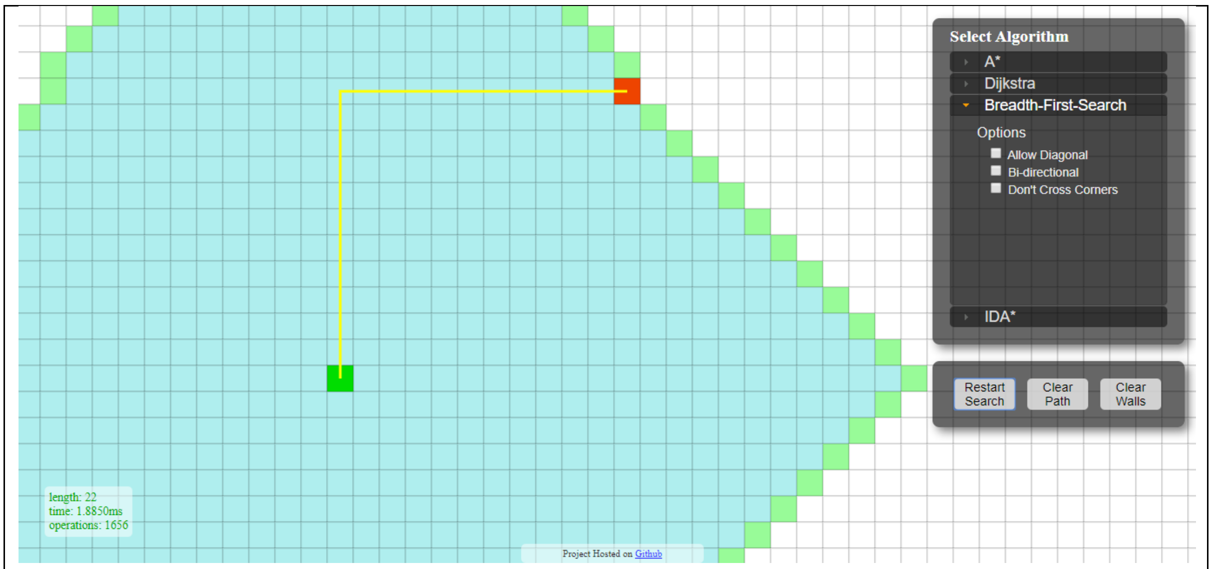
Select Algorithm
- A*
- Dijkstra
- Breadth-First-Search
- IDA*
  - Heuristic
    - ○ Manhattan
    - ● Euclidean
  - Options
    - [✓] Allow Diagonal
    - [ ] Don't Cross Corners
    - 1 Weight
    - 10 Seconds limit
    - [✓] Visualize recursion

length: 19
time: 0.3000ms
operations: 57

The most efficient algorithm in this case is IDA* Algorithm with only 57 operations. Breadth-First Search provided the result after 1624 operations. We can see from the images about how much area each search technique covered.

2. No obstacles, Diagonal Path

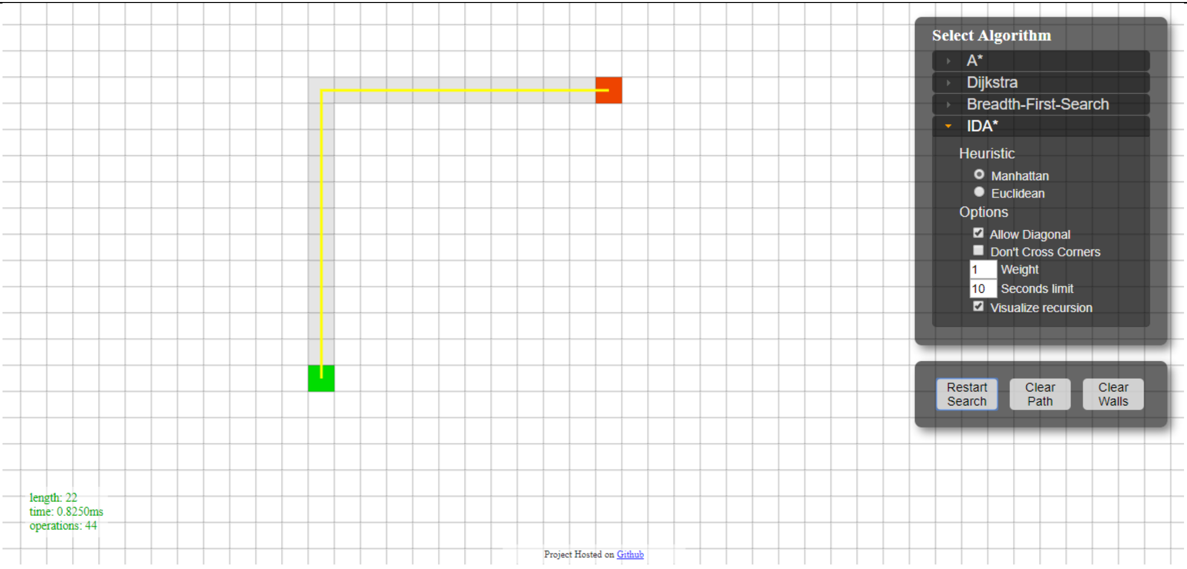
A* Algorithm
Number of operations: 199, Length: 22
 Select Algorithm A* Heuristic Manhattan Euclidean Options Allow Diagonal Bi-directional Don't Cross Corners 1 Weight Dijkstra Breadth-First-Search IDA* Restart Search Clear Path Clear Walls length: 22 time: 0.4300ms operations: 199
Dijkstra's Algorithm
Number of operations: 1720, Length: 22
 Select Algorithm A* Dijkstra Options Allow Diagonal Bi-directional Don't Cross Corners Breadth-First-Search IDA* Restart Search Clear Path Clear Walls Project Hosted on Github length: 22 time: 13.6050ms operations: 1720
Breadth-First Search Algorithm
Number of operations: 1656, Length: 22



