

Introduction & Background

Prometheus is an AI system designed to control and coordinate multiple **robots** based on their sensor data. Its architecture is loosely inspired from the structure of the human brain, and is composed of the following four layers: the **Neural Network (NN)**, the **Knowledge Node Network (KNN)**, the **Expert System (ES)**, and the **Meta Reasoner (META)**. The **Neural Network** layer is the interface to the robots' sensors and provides informational tags to the KNN.

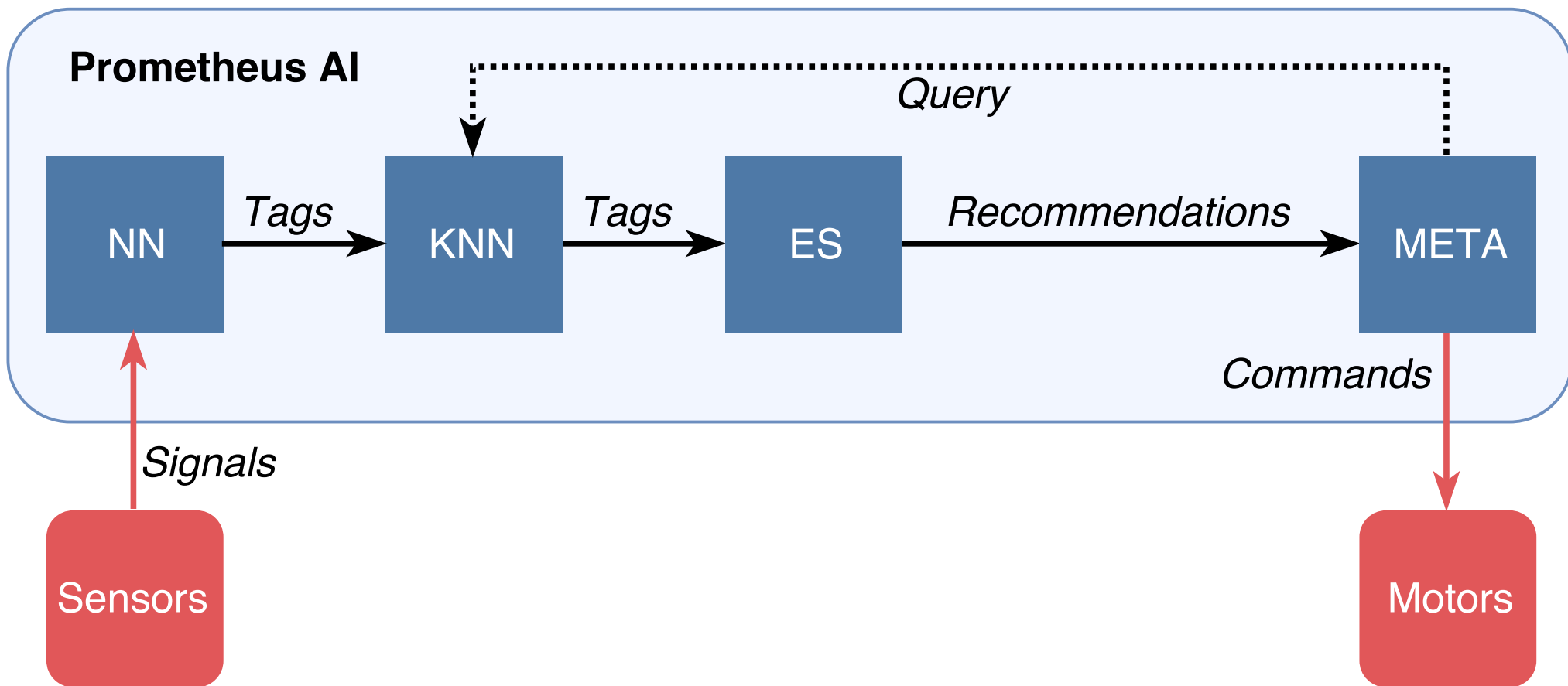


Figure 1: Prometheus AI model with labeled input and output.

