Prometheus AI

Phase 1

Sean Stappas
Supervised by: Prof. Joseph Vybihal

ECSE-498 Presentation

Table of Contents

Table of Contents

Swarm Robotics

Coordinating multiple robots

The aim is to create an intelligent system controlling multiple agents simultaneously working in a swarm. Possible applications include robots in hazardous environments:

1. Outer space (Moon, Mars)
2. Nuclear disaster aftermath
3. Oil pipeline inspection
4. Military zones

Table of Contents

Prometheus AI

A model of the human brain

The Prometheus AI model aims to mimic the basic structure of the human brain with four layers:

1. Neural Network
2. Knowledge Node Network
3. Expert System
4. Meta Reasoner

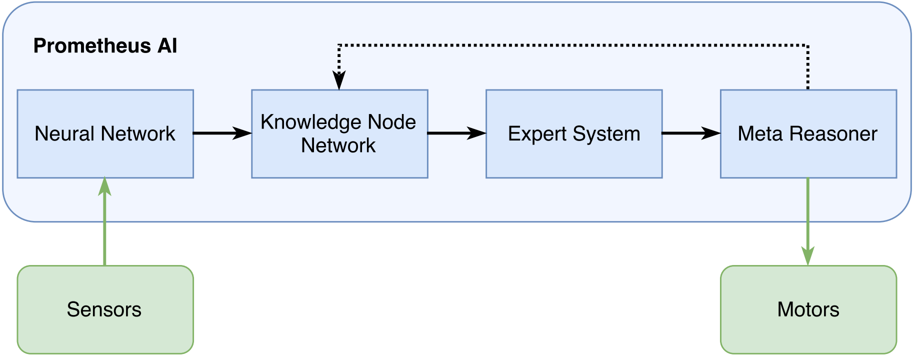

Figure 1: Full AI model.

1. Neural Network

The Neural Network (NN) is a **signal classifier** and the first interface between the sensors and the rest of the system.

input Signals from the robots' sensors (through an external API).

output Tags to be used by the Knowledge Node Network.

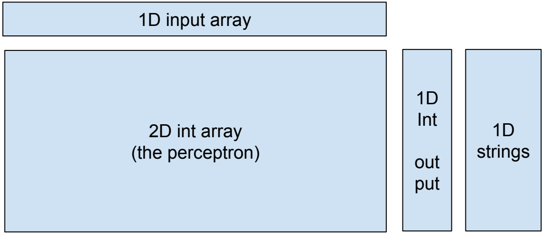

Figure 2: A perceptron[1].

[1] Joseph Vybihal. *Full AI Model*. 2016.

2. Knowledge Node Network

The Knowledge Node Network (KNN) represents **memory** with the following incomplete list of features[2]:

- Aging
- Spread of activation
- Thresholds
- Weighted connections
- Learning
- Effort
- Belief

input Tags from the Neural Network.
Commands from the Meta Reasoner.

output Tags (facts, recommendations and rules).

[2] Joseph Vybihal. *Knowledge Nodes*. 2017.

Knowledge Node

The Knowledge Node Network is centered around the concept of **Knowledge Nodes**, a one-to-many structure similar to neurons. The input tag represents information and the output tag represents information related to the input tag.

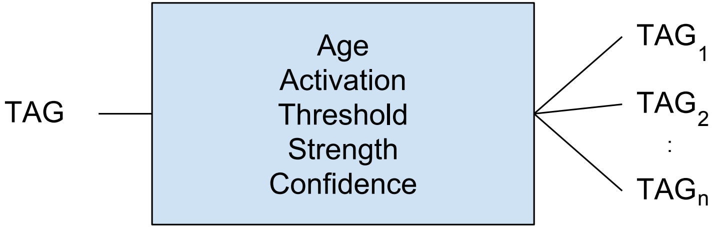

Figure 3: An abstract view of the Knowledge Node[3].

[3] Joseph Vybihal. *Full AI Model*. 2016.

3. Expert System

The Expert System (ES) is a **basic logic reasoner**, unaware of context.

input Tags from the Knowledge Node Network.

output Recommendations for actions to take (such as “turn left”).

