Experiment 1 - Jason NetworkScaler- : Understanding and Analyzing Resource Scaling with AgentSpeak Code
Objective:

This experiment aims to analyze the given AgentSpeak code for a resource scaling system. The goal is to understand how the code manages workloads and adjusts resources based on response time.

Story

The Jason code implements a resource scaling system that monitors workload levels and response times to make intelligent scaling decisions. Initially, the system checks whether the workload has arrived and evaluates the current workload status using simple print statements for logging. Based on the workload level (low, medium, or high) and the corresponding response time (good, ok, or bad), the system applies a specific scaling rule to adjust resource usage using a scale factor. After making the necessary adjustments, it consumes the workload and provides clear feedback through print statements to indicate which rule was applied.

Initialization and Basic Execution:

The code starts with a basic print statement (.print("asd")) upon the initial trigger +!startt plan.

Afterwards, any alternative of startt plan should trigger **arrageSource** to process incoming workload and response time, using the `workLoad(X) and responseTime(Y) beliefs as conditions. The plan then runs the following table.`

Rule	workLoad	responseTime	scaleFactor	ExtraAction
RULE 1	Low	Good	-15	consumeWorkLoad
RULE 2	Low	Ok	-10	consumeWorkLoad
RULE 3	Low	Bad	10	consumeWorkLoad
RULE 4	Medium	Good	-10	consumeWorkLoad
RULE 5	Medium	Ok	0	consumeWorkLoad
RULE 6	Medium	Bad	10	consumeWorkLoad
RULE 7	High	Good	0	consumeWorkLoad
RULE 8	High	Ok	10	consumeWorkLoad
RULE 9	High	Bad	15	consumeWorkLoad

Workload Management:

The system checks the current workload using **?currentWorkLoad(CCWL)** and prints the value. **In jason a belief that contains any value can be accessed by queries, for example currentWorkLoad(500), is a variable that holds 500 at that moment. ?currentWorkLoad(CCWL) assings 500 to CCWL.**

If the workload is zero or no tasks have arrived, it performs checks using getWorkLoadBool and checkWorkLoadBool. **For any alternative of +!startt plan, +!arrageSource plan should be triggered.**

Resource Scaling Logic:

There is a plan namely arrangeResources, which runs the following statements. In Jason a plan is written as "**+!planName: condition <- action1(55); action2.**", for example **+!openDoor: neighbour(true) & doorbell(true) & happiness (high) <- action1(55); action2.**

On each iteration of **arrangeSource, it returns back to the +!startt plan.**
The program executes continuously, as many iterations, when the all resources are consumed.