INFOM451 - Conception d'Applications Mobiles

Rapport

Groupe Environnement

Auteurs :
Cyril Carlier
Axel Halin
Dorian Lecomte

9 mai 2016

Table des matières

1 Introduction **3**

2 Technologies **3**

3 Contenu **3**

3.1 Première Question 3

3.1.1 gui.scala 3

3.1.2 AutonomousAgentGUI.scala 4

3.1.3 InteractiveAgentGUI.scala 6

3.1.4 Modification de bacht_cli.scala 7

3.2 Deuxième Question 8

Annexe A Code Source **11**

A.1 AutonomousAgentGUI.scala 11

A.2 bacht_cli.scala 13

A.3 gui.scala 18

A.4 InteractiveAgentGUI.scala 21

A.5 VisualBlackboard.scala 24

1 Introduction

Ce projet, développé dans le cadre du cours *INFOM451 - Conception d'applications mobiles*, consiste en l'implémentation d'une interface pour l'interpréteur Bach-T. Dans un deuxième temps, il nous était demandé d'implémenter un langage permettant l'affichage d'image(s) en fonction du contenu du tableau.

2 Technologies

L'ensemble du projet a été réalisé en ***Scala***. Nous avons utilisé, pour l'interface graphique, la bibliothèque ***Swing Api*** (c.f : http ://www.scala-lang.org/api/2.11.2/scala-swing/.

Cette interface est construite au dessus de l'interpéteur Bach-T dont le code source nous as été fourni et dont une copie se trouve en annexe. Les modifications apportées au code source de base sont reprises et détaillées à la section suivante.

3 Contenu

3.1 Première Question

Cette première question concernait le design et l'implémentation d'une interface graphique. Des images d'une interface réalisée pour une extension de BachT nous étaient proposées en inspiration. Notre interface s'inspire grandement des images proposées et est composée de quatres fenêtres (nous détaillons dans cette section les trois principales, la quatrième sera présentée à la section suivante).

3.1.1 gui.scala

La classe ***gui.scala*** défini la fenêtre principale de l'application. Elle est définie comme suit :

— ***Content of the blackboard*** : *TextArea* affichant le contenu du tableau. Cet objet est mis à jour à chaque exécution d'une primitive ou d'un agent (autonome ou interactif).

— ***Clear*** : *Button* pour effacer le contenu du tableau ainsi que l'affichage du tableau.
— ***Token & Density*** : *TextFields* indiquant le nom ainsi que le nombre de tokens de la primitive.
— ***Primitives*** : *Buttons* pour exécuter une primitive. Ces primitives peuvent être de quatre types : tell, ask, get, nask. Elles sont exécutées via l'objet *BachTStore* de la classe ***bacht_cli.scala***.
— ***Agents*** : *Buttons* pour instantier un nouvel agent. Cet agent peut être soit autonome soit interactif (voir ci-dessous).

Cette fenêtre est visible à tout moment et sa fermeture entraine l'arrêt de l'application. Toute modification du tableau par un agent interactif ou autonome sera affiché sur cette page.

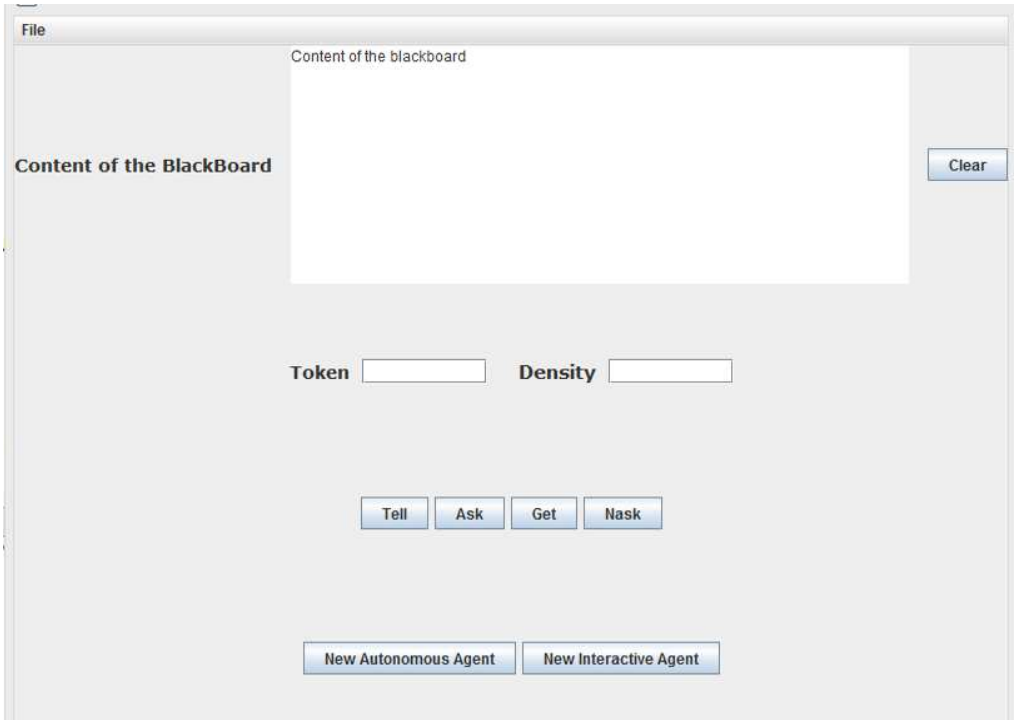

FIGURE 1 – gui.scala - Page d'accueil

3.1.2 AutonomousAgentGUI.scala

La classe ***AutonomousAgentGUI.scala*** défini la fenêtre relative aux agents autonomes. Un agent autonome permet à l'utilisateur de déclarer un

agent et de l'exécuter. Cette exécution peut se faire soit étape par étape, soit en une fois (auquel cas le tableau sera mis à jour à la fin de l'exécution et un message notifiera la réussite/l'échec de l'exécution).

La fenêtre est définie avec les composants suivants :

- ***Agent*** : *TextArea* pour déclarer un nouvel agent.
- ***Submit*** : *Button* pour valider l'agent. Une vérification est effectuée sur cet agent via la méthode de parsing *parse_agent* de la classe *BachTSimulParser*.
- ***Current Agent*** : *Label* affichant l'agent actuel. Cet agent est mis à jour à chaque exécution de la prochaine primitive.
- ***Run & Next*** : *Buttons* permettant respectivement d'exécuter entièrement l'agent ou d'exécuter la prochaine primitive.

Plusieurs instances d'agents autonomes sont possibles en parallèle. Ceux-ci partagent le même tableau.

