Prometheus AI Integration

ECSE 499 HT 10
Vera Tang 260624313
Supervisor: Joseph Vybihal

Abstract

The objective of building up the Prometheus AI brain is to enable the brain with the capabilities of exploring and learning its surroundings by the sensor data input from its controlling robots, therefore it is able to control these robots to complete their designated missions by making right decision based on sufficient knowledge of the environment. This system have a great potential on future application, since it is a system that is capable of working in hazardous environments, such as the outer space exploration and search and rescue after disasters happens. The system is composed by 4 layers: the Neural Networks (NN), the Knowledge Node Networks (KNN), the Expert System (ES) and the Meta Reasoned. The NN processes the raw data collected from each sensor and classifies this information into a set of Tags, which will be the input of KNN. The KNN will search through its network for relevant information and the ES will give out a recommendation based on all this information. The Meta is the only layer that has accessed to all other layer and has goal list as well as exploration map inside, which will determine whether to take this recommendation or not. The two layers, KNN and ES, have been well established by the former scholars, whereas this thesis will be focused on integrating all 4 together to perform and test map exploration and object avoidance in Java.

Acknowledgement

The design project team (Jiachen Ju, Li Jian, and Zisheng Wang) who are working on the fourth layer META Reasoner last semester, have given me a lot of advices about their works.

Table of Contents

List of Figures

List of Abbreviation

NN	NeuralNetwork
KN	Knowledge Node
KNN	Knowledge Node Network
ES	Expert System
META	Meta Reasoner

1.Introduction

The objective of this project is to build up an artificial intelligence system called Prometheus, which is capable of controlling a set of robots to accomplish a predetermined goal. This system takes the data from sensors of the robots as input and controls the motors of them. It works as the brain of robots; for example, it can explore and learn from its surrounding then response accordingly. This system can be applied to several of fields, especially for sending robots to work on dangerous situation, such as outer space exploration, search and rescue people after earthquake and so on.

The architecture of the system is shown in figure 1; It is composed of four layers: Neural Network, Knowledge Network, Expert System and Meta reasoner[1]. a detailed explanation of the existed functions of each layer individually will be described in the section 2 background. This thesis is mainly focus on integrating and testing all 4 existed layers in a simulated environment. The detailed requirements of building and integration will be described in Section 3. The implementation and progress achieved this year will be discussed in Section 4. Finally, all the possible impact on society and the environment conducted by this thesis will be described in section 5.

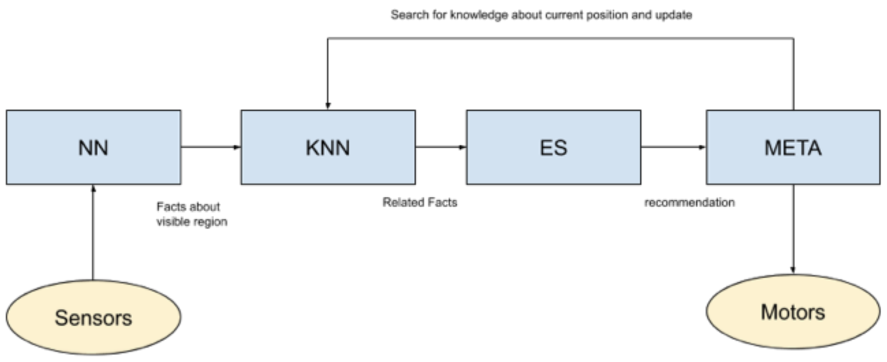
Search for knowledge about current position and update
NN
KNN
ES
META
Facts about visible region
Related Facts
recommendation
Sensors
Motors

Figure 1. Architecure and Flow Chart of the Prometheus

2. Background

In this section, the basic idea of NN will be given and further implementation and design methods will be discussed in sections 4. The other layers' functions and basic structure will be described in this section; the integration plan of all four layers will be shown in the section 5.

2.0 Tags

Tag is an important component to the system, because it is the data structure to be passed between second and third layer. The final data structure passing between all four layer is designed based on tags. The relationship between each class in tag package is described in figure 2. The tag class is the parent class of predicate, rule and reading. The rule is a relationship among facts and facts or facts and recommendation, thus the ES will think and output recommendation based on it.

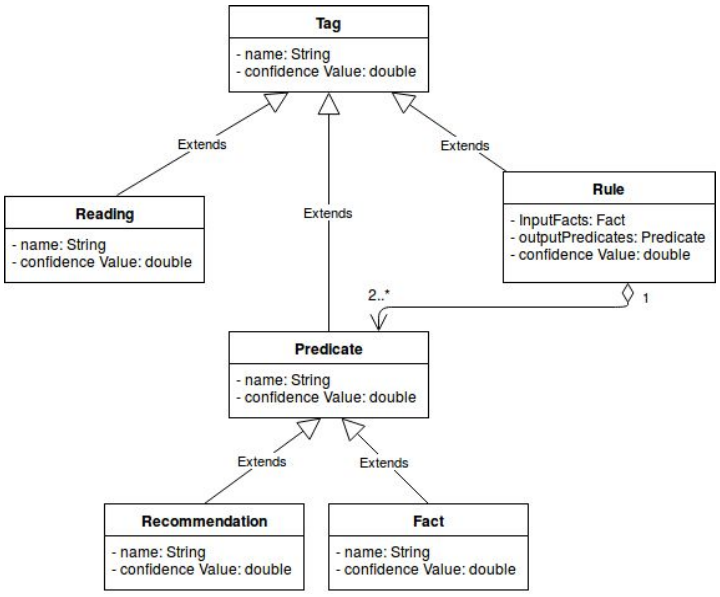
Tag
- name: String
- confidence Value: double
Extends
Extends
Extends
Reading
- name: String
- confidence Value: double
Rule
- InputFacts: Fact
- outputPredicates: Predicate
- confidence Value: double
2..*
1
Predicate
- name: String
- confidence Value: double
Extends
Extends
Recommendation
- name: String
- confidence Value: double
Fact
- name: String
- confidence Value: double

Figure 2. UML Class Diagram of Package Tags

2.1 Neural Network

The Neural Network layer consists of array based initial weights for training the robot. Sensors object are used to provide raw data to the NN and the output will be the prediction of types of object in front of the robot in its vision. For now NN is only dependent on ultrasonic sensors to distinguish object between immovable and movable object. It might require further development for the brain to do more sophisticated thinking process.

