



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
3.2	Deuxième Question	7

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'interpréteur Bach-T dont le code source nous a 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 quatre 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* définit 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 ***bach_t_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 entraîne l’arrêt de l’application. Toute modification du tableau par un agent interactif ou autonome sera affiché sur cette page.

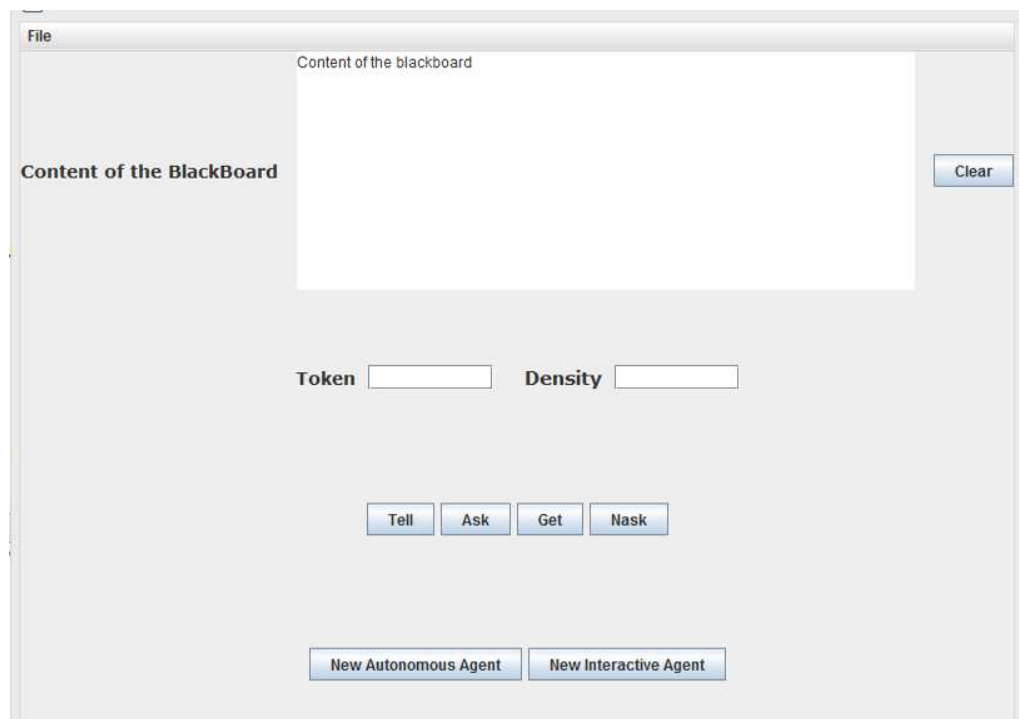


FIGURE 1 – gui.scala - Page d’accueil

3.1.2 AutonomousAgentGUI.scala

La classe ***AutonomousAgentGUI.scala*** définit 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.

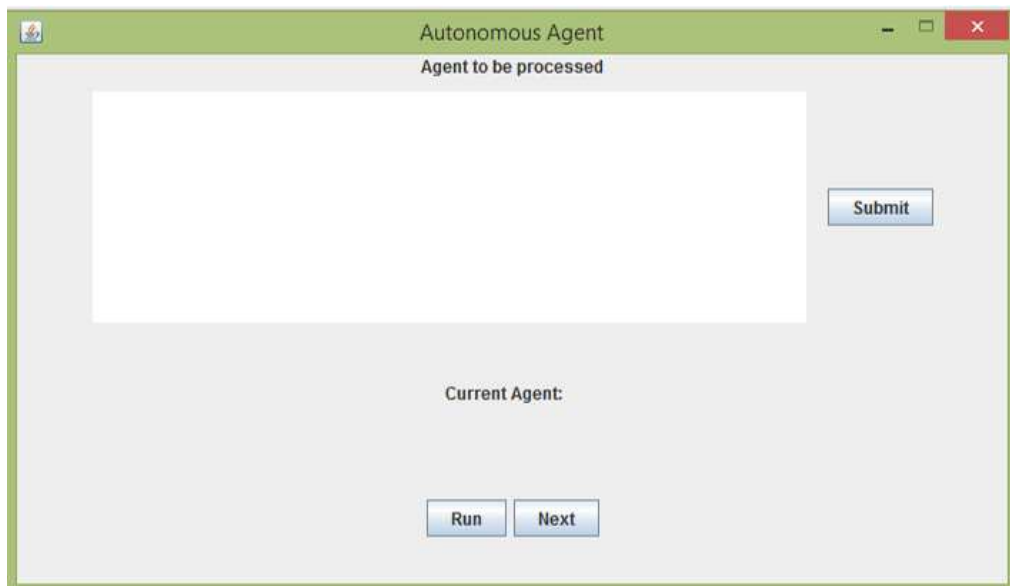


FIGURE 2 – AutonomousAgentGUI.scala - Ecran Agent Autonome

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 renvoie 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

3.2 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 choisis un affichage simple : un conteneur (un bocal en l’occurrence) 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).