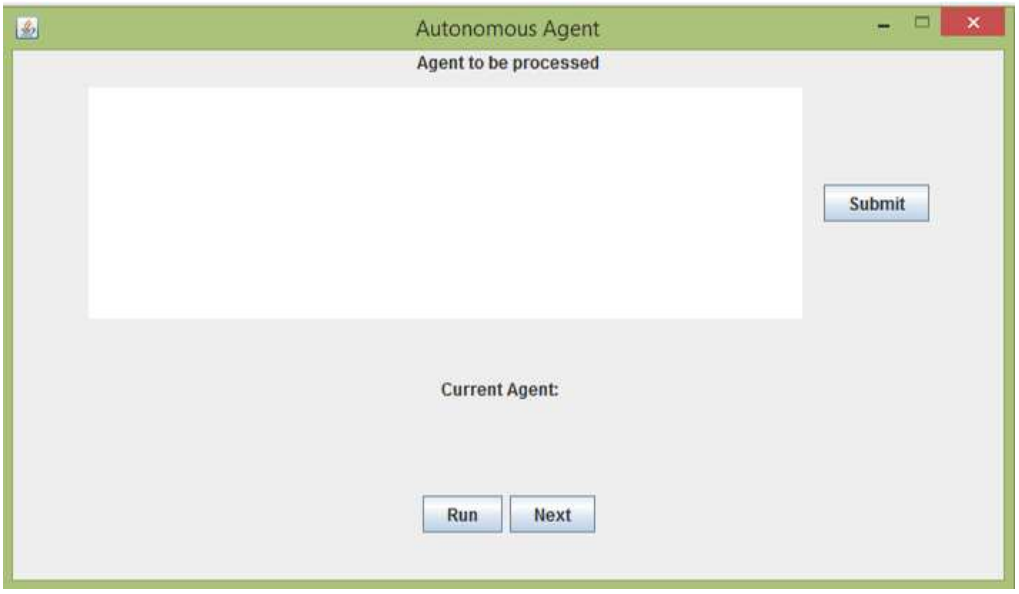

FIGURE 2 – AutonomousAgentGUI.scala - Ecran Agent Autonome

3.1.3 InteractiveAgentGUI.scala

La classe ***InteractiveAgentGUI.scala*** gère la fenêtre relative aux agent interactifs. Un tel agent permet à l'utilisateur de sélectionner lui même la primitive qu'il veut exécuter. Les primitives accessibles (c'est à dire les primitives dont l'exécution renvoit un *success*) sont accentuées par un bouton.

Elle est structurée de manière similaire à la classe de l'agent autonome :

- ***Agent*** : *TextArea* pour déclarer un nouvel agent.
- ***Submit*** : *Button* pour valider l'agent. Une vérification est effectuée sur cet agent via la méthode de parsing *parse_agent* de la classe *BachTSimulParser*.
- ***Current Agent*** : *BoxPanel* affichant l'agent actuel. L'agent affiché est un mix de boutons et de label où les boutons représentent les primitives pouvant être exécutées. Cet agent est mis à jour à chaque exécution de la prochaine primitive.

FIGURE 3 – InteractiveAgentGUI.scala - Ecran Agent Interactif

3.1.4 Modification de bacht_cli.scala

Nous n'avons que très légèrement modifié le code source de l'interpréteur *bacht_cli.scala*. Ces modifications sont majoritairement des ajouts avec une ou deux modifications.

Expr

Le premier ajout que nous avons fait s'applique à l'objet *Expr* ainsi qu'à ses sous-types.
Nous avons déclarer deux fonctions utilitaires : une redéfinition de toString() et une fonction remove(expression) permettant de supprimer une primitive de l'expression.

```
1  class Expr {
2    def remove(expression:bacht_ast_primitive) = this
3  }
4  case class bacht_ast_empty_agent() extends Expr
5  case class bacht_ast_primitive(primitive: String, token: String) extends Expr {
6    override def toString:String = {
7      "[ " + primitive + "(" + token + ")" + " ]"
8    }
9
10   override def remove(expression:bacht_ast_primitive) = if (expression.primitive.equals(
         primitive) && expression.token.equals(token)) new bacht_ast_empty_agent
11                                                         else this
12 }
13 case class bacht_ast_agent(op: String, agenti: Expr, agentii: Expr) extends Expr{
14   override def toString:String = {
15     "[ " + agenti.toString + " " + op + " " + agentii.toString + " ]"
16   }
17
18   override def remove(expression:bacht_ast_primitive) = {
19     op match {
20       case ";" => if (agenti.remove(expression).isInstanceOf[bacht_ast_empty_agent]) agentii
21                   else bacht_ast_agent(op, agenti, agentii.remove(expression))
22       case _ =>     bacht_ast_agent(op, agenti.remove(expression), agentii.remove(expression))
23     }
24
25   }
26 }
```

BachTSimulParser

La deuxième modification apportée concerne l'étape de parsing. Nous avons modifié le "case failure :NoSuccess" pour que la fonction retourne null plutôt que de lancer une erreur.
Lors de l'appel de cette méthode une vérification sur la valeur de retour est donc nécessaire (et cruciale) pour afficher un message à l'utilisateur en cas de parsing incorrect.

```
1   def parse_primitive(prim: String) = parseAll(primitive,prim) match {
2         case Success(result, _) => result
3         case failure : NoSuccess => null
4   }
5
6   def parse_agent(ag: String) = parseAll(agent,ag) match {
7         case Success(result, _) => result
8         case failure:NoSuccess => null
9   }
```

BachTStore

Enfin, les dernières modifications concernent le store. Nous avons ajouter deux nouvelles méthodes utilitaires permettant respectivement d'avoir le contenu du tableau en une chaine de caractères et de connaître la densité d'un token particulier.

```
1   def getContent:String ={
2       var res = ""
3       for((t,d) <- theStore) res += t+"("+d.toString+")"
4       res = "{" + res + "}"
5       res
6   }
7
8   def findDensity(token:String):Int = {
9       try{
10          val density = theStore(token)
11          density
12      } catch{
13        case e:java.util.NoSuchElementException => 0
14      }
15  }
```

3.2 Deuxième Question

Pour cette deuxième question, il nous était demandé de concevoir et implémenter un langage permettant d'afficher des images selon le contenu du tableau. Ceci a été réalisé au moyen de la classe *VisualBlackboard.scala*, composée d'un ***GridPanel*** et d'un ***ScrollPane*** permettant l'affichage de plusieurs images.