2.2 Knoledge Node Network

The design of KNN is to potentially shadow people's memory [2]. In the KNN, each predicate tag serves as a knowledge node inside the network, just as a neuron in people's brain. Once the input tag received, it will fires the predicate tag that connected to it. The KNN need to search through the network to collect all active tags. Active tag means that the information is considered to be true. For example, when the tag “large dog” is input, the tags “is animal” might be excited. In order to search for relevant tags, there are 4 different type of search implemented in KNN as shows in figure 3 [3]. The direct search only outputs the immediate next tag to the input. Forward search is to search through all the nodes that connected to input tag along the forward direction. Whereas backward search is more likely to be used when the researcher is trying to abstract from the facts received. There will be less tag output then input because people have to sum up a tag that are relevant to all of input tags. The lamda search is first to search backward and then search forward so that all kind of relevant information will be gathered as the output to the ES Layer. All forms of searching can continue until there are no more nodes can be set to active. The searching can also be set to be fix number of cycles, which is faster and more similar to how humans think.

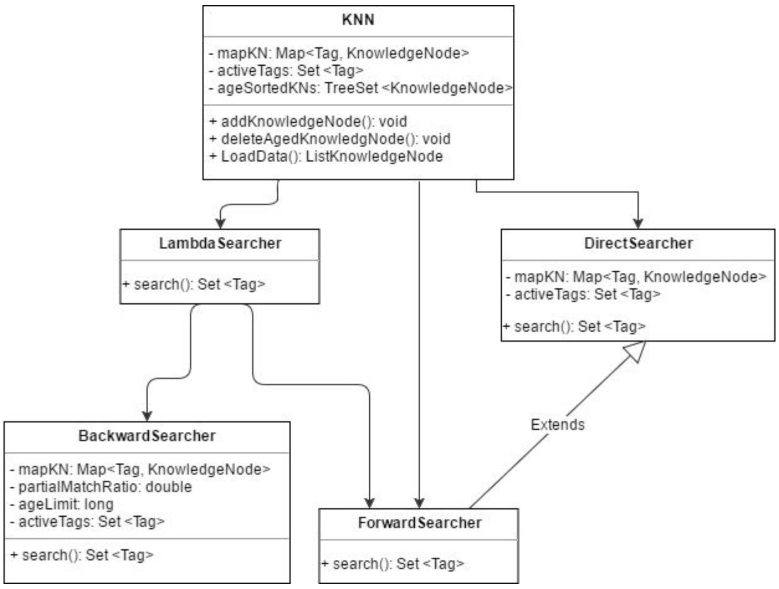
KNN
- mapKN: Map<Tag, KnowledgeNode>
- activeTags: Set <Tag>
- ageSortedKNs: TreeSet <KnowledgeNode>
+ addKnowledgeNode(): void
+ deleteAgedKnowledgNode(): void
+ LoadData(): ListKnowledgeNode
LambdaSearcher
+ search(): Set <Tag>
DirectSearcher
- mapKN: Map<Tag, KnowledgeNode>
- activeTags: Set <Tag>
+ search(): Set <Tag>
Extends
BackwardSearcher
- mapKN: Map<Tag, KnowledgeNode>
- partialMatchRatio: double
- ageLimit: long
- activeTags: Set <Tag>
+ search(): Set <Tag>
ForwardSearcher
+ search(): Set <Tag>

Figure 3. UML Diagram of KNN

For each node, there is an age associate to it, which will be incremented every time. There is a sorting tree in KNN so that the aged knowledge node will be removed to keep enough speed for searching and space for adding new nodes, which has same working mechanism as human brain [4]. For each pair of node, there is a strength associate with this relationship. The strength denoted how fast this node is been fired.

2.3 Expert System

The ES layer is a basic logic reasoner, which has no knowledge of any context, which takes in the tags provided by the KNN and interprets them as either facts, recommendations or rules [5]. The expert system will output a recommendation based on the facts received. For example, the facts tags, expert system received is “obstacle in front is wall, distance=x, angle=y”, so the recommendation based on these tags might be “mark it as wall” and “stop and turn to -y direction”.

The general steps for the ES to run are: Reset, Add facts and rules, Think, and Send recommendations to META. The basic class of complete each step is described in figure 4. In the reasoner, it will clear all rules and facts obtained from last run using the rester. The class teacher will be used to translate human language such as “if .. then” to a rule tag and then add facts and rules to ES. When all data are ready, the thinker will perform the logic reasoning and output the recommendation to META.

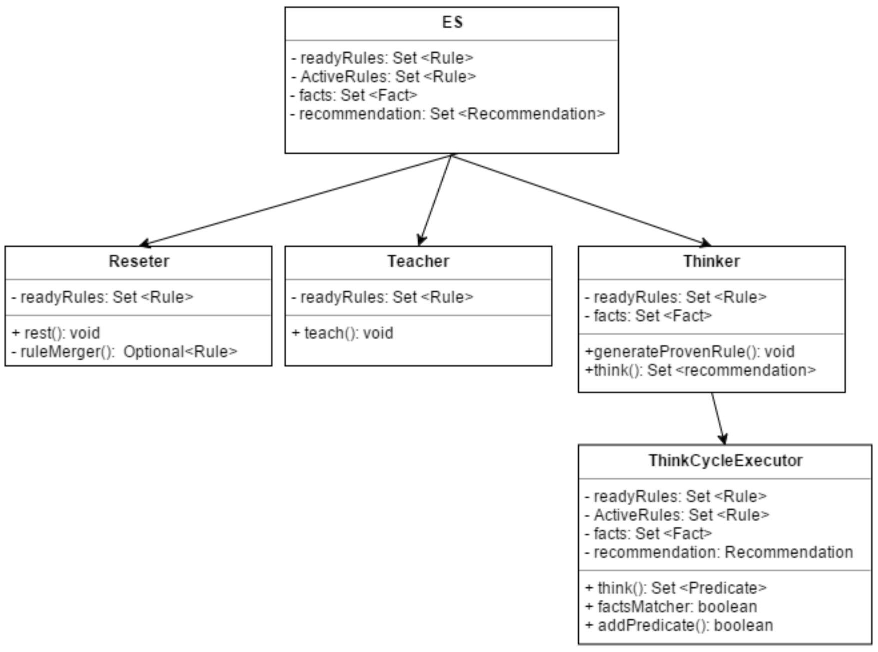

Figure 4. UML Diagram of ES

2.4 Meta Resoner

The Meta reasoned layer is the layer make final decision based on the goal list store in memory and the initial map received from simulator, which shows in figure 5. Throughout the process of exploration, META will utilize the data from former layers, use the method scan () to decide the direction of movement while constructing a tree map that describe more details about surroundings. The tree map is a graph that formed by connecting waypoints which is the node object in figure 5. New node is added every time the robots changes moving direction or detecting objects and then node will be stored into the tree map. The Meta is interfacing with the simulator using simulator controller to control the virtual robot.