length: 22
time: 1.8850ms
operations: 1656

IDA* Algorithm

Number of operations: 44, Length: 22

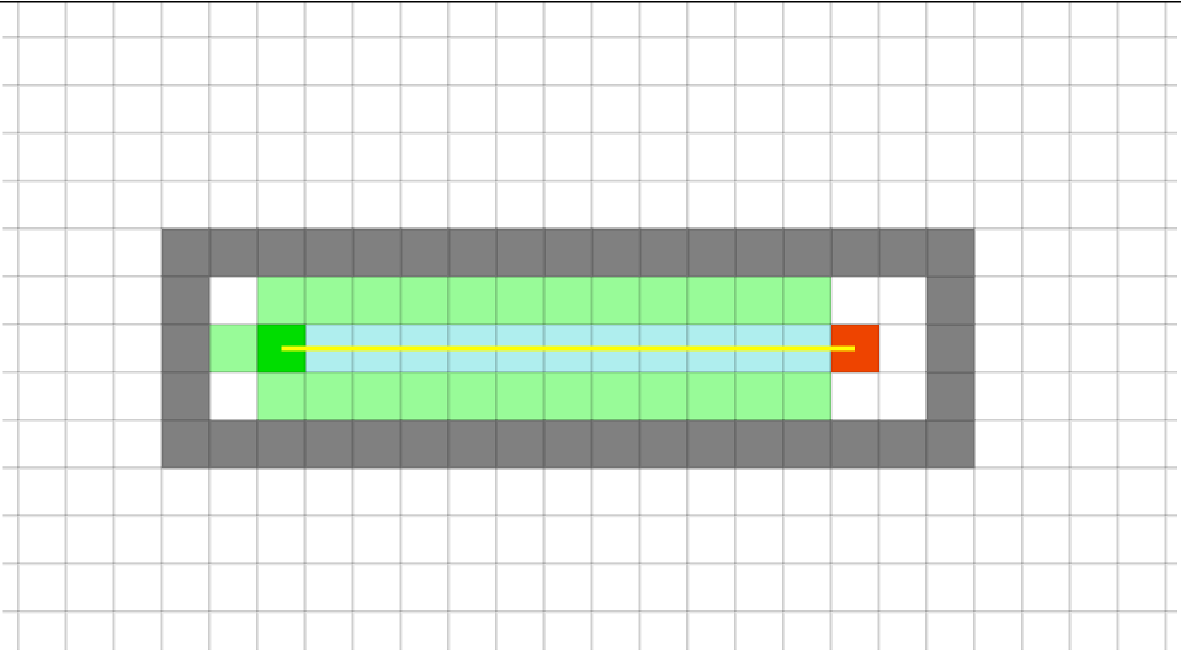
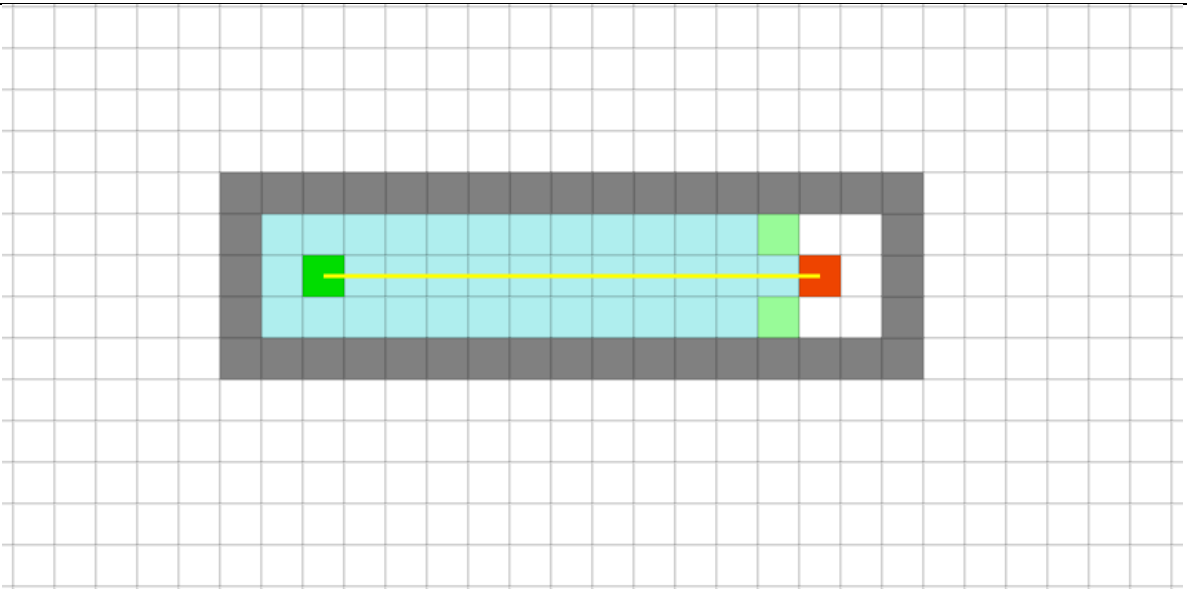