Facts	Ready Rules	Activated Rules	Fired Tags
$(A),(B)$	$(A)(B) \to (D)$ $(D)(B) \to (E)$ $(D)(E) \to (F)$		
$(A),(B)$	$(D)(B) \to (E)$ $(D)(E) \to (F)$	$(A)(B) \to (D)$	(D)
$(A),(B),(D)$	$(D)(E) \to (F)$	$(A)(B) \to (D)$ $(D)(B) \to (E)$	(E)
$\vdots$	$\vdots$	$\vdots$	$\vdots$

Table 1: Example of cascading in a simple expert system[4]

[4] Joseph Vybihal. *Expert Systems*. 2016.

4. Meta Reasoner

High-level thinking

The Meta Reasoner (META) receives recommendations from the ES and makes an intelligent decision based on its own **paranoid** view of the world. The META is aware of context and may reject the KNN's recommendation based on how that action would affect the current reality.

This layer represents **high-level thinking** in humans.

- **input** Recommendations from the Expert System.
- **output** Commands to the robots' motors (through an external API). Commands to the Knowledge Node Network (think cycle).

Summary

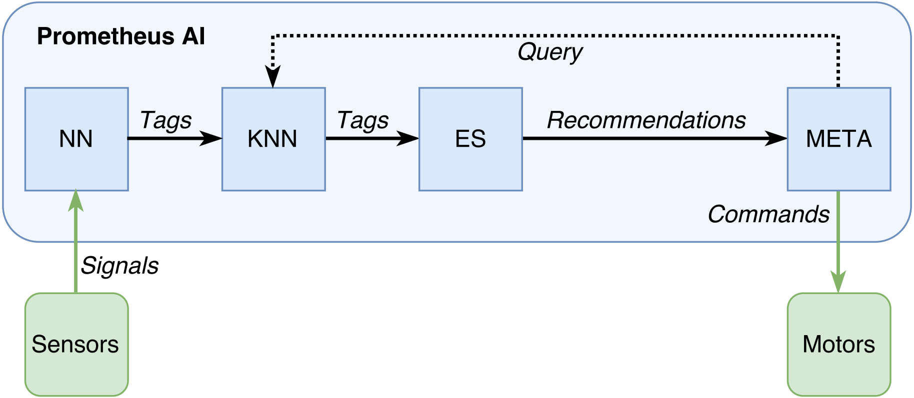

Figure 4: Summary of the Prometheus AI layers, with emphasis on the input/output of each layer.

Table of Contents

Assigned Task

The task assigned to me was to implement the **Expert System** and **Knowledge Node Network** in Java based on some preliminary specifications.

The other two layers (Neural Network and Meta Reasoner) were to be done in parallel by another student.

Table of Contents

Weekly Meetings

Weekly meetings were conducted to assess progress in the project.

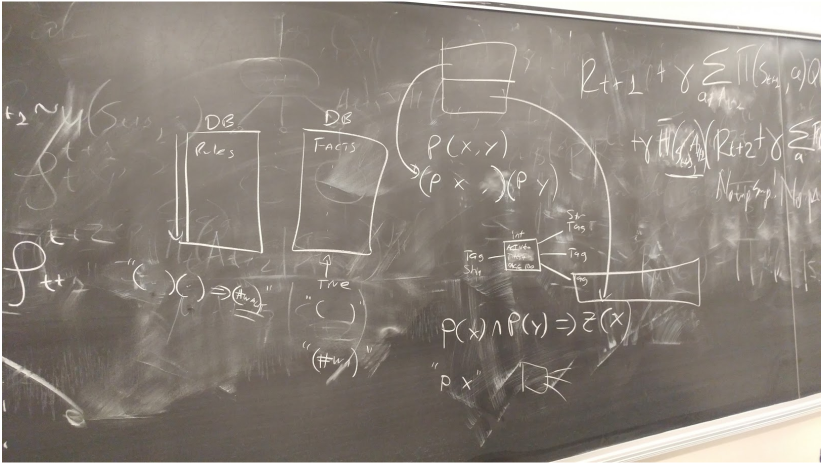

Figure 5: One of the weekly discussions on the blackboard.

Design Criteria

The design of both the Expert System and Knowledge Node Network had many criteria:

- Object-Oriented Design
  - Encapsulation
  - Abstraction
  - Inheritance
  - Polymorphism
- Efficiency (space & time)
- Documentation (Javadoc & UML)
  `http://cs.mcgill.ca/~sstapp/prometheus/index.html`

Tags

The entire system revolves around tags, which can be one of three types:

Fact Simple calculus predicate (essentially boolean expression).
e.g.: (A), $(A < 1)$, *or* $(A = 1)$.