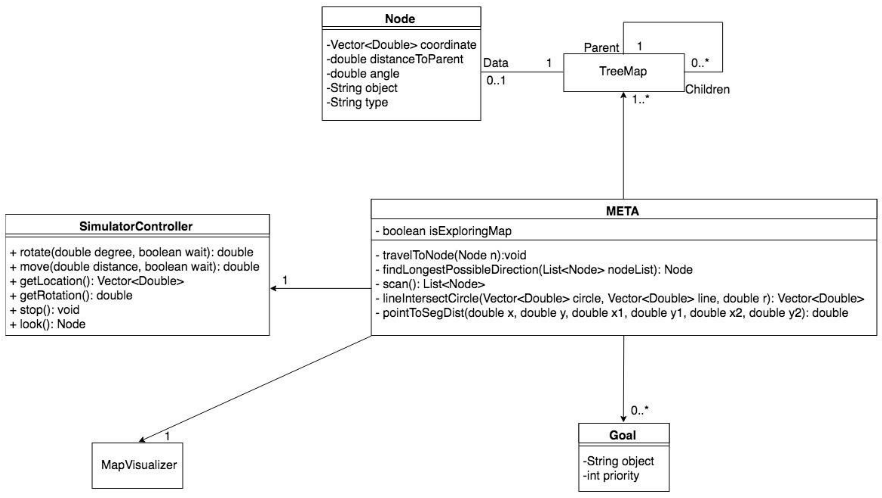
Node
-Vector<Double> coordinate
-double distanceToParent
-double angle
-String object
-String type
Data
0..1
1
TreeMap
Parent
1
0..*
Children
1..*
META
- boolean isExploringMap
- travelToNode(Node n):void
- findLongestPossibleDirection(List<Node> nodeList): Node
- scan(): List<Node>
- lineIntersectCircle(Vector<Double> circle, Vector<Double> line, double r): Vector<Double>
- pointToSegDist(double x, double y, double x1, double y1, double x2, double y2): double
SimulatorController
+ rotate(double degree, boolean wait): double
+ move(double distance, boolean wait): double
+ getLocation(): Vector<Double>
+ getRotation(): double
+ stop(): void
+ look(): Node
1
MapVisualizer
1
0..*
Goal
-String object
-int priority

Figure 5. UML Diagram of Meta

3. Problem

The main task for this thesis is to integrate existing layers into one software system, so that the robot can explore and learn from the environment while avoiding serious collision with immovable object, wait for movable object to move away and save the movable object locations in KNN. By exploring the environment, the integrated AI brain is able to construct a global map consisted with abstract information in Meta from local map, which consist of detailed information and are saved in the KNN for future use. The AI brain need to perform the functionality of each layer, which has been described in section 2. It also need to be integrated with the a newly created environment simulator for testing and visualizing the local and global map constructed by AI brain. Therefore, it is also important to communicate with other undergraduate student who have previously worked on all layers to ensure all the codes are fast, object oriented, heavily tested and easy to integrate.

4.Implementation

As the problem described in chapter 3, all tasks of finishing this project can be break down into 3 main parts, which are integrating four layers, adding functionality in META and KNN for global and local map construction and visualization, and testing with simulated environments.

4.1 Data Sturcture Interfacing

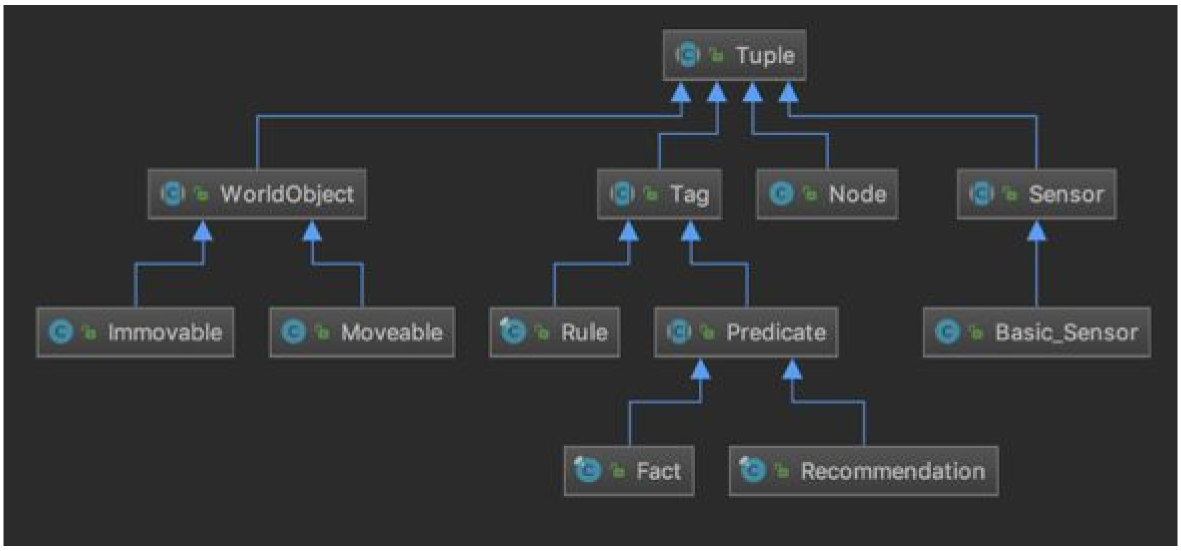

Figure 6. UML Class Diagram of Tuple

For each existed layer developed by 3 groups worked on it, each of the group used different data structure based on their needs. In order to allow information passed by each layer and to be easily understood and utilized. The abstract class Tuple is created and inheriting by each data structure as the figure 6 described.

The data structure Tuple is composed of three attributes: Label in String, integer parameters in int array and its according description in String[]. Each actual object inheriting Tuple everytime when instantiate will set up Tuple's attributes in its constructor by default. Therefore, their information can be easily translated and the Tuple object is used when passing around different layers conveniently.

In order to encapsulate the actual implementation of thinking process and data structure interfacing of each layer, the interfaces are used and will be inside of the api package for clients to look at as figure 7 uml class diagram of prometheus package described.