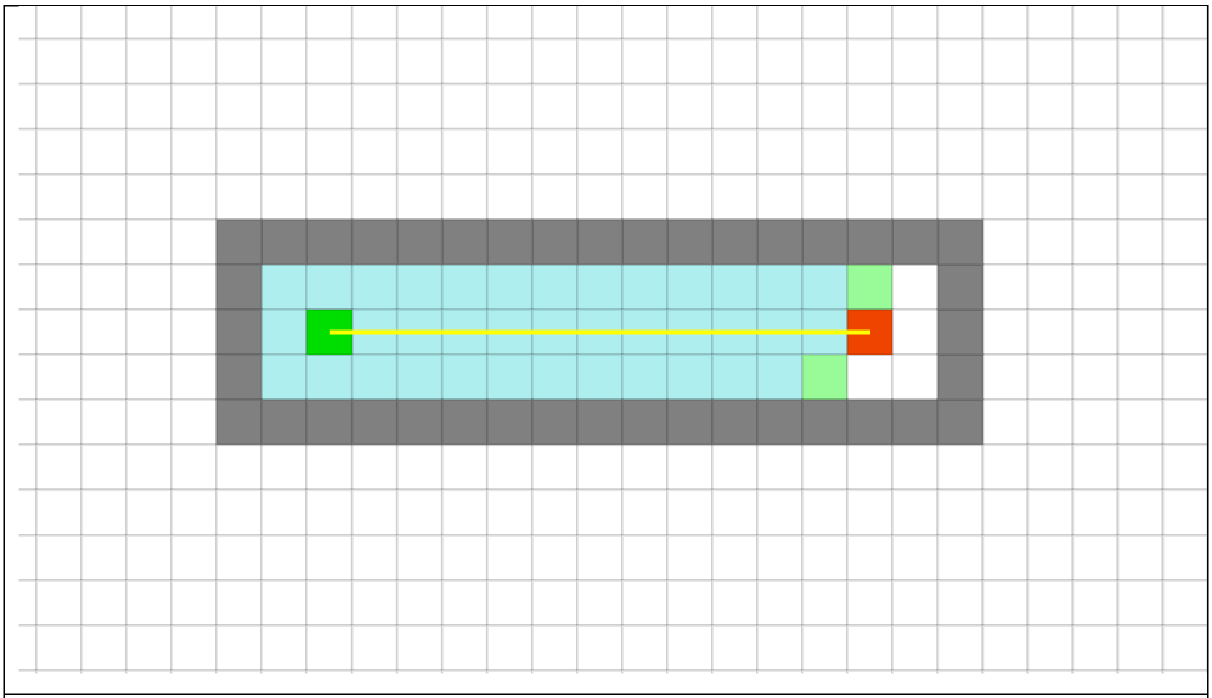
Heuristic
Manhattan
Euclidean
Options
[✓ Allow Diagonal]
Don't Cross Corners
1 Weight
10 Seconds limit
[✓ Visualize recursion]

length: 22
time: 0.8250ms
operations: 44

In this situation as well, IDA* is leading the numbers with 44 operations.