The **Knowledge Node Network** layer is the analog to **memory** in the human brain. It takes in the tags provided by the NN and outputs tags based on its knowledge. It consists of interconnected **Knowledge Nodes** (KNs), which are abstract structures representing **memories** and their connections to other memories. There are many concepts associated with KNs, such as **activation**, **firing**, **strength**, **aging** and **belief**. The KNN has 4 ways of searching to fire KNs and activate tags: **direct**, **forward**, **backward** and **lambda**.

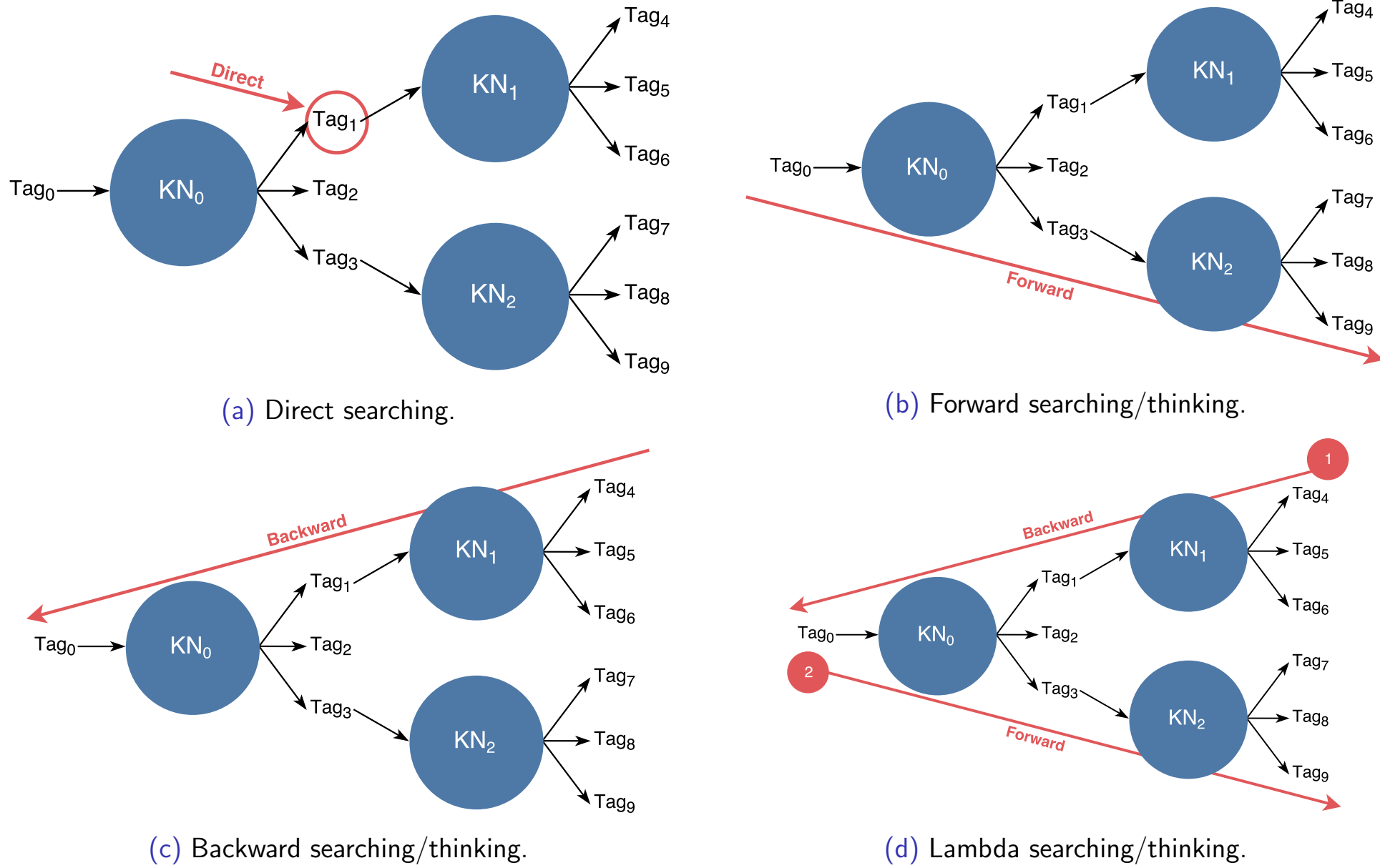


Figure 2: Methods of searching and thinking in the KNN.