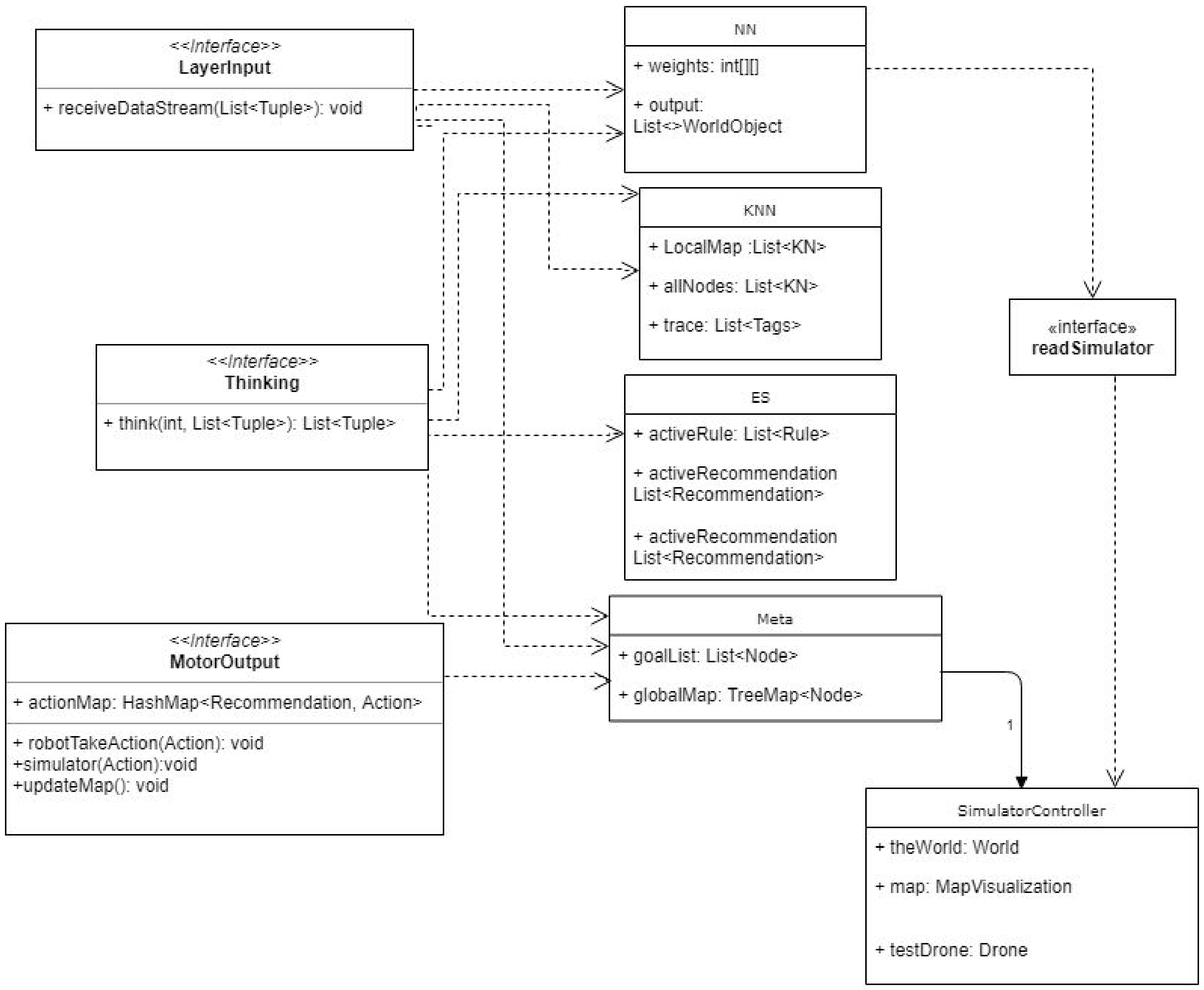

Figure 7. UML diagram of prometheus package

4.2 LocalMap and GlobalMap

Global Map is a treeNode map as background explained and it only consists of the Waypoint information. Waypoint is the center of a local map, which has a size predefined and a new waypoint is only created when the robot has leaved all existed local maps covered region. This design is to mimic the memory system of human being that we only remember very abstract information in our surface memory.

The Local Map is constructed while exploring but it will not existed in Meta but saved in KNN as a deeper memory. While if needed by Meta, it can be retrieved and updated when visited again. Therefore the localMap is in a string format that can be both stored as tree and KN. The local maps have overlapped region, because the new waypoint is set to be current position of robot when the robot left covered region. This design is based on the drone should have any knowledge about the size of the environment and a waypoint should be a place that is not occupied by any immovable object.

Since the local map is overwritten when revisited, some important information such as the Waypoint and location movable object can be found. Therefore those information is also store as KN in order to be retrieved by Meta for future use such as finding the movable object.

The visualization of the global map are the tree nodes drawed when visited and it not only draws out the waypoint as well as all nodes that attached to the waypoint in the localmap, and the local map is draws out as a n*n grid as the simulated world looks like. All the visualizations are further described and shown in figure 9 and 10 in 4.4 testing result.

4.3 Integration Structures

The basic idea of how the four layer communicate with each other is described in 4.1 and the detailed integration class diagram can be found in figure 8. The prometheus class will has the 4 layers as attributes and each layer is capable of thinking in fixed iterations and by take the output of last layer as input using think() and each layer uses receivedatastream() to do necessary pre-process of data.

Just as the sequence diagram in figure shown, the setup of all the layers are finished sequentially and after setting up the initial weights ,pre-loaded knn and environments of simulator, the AI brain starts to think. The thinking process of four layers is also sequentially. Firstly the NN read data from simulator and think() then send KNN the predicted object in robot's vision. KNN use the information of possible obstacles to search for relative facts and rules about the current situation. For example when input is three immovable knn will find the rule that all immovable -> turn right and all immovable-> turn left to successfully avoid collision. The ES will get all recommendations and select one to Meta. The META is the decision maker and map builder that has access to goalList from client input. In addition, from all four layer, Meta has the highest power to request for data from KNN as well as update KNN about localMap information and visualization. After Meta takse action, it updates the simulator as well as the global map for visualization.