La gestion de l'affichage des images s'effectue simplement, au moyen d'un listener sur le contenu du tableau. Ainsi, à chaque mise à jour de celui-ci (que ce soit l'ajout ou le retrait d'un token), l'affichage des images est recalculé. Dans cette version, nous avons choisit un affichage simple : un conteneur

(un bocal en l'occurence) qui se remplit en fonction du nombre de token sur le tableau. Les figures 4 et 5 présentent les différents cas possibles (à savoir les cas où 0,1,2,3,4 ou 5 tokens sont présents sur le tableau).
Pour des raisons de simplicités, nous avons traité les cas où la densité est supérieure ou égale à 5 comme similaire (partageant une même image).

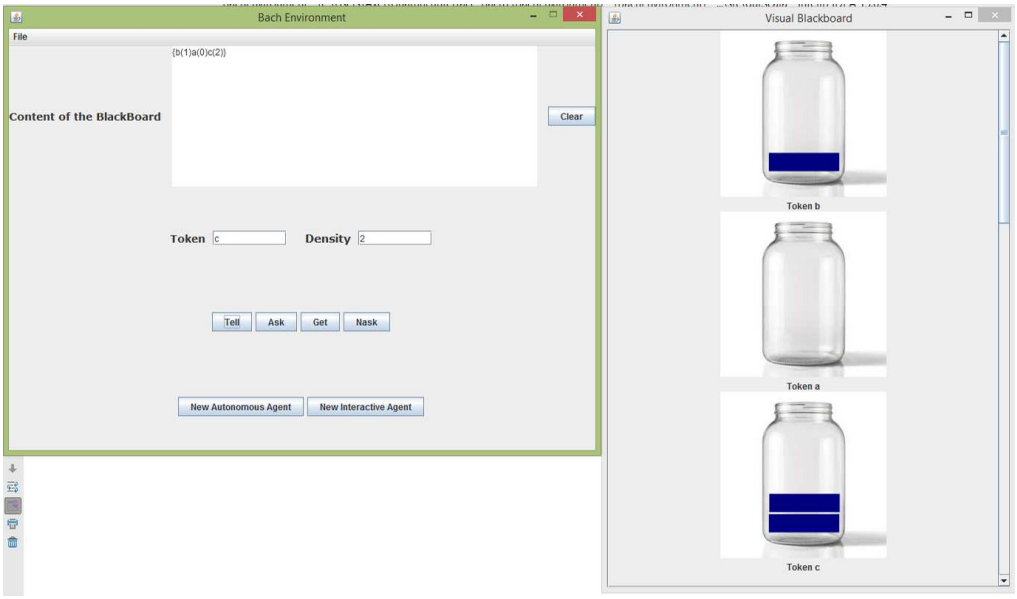

FIGURE 4 – VisualBlackboard.scala - 0 à 2 tokens

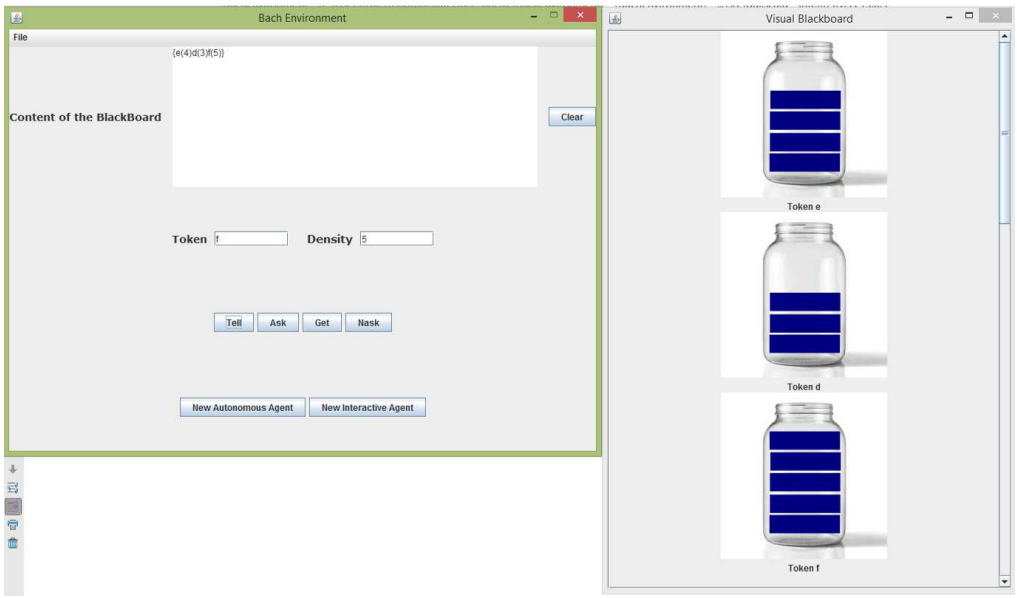

FIGURE 5 – VisualBlackboard.scala - 3 à 5 tokens

A Code Source

A.1 AutonomousAgentGUI.scala

```
1   import java.awt.{Dimension, Font}
2
3  import scala.swing._
4  import scala.swing.event.ButtonClicked
5
6
7  /**
8    * Autonomous Agent window of the application.
9    * This window allows the user to create an autonomous agent.
10   * The user can then either execute the whole agent at once or each primitive at a time.
11   * In the event of a bad execution (failure returned) a Dialog is displayed.
12   *
13   * Created by Axel on 23-04-16.
14   */
15 class AutonomousAgentGUI(blackboard:BachTStore, blackboardDisplay:TextArea) extends MainFrame
       {
16
17   var agent = new BachTSimul(blackboard)
18   var expression = new Expr
19
20   // Components definition
21   val titleFont = new Font("Verdana", java.awt.Font.BOLD, 14)
22
23   val agentTitleLabel = new Label(){
24                           text = "Agent to be processed"
25                         }
26
27   val currentAgentLabel = new Label(){
28                             text = "Current Agent: "
29                           }
30
31   val currentAgentValue = new Label(){
32                             tooltip = "Agent to be executed"
33                             text = ""
34                           }
35
36   val agentTextArea = new TextArea(){
37                         editable = true
38                         preferredSize = new Dimension(100, 100)
39                         maximumSize = new Dimension(500, 300)
40                         resizable = false
41                       }
42
43   val submitButton = new Button(){
44                         text = "Submit"
45                         tooltip = "Submit the new Agent"
46                     }
47
48   val runButton = new Button(){
49                     text = "Run"
50                     tooltip = "Run all primitives of the agent"
51                   }
52
53   val nextButton = new Button(){
54                     text = "Next"
55                     tooltip = "Run next primitive"
56                   }
57
58
59   title = "Autonomous Agent"
60   centerOnScreen()
61   preferredSize = new Dimension(700, 400)
62   resizable = false
63
```

```
64  contents = new BoxPanel(Orientation.Vertical){
65    contents += new BoxPanel(Orientation.Horizontal){
66      contents += agentTitleLabel
67    }
68
69    // Vertical Space
70    contents += Swing.VStrut(10)
71
72    // Second line: Tokens
73    contents += new BoxPanel(Orientation.Horizontal){
74      contents += agentTextArea
75        contents += Swing.HStrut(15)
76        contents += submitButton
77      }
78
79    // Vertical Space
80    contents += Swing.VStrut(10)
81
82    // Third Line: Buttons
83    contents += new BoxPanel(Orientation.Horizontal){
84      contents += currentAgentLabel
85        contents += Swing.HStrut(10)
86        contents += currentAgentValue
87    }
88
89    // Vertical Space
90    contents += Swing.VStrut(5)
91
92    contents += new BoxPanel(Orientation.Horizontal){
93      contents += runButton
94        contents += Swing.HStrut(5)
95        contents += nextButton
96    }
97  }
98
99  listenTo(submitButton)
100 listenTo(nextButton)
101 listenTo(runButton)
102
103 reactions += {
104   case ButtonClicked(component) if component == submitButton
105   =>  // Check the input content
106       if(agentTextArea.text == ""){
107           Dialog.showMessage(submitButton,"Please enter the value of the agent","Error:
                Agent can't be empty")
108       } else{
109           val agentParsed = BachTSimulParser.parse_agent(agentTextArea.text)
110           if (agentParsed == null) Dialog.showMessage(submitButton,"The agent value is
                incorrect.\nThe agent wasn't recognised.", "Error: Agent not recognized")
111           else {currentAgentValue.text = agentParsed.toString; expression = agentParsed}
112       }
113   case ButtonClicked(component) if component == nextButton
114   =>  if (currentAgentValue.text == "") Dialog.showMessage(nextButton, "The agent is empty.
        It cannot be run", "Error: Agent is empty")
115       else{
116           val res = agent.run_one(expression)
117           if (res._1) {
118               expression = res._2
119               if (!expression.isInstanceOf[bacht_ast_empty_agent]){
120                   currentAgentValue.text = expression.toString
121               }
122               else currentAgentValue.text = ""
123               blackboardDisplay.text = blackboard.getContent
124           }
125           else Dialog.showMessage(nextButton, "The execution of the next primitive returns a
                failure.","Error: next primitive failure")
126       }
127   case ButtonClicked(component) if component == runButton
```

```
128     =>  if (currentAgentValue.text == "") Dialog.showMessage(runButton, "The agent is empty.
            It cannot be run", "Error: Agent is empty")
129         else{
130             val res = agent.bacht_exec_all(expression)
131             if (res){
132                 currentAgentValue.text = ""
133                 blackboardDisplay.text = blackboard.getContent
134                 Dialog.showMessage(runButton, "The execution of the agent is a success !", "
                        Success")
135             } else Dialog.showMessage(runButton, "The execution of the agent returns a failure
                    ","Error: agent failure")
136         }
137   }
138
139   override def closeOperation : scala.Unit = this.close()
140 }
```