The **Expert System** layer is a basic logic reasoner. It is not aware of its current reality or any context. It takes in the tags provided by the KNN and interprets them as either **facts**, **recommendations** or **rules**. Based on these rules, it performs a thinking routine to activate recommendations and provide them to the Meta Reasoner.

Fact	Meaning
$P(x)$	x is true or active.
$P(x = 1)$	x is equal to 1.
$P(x \neq 1)$	x is not equal to 1.
$P(x > 1)$	x is greater than 1.
$P(x < 1)$	x is less than 1.
$P(\&x)$	x can take any integer value.

(a) Examples of simple facts in the ES.

$$Fact := P(A_1, \dots, A_n)$$

(b) The form of a fact in the ES. P is a predicate, and A_i is an argument.

$$Recommendation := @P(A_1, \dots, A_n)$$

(c) The form of a recommendation in the ES. P is a predicate, and A_i is an argument..

$$Rule := Fact_1 \dots Fact_m \rightarrow Tag_1 \dots Tag_n$$

(d) The form of a rule in the ES, with m input facts and n output tags. The output tags must either be facts or recommendations.

Figure 3: Facts, recommendations and rules in the ES.

The **Meta Reasoner** represents high-level reasoning in the human brain. It is aware of its environment and context, and makes informed decisions to send actuator commands to the robots.

Prometheus AI

Problem

There are two main assigned tasks over the past two semesters:

- Design and implement the **Expert System** and **Knowledge Node Network** in Java.
- Supervise undergraduate students working on Prometheus.

Design & Implementation

The high-level **Prometheus** interface has dependencies on the NeuralNetwork, KnowledgeNodeNetwork, ExpertSystem and MetaReasoner interfaces. The KnowledgeNodeNetwork and ExpertSystem layers are the ones that are currently implemented. The code base was designed to be **efficient**, **modular** and easily **testable**. The **tags** used throughout the system are implemented using a Tag Java class, with Predicate and Rule subclasses. The Predicate class is a superclass of the Fact and Recommendation and represents the possible output tags of a rule.

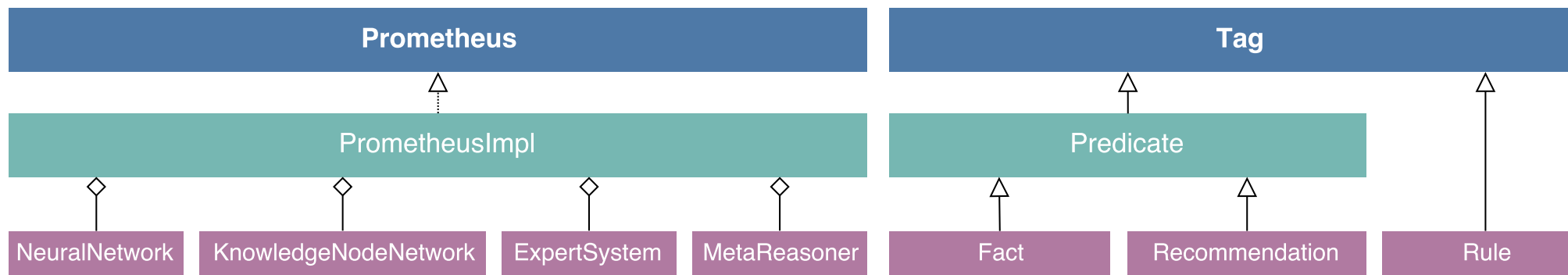


Figure 4: UML diagrams of the Prometheus and Tag packages.

The most important method in the **Expert System** is think(), which executes **thinking cycles** and output recommendations. This is performed by the Thinker class and its associated ThinkCycleExecutor. The rest() method allows the creation of **new rules** based on existing ones, with the associated Rester and RuleMerger classes. The teach() method parses simple "if-then" **sentences** to generate rules by using the Teacher class.

