Michael Della Donna
CS 4341 – Artificial Intelligence
Project 1

README

The project is divided into 5 packages. The package `project1` contains runnable demonstration programs. The `ri` package contains implementations of the Rule Interpreter and the Interface Engine. The `ri` package has three sub packages that contain implementations of rules, actions, and predicates.

Architecture

The Data Structure for this program is centered around the `RuleInterpreter` Class. The `RuleInterpreter` Class maintains a list of rules and a JavaScript engine that has been loaded with the working memory and function definitions. A `Rule` consists of a `Predicate` and a list of `Action`s. An `Action` only consists of an executable snippet of JavaScript. The `Predicate` class is an interface that has several implementing classes, including `And`, `Not`, `Or`, and `JavaScript`. In this way a `Predicate` can be composed of several nested logical statements. The `InferenceEngine` contains only one static method that serves to process the rules. A graphical representation is available in Appendix A.

Program Operation

Overview

The Rules, initial Working Memory, and functional definitions for the predicates and actions are passed to the Rule Interpreter as Strings.
In the demonstration program, these are loaded in from files, but they do not need to be.
The rule interpreter parses each rule into at rule object and loads the function definitions and initial working memory state into the context of the script engine.
The rule interpreter then uses the inference engine to evaluate the set of rules, modifying the working memory as directed.
The final state of the working memory can then be retrieved as the solution to the particular set of given rules and initial state.

Rule Language Design

The rules themselves consist of modified JavaScript.

```
IF (colorIs('green') AND (NOT isHeavy())) THEN setType('grape')
```

The actions and predicates correspond to defined JavaScript functions. Predicates are the group between IF and THEN, and actions are anything after THEN. Predicates must return a Boolean value and actions are usually void functions that act on the working memory. Predicates can be nested with the logical keywords (AND), (NOT), and (OR). The rule interpreter uses these keywords to build up a predicate data structure for each rule. The Predicate interface has only one method, getResult(), that returns a boolean value. The Predicate interface is then implemented by classes that represent the logical keywords. For example, the And class is created with two other predicates. When asked to produce a response, it returns the logical AND of its two fields. At the root of these classes is the Javascript class, which also implements the predicate interface. It's getResult() method evaluates the JavaScript function that it represents and returns the result. In this way, the predicates can be very complicated while still maintaining a readable form. Actions are similar, in that each action is a JavaScript function, however, it is only necessary to list them sequentially.

Because the basis for each predicate and action is a JavaScript function, predicate and action definitions can be loaded in together in the same file. This implementation also allows for extremely powerful predicate definitions represented by easily read rules. For example, the predicate isToastReady() might be defined as connecting to a webcam and determining if the toaster it is pointed at has finished its toasting cycle.

The Working Memory has been implemented as a JavaScript Object called `wm`. This simplifies working memory initialization, as well as allowing on the fly creation of fields.

Inference Engine Algorithm

- while rules exist and there are rules that match the current working memory
    - make a new set of triggered rules
    - put rules that match the current working memory into the triggered set
    - check if any rules matched the current wm
        - if none of them did then halt the IE
    - the set of triggered rules is stored as a heap, with the rules being sorted by how many individual functions were evaluated for their predicate. Therefore the top of the heap is the most specific rule that was encountered first
    - evaluate the actions of the fired rule
        - if the built in stop command is encountered, halt the IE
    - removed the fired rule from the list

Demonstration of the provided test case is included in Appendix B

Part 2

Problem Description

The problem that was chosen to be solved was to diagnose a landscape based on the obstacles that it contained, and then suggest a list of special gear that would be appropriate for some hiking through that particular landscape. This is all contained inside the domain specified in project0.
The domain is classified by two variables, type of obstacles (ecology), and relative height(landscape).

Ecology	**Landscape**
Forest (>50% trees with water and no vehicles)	High (max height > 10)
Tundra (>40%trees without water and vehicles)	Mid (height difference<2 and(4 <max height < 11))
Desert (<15%trees no water no vehicles)	Low (max height < 5)
Plains(<40%trees no vehicles)	
Urban (> 2 buildings and has vehicles)	

Using these two intermediate values, the climate of the domain is determined, as well as other factors that would affect a trip through the domain. These environmental factors are analyzed, and appropriate domain specific equipment is selected and suggested to the User.