A.2 bacht_cli.scala

```
1   /* ------------------------------------------------------------------------
2
3    Complete code for the command-line interpreter
4
5
6    AUTHOR : J.-M. Jacquet and D. Darquennes
7    DATE   : March 2016
8
9   ------------------------------------------------------------------------*/
10
11  class Expr {
12    def remove(expression:bacht_ast_primitive) = this
13  }
14  case class bacht_ast_empty_agent() extends Expr
15  case class bacht_ast_primitive(primitive: String, token: String) extends Expr {
16    /**
17      * Returns the primitive in a structured String
18      *
19      * @return a String object of the token and its density.
20      * @author Axel Halin
21      */
22    override def toString:String = {
23      "[ " + primitive + "(" + token + ")" + " ]"
24    }
25
26    /**
27      * Returns a bacht_ast_empty_agent if expression matches the bacht_ast_primitive.
28      * Returns the bacht_ast_primitive otherwise.
29      *
30      * @param expression Expression to compare to.
31      * @return bacht_ast_empty_agent if it matches expression; self otherwise.
32      */
33    override def remove(expression:bacht_ast_primitive) = if (expression.primitive.equals(
          primitive) && expression.token.equals(token)) new bacht_ast_empty_agent
34                                                          else this
35  }
36  case class bacht_ast_agent(op: String, agenti: Expr, agentii: Expr) extends Expr{
37    /**
38      * Returns the agent in a structured String
39      *
40      * @return a String object of the operator between the two subagents
41      * @author Axel Halin
42      */
43    override def toString:String = {
44      "[ " + agenti.toString + " " + op + " " + agentii.toString + " ]"
45    }
46
47    /**
48      * Removes the occurence of expression from the bacht_ast_agent
49      *
```

```
50      * @param expression Expression to remove
51      * @return The bacht_ast_agent without expression.
52      * @author Axel Halin
53      */
54    override def remove(expression:bacht_ast_primitive) = {
55      op match {
56        case ";" => if (agenti.remove(expression).isInstanceOf[bacht_ast_empty_agent]) agentii
57                    else bacht_ast_agent(op, agenti, agentii.remove(expression))
58        case _ =>     bacht_ast_agent(op, agenti.remove(expression), agentii.remove(expression))
59      }
60
61    }
62  }
63
64  import scala.util.parsing.combinator._
65  import scala.util.matching.Regex
66
67  class BachTParsers extends RegexParsers {
68
69    def token   : Parser[String] = ("[a-z][0-9a-zA-Z_]*").r ^^ {_.toString}
70
71    val opChoice  : Parser[String] = "+"
72    val opPara    : Parser[String] = "||"
73    val opSeq     : Parser[String] = ";"
74
75    def primitive : Parser[Expr]   = "tell("~token~")" ^^ {
76          case _ ~ vtoken ~ _  => bacht_ast_primitive("tell",vtoken) }  |
77                                  "ask("~token~")" ^^ {
78          case _ ~ vtoken ~ _  => bacht_ast_primitive("ask",vtoken) }   |
79                                  "get("~token~")" ^^ {
80          case _ ~ vtoken ~ _  => bacht_ast_primitive("get",vtoken) }   |
81                                  "nask("~token~")" ^^ {
82          case _ ~ vtoken ~ _  => bacht_ast_primitive("nask",vtoken) }
83
84    def agent = compositionChoice
85
86    def compositionChoice : Parser[Expr] = compositionPara~rep(opChoice~compositionChoice) ^^ {
87          case ag ~ List() => ag
88          case agi ~ List(op~agii)  => bacht_ast_agent(op,agi,agii) }
89
90    def compositionPara : Parser[Expr] = compositionSeq~rep(opPara~compositionPara) ^^ {
91          case ag ~ List() => ag
92          case agi ~ List(op~agii)  => bacht_ast_agent(op,agi,agii) }
93
94    def compositionSeq : Parser[Expr] = simpleAgent~rep(opSeq~compositionSeq) ^^ {
95          case ag ~ List() => ag
96          case agi ~ List(op~agii)  => bacht_ast_agent(op,agi,agii) }
97
98    def simpleAgent : Parser[Expr] = primitive | parenthesizedAgent
99
100   def parenthesizedAgent : Parser[Expr] = "("~>agent<~")"
101
102 }
103
104 object BachTSimulParser extends BachTParsers {
105
106   // return nothing instead of scala.sys.error
107
108   def parse_primitive(prim: String) = parseAll(primitive,prim) match {
109         case Success(result, _) => result
110         //case failure : NoSuccess => scala.sys.error(failure.msg)
111         case failure : NoSuccess => null
112   }
113
114   def parse_agent(ag: String) = parseAll(agent,ag) match {
115         case Success(result, _) => result
116         //case failure : NoSuccess => scala.sys.error(failure.msg)
117         case failure:NoSuccess => null
118   }
```

```
119
120 }
121 import scala.collection.mutable.Map
122 import scala.swing._
123
124 class BachTStore {
125
126    var theStore = Map[String,Int]()
127
128    def tell(token:String):Boolean = {
129       if (theStore.contains(token))
130         { theStore(token) = theStore(token) + 1 }
131       else
132         { theStore = theStore ++ Map(token -> 1) }
133       true
134    }
135
136
137    def ask(token:String):Boolean = {
138       if (theStore.contains(token))
139              if (theStore(token) >= 1) { true }
140              else { false }
141       else false
142    }
143
144
145    def get(token:String):Boolean = {
146       if (theStore.contains(token))
147              if (theStore(token) >= 1)
148                { theStore(token) = theStore(token) - 1
149                  true
150                }
151              else { false }
152       else false
153    }
154
155
156    def nask(token:String):Boolean = {
157       if (theStore.contains(token))
158              if (theStore(token) >= 1) { false }
159              else { true }
160       else true
161    }
162
163    def print_store {
164       print("{ ")
165       for ((t,d) <- theStore)
166          print ( t + "(" + theStore(t) + ")" )
167       println(" }")
168    }
169
170    def clear_store {
171       theStore = Map[String,Int]()
172    }
173
174   /**
175     * Returns the content of the BlackBoard as a String
176     *
177     * @author Axel Halin
178     */
179   def getContent:String ={
180       var res = ""
181       for((t,d) <- theStore) res += t+"("+d.toString+")"
182       res = "{" + res + "}"
183       res
184   }
185
186
187   /**
```

```
188     * Returns the density of the specified token.
