Winter 2019

Project Description

AI Integration

For Vera Tang, Evangeline, and Pentcho

Due: April 14, 2019

Overview / Goal:

To integrate the 4 AI technologies into a single software system.

The Prometheus AI model contains four technologies. The lowest layer is the neural network (NN) and it is directly connected to the sensors (robotic or simulated). It takes the raw data and converts it into an appropriate format that can be used by the remaining technologies. The NN is directly connected to the Knowledge Node layer (KN) which mimics cognitive science theories about animal memory. Data from NN and KN are passed to the third layer, the Expert System (ES). It functions as a recommendation system. It considers the data it received and proposes a course of action. The output from the NN, KN, and ES is passed to the last layer, the Meta Reasoner (META). The NN, KN, and ES layers are viewed as the subconscious. The META is viewed as conscience, or at least aware of its surroundings, goals, priorities, interests and purpose. The META is connected to the motors and must make the final decision of whether to follow the recommendation or to think some more.

Your job is to get the four technologies into a configuration where it behaves as described in the previous paragraph. To prove this, an experiment must be conducted that shows the extent of your success.

Requirements:

You will need to get, from the professor, the code. Each layer is a separate code base. Each layer has a report written by the previous student who worked on it. Spend time to understand the report and the code.

You must be proficient in Java programming, data structures, and algorithms.

A course in AI or the willingness to read background material on AI is an asset.

Background:

Read the Google spreadsheet docs: “Full AI Model” and “Simulator”. Also read the NN, KN, ES, and META sections.

Install, run, understand how to use, and understand how to modify:

- Chris O’Conner’s Fall 2018 NN Project.
- Zisheng Wang’s Fall 2018 META Project
- KN and ES 2017 Project

Once that is done, you should read and understand the assignment description, below, and come to meet with me to go over additional requirements from me or to answer your questions. Then, you can start the project.

Assignment Description:

Step 1: Directories

Create a folder called **Integration** and within that folder four subfolders called: **NN**, **KN**, **ES**, and **META**. Copy into those subfolders the relevant student projects.

Continue to Step 2 once you have created these folders.

Step 2: Old Projects

In the case of NN, remove the text simulator and only retain the neural network code, which probably won't be too large. In the case of META, the project was not complete and may be difficult to integrate. There is a new student this semester also working on the META layer. Do not use the new student's code unless you have too but keep in contact with him so that his code is compatible with what you are doing. I might also work on META this semester, if I have time. You might want to ignore my code as well, but you will need to keep me informed so that my code is also compatible with what you are doing.

Continue to Step 3 once you have extracted the parts of the code you need, compiled those parts successfully, and made each project run on its own.

Step 3: Understanding Integration

Each layer of the AI must implement Java interfaces to maintain communication consistency between the different technologies. This will be handled by defining special interfaces and special data structures for managing the passing of data from one later of the AI to the next layer. The code presented here is not complete but serves as a guideline.

See the following:

INTERFACE DEFINITION

```
interface LayerInput {
     void receiveDataStream(Tuple x);
}

interface LayerOutput {
     void sendDataStream(Tuple x);
}

interface SensorInput {
     void receiveDataStream(int nnID, int nnStruct, int data[]);
}
```

```
interface MotorOutput {
     void sendCommand(String motor, String command);
}
```

INTERFACE USAGE

```
class NeuralNetwork implements SensorInput, LayerOutput { … }

class KnowledgeNode implements LayerInput, LayerOutput { … }

class EspertSystem implements LayerInput, LayerOutput { … }

class Meta implements LayerInput, MotorOutput { … }
```

The purpose of the above interfaces is to establish a common public method signature when a layer wants to communicate (pass data) with another layer. It also serves to establish a common way of thinking about sharing and of thinking about what is inside the AI versus what is outside the AI. For example, the simulator or robot will communicate with the MotorOutput and SensirInput interfaces, while the LayerInput and LayerOutput interfaces deal with communication that happens within the AI.

The above interface definition is minimal. You can add additional signatures.

THE FORMAT OF DATA PASSED BETWEEN LAYERS

It is important to note that each layer treats data differently. For example, NN uses arrays, KN uses a linked list of strings, etc. We want each layer to keep its data format. We also want a common way to transmit data between layers. Therefore, it is assumed that the interface methods will translate the data format of a layer into the Tuple format when transmitting. Then the receiving interface method will translate the Tuple format into the format used by the layer the data was sent to.