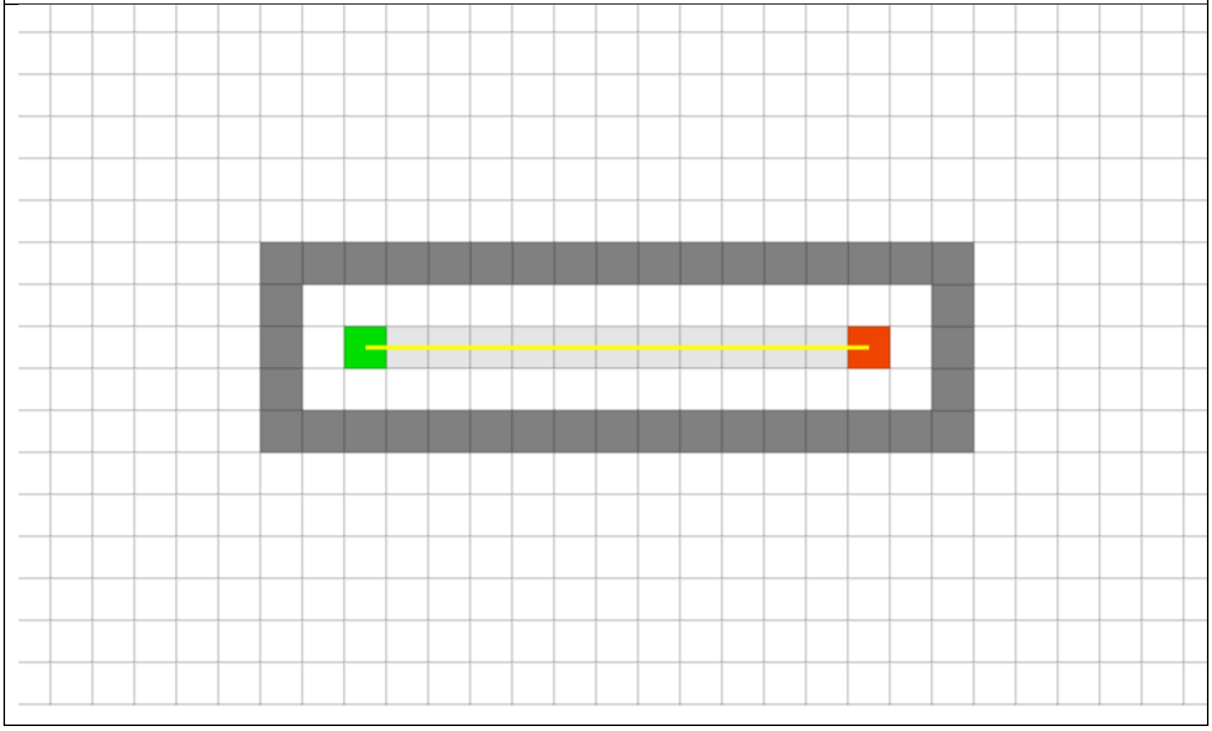
3. Surrounded by walls, Straight Path

A* Algorithm Number of operations: 51

Dijkstra’s Algorithm Number of operations: 78

Breadth-First Search Algorithm Number of operations: 80



IDA* Algorithm

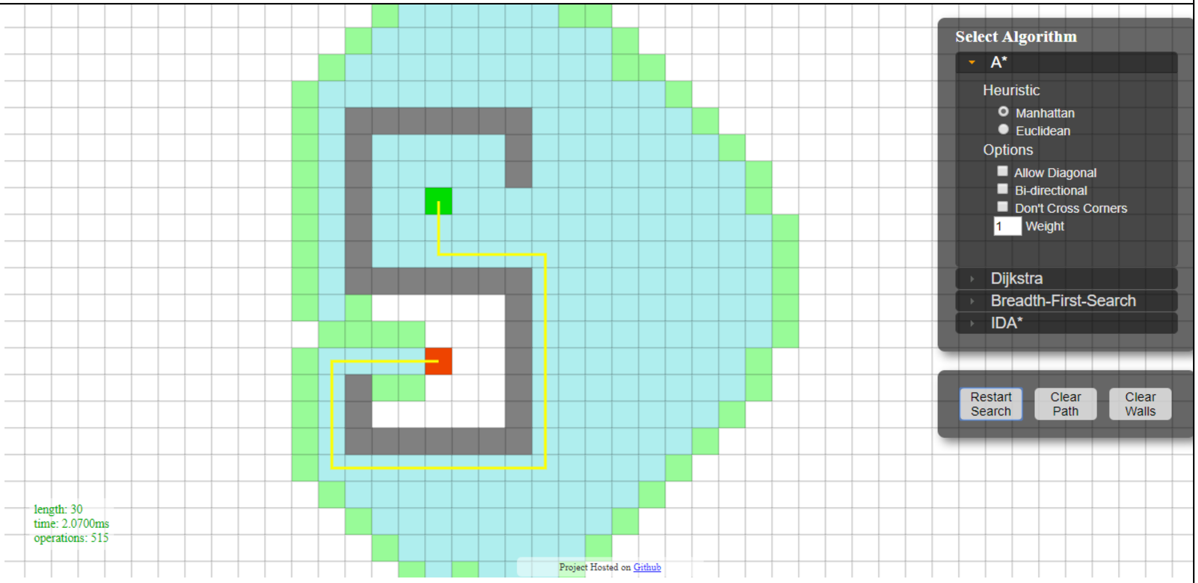
Number of operations: 36