189     * If the token is not in the Store, returns 0
190     *
191     * @param token Token to search in the store
192     * @return The density of the token
193     * @author Axel Halin
194     */
195   def findDensity(token:String):Int = {
196       try{
197           val density = theStore(token)
198           density
199       } catch{
200         case e:java.util.NoSuchElementException => 0
201       }
202   }
203 }
204
205 object bb extends BachTStore {
206
207    def reset { clear_store }
208
209 }
210 import scala.util.Random
211 import language.postfixOps
212
213 class BachTSimul(var bb: BachTStore) {
214
215    val bacht_random_choice = new Random()
216
217    def run_one(agent: Expr):(Boolean,Expr) = {
218
219       agent match {
220          case bacht_ast_primitive(prim,token) =>
221             {  if (exec_primitive(prim,token)) { (true,bacht_ast_empty_agent()) }
222                else { (false,agent) }
223             }
224
225          case bacht_ast_agent(";",ag_i,ag_ii) =>
226             {  run_one(ag_i) match
227                   { case (false,_) => (false,agent)
228                     case (true,bacht_ast_empty_agent()) => (true,ag_ii)
229                     case (true,ag_cont) => (true,bacht_ast_agent(";",ag_cont,ag_ii))
230                   }
231             }
232
233          case bacht_ast_agent("||",ag_i,ag_ii) =>
234             {  var branch_choice = bacht_random_choice.nextInt(2)
235                if (branch_choice == 0)
236                  { run_one( ag_i ) match
237                       { case (false,_) =>
238                           { run_one( ag_ii ) match
239                               { case (false,_)
240                                   => (false,agent)
241                                 case (true,bacht_ast_empty_agent())
242                                   => (true,ag_i)
243                                 case (true,ag_cont)
244                                   => (true,bacht_ast_agent("||",ag_i,ag_cont))
245                               }
246                           }
247                         case (true,bacht_ast_empty_agent())
248                                   => (true,ag_ii)
249                         case (true,ag_cont)
250                                   => (true,bacht_ast_agent("||",ag_cont,ag_ii))
251                       }
252                  }
253                else
254                  { run_one( ag_ii ) match
255                       { case (false,_) =>
256                           { run_one( ag_i ) match
```

```
257                             { case (false,_) => (false,agent)
258                               case (true,bacht_ast_empty_agent()) => (true,ag_ii)
259                               case (true,ag_cont)
260                                     => (true,bacht_ast_agent("||",ag_cont,ag_ii))
261                             }
262                         }
263                       case (true,bacht_ast_empty_agent())
264                         => (true,ag_i)
265                       case (true,ag_cont)
266                         => (true,bacht_ast_agent("||",ag_i,ag_cont))
267                     }
268                 }
269
270           }
271
272
273         case bacht_ast_agent("+",ag_i,ag_ii) =>
274           {  var branch_choice = bacht_random_choice.nextInt(2)
275              if (branch_choice == 0)
276                { run_one( ag_i ) match
277                     { case (false,_) =>
278                         { run_one( ag_ii ) match
279                             { case (false,_) => (false,agent)
280                               case (true,bacht_ast_empty_agent())
281                                 => (true,bacht_ast_empty_agent())
282                               case (true,ag_cont)
283                                 => (true,ag_cont)
284                             }
285                         }
286                       case (true,bacht_ast_empty_agent())
287                         => (true,bacht_ast_empty_agent())
288                       case (true,ag_cont)
289                         => (true,ag_cont)
290                     }
291                 }
292              else
293                { run_one( ag_ii ) match
294                     { case (false,_) =>
295                         { run_one( ag_i ) match
296                             { case (false,_)
297                                 => (false,agent)
298                               case (true,bacht_ast_empty_agent())
299                                 => (true,bacht_ast_empty_agent())
300                               case (true,ag_cont)
301                                 => (true,ag_cont)
302                             }
303                         }
304                       case (true,bacht_ast_empty_agent())
305                         => (true,bacht_ast_empty_agent())
306                       case (true,ag_cont)
307                         => (true,ag_cont)
308                     }
309                 }
310           }
311       }
312   }
313
314   def bacht_exec_all(agent: Expr):Boolean = {
315
316       var failure = false
317       var c_agent = agent
318       while ( c_agent != bacht_ast_empty_agent() && !failure ) {
319          failure = run_one(c_agent) match
320               { case (false,_)          => true
321                 case (true,new_agent)  =>
322                    { c_agent = new_agent
323                      false
324                    }
325               }
```

```
326             bb.print_store
327             println("\n")
328         }
329
330         if (c_agent == bacht_ast_empty_agent()) {
331             println("Success\n")
332             true
333         }
334         else {
335             println("failure\n")
336             false
337         }
338     }
339
340     def exec_primitive(prim:String,token:String):Boolean = {
341         prim match
342           { case "tell" => bb.tell(token)
343             case "ask"  => bb.ask(token)
344             case "get"  => bb.get(token)
345             case "nask" => bb.nask(token)
346           }
347     }
348
349 }
350
351 object ag extends BachTSimul(bb) {
352
353   def apply(agent: String) {
354      val agent_parsed = BachTSimulParser.parse_agent(agent)
355      ag.bacht_exec_all(agent_parsed)
356   }
357   def eval(agent:String) { apply(agent) }
358   def run(agent:String) { apply(agent) }
359 }
```