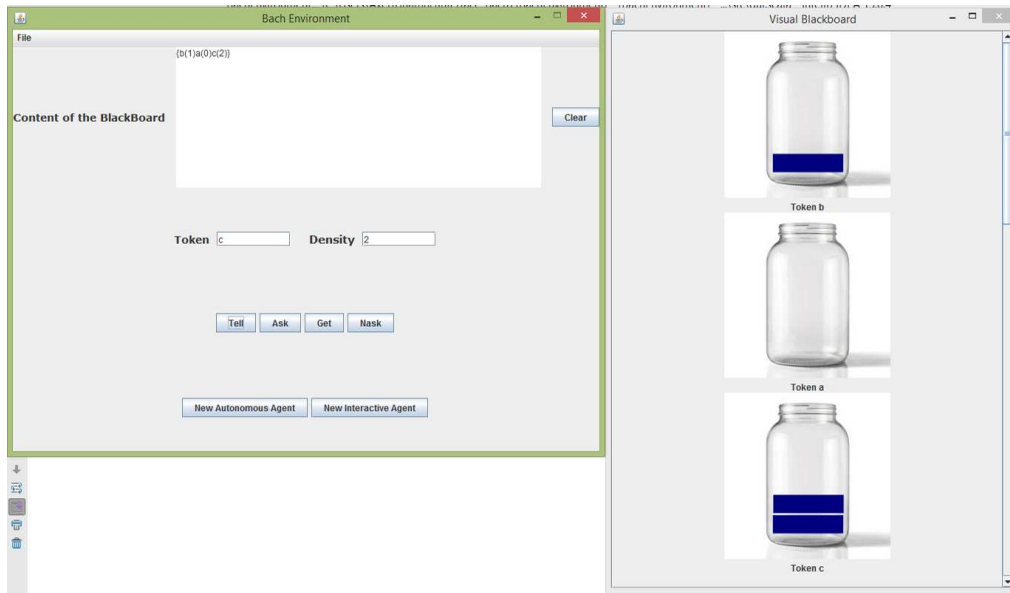


FIGURE 4 – VisualBlackboard.scala - 0 à 2 tokens

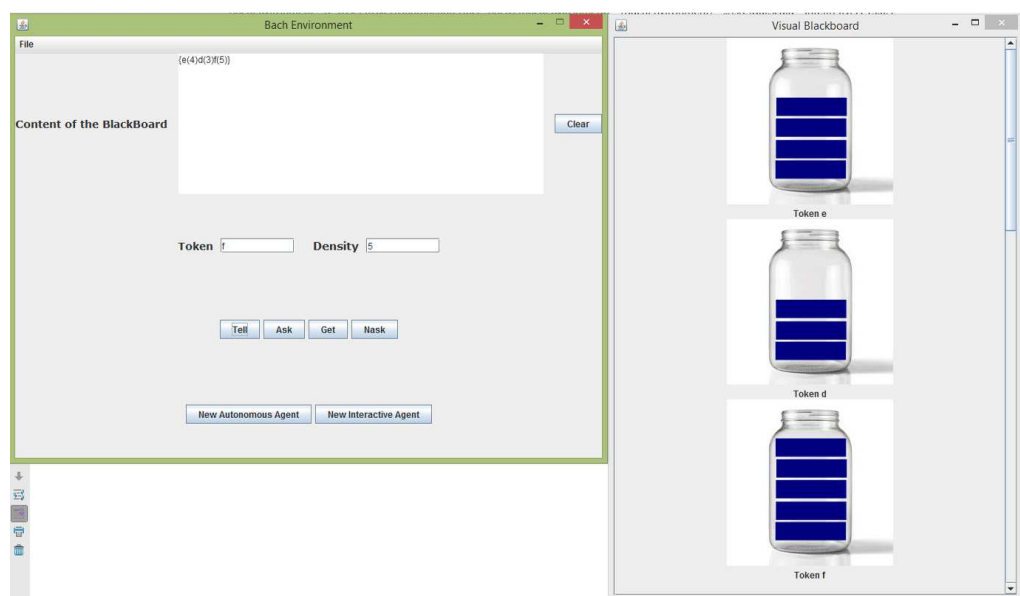


FIGURE 5 – VisualBlackboard.scala - 3 à 5 tokens