4. Surrounded by walls, S-Shaped Walls

A* Algorithm

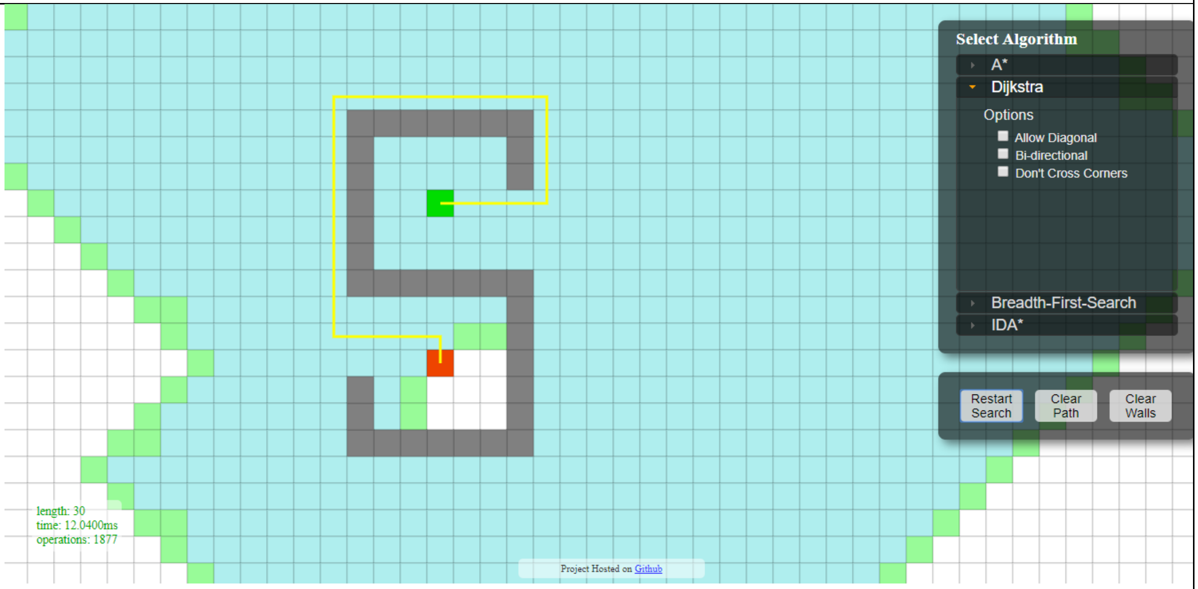
Number of operations: 515, Length: 30



```
length: 30
time: 2.0700ms
operations: 515
```

Dijkstra's Algorithm

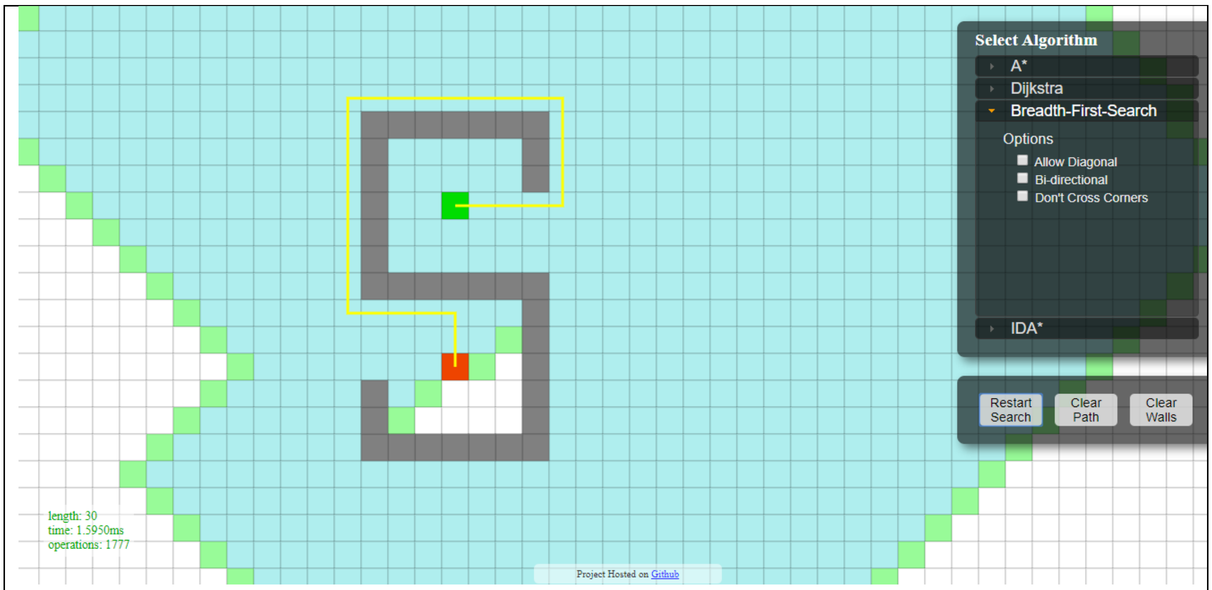
Number of operations: 1877, Length: 30



```
length: 30
time: 12.0400ms
operations: 1877
```