For part 2, the only change made to the Rule Interpreter was the addition of this method:
public void addExternalObject(String name, Object var)
to allow the predicate functions direct access to the Land data structure from project0

Demonstrations
This Domain contained three trees and a river

Before Evaluation

```
{"water":false}
```


After Evaluation

```
{"water":true,
"hasRiver":true,
"hasLake":false,
"gear":["rope",
"waterproofBag",
"defensiveWeapon",
"poncho"],
"hasVehicle":false,
"ecology":"forest",
"hasMudpit":false,
"danger":true,
"climate":"moderate",
"rainy":true,
"landscape":"low",
"foodFactor":5}
```


This domain contained only one tree

Before Evaluation

```
{"water":false}
```


After Evaluation

```
{"water":false,
"hasVehicle":false,
"hasRiver":false,
"hasLake":false,
"landscape":"low",
"ecology":"tundra",
"gear":["extraWater",
"defensiveWeapon",
"extraFood"],
"climate":"continental",
"danger":true,
"foodFactor":3}
```


This domain contained 9 trees, 4 rivers, 1 lake, 1 rock, and 1 mudpit

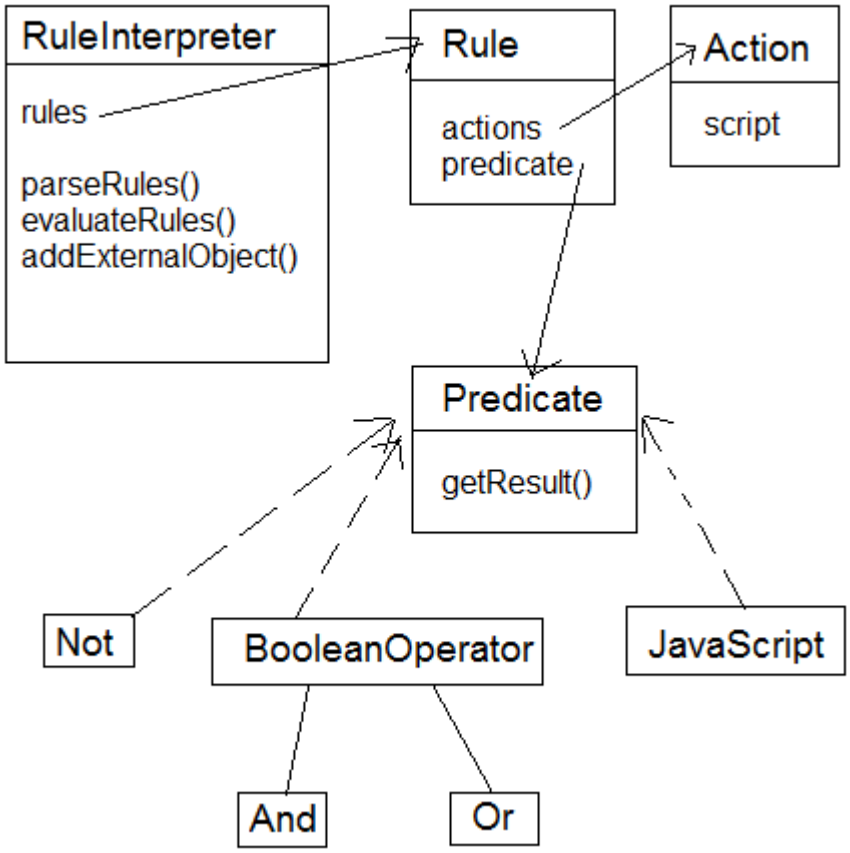
Before Evaluation

```
{"water":false}
```


After Evaluation

```
{"water":true,
"hasRiver":true,
"hasLake":true,
"gear":["rope","waterproofBag","defensiveWeapon",
"antiVenom","poncho","climbingHarness","airTank",
"fishingGear"],
"hasVehicle":false,
"ecology":"forest",
"hasMudpit":true,
"danger":true,
"climate":"tropical",
"poisonousAnimals":true,
"rainy":true,
"landscape":"high",
"foodFactor":7}
```

Appendix A



Appendix B

Rules

```
IF (colorIs('green') AND (NOT isHeavy())) THEN setType('grape')
IF (shapeIs('round') AND (colorIs('red') OR colorIs('green')))    THEN setType('apple')
IF colorIs('red') THEN setType('tomato')
IF typeIs('tomato') THEN setPlant('yes') setHeight('short')
IF (NOT ripe()) THEN setColor('green')
IF canCarry() THEN setHeavy('no')
IF typeHasValue() THEN stop()
```

Initial Working Memory

```
wm.color = null;
wm.carry = true;
wm.shape = 'round';
wm.type = null;
wm.plant = null;
wm.vegetable = null;
wm.fruit = null;
wm.ripe = false;
wm.heavy = null;
wm.weight = 2;
wm.height = null;
```