Below is a description of the Tuple data structure:

```
abstract class Tuple {
     // All AI knowledge structures will be based on this class.
     // Based on the following format: (label, parameters)
     // Where "label" is a string
     // Where "parameters" are either strings or integers
     // The integers range from -100 to +100 and are a score or
     //    a measurement

     final int minScore = -100;
     final int maxScore = 100;

     String label;       // The "label" from (label, params)
     String sParams[];   // The "parameters" from (label,params)
     int iParams[];      // Int parameters from (label, params)

     void setTuple(String label,String[] sParams,int[] iParams);
     String getLabel();
     String[] getSParams();
```

```
     int[] getIParams();
}

interface IterateTuple {
   Tuple firstTuple();     // Returns NULL when list is empty
   Tuple nextTuple();      // Returns NULL when at end of list
   boolean isEmptyTuple(); // False when nextTuple() is NULL
}
```

HOW DOES A LAYER THINK?

It is important to standardize the way each layer thinks, from a method signature point of view. This will give us a common abstract way of understanding this process.

Please consider the following code:

```
interface Thinking {
     // All layers: NN, KN, ES, META implement this interface.
     // It is the "main method" for each layer. It initiates the
     // thinking process for every layer and defines how each
     // layer interacts with the other.  The interface assumes
     // the following abstract view of the method:
     // output_to_next_layer think(input_to_current_layer)

     Tuple[] think(int iterate, Tuple tuples[]);
}
```

The Think interface works as a chain reaction: the robot (or simulator) calls the think() method of the NN layer. The NN layer then calls the sensor input methods and performs its AI computations. It then creates a Tuple of the results and calls the think() method of the KN layer passing the Tuple. KN does its computations and then calls the think() method of ES passing the Tuple containing the data from NN and KN. The ES think() method does its computations and calls the think() method of META passing the NN, KN, and ES outputs. Now META computes and decides if it should call the think() method of the KN to rethink things or to call the methods of the motors to carry out the action.

Continue to Step 4 once you have understood the meaning and purpose of the above interfaces and classes.

Step 4: Integration Part 1

In this step we will ignore the SensorInput and MotorOutput interfaces. We will focus on the LayerInput, LayerOutput and Tuple data structure.

Rework the individual code bases so that they implement the above while not integrated.

Continue to Step 5 once you can show a working experiment that uses the new code. This is done with each layer individually. Write or use the existing main method in each project to

prove that simple data communication is happening. To do this: provide an in-code way to input sample data to the layer, and then have the program output to the screen the events that happen at each step that proves the program is functioning as expected.

Demo to the prof.

Step 5: Integration Part 2

In this step you will incorporate the Think interface and you will merge the 4 projects into a single application.

Rework the individual codebases into a single merged code base within the folder Integration under a subdirectory called Source. You are implementing the Think interface as the common method to pass data from one layer to the next.

Continue to Step 6 once you can show that the integrated project is working. Do this in a similar way you proved Step 4.

Demo to the prof.

Step 6: Integration Part 3 – The Experiment

In this step you will implement the SensorInput and MotorOutput interfaces within the integrated software system completed in Step 5. To do this we need to determine, together with the professor, what sensor-input and motor-output mean.

To prove that this works, create a main method that pretends to be a simulator and calls the think() method of the NN layer. Then the chain of events should happen automatically. Since there are no motors the program will output to the screen what it would have done given the input from the main method.

Your experiment will be to handle the following situation: assume the robot is stationary. Assume the robot is looking ahead. It sees one of three possibilities: (1) nothing in front of it, (2) a stationary object in front of it, or (3) a moving object in front of it. The AI should respond with the following outputs: (1) move forward one step, (2) turn left (or right) 90 degrees, (3) wait one-time unit (for the moving object to go away).

Continue to Step 7 once you have successfully demoed this to the prof.