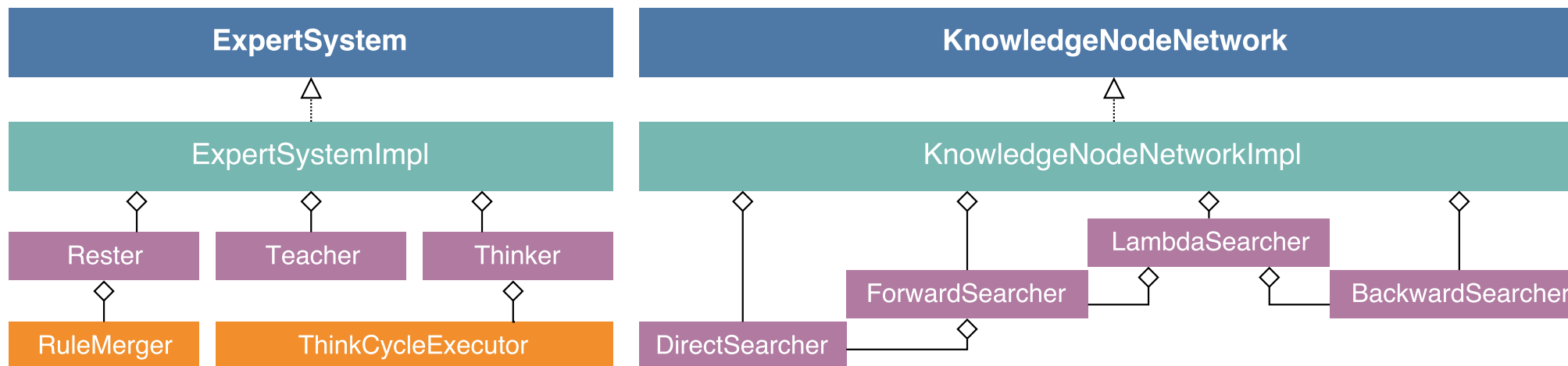


Figure 5: UML diagrams of the Expert System and KnowledgeNodeNetwork packages.

Searching in the **Knowledge Node Network** is done by the directSearch(), forwardSearch(), backwardSearch() and lambdaSearch() methods. These are controlled by the DirectSearcher, ForwardSearcher, BackwardSearcher and LambdaSearcher classes, respectively. These **modular** classes allowed great **re-use**, with the LambdaSearcher using the ForwardSearcher and BackwardSearcher, and the ForwardSearcher using the DirectSearcher.

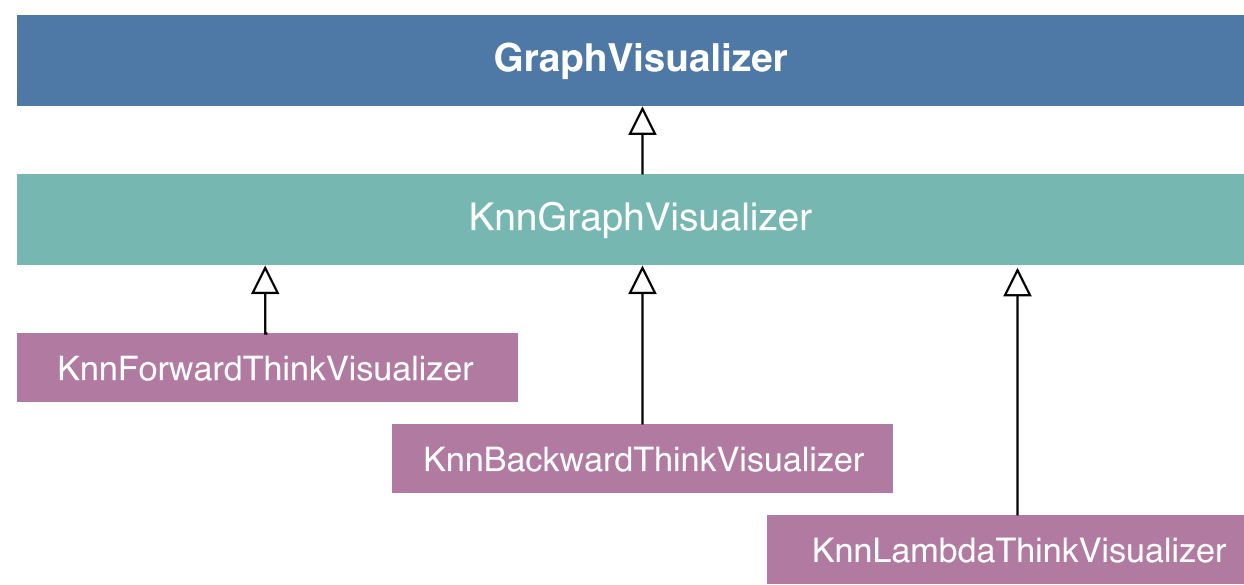


Figure 6: UML diagram of the graphing package.