Functions(Predicate and Action definitions)

```
function shapeIs(shape)
{return wm.shape == shape;}

function setHeight(height)
{wm.height = height;}

function setPlant(plant)
{wm.plant = plant;}

function ripe()
{return wm.ripe;}

function colorIs(color)
{return wm.color == color;}

function setColor(color)
{wm.color = color;}

function setType(type)
{wm.type = type;}

function typeIs(type)
{return wm.type == type;}

function typeHasValue()
{return wm.type != null;}

function canCarry()
{return wm.carry;}

function isHeavy()
{return wm.heavy;}

function setHeavy(heavy)
{wm.heavy =
convertYesNoToTrueFalse(heavy);}

function
convertYesNoToTrueFalse(aString)
{if(aString == 'no' || aString == 'NO'
|| aString == 'No')
       {return false;}
 else{
       return true;}
}
```

Output

Initial Memory

```
{"color":null,
"carry":true,
"shape":"round",
"type":null,
"plant":null,
"vegetable":null,
"fruit":null,
"ripe":false,
"heavy":null,
"weight":2,
"height":null}
```

Memory After Rule Evaluation

```
{"color":"green",
"carry":true,
"shape":"round",
"type":"grape",
"plant":null,
"vegetable":null,
"fruit":null,
"ripe":false,
"heavy":false,
"weight":2,
"height":null}
```

Inference Engine Log

```
Beginning rule evaluation
Creating a list of triggered rules
Evaluating rule :IF ( colorIs('green'); AND (NOT isHeavy();) ) THEN setType('grape');
Evaluating rule :IF ( shapeIs('round'); AND ( colorIs('red'); OR colorIs('green'); ) ) THEN setType('apple');
Evaluating rule :IF colorIs('red'); THEN setType('tomato');
Evaluating rule :IF typeIs('tomato'); THEN setPlant('yes');setHeight('short');
Evaluating rule :IF (NOT ripe();) THEN setColor('green');
Rule has been triggered
Evaluating rule :IF canCarry(); THEN setHeavy('no');
Rule has been triggered
Evaluating rule :IF typeHasValue(); THEN stop();
Most specific rule is: IF (NOT ripe();) THEN setColor('green');
Executing action: setColor('green');
Removing triggered rule
re evaluating all remaining rules
Creating a list of triggered rules
Evaluating rule :IF ( colorIs('green'); AND (NOT isHeavy();) ) THEN setType('grape');
Evaluating rule :IF ( shapeIs('round'); AND ( colorIs('red'); OR colorIs('green'); ) ) THEN setType('apple');
Rule has been triggered
Evaluating rule :IF colorIs('red'); THEN setType('tomato');
Evaluating rule :IF typeIs('tomato'); THEN setPlant('yes');setHeight('short');
```

```
Evaluating rule :IF canCarry(); THEN setHeavy('no');
Rule has been triggered
Evaluating rule :IF typeHasValue(); THEN stop();
Most specific rule is: IF ( shapeIs('round'); AND ( colorIs('red'); OR colorIs('green'); ) ) THEN
setType('apple');
Executing action: setType('apple');
Removing triggered rule
re evaluating all remaining rules
Creating a list of triggered rules
Evaluating rule :IF ( colorIs('green'); AND (NOT isHeavy();) ) THEN setType('grape');
Evaluating rule :IF colorIs('red'); THEN setType('tomato');
Evaluating rule :IF typeIs('tomato'); THEN setPlant('yes');setHeight('short');
Evaluating rule :IF canCarry(); THEN setHeavy('no');
Rule has been triggered
Evaluating rule :IF typeHasValue(); THEN stop();
Rule has been triggered
Most specific rule is: IF canCarry(); THEN setHeavy('no');
Executing action: setHeavy('no');
Removing triggered rule
re evaluating all remaining rules
Creating a list of triggered rules
Evaluating rule :IF ( colorIs('green'); AND (NOT isHeavy();) ) THEN setType('grape');
Rule has been triggered
Evaluating rule :IF colorIs('red'); THEN setType('tomato');
Evaluating rule :IF typeIs('tomato'); THEN setPlant('yes');setHeight('short');
Evaluating rule :IF typeHasValue(); THEN stop();
Rule has been triggered
Most specific rule is: IF ( colorIs('green'); AND (NOT isHeavy();) ) THEN setType('grape');
Executing action: setType('grape');
Removing triggered rule
re evaluating all remaining rules
Creating a list of triggered rules
Evaluating rule :IF colorIs('red'); THEN setType('tomato');
Evaluating rule :IF typeIs('tomato'); THEN setPlant('yes');setHeight('short');
Evaluating rule :IF typeHasValue(); THEN stop();
Rule has been triggered
Most specific rule is: IF typeHasValue(); THEN stop();
Executing action: stop();
Stop action detected, halting
Removing triggered rule
re evaluating all remaining rules
Rule evaluation finished
```