A.3 gui.scala

```
1   import scala.swing._
2 import scala.swing.event.ButtonClicked
3 import swing.Swing
4
5 /**
6   * Main window of the application.
7   * This window displays the blackboard and allows the user to execute primitives.
8   *
9   * Created by Axel on 21-04-16.
10  */
11 object gui extends SimpleSwingApplication {
12
13   val blackBoard = new BachTStore
14
15   // Components definition
16   val titleFont = new Font("Verdana", java.awt.Font.BOLD, 14)
17
18   val contentLabel = new Label(){
19                        text = "Content of the BlackBoard"
20                        font = titleFont
21                      }
22
23   val blackBoardContent = new TextArea(){
24                             text = "Content of the blackboard"
25                             editable = false
26                             preferredSize = new Dimension(100,100)
27                             maximumSize = new Dimension(800, 400)
28                           }
29
30   val clearButton = new Button(){
31                        text = "Clear"
32                        tooltip = "Click to clear the content of the BlackBoard"
```

```
33                    }
34    val tokenLabel = new Label(){
35                       text = "Token"
36                       font = titleFont
37                    }
38    val tokenTextField = new TextField(){
39                         columns = 1
40                         preferredSize = new Dimension(30, 20)
41                         maximumSize = new Dimension(100, 20)
42                       }
43    val densityLabel = new Label(){
44                         text = "Density"
45                         font = new Font("Verdana", java.awt.Font.BOLD, 15)
46                         tooltip = "Number of token"
47                       }
48    val densityTextField = new TextField(){
49                           columns = 1
50                           preferredSize = new Dimension(30, 20)
51                           maximumSize = new Dimension(100, 20)
52                         }
53    val tellButton = new Button(){
54                     text = "Tell"
55                     tooltip = "Write a token on the blackboard."
56                   }
57    val getButton = new Button(){
58                     text = "Get"
59                     tooltip = "Remove a token from the blackboard."
60                   }
61    val askButton = new Button(){
62                     text = "Ask"
63                     tooltip = "Ask if a token is on the blackboard."
64                   }
65    val naskButton = new Button(){
66                     text = "Nask"
67                     tooltip = "Ask if a token is not on the blackboard."
68                   }
69    val autonomousAgentButton = new Button{
70                                  text = "New Autonomous Agent"
71                                  tooltip = "This button lets you create an autonomous agent.
                                        You then can choose to let it run and see the result, or
                                         run it step by step."
72                                }
73    val interactiveAgentButton = new Button{
74                                  text = "New Interactive Agent"
75                                  tooltip = "This button lets you create an agent and choose
                                        which (possible) instruction to execute."
76                                }
77    // End of components definition
78
79    val visualBlackboard = new VisualBlackboard(blackBoard, blackBoardContent)
80
81    def top = new MainFrame {
82                  title = "Bach Environment"
83                  preferredSize = new Dimension(800, 600)
84                  resizable = false
85
86                  contents = new BoxPanel(Orientation.Vertical){
87                      contents += new BoxPanel(Orientation.Horizontal){
88                          contents += contentLabel
89                          contents += Swing.HStrut(15)
90                          contents += blackBoardContent
91                          contents += Swing.HStrut(15)
92                          contents += clearButton
93                      }
94
95                      // Vertical Space
96                      contents += Swing.VStrut(15)
97
98                      // Second line: Tokens
```

```
99                  contents += new BoxPanel(Orientation.Horizontal){
100                   contents += tokenLabel
101                   contents += Swing.HStrut(10)
102                   contents += tokenTextField
103                   contents += Swing.HStrut(25)
104                   contents += densityLabel
105                   contents += Swing.HStrut(10)
106                   contents += densityTextField
107                 }
108
109                 // Third Line: Buttons
110                 contents += new BoxPanel(Orientation.Horizontal){
111                   contents += tellButton
112                   contents += Swing.HStrut(5)
113                   contents += askButton
114                   contents += Swing.HStrut(5)
115                   contents += getButton
116                   contents += Swing.HStrut(5)
117                   contents += naskButton
118                 }
119
120                 contents += new BoxPanel(Orientation.Horizontal){
121                   contents += autonomousAgentButton
122                   contents += Swing.HStrut(5)
123                   contents += interactiveAgentButton
124                 }
125             }
126
127             // Listener on buttons
128             listenTo(clearButton)
129             listenTo(tellButton)
130             listenTo(getButton)
131             listenTo(askButton)
132             listenTo(naskButton)
133             listenTo(autonomousAgentButton)
134             listenTo(interactiveAgentButton)
135
136             // Defines the reactions of events (e.g buttonclicked)
137             reactions += {
138                 case ButtonClicked(component) if component == clearButton
139                   =>  blackBoard.clear_store
140                       blackBoardContent.text = ""
141                       visualBlackboard.repaint()
142                       visualBlackboard.flowPanel.revalidate()
143                 case ButtonClicked(component) if component == tellButton
144                   =>  if (tokenTextField.text != "" && densityTextField.text != ""){
145                           for (i <- 1 to densityTextField.text.toInt) blackBoard.tell(
                                    tokenTextField.text)
146                         } else{
147                           Dialog.showMessage(tellButton,"Please indicate a token and a
                                  density.","An error occured")
148                         }
149                       blackBoardContent.text = blackBoard.getContent
150                       visualBlackboard.repaint()
151                       visualBlackboard.flowPanel.revalidate()
152                 case ButtonClicked(component) if component == getButton
153                   =>  if (tokenTextField.text != "" && densityTextField.text != ""){
154                             if (densityTextField.text.toInt <= blackBoard.findDensity(
                                    tokenTextField.text)){
155                               for (i <- 1 to densityTextField.text.toInt) blackBoard.get
                                      (tokenTextField.text)
156                             } else{
157                               Dialog.showMessage(getButton, "There isn't enough of that
                                      token on the blackboard", "The primitive can't be
                                      treated")
158                             }
159                         } else{
160                           Dialog.showMessage(getButton, "Please specify a token and a
                                  density.","An error occured")
```

```
161                         }
162                         blackBoardContent.text = blackBoard.getContent
163                         visualBlackboard.repaint()
164                         visualBlackboard.flowPanel.revalidate()
165                     case ButtonClicked(component) if component == askButton
166                       =>  if(tokenTextField.text != "" && densityTextField.text != ""){
167                             if (densityTextField.text.toInt <= blackBoard.findDensity(
                                    tokenTextField.text)){
168                                 blackBoard.ask(tokenTextField.text)
169                                 Dialog.showMessage(askButton,"There is enough tokens on the
                                        blackboard !", "Success")
170                             } else{
171                                 blackBoard.ask(tokenTextField.text)
172                                 Dialog.showMessage(askButton,"Unfortunately, there isn't
                                        enough tokens on the blackboard...","Failure")
173                             }
174                         } else{
175                           Dialog.showMessage(askButton, "Please specify a token and a
                                  density.","An error occured")
176                         }
177                         blackBoardContent.text = blackBoard.getContent
178                         visualBlackboard.repaint()
179                         visualBlackboard.flowPanel.revalidate()
180                     case ButtonClicked(component) if component == naskButton
181                       =>  if(tokenTextField.text != "" && densityTextField.text != ""){
182                             if (densityTextField.text.toInt <= blackBoard.findDensity(
                                    tokenTextField.text)){
183                               blackBoard.ask(tokenTextField.text)
184                               Dialog.showMessage(askButton,"Unfortunately, there is enough
                                      tokens on the blackboard...","Failure")
185                             } else{
186                               blackBoard.ask(tokenTextField.text)
187                               Dialog.showMessage(askButton,"There isn't enough tokens on the
                                      blackboard !", "Success")
188                             }
189                         } else{
190                           Dialog.showMessage(askButton, "Please specify a token and a
                                  density.","An error occured")
191                         }
192                         blackBoardContent.text = blackBoard.getContent
193                         visualBlackboard.repaint()
194                         visualBlackboard.flowPanel.revalidate()
195                     case ButtonClicked(component) if component == autonomousAgentButton
196                       =>  new AutonomousAgentGUI(blackBoard, blackBoardContent).open()
197                     case ButtonClicked(component) if component == interactiveAgentButton
198                       =>  new InteractiveAgentGUI(blackBoard, blackBoardContent).open()
199                 }
200
201       menuBar = new MenuBar {
202         contents += new Menu("File") {
203           contents += new MenuItem(Action("Exit") {
204             sys.exit(0)
205           })
206           contents += new MenuItem(Action("Interactive Blackboard"){
207               visualBlackboard.open()
208           })
209         }
210       }
211     }
212 }
```