The Java **graph visualization** library GraphStream was used to visualize the KNN and the process of searching through it. At each thinking or searching cycle, the graph can be updated in the user-interface and saved to SVG screenshots.

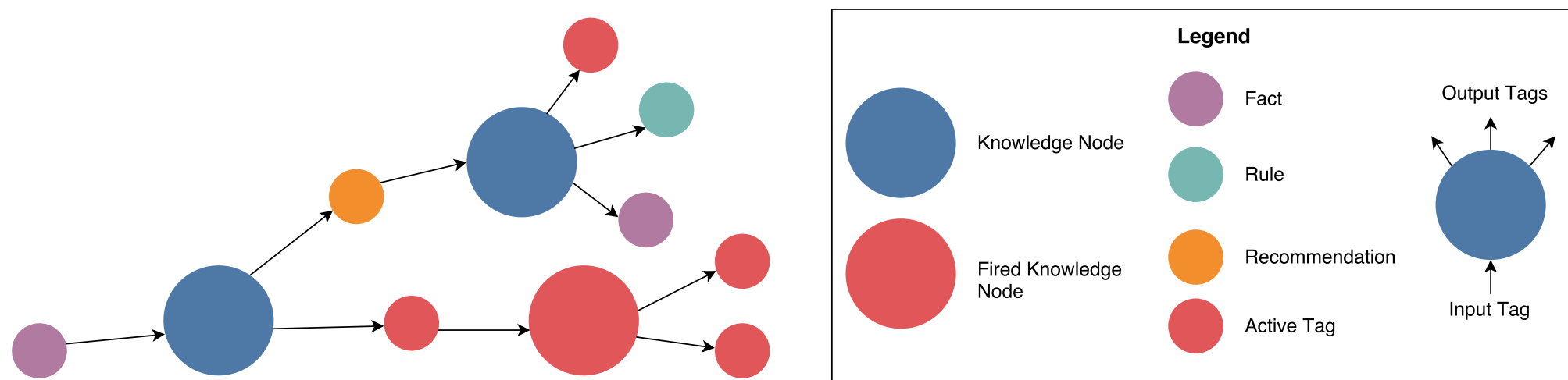


Figure 7: Legend for the graph visualization of the KNN.

Prometheus is a Java **Maven** project, which allows very simple use of many open-source libraries. Some of the libraries used on this project include: **Google Guice** for modeling and building the dependencies between the layers and their modules, **Mockito** for mocking objects in unit tests, **TestNG** for running tests and summarizing their results and **Apache Commons Lang 3** for standardized toString(), hashCode() and equals() methods.

Results & Tests

Unit tests were created using **Mockito** and **TestNG**. These tests test the functionality of isolated methods. **Integration tests** were also created with **TestNG**, testing end-to-end behavior. All of these tests are part of a continuous integration test suite which runs on the GitHub repository whenever a commit is made. This is done using **Travis CI**. Figure 8 shows a simple Expert System integration test.

State	Ready Rules	Active Rules	Active Facts	Active Recommendations
Initial	$(A)(B) \rightarrow (D)$ $(D)(B) \rightarrow (E)$ $(D)(E) \rightarrow (F)$ $(G)(A) \rightarrow (H)$ $(E)(F) \rightarrow (\#Z)$		$(A), (B)$	$(\#X), (\#Y)$
⋮	⋮	⋮	⋮	⋮
Final	$(G)(A) \rightarrow (H)$	$(A)(B) \rightarrow (D)$ $(D)(B) \rightarrow (E)$ $(D)(E) \rightarrow (F)$ $(E)(F) \rightarrow (\#Z)$	$(A), (B),$ $(D), (E)$ (F)	$(\#X), (\#Y), (\#Z)$

Figure 8: ES test setup representing simple rules and facts that must be brought to quiescence.

The forwardSearch(), backwardSearch() and lambdaSearch() were visualized using the created graph visualization tool. These can be seen in Figures 9 to 11. Please refer to Figure 7 for the meaning of the nodes and their colors.