Rule Logical combination of Tags, used by the Expert System.
e.g.: $(A)(B) \rightarrow (C)$.

Recommendation Action to be performed.
e.g.: *(#turn_left).*

To follow object-oriented design, **Fact**, **Rule** and **Recommendation** classes were created, which all subclass an abstract **Tag** class.

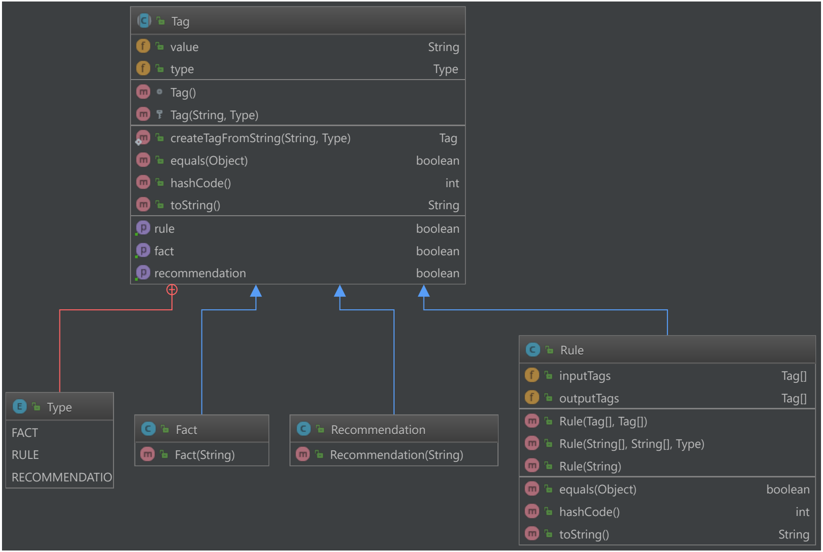

Figure 6: UML diagram of the tags package.

Table of Contents

Terminology

We will assume the following terminology for the Knowledge Node Network:

KNN Knowledge Node Network.

KN Knowledge Node.

active Describes a Tag that is seen as true by the KNN. If the Tag is the input of a Knowledge Node, it will excite that node (increasing its `activation` parameter).

fired Describes a Knowledge Node whose `activation` $\geq$ `threshold`.

Knowledge Node

The Knowledge Node has the following important fields:

`inputTag` Represents information and is used as an index.

`outputTags` Array of output Tags.

`activation` Integer starting at 0, incrementing when KN is excited. Could also be implemented with sigmoid.

`threshold` Threshold such that `activation` $\geq$ `threshold` causes firing of the KN.

`strength` Biases the activation of a KN, causing early firing. Related to learning. Simple implementation: if `activation * strength` $\geq$ `threshold` then fire the KN.

`confidence` Belief that `inputTag` is true (0 to 100%). Could be associated with each Tag.

`age` Age of the KN. If greater than some threshold, the KN is discarded.

Data Structures

The Knowledge Node Network has the following important data structures:

`mapKN` Map of input Tags to associated Knowledge Nodes

`activeTags` Set of active Tags, corresponding to input Tags of fired Knowledge Nodes

think()

Starts the activation process. The KNN has three main ways of thinking:

- `thinkForwards()`
- `thinkBackwards()`
- `thinkLambda()`

The KNN picks the correct way of thinking based on a command from META.

parameters: A **number of cycles** can be passed as a parameter to an overloaded version of `think()`. This number of cycles represents the amount of *effort* being put into thinking, running until energy-based quiescence[5] occurs. Otherwise, `think()` takes no parameters and runs to natural quiescence.

returns: The Set of Tags activated as a result of thinking.

[5] Joseph Vybihal. *Knowledge Nodes*. 2017.

thinkForwards()

Forwards thinking is the simplest method. It is based on simple forward activation, similar to the Expert System. If the input Tag of a Knowledge Node is known to be true, then the output Tags become *active* and all the Knowledge Nodes associated with those Tags are *fired*.