A.4 InteractiveAgentGUI.scala

```
1   import java.awt.{Dimension, Font}
2 import scala.swing._
3 import scala.swing.event.{ValueChanged, ButtonClicked}
4
5 /**
6   * Interactive Agent window of the application.
```

```
7   * This window allows the user to create and run an autonomous agent.
8   * The user can choose which primitive to execute (the system only proposes primitives which
        would lead to a success).
9   * In the event of a bad execution (failure returned) a Dialog is displayed.
10  * A bad execution would happen only if the blackboard content is altered in another view.
11  *
12  * Created by Axel on 23-04-16.
13  */
14 class InteractiveAgentGUI(blackboard:BachTStore, blackboardDisplay:TextArea) extends MainFrame
        {
15
16  var expression = new Expr
17  var agent = new BachTSimul(blackboard)
18
19  // Components definition
20  val titleFont = new Font("Verdana", java.awt.Font.BOLD, 14)
21
22  val agentTitleLabel = new Label(){
23    text = "Agent to be processed"
24  }
25
26  val currentAgentLabel = new Label(){
27    text = "Current Agent: "
28  }
29
30  val currentAgentValue = new Label(){
31    tooltip = "Agent to be executed"
32    text = ""
33  }
34
35  val agentTextArea = new TextArea(){
36    editable = true
37    preferredSize = new Dimension(100, 100)
38    maximumSize = new Dimension(600, 300)
39    resizable = false
40  }
41
42  val submitButton = new Button(){
43    text = "Submit"
44    tooltip = "Submit the new Agent"
45  }
46
47  val agentLine = new BoxPanel(Orientation.Horizontal)
48
49  title = "Interactive Agent"
50  centerOnScreen()
51  preferredSize = new Dimension(700, 400)
52  resizable = false
53
54  contents = new BoxPanel(Orientation.Vertical){
55    contents += new BoxPanel(Orientation.Horizontal){
56      contents += agentTitleLabel
57    }
58
59    // Vertical Space
60    contents += Swing.VStrut(10)
61
62    // Second line: Tokens
63    contents += new BoxPanel(Orientation.Horizontal){
64      contents += agentTextArea
65      contents += Swing.HStrut(15)
66      contents += submitButton
67    }
68
69    // Vertical Space
70    contents += Swing.VStrut(10)
71
72    // Third Line: Buttons
73    contents += new BoxPanel(Orientation.Horizontal){
```

```
74        contents += currentAgentLabel
75      }
76
77      contents += Swing.VStrut(3)
78
79      contents += agentLine
80    }
81
82    listenTo(submitButton)
83    listenTo(blackboardDisplay)
84
85
86    reactions += {
87        case ButtonClicked(component) if component == submitButton
88        =>  if (agentTextArea.text == "") Dialog.showMessage(submitButton, "Please enter the
               value of the agent.", "Error: Agent can't be empty")
89            else{
90                val agentParsed = BachTSimulParser.parse_agent(agentTextArea.text)
91                if (agentParsed == null) Dialog.showMessage(submitButton, "The agent value is
                     incorrect.\nThe agent wasn't recognised","Error: Agent not recognized")
92                else {
93                    print(agentParsed.toString())
94                    expression = agentParsed
95                    agentLine.contents.clear()
96                    displayExpression(expression)
97                    // Refresh the displaying
98                    agentLine.revalidate()
99                }
100           }
101       case ValueChanged(component) if component == blackboardDisplay
102       =>  agentLine.contents.clear()
103           displayExpression(expression)
104           agentLine.revalidate()
105       // If another button then submit is pressed.
106       case ButtonClicked(component)
107       =>  val componentExpression = BachTSimulParser.parse_agent(component.text)
108           componentExpression match {
109             case bacht_ast_primitive(primitive, token)
110             => primitive match {
111               case "tell" => blackboard.tell(token)
112               case "ask" => blackboard.ask(token)
113               case "get" => blackboard.get(token)
114               case "nask" => blackboard.nask(token)
115             }
116             case _ => Dialog.showMessage(component, "Component is not a primitive.", "Error")
117           }
118           // Remove primitive from expression
119           component.visible = false
120           expression = expression.remove(BachTSimulParser.parse_primitive(component.text).
                asInstanceOf[bacht_ast_primitive])
121           blackboardDisplay.text = blackboard.getContent
122
123   }
124
125   override def closeOperation : scala.Unit = this.close()
126
127   /**
128     * Uses pattern matching on case classes to highlight which primitives can be executed.
129     */
130   private def displayExpression(expression:Expr):Unit = {
131       expression match {
132           case bacht_ast_empty_agent() => print("done")
133           case bacht_ast_primitive(primitive: String, token: String) => { agentLine.contents
                += new Label("[")
134               primitive match{
135                 case "tell" => val button = new Button(primitive + "(" + token +")");
                       agentLine.contents += button; this.listenTo(button)
136                 case "get" =>   val density = blackboard.findDensity(token)
```

```
137                                 if (density > 0) {val button = new Button(primitive + "(" +
                                        token + ")"); agentLine.contents +=button; this.listenTo
                                        (button)}
138                                 else agentLine.contents += new Label(primitive + "(" + token +
                                         ")")
139                 case "ask" =>   val density = blackboard.findDensity(token)
140                                 if (density > 0){val button = new Button(primitive + "(" +
                                        token + ")"); agentLine.contents += button; this.
                                        listenTo(button)}
141                                 else agentLine.contents += new Label(primitive + "(" + token +
                                         ")")
142                 case "nask" =>  val density = blackboard.findDensity(token)
143                                 if (density > 0) agentLine.contents += new Label(primitive + "
                                        (" + token + ")")
144                                 else{ val button = new Button(primitive + "(" + token + ")");
                                        agentLine.contents += button; this.listenTo(button)}
145             }
146             agentLine.contents += new Label("]")
147         }
148         case bacht_ast_agent(op: String, agenti: Expr, agentii: Expr)
149         => op match{
150             case ";" => agentLine.contents += {
151                                                 displayExpression(agenti)
152                                                 new Label(agentii.toString)
153                                               }
154             case _ => agentLine.contents +={
155                         displayExpression(agenti);
156                         new Label(op)
157                         displayExpression(agentii)
158                         new Label("")
159                       }
160           }
161         case _ => Dialog.showMessage(submitButton, "An unknown error has occured", "Unknown
                Error")
162     }
163   }
164 }
```