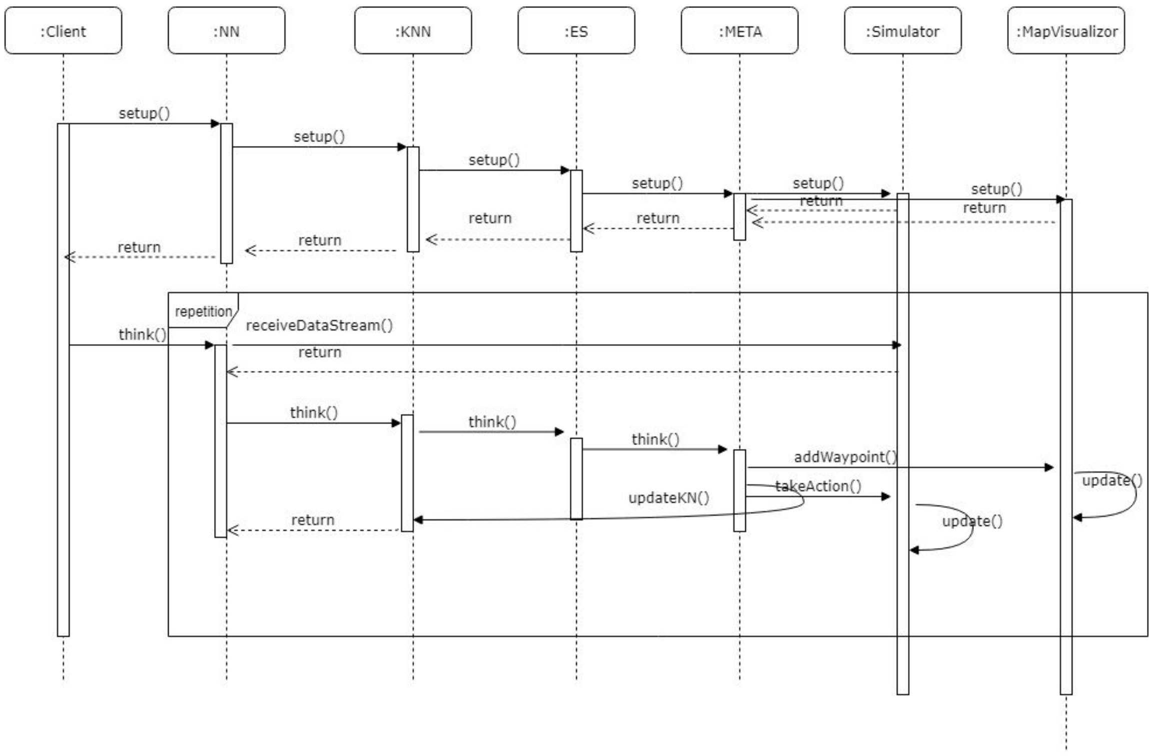

Figure 8. Integrated UML Sequence Diagram of Prometheus AI

4.4 Testings and Results

In order to test the functionality of the AI brain, an simulated square world is created. It is assumed to be an world in a grid sense and Immovable and movable along a fixed path can be added according to different requirement. In the final experiment we performed the world is generated as figure 17 shown. Note: The grey area containing character I is immovable objects. The Movable object is denoted as M inside grid and move along the fixed path noted by red arrows. The char "^" inside grid is the robot controlled by the AI brain and its only visible region are marked in green.

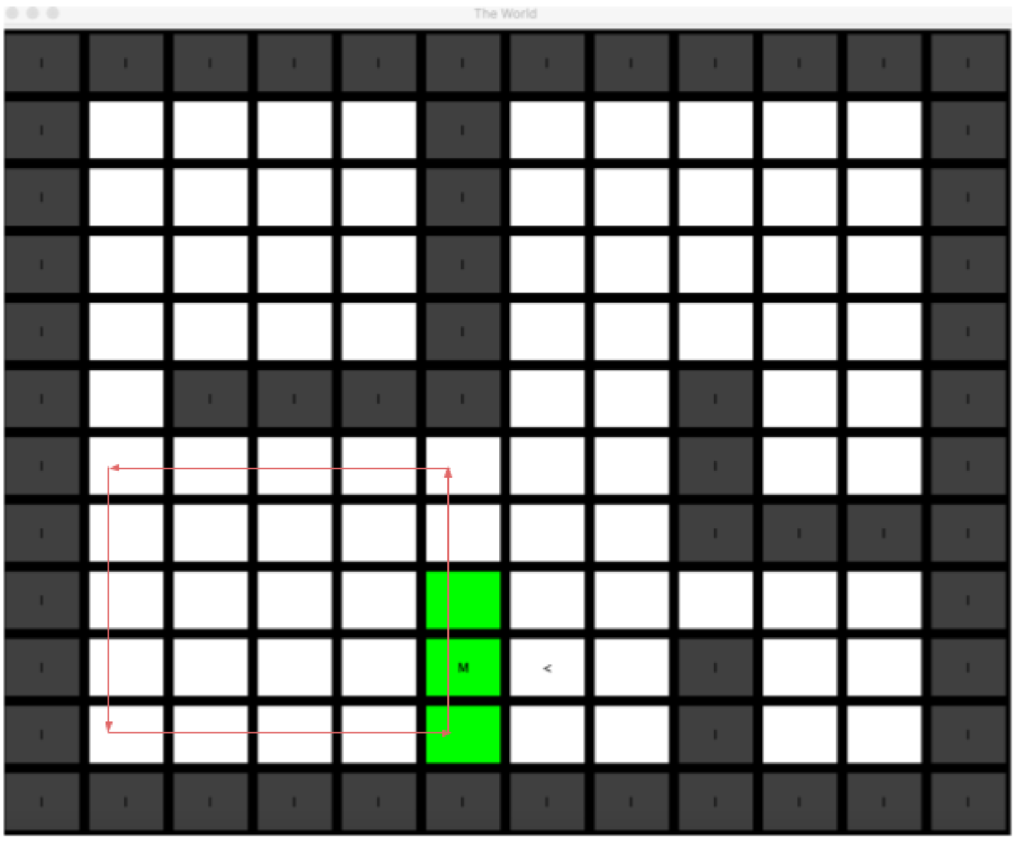

Figure 9 Simulated environemnts

For this experiment the setup of KNN can be found in appendix. The robot is expected to turn to left or right when all visible region is blocked, otherwise goes to any green region, where obstacle does not present. It is also expected to wait for the movable to move away. While the robot roaming through the environment, it is going to build up a new 7*7 grid local map every times it enters area that existed local map does not covers and the center of the local map is called waypoint. Waypoints will be the only object saved into Meta's memory of global map as a treeMap of node and all other detailed information in local map is saved into KNN. While continuously exploring the simulated world, the robot will update its local map in KNN and visualization if anything has changed in that area. The visualization of global map will show current information as local map described in figure 18 left and the important information which should not be overwritten even when environment changed for example the trace of the movable object will also be saved in KNN and the visualization of them are in another window as figure 18 right. All information saved in KNN can be found in appendix.