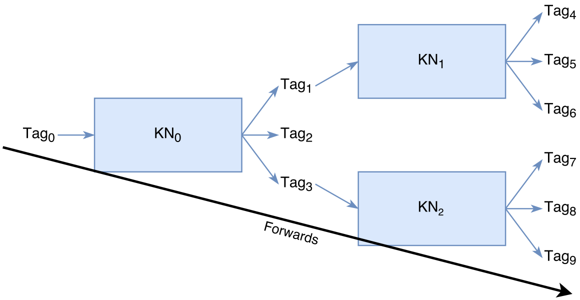

Figure 7: Thinking forwards.

thinkBackwards()

Backwards thinking looks at the output Tags of Knowledge Nodes and propagates activation backwards. This type of thinking occurs constantly in the background in humans[6].

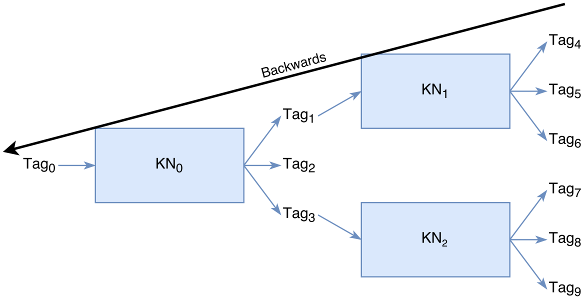

Figure 8: Thinking backwards.

[6] Joseph Vybihal. *Knowledge Nodes*. 2017.

thinkLambda()

The most complex way of thinking, representing a combination of `thinkBackwards()` and `thinkForwards()`. Commonly used in humans when using analogical reasoning to solve a problem[7].

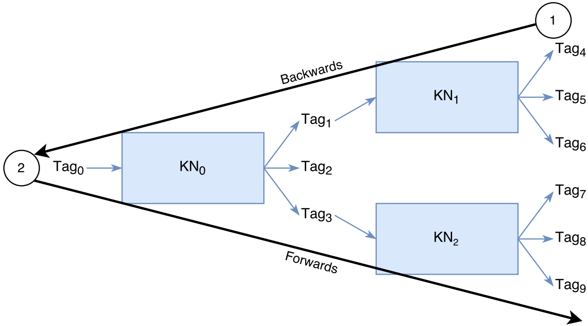

Figure 9: Lambda (Λ) thinking.

[7] Joseph Vybihal and Thomas R. Shultz. *Search in Analogical Reasoning*. 1990.

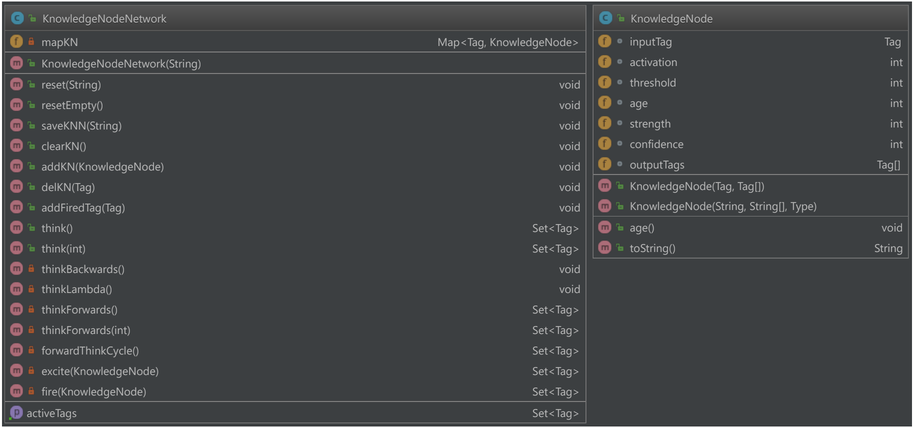

Figure 10: UML diagram of the `knn` package.

Table of Contents

Terminology

We will assume the following terminology for the Knowledge Node Network:

ES Expert System.

active Tag A Fact or Recommendation that is seen as true by the ES. If all input Tags of a Rule are active, that Rule will become active.

active Rule A Rule whose input Tags are active, making its output Tags active.

ready Rule A Rule that has not yet become active.

to activate To make a Tag or Rule active.

Rule

The Expert System is based around cascaded activation of Rules, a many-to-many structure. A Rule has the following important fields:

`inputTags` Array of input Tags representing the conditions of the Rule.