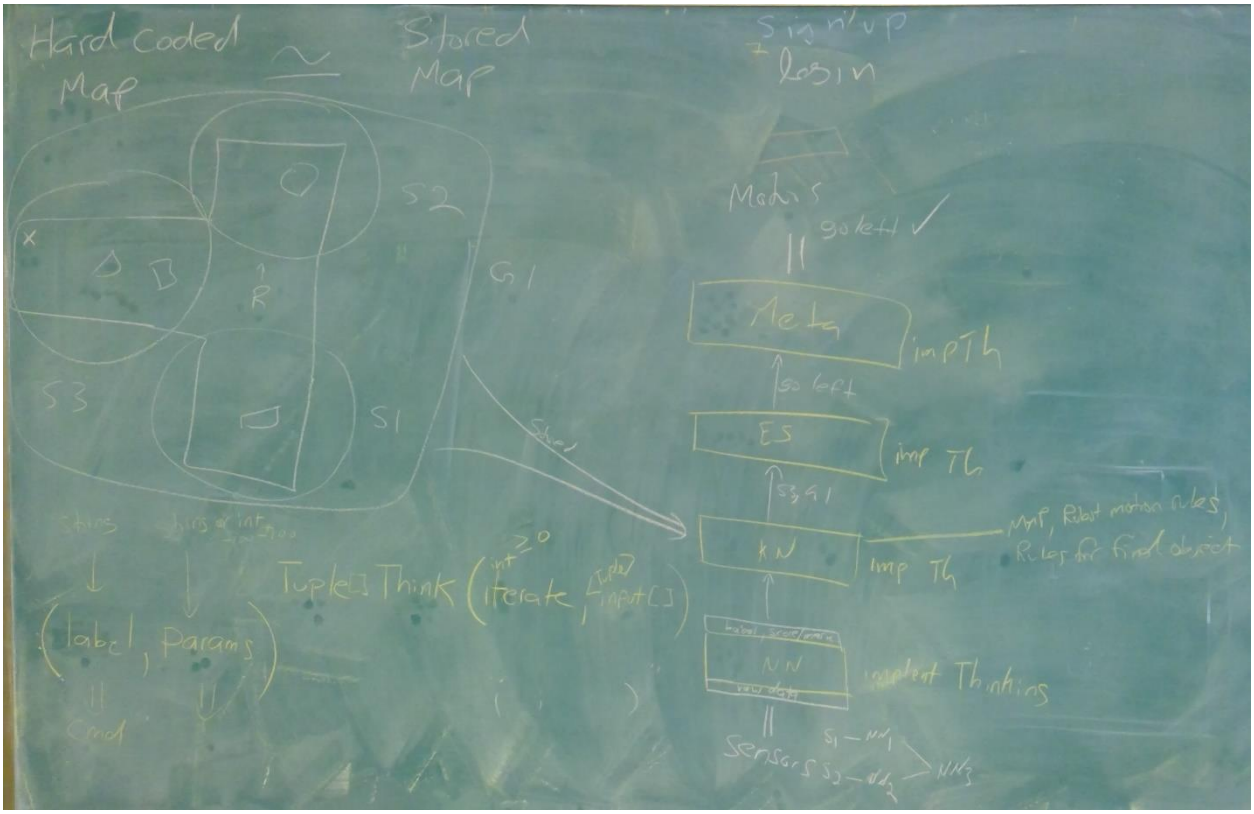
Step 7: Integration Part 3 – The Text Simulator

This step is optional.

Your project is complete when you successfully finish Step 6, however, if you have time and motivation, then integrating your AI into the text simulator and performing a more advanced experiment is the next step.

If you are interested in this, then make an appointment with the professor to talk about Step 7.

RANDOM NOTES FROM THE BLACKBOARD



The Step 7 Experiment

Using text files to help boot the software and initialize the experiment. Specifically, the text file(s) will contain configuration parameters and AI knowledge for each layer: NN, KN, ES, and Meta. For example: the map (S1, S2, S3 and G1) will be stored in KN. Rules for robotic motion, navigation, and search will also be stored in KN. Previously learned matrix parameters for the NN will also be stored in the text file and then loaded into the NN at the beginning. Similarly, goals for Meta will exist in the text file.

Three previously learned maps S1, S2, S3 and a graph G1 linking the three maps will be stored in KN. This "learned" map will have errors. For example, when it learned the map for S1 there was only one object in the room but now there are two. When it learned the size of the room the sensor incorrectly detected the distance to the walls.

One of the maps will have an X. The robot will have to get to that location from its current location. In the setup text files an initial position for the robot will be specified.

The **application** will have a hard-coded map built into the software that will be almost like the one **previously learned** by the AI. By previously learned, we mean hand written in the text file purposely with minor errors.

Starting up the experiment will follow this order:

1. Start the application
2. It builds all internal hard-coded data structures (like the “real” map)
3. It loads all the AI knowledge from the text files into the layers
4. It loads the parameters of the experiment from the text file, for example the location of the X and the initial location and direction of the robot
5. Display the “real map” (the entire world as hard-coded in app)
6. Display the Meta map (what it thinks the world is like) (map of path from robot to X)
7. Display the Meta immediate-surroundings-map (30cm around the robot, or more)
8. Display the Meta goal
9. Experiment starts…. Real and Meta maps are updated as the experiment runs until the robot finds X.

The experiment needs to be robust enough that the X can be moved to other locations. The robot’s starting location can be moved to other locations. The remembered map can be changed with random errors.

The knowledge stored in the AI should be able to handle all situations.