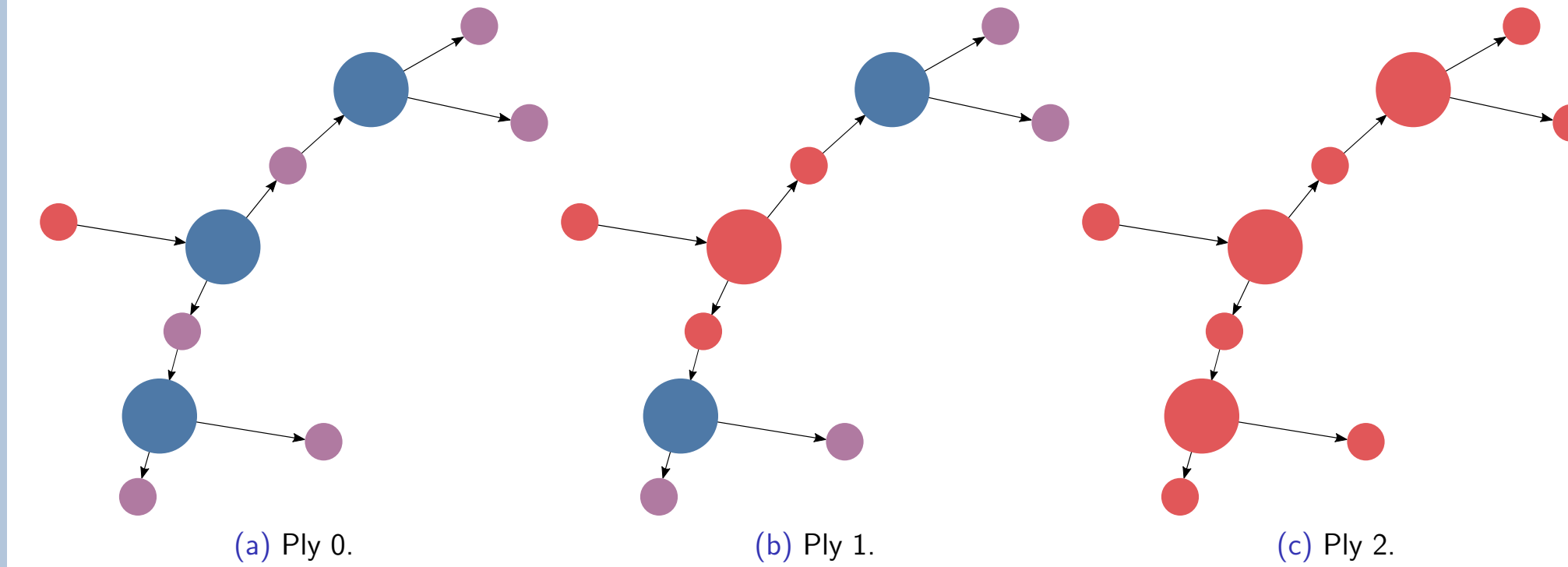


Figure 9: Forward search visualization in the KNN.