`outputTags` Array of output Tags, which become active when **all** condition Tags are active (logical AND).

$$(Tag_{in_1}) \cdots (Tag_{in_m}) \rightarrow (Tag_{out_1}) \cdots (Tag_{out_n}) \tag{1}$$

Data Structures

The Expert System has the following important data structures:

`readyRules` Set of Rules that have not been activated yet.

`activeRules` Set of active Rules.

`facts` Set of active Facts.

`recommendations` Set of active Recommendations.

think()

Starts the activation process, similarly to the KNN, with the following important differences:

- A rule is either active or not; there is no activation threshold.
- Rules are many-to-many; Knowledge Nodes are one-to-many.
- The propagation is only forwards (no backwards or lambda).

parameters: A number of cycles can be passed, similarly to the KNN. Otherwise, `think()` takes no parameters and runs to natural quiescence.

returns: The Set of **Recommendations** activated as a result of thinking. These Recommendations are passed on to the META layer.

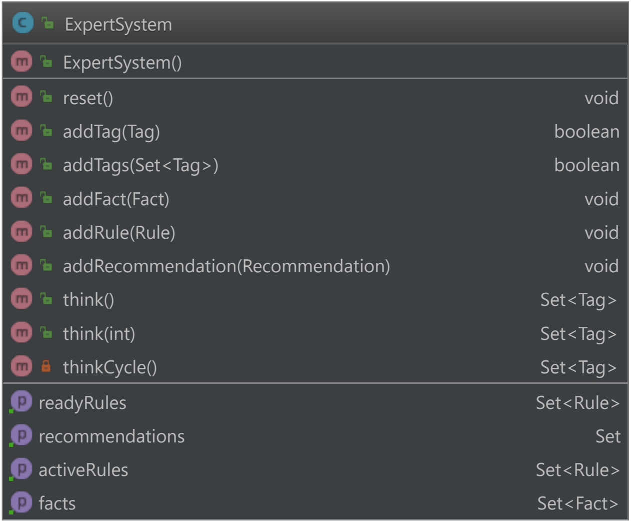

Figure 11: UML diagram of the es package.

Table of Contents

Tests

Tests were created with the TestNG framework

`TestES` Unit tests of all the Expert System methods.

`TestKNN` Unit tests of the Knowledge Node Network methods.

`TestIntegration` High-level tests of the ES and KNN (Demo).

- `testKNN()`
- `testES()`
- `testESandKNN()`

testES()

Ready Rules	Active Rules	Active Facts	Active Recommendations
$(A)(B) \to (D)$ $(D)(B) \to (E)$ $(D)(E) \to (F)$ $(G)(A) \to (H)$ $(E)(F) \to (\#Z)$		$(A), (B)$	$(\#X), (\#Y)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$(G)(A) \to (H)$	$(A)(B) \to (D)$ $(D)(B) \to (E)$ $(D)(E) \to (F)$ $(E)(F) \to (\#Z)$	$(A), (B),$ $(D), (E)$ (F)	$(\#X), (\#Y), (\#Z)$

Table 2: Test setup for `testES()`.

testKNN()

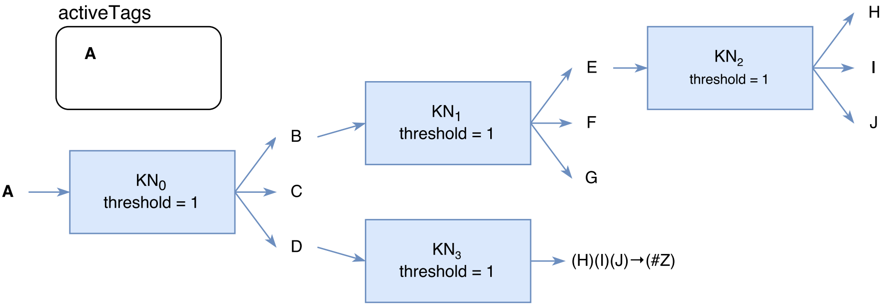

Figure 12: Test setup for `testKNN()`.

testKNNandES()

Same setup as testKNN(), but the activated Tags are passed on to an Expert System.