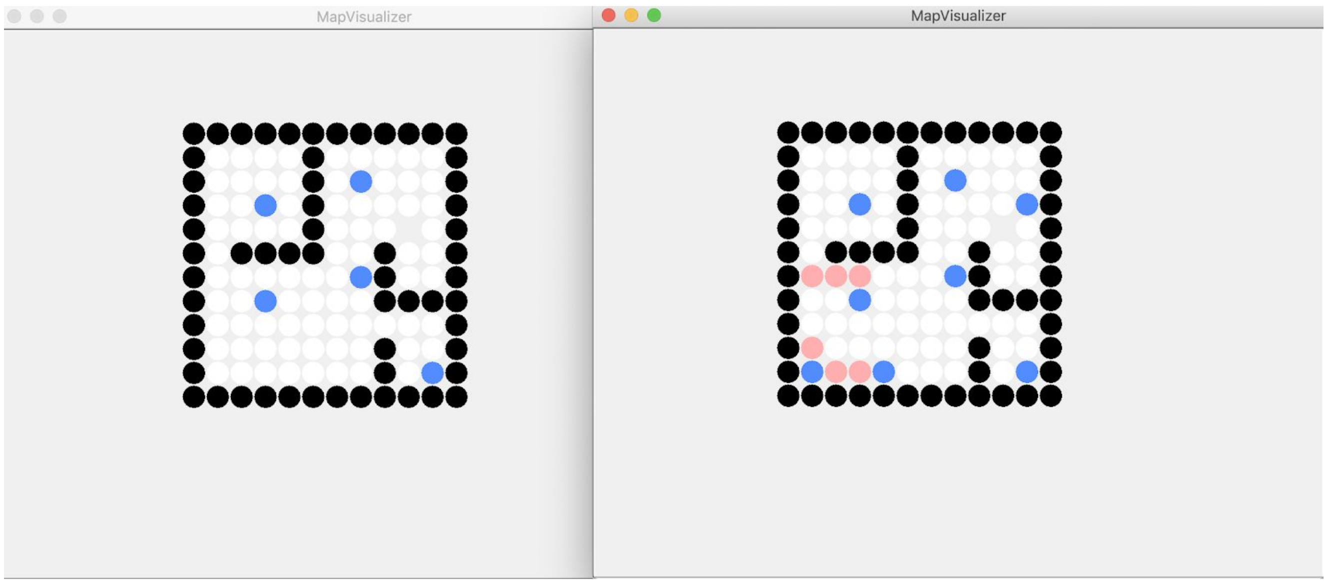

Figure 10. Global Map explored after 300 iterations

As the figure 18 shown, there are 7 waypoints painted in blue, immovable objects in black and movable object in pink. According the experiment result, there are 7 different local map constructed as figure 19 shown. Inside of the local map, the waypoint is painted as blue and immovable region are painted in dark gray and character "I" is denoted inside the grid. For movable object, it is denoted in pink region with character "M". White background and character "N" is the non-obstacle region and for unvisited region it is leaved blank without border.

It is worth to note that local maps have overlapped region, because the robot is not aware of the boundary of the world while exploring and it can only see the three grid in front of itself. Therefore, it creates a waypoint and its localmap every time it enters the region not covered by existed local map. Moreover, for the overlapped region, both local map covered that region will be stored. It is possible for the future student working on this project to merge overlapped region in localMap for saving memory in KNN.

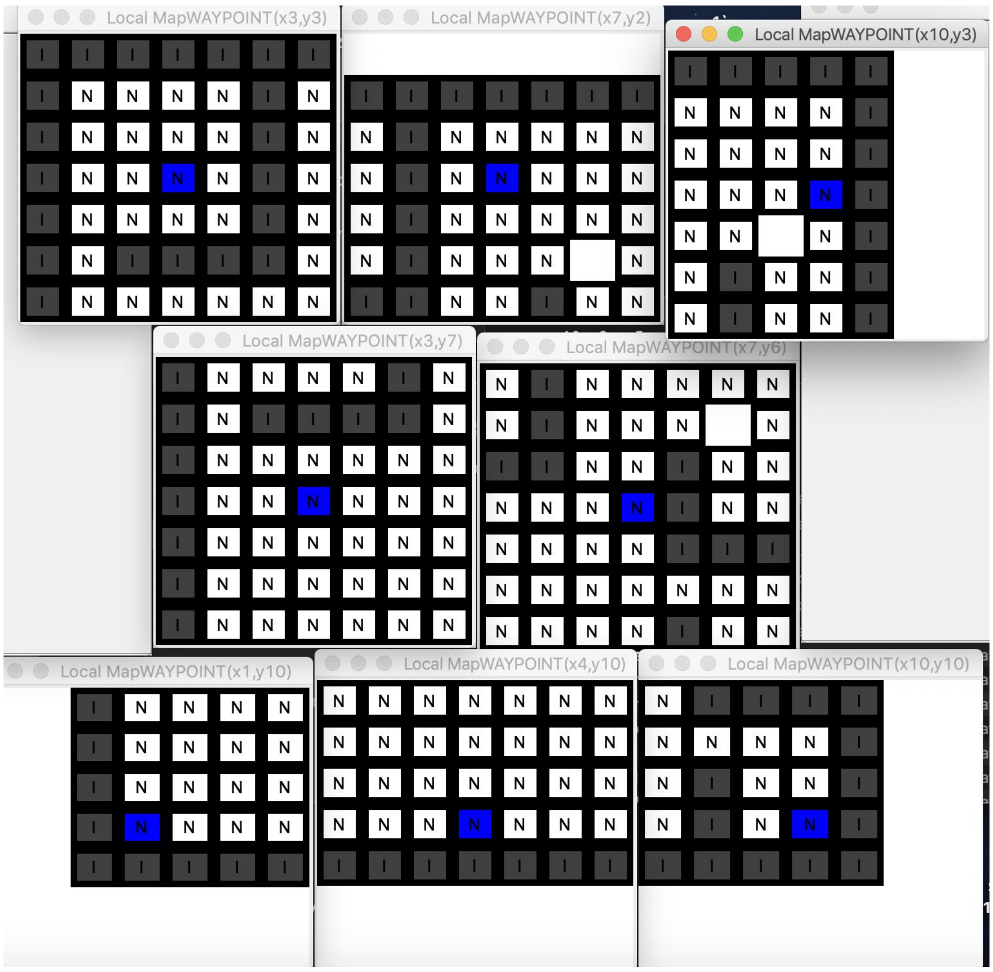
Local MapWAYPOINT(x3,y3)
Local MapWAYPOINT(x7,y2)
Local MapWAYPOINT(x10,y3)
Local MapWAYPOINT(x3,y7)
Local MapWAYPOINT(x7,y6)
Local MapWAYPOINT(x1,y10)
Local MapWAYPOINT(x4,y10)
Local MapWAYPOINT(x10,y10)

Figure 11. Local Map updated after 300 iterations.