Breadth-First Search Algorithm

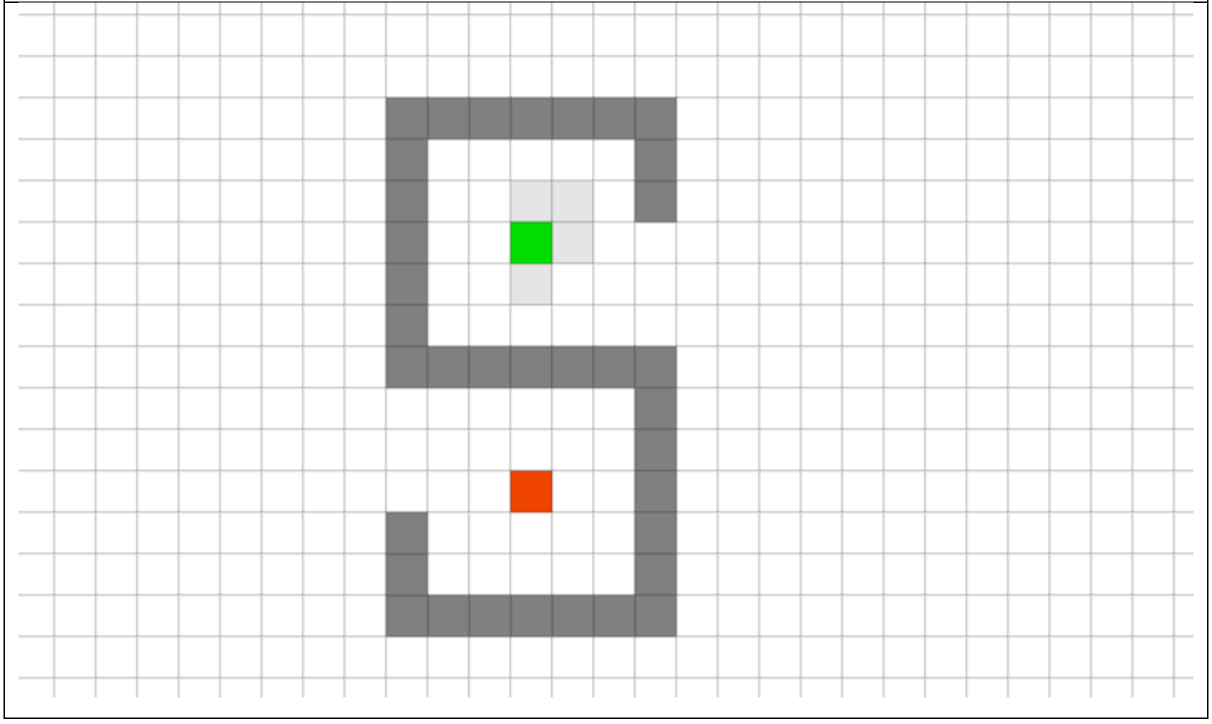
Number of operations: 1777, Length: 30



length: 30
time: 1.5950ms
operations: 1777

IDA* Algorithm

Number of operations: Failed, Went to a permanent recursion

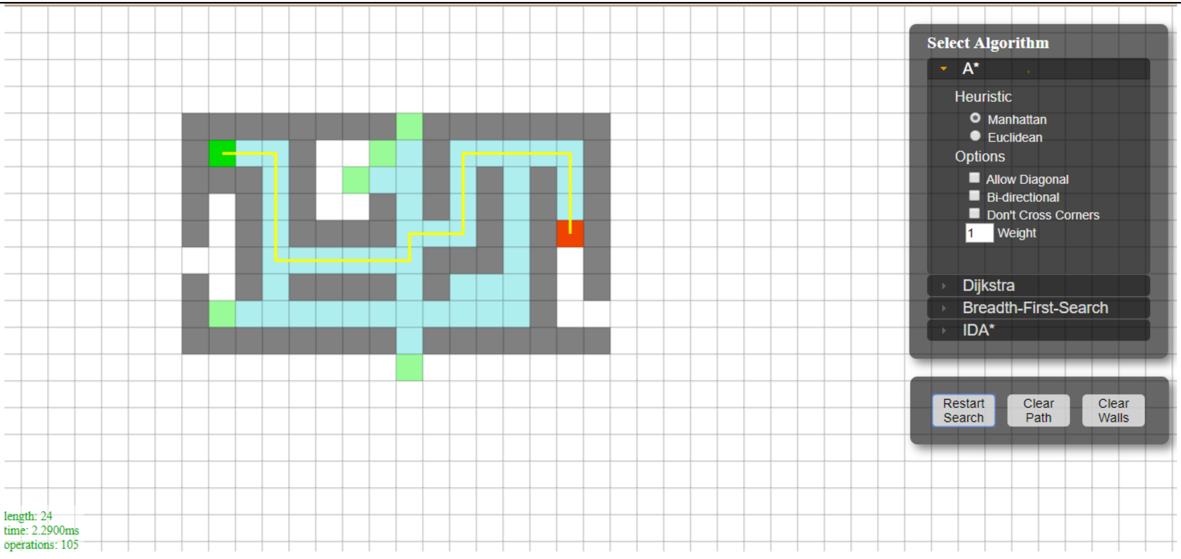


The IDA* algorithm failed to provide a suitable path and went inside a never-ending loop. This shows how IDA*'s main problem of running through the same path could affect in implementation.