A.5 VisualBlackboard.scala

```
1   import java.awt.image.BufferedImage
2 import java.awt.{Graphics, Component}
3 import javax.swing.{ImageIcon, Icon}
4
5 import scala.swing._
6 import scala.swing.event.ValueChanged
7
8 /**
9   * MainFrame to display the visual blackboard.
10  * This version of the blackboard displays images depending on the content of the blackboard.
11  *
12  * Created by Axel on 07-05-16.
13  */
14 class VisualBlackboard(blackboard:BachTStore, blackboardDisplay:TextArea) extends MainFrame{
15
16   val flowPanel = new GridPanel(9,3)
17
18   contents = new ScrollPane(){
19     contents = flowPanel
20   }
21
22   title = "Visual Blackboard"
23   size = new Dimension(1000,600)
24
25   listenTo(blackboardDisplay)
26
27   reactions +={
28       case ValueChanged(component) if component == blackboardDisplay
29       =>  {
```

```
30              flowPanel.contents.clear()
31              val exprList = blackboard.theStore
32              if (exprList.isEmpty){}
33              else {
34                for (i <- exprList) {
35                  i._2 match {
36                      case 0 => flowPanel.contents += new Label("Token " + i._1, new ImageIcon
                            ("./resources/jar.jpg"), Alignment.Center) {
37                        verticalTextPosition = Alignment.Bottom
38                        horizontalTextPosition = Alignment.Center
39                      }
40                      case 1 => flowPanel.contents += new Label("Token " + i._1, new ImageIcon
                            ("./resources/jar1.png"), Alignment.Center){
41                          verticalTextPosition = Alignment.Bottom
42                          horizontalTextPosition = Alignment.Center
43                      }
44                      case 2 => flowPanel.contents += new Label("Token " + i._1, new ImageIcon
                            ("./resources/jar2.png"), Alignment.Center){
45                          verticalTextPosition = Alignment.Bottom
46                          horizontalTextPosition = Alignment.Center
47                      }
48                      case 3 => flowPanel.contents += new Label("Token " + i._1, new ImageIcon
                            ("./resources/jar3.png"), Alignment.Center){
49                          verticalTextPosition = Alignment.Bottom
50                          horizontalTextPosition = Alignment.Center
51                      }
52                      case 4 => flowPanel.contents += new Label("Token " + i._1, new ImageIcon
                            ("./resources/jar4.png"), Alignment.Center){
53                          verticalTextPosition = Alignment.Bottom
54                          horizontalTextPosition = Alignment.Center
55                      }
56                      case _ => flowPanel.contents += new Label("Token "+i._1, new ImageIcon("
                            ./resources/jar5.png"), Alignment.Center){
57                          verticalTextPosition = Alignment.Bottom
58                          horizontalTextPosition = Alignment.Center
59                      }
60                  }
61                }
62              }
63              blackboardDisplay.revalidate()
64          }
65    }
66
67    override def closeOperation : scala.Unit = this.close()
68  }
```