Overall, the test is very successful, the robot behaves as expected, it always successfully avoiding collision to Immovable object and explored the map as well as providing information enough for tracking down the movable object.

5. Future Works

Although the system is successfully performed its exploration tasks, there still are many additional functions can be added to the AI brain to perform more complicated mission in more complex environments.

Firstly, the NN need to be further developed to analyze or recognize the object in its visible region. For example, it is noted that during the experiment that there is a possibility for movable object and robot to collide when they are both out of each other’s sight. Therefore, it is bette the touch sensor can be considered in NN and therefore KNN and ES can give recommendation based on that.

In addition, if the NN is able to recognize object, KNN will need to load more data before running the AI so memory management of KNN should be taken care of.

There are some untested additional function of META including management of Goal List and route-planning can be finished and tested since exploration mission is successfully completed. This may help to complete mission more efficiently.

6.Impact on Society and Environments

6.1 Use of Non-renewable Resources

As a project mainly depends on software development, there are limited physical materials will be used for constructing this system. The only related physical materials might need to take in consideration is the power source of the robot. The design of the physical robot should take in consideration of using non-renewable resources such as using solar panel as battery.

6.2 Environmental Benefits

AI system like Prometheus can be implemented as a tool to protect environments. For example it can be used to control the robots for wildlife monitoring and protection. Indeed such implementation has been achieved by in their 2016's research. They use an autonomous Unmanned Aerial Vehicle (UAV) to localize a tagged animal and taking pictures to document the interaction between the bear and environment[6].

6.3 Benefits to Society

This type of system could be implemented in wide range of field to benefits the society. For instance, the system could be used to send and control robots in hazardous environment for humans. Just as the tests performed in this project, the robot is able to find the trace of movable objects and record all possible location in their memory. For example, robots are often used in the aftermath of disasters to prevent in order to protect or potentially rescue lives. For example, in the nuclear disaster of Fukushima, if robots can be employed the use of robots, it will have faster actions and avoid losses of human life. With system like prometheus, robots could also be used for outer space exploration such as Mars explorer Curiosity. The Mars Curiosity rover recently have developed its own AI system for image processing and neural network to identify types of rocks. This system could potentially help with cutting edge discoveries in outer space exploration.

7.Conclusion

The prototypes of the integrated Prometheus AI model consisting NN, KNN, ES and meta were completed in Java based on personal design criteria and feedback from students working on other layer and supervisor. Some additional functions required for complete the mission of exploration is also added to each layer. The design of the prototypes has required a great deal of thought and planning to ensure efficiency and proper behaviour. These prototypes were tested using the generated environments with positive results but more test cases might needed.Possible application and impact on the environment and society were also discussed. Some possible future work including developing NN and META is also discussed in chapter 5.

Reference

[1] J. Vybihal, “Full AI model,” 2016.

[2] J. Vybihal, “Knowledge nodes,” 2017.

[3] M. P. Mattson and T. Magnus, “Ageing and neuronal vulnerability,” Nature Reviews Neuroscience, vol. 7, no. 4, pp. 278–294, 2006.

[4] J. Vybihal and T. R. Shultz, “Search in analogical reasoning,” 1990

[5] J. Vybihal, “Expert systems,” 2016.

[6]Bayram, H., Doddapaneni, K., Stefas, N., Isler, V., & 2016 IEEE International Conference on Automation Science and Engineering (CASE). (August 01, 2016). Active localization of VHF collared animals with aerial robots. 934-939.

Appendix

knn data after the experiments:

KnowledgeNode[inputTag=WAYPOINT(x3,y3),outputTags=[Nothing(x1,y6), Nothing(x1,y5), Nothing(x1,y4), Nothing(x1,y3), Nothing(x5,y6), Immovable(x5,y4), Immovable(x1,y0), Nothing(x1,y2), Immovable(x5,y5), Nothing(x1,y1), Immovable(x5,y2), Immovable(x5,y3), Immovable(x5,y0), Immovable(x5,y1), Immovable(x0,y5), Immovable(x0,y6), Immovable(x0,y3), Immovable(x0,y4), Immovable(x0,y1), Immovable(x4,y5), Nothing(x4,y6), Immovable(x0,y2), Nothing(x4,y4), Immovable(x0,y0), Nothing(x4,y3), Nothing(x4,y2), Nothing(x4,y1), Immovable(x4,y0), Nothing(x3,y6), Immovable(x3,y5), Nothing(x3,y4), Nothing(x3,y3), Nothing(x3,y2), Immovable(x3,y0), Nothing(x3,y1), Immovable(x2,y5), Nothing(x2,y6), Nothing(x2,y4), Nothing(x2,y3), Nothing(x6,y6), Nothing(x2,y2), Nothing(x6,y5), Nothing(x2,y1), Nothing(x6,y4), Immovable(x2,y0), Nothing(x6,y3), Nothing(x6,y2), Nothing(x6,y1), Immovable(x6,y0)]]

KnowledgeNode[inputTag=WAYPOINT(x7,y6),outputTags=[Nothing(x5,y9), Nothing(x5,y8), Nothing(x5,y7), Nothing(x7,y9), Nothing(x5,y6), Nothing(x7,y8), Immovable(x5,y4), Immovable(x5,y5), Nothing(x7,y7), Nothing(x9,y9), Nothing(x9,y8), Nothing(x7,y6), Immovable(x9,y7), Nothing(x7,y5), Immovable(x5,y3), Nothing(x7,y4), Nothing(x9,y6), Nothing(x7,y3), Nothing(x9,y5), Nothing(x9,y3), Immovable(x10,y7), Nothing(x4,y9), Nothing(x4,y8), Nothing(x4,y7), Immovable(x8,y9), Nothing(x6,y9), Immovable(x4,y5), Nothing(x4,y6), Nothing(x6,y8), Nothing(x6,y7), Immovable(x8,y7), Nothing(x6,y6), Nothing(x8,y8), Nothing(x4,y4), Nothing(x6,y5), Immovable(x8,y5), Nothing(x4,y3), Immovable(x8,y6), Nothing(x6,y4), Nothing(x6,y3), Nothing(x8,y4), Nothing(x10,y3), Nothing(x8,y3), Nothing(x10,y4), Nothing(x10,y9), Nothing(x10,y5), Nothing(x10,y6), Nothing(x10,y8)]]