5. Random Path Mimicking Real-World Roads

A* Algorithm

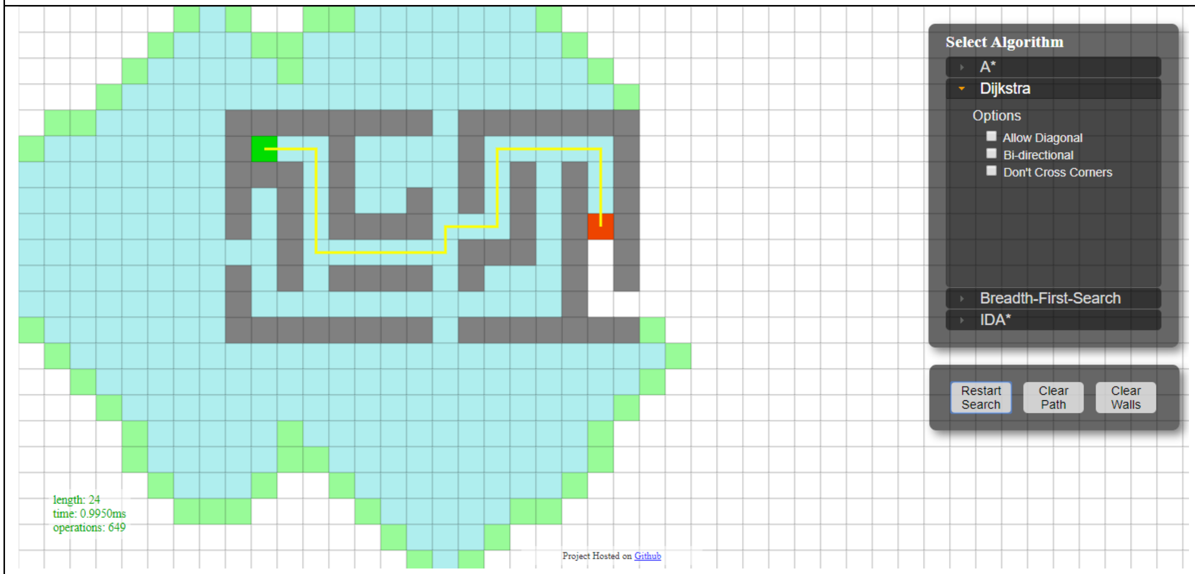
Number of operations: 105, Length: 24



length: 24
time: 2.2900ms
operations: 105

Dijkstra’s Algorithm

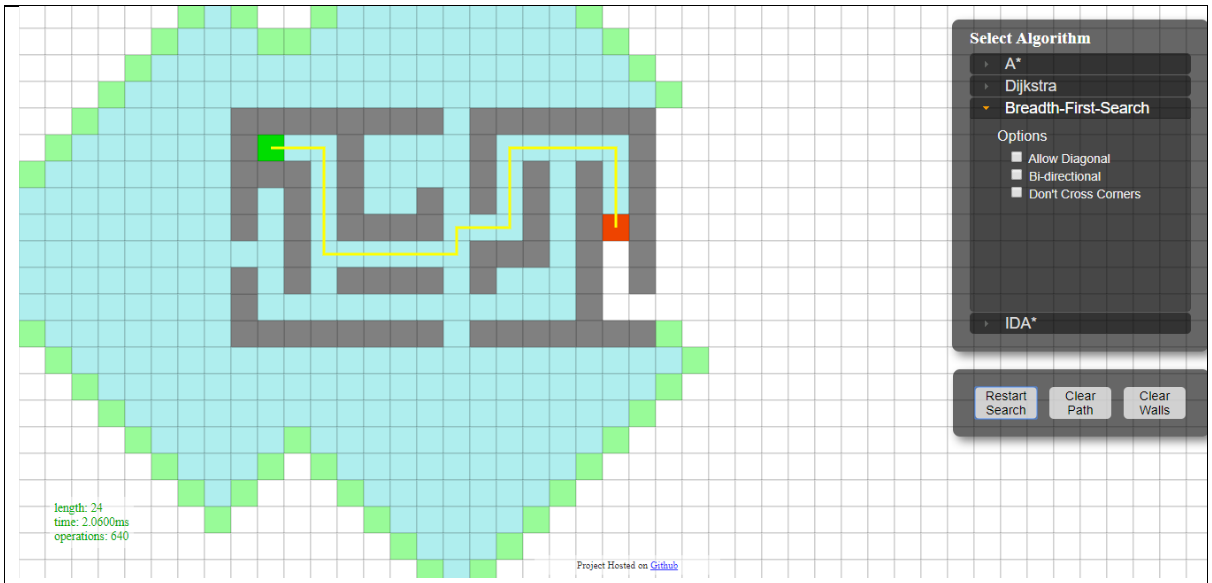
Number of operations: 649, Length: 24



length: 24
time: 0.9950ms
operations: 649

Breadth-First Search Algorithm

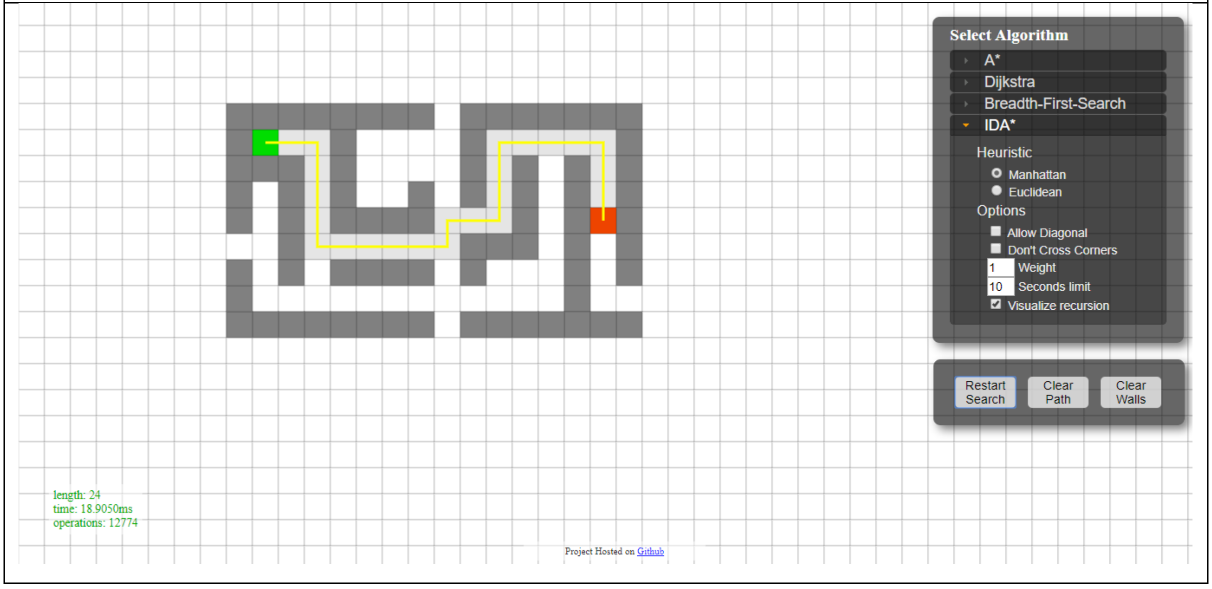
Number of operations: 640, Length: 24



length: 24
time: 2.0600ms
operations: 640

IDA* Algorithm

Number of operations: 12744, Went to a long recursion



Weight: 1
Seconds limit: 10
Visualize recursion: ✓

length: 24
time: 18.9050ms
operations: 12774

In this case, IDA* went through a very long recursion before being able to find the path. Although the path was optimum, it required way more operations (more than 20 times) than even Breadth-First Search and Dijkstra's Algorithm and 120 times more than A* Search. We can interpret that in such conditions, IDA* is not very effective.