Ready Rules	Active Rules	Active Facts	Active Recommendations
$(H)(I)(J) \rightarrow (\#Z)$		$(B),(C),(D)$ $(E),(F),(G)$ $(H),(I),(J)$	
	$(H)(I)(J) \rightarrow (\#Z)$	$(B),(C),(D)$ $(E),(F),(G)$ $(H),(I),(J)$	$(\#Z)$

Table 3: Test setup for testKNNandES().

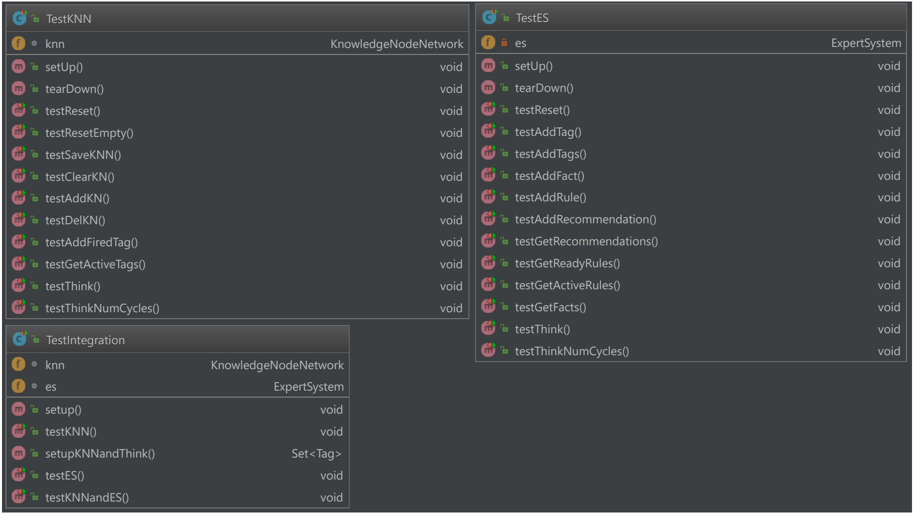

Figure 13: UML diagram of the `test` package.

Table of Contents

Finalization

First month

Features of the **Knowledge Node Network** yet to be implemented:

- Lambda thinking
- Confidence and strength
- Sigmoid activation
- Timestamp aging
- Interface with Neural Network

Features of the **Expert System** yet to be implemented:

- Fact token matching ($= \quad < \quad >$)

Table of Contents

Integration

Second month

The ES and KNN layers will have to be merged with the NN and META, which will probably lead to some conflicts. Time will be required to ensure proper functionality.

Table of Contents

Simulation

Third month

The Java simulation library Simbad will be used to test the entire system on virtual robots.

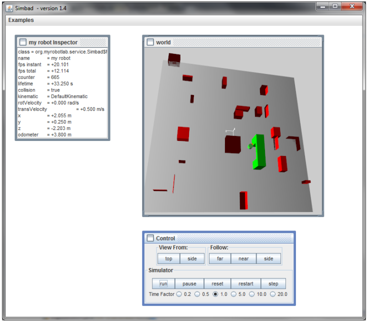

Figure 14: Simbad Java simulator.[8]

---

[8] *Simbad 3D Robot Simulator*. http://simbad.sourceforge.net/index.php. (Accessed on 03/27/2017).

Physical Testing

Third month

Prof. Vybihal's lab has multiple simple robots with ultrasonic sensors that can be tested on.

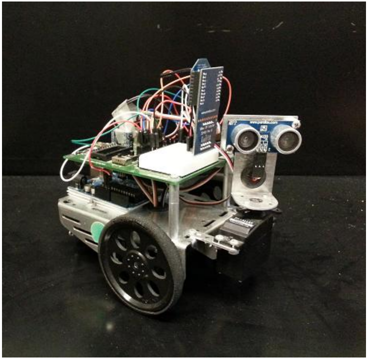

Figure 15: Robot in the lab.

References I

- *Simbad 3D Robot Simulator*. http://simbad.sourceforge.net/index.php. (Accessed on 03/27/2017).
- Joseph Vybihal. *Expert Systems*. 2016.
- Joseph Vybihal. *Full AI Model*. 2016.
- Joseph Vybihal. *Knowledge Nodes*. 2017.
- Joseph Vybihal and Thomas R. Shultz. *Search in Analogical Reasoning*. 1990.