KnowledgeNode[inputTag=WAYPOINT(x10,y10),outputTags=[Nothing(x7,y9), Nothing(x7,y8), Nothing(x7,y7), Nothing(x9,y9), Nothing(x9,y8), Immovable(x9,y7), Immovable(x11,y10), Immovable(x11,y11), Nothing(x9,y10), Immovable(x8,y10), Nothing(x7,y10), Immovable(x8,y11), Immovable(x10,y7), Immovable(x8,y9), Immovable(x8,y7), Nothing(x8,y8), Immovable(x10,y11), Nothing(x10,y10), Immovable(x11,y9), Nothing(x10,y9), Immovable(x9,y11), Immovable(x11,y7), Immovable(x7,y11), Nothing(x10,y8), Immovable(x11,y8)]],
KnowledgeNode[inputTag=WAYPOINT(x7,y2),outputTags=[Immovable(x5,y4), Immovable(x5,y5), Immovable(x5,y2), Nothing(x7,y5), Immovable(x5,y3), Nothing(x7,y4), Immovable(x5,y0), Nothing(x7,y3), Immovable(x5,y1), Nothing(x9,y5), Immovable(x7,y0), Nothing(x7,y2), Nothing(x9,y3), Nothing(x7,y1), Nothing(x9,y2), Immovable(x9,y0), Nothing(x9,y1), Immovable(x4,y5), Nothing(x4,y4), Nothing(x6,y5), Immovable(x8,y5), Nothing(x4,y3), Nothing(x6,y4), Nothing(x4,y2), Nothing(x6,y3), Nothing(x4,y1), Immovable(x10,y0), Nothing(x10,y1), Nothing(x6,y2), Nothing(x8,y4), Nothing(x10,y2), Immovable(x4,y0), Nothing(x10,y3), Nothing(x6,y1), Nothing(x8,y3), Nothing(x10,y4), Nothing(x8,y2), Immovable(x6,y0), Nothing(x8,y1), Immovable(x8,y0), Nothing(x10,y5)]],
KnowledgeNode[inputTag=WAYPOINT(x10,y3),outputTags=[Nothing(x7,y6), Nothing(x7,y5), Nothing(x7,y4), Nothing(x9,y6), Immovable(x11,y1), Nothing(x7,y3), Nothing(x9,y5), Immovable(x11,y2), Immovable(x11,y3), Immovable(x7,y0), Nothing(x7,y2), Immovable(x11,y4), Nothing(x9,y3), Nothing(x7,y1), Nothing(x9,y2), Immovable(x9,y0), Nothing(x9,y1),

Immovable(x11,y0), Immovable(x8,y5), Immovable(x8,y6), Immovable(x10,y0), Nothing(x10,y1), Nothing(x8,y4), Nothing(x10,y2), Nothing(x10,y3), Nothing(x8,y3), Nothing(x10,y4), Nothing(x8,y2), Nothing(x8,y1), Immovable(x8,y0), Nothing(x10,y5), Immovable(x11,y5), Nothing(x10,y6), Immovable(x11,y6)]],
KnowledgeNode[inputTag=WAYPOINT(x4,y10),outputTags=[Nothing(x3,y8), Nothing(x3,y7), Nothing(x5,y9), Nothing(x5,y8), Nothing(x5,y7), Nothing(x7,y9), Nothing(x7,y8), Nothing(x7,y7), Nothing(x7,y10), Nothing(x5,y10), Immovable(x6,y11), Nothing(x3,y10), Immovable(x4,y11), Nothing(x1,y10), Immovable(x2,y11), Nothing(x1,y9), Nothing(x1,y8), Nothing(x1,y7), Nothing(x3,y9), Nothing(x2,y7), Nothing(x4,y9), Nothing(x4,y8), Nothing(x4,y7), Nothing(x6,y9), Nothing(x6,y8), Nothing(x6,y7), Nothing(x6,y10), Immovable(x7,y11), Nothing(x4,y10), Immovable(x5,y11), Nothing(x2,y10), Immovable(x3,y11), Immovable(x1,y11), Nothing(x2,y9), Nothing(x2,y8)]]

KnowledgeNode[inputTag=WAYPOINT(x3,y7),outputTags=[Nothing(x1,y6), Nothing(x1,y5), Nothing(x5,y9), Nothing(x5,y8), Nothing(x1,y4), Nothing(x5,y7), Nothing(x5,y6), Immovable(x5,y4), Immovable(x5,y5), Nothing(x5,y10), Immovable(x0,y9), Nothing(x1,y10), Immovable(x0,y7), Nothing(x1,y9), Immovable(x0,y8), Immovable(x0,y5), Nothing(x1,y8), Nothing(x1,y7), Immovable(x0,y6), Nothing(x4,y9), Nothing(x4,y8), Immovable(x0,y4), Nothing(x4,y7), Immovable(x4,y5), Nothing(x4,y6), Nothing(x4,y4), Nothing(x4,y10), Nothing(x3,y8), Nothing(x3,y7), Nothing(x3,y6), Immovable(x3,y5), Nothing(x3,y4), Nothing(x3,y10), Immovable(x0,y10), Nothing(x3,y9), Nothing(x2,y7), Immovable(x2,y5), Nothing(x2,y6), Nothing(x6,y9), Nothing(x6,y8), Nothing(x2,y4), Nothing(x6,y7), Nothing(x6,y6), Nothing(x6,y5), Nothing(x6,y4), Nothing(x6,y10), Nothing(x2,y10), Nothing(x2,y9), Nothing(x2,y8)]]

KnowledgeNode[inputTag=WAYPOINT(x1,y10),outputTags=[Nothing(x3,y8), Nothing(x3,y7), Nothing(x3,y10), Immovable(x4,y11), Immovable(x0,y9), Nothing(x1,y10), Immovable(x2,y11), Immovable(x0,y7), Immovable(x0,y10), Nothing(x1,y9), Immovable(x0,y8), Immovable(x0,y11), Nothing(x1,y8), Nothing(x1,y7), Nothing(x3,y9), Nothing(x2,y7), Nothing(x4,y9), Nothing(x4,y8), Nothing(x4,y7), Nothing(x4,y10), Nothing(x2,y10), Immovable(x3,y11), Immovable(x1,y11), Nothing(x2,y9), Nothing(x2,y8)]]]

KnowledgeNode[inputTag=Trace(Movable,WAYPOINT), outputTags=[WAYPOINT(x3,y3), WAYPOINT(x7,y6), Moveable(x1,y9), WAYPOINT(x7,y2), Moveable(x2,y6), Moveable(x1,y6), Moveable(x1,y7), Moveable(x3,y6), Moveable(x3,y10), Moveable(x2,y10), WAYPOINT(x10,y10), WAYPOINT(x10,y3), WAYPOINT(x4,y10), WAYPOINT(x3,y7), WAYPOINT(x1,y10)