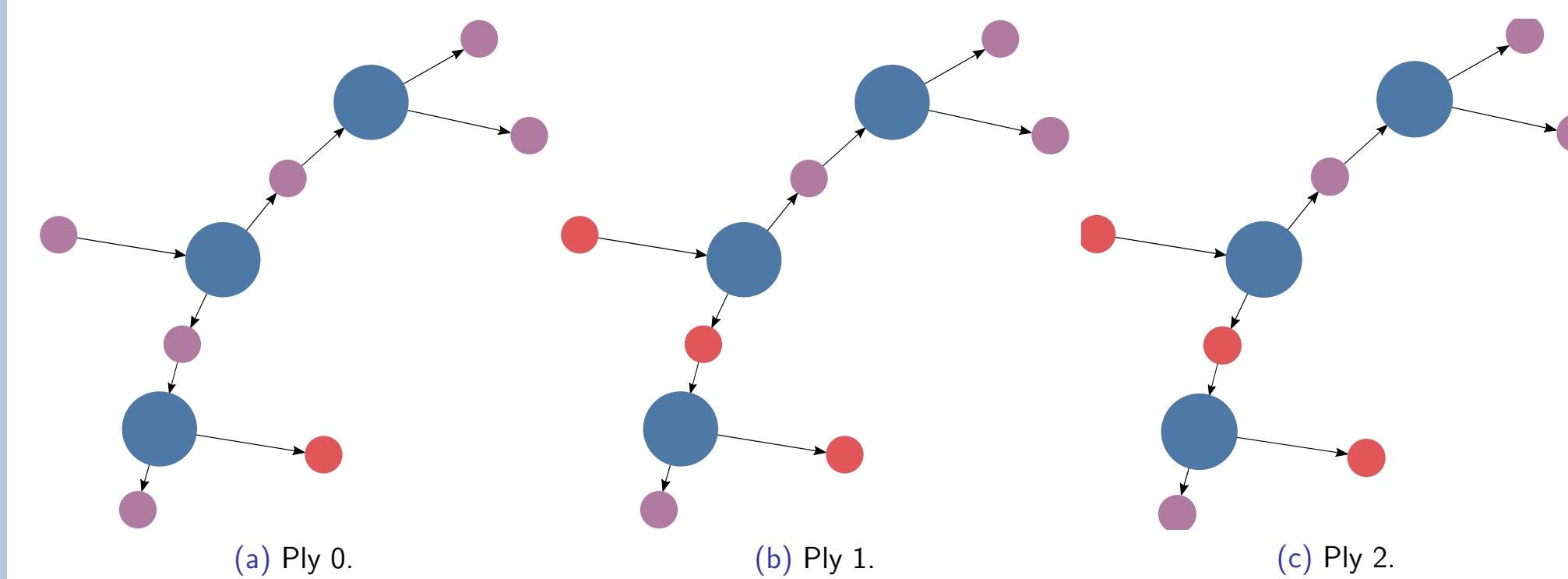


Figure 10: Backward search visualization in the KNN.

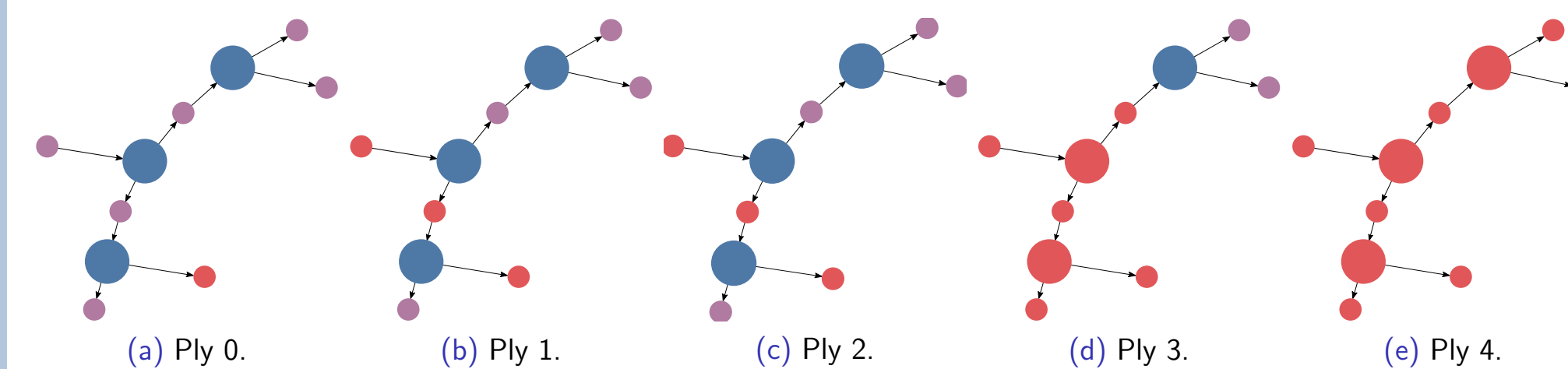


Figure 11: Lambda search visualization in the KNN.

Conclusion

Over the course of this project, I learned a great deal of skills relating to **designing** and **implementing** large-scale software projects, specifically with Java. I also learned a great deal about various **cognitive science** topics relating to Prometheus. Finally, setting up systems for software project **collaboration** was also learned. Possible further work relating to this project could be implementing the missing **NN** and **META** layers, as well as implementing more features in the KNN, such as **learning** and **attention**.