Mehdi Haddad – Florent Catiau-Tristant

Yoann Boyère – Romain Ringenbach

Le 6 mars 2015

Rapport Machine de Turing

IUT de Vannes

A l'attention de M.Fleurquin

Table des matières

Mehdi Haddad – Florent Catiau-Tristant

Yoann Boyère – Romain Ringenbach

Le 6 mars 2015

I) Architecture de l'application

```
/src
  |-- ctrl
  |    `-- TMCtrl
  |    `-- TMListener
  |-- data
  |    `-- Machine
  |    `-- Transition
  |    `-- TuringIO
  |    `-- TuringSyntaxe
  |-- view
  |    `-- TMView
  |    `-- Tape
  |    `-- TransitionTableModel
  |-- test
  |    `-- TestConfigurations
```

Pour effectuer cette application, nous avons choisi d'utiliser le modèle MVC.

1) La partie « data »

Le package 'data' représente le fonctionnement interne d'une machine de Turing. Les *'Transitions'* étant le cœur de la machine, une classe entière y fut dédiée.

La *'Machine'* contient toutes les données entrées dans le fichier de configuration.

Les fichiers *'TuringIO'* et *'TuringSyntaxe'* permettent la lecture de ce fichier et d'en vérifier la conformité, selon les critères syntaxiques prédéfinis.

2) La partie « view »

Ce package constitue l'interface visuelle de l'application. *'TMView'* représente l'interface générale de la fenêtre : boutons, listes, menus, etc.

'Tape' est une classe dédiée à la représentation du ruban de la machine. Ce ruban étant dynamique (affichage de la tête de lecture, modification des caractères), il nécessitait un fichier à part entière.

La classe *'TransitionTableModel'* est simplement utilisée pour personnaliser l'affichage du tableau représentant la liste des transitions de la machine.

3) La partie « ctrl »

Cette partie est divisée en deux classes :

- *'TMListener'*, implémentant toutes les actions liées à l'interface graphique.
- *'TMCtrl'*, permettant les différents lancements de la machine.

Le contrôle de la machine récupère les données depuis *'Machine'* et effectue le lancement du programme tout en mettant à jour l'interface. Il initialise, met à jour, et remet à zéro la machine.

Il constitue donc le lien entre les données et l'interface.

II) Choix d'implantation

1) Le ruban

Le ruban est affiché sur un panel dédié. C'est une liste de JLabel, dessinée sous forme de cases (grâce aux bordures). Le ruban étant normalement infinie sur la droite, nous avons décidé de limiter les cases vides sur la droite, une implémentation illimitée n'étant pas optimisée. Le nombre de cases vides sur la droite est par défaut de 50, mais il est possible de le modifier dans le menu paramètres.

Chaque case est numérotée à partir de 1 afin de mieux se repérer sur le ruban, surtout lors du scroll automatique du ruban. En effet, l'interface reste fixée sur la tête de lecture, afin de ne pas la perdre de vue elle dépasse les cases initialement affichées.

La tête de lecture est colorée en jaune, par défaut, afin de la repérer (La couleur est modifiable dans les paramètres).

D'un point de vue programmation, la tête de lecture est représentée par un entier comprit entre 0 et la taille du ruban. Lors d'un déplacement à droite, l'entier est incrémenté de 1. Il est décrémenté lors d'un déplacement à gauche, sauf s'il est égal à zéro. Cet entier équivaut à l'index de la case où la tête doit être placée.

2) La fonction de transition

Les transitions sont effectuées par la classe *'TMCtrl'*. Une simple méthode 'doTransition()' lance la prochaine transition, selon l'état courant et le symbole lu sur la tête de lecture. Elle modifie ainsi les données courantes du programme, et l'interface graphique.

Les données sont entièrement chargées dans la classe *'Machine'*. A partir de celle-ci, les transitions sont effectuées une à une, en conservant dans des attributs l'état courant, le symbole lu par la tête de lecture, la transition venant d'être effectuée et l'état suivant de cette transition.

Depuis ces attributs, la méthode récupère la transition suivante (qui est unique) et l'effectue en déplaçant la tête de lecture sur le ruban, écrivant le nouveau symbole et plaçant la machine dans l'état suivant.

3) Le format des fichiers

a) Configurations

Les fichiers de configuration initiale de la machine ont tous la même forme. Chaque partie de la configuration est séparée par des « balises » afin de le rendre à la fois clair pour l'utilisateur et efficace à la récupération des données.

Les espaces, tabulations, et retour chariots ne sont pas pris en compte à l'intérieur d'une balise (on peut ainsi écrire les valeurs en ligne, colonne, ou les deux).

Une balise se présente de la forme **:balise:**. Le fichier en comporte 7, sans compter la balise indiquant la fin du fichier :

- **states :** (Attention ! Cette balise n'inscrit pas de « : » avant, étant la première balise)

La liste des états du programme. Indique aussi s'ils sont stoppant ou non (on ajoute un S majuscule avant la virgule).

Les états sont des chaines de caractères et sont séparés par une virgule.

- **machine_alphabet :**

La liste des caractères que l'utilisateur peut entrer sur le ruban initial.

Les caractères ne peuvent être des chaines de caractères et ne sont pas séparés par une virgule.

- **tape_alphabet :**

La liste des caractères que la machine de Turing peut inscrire sur le ruban. Il est nécessaire que cet alphabet contienne tous les caractères de l'alphabet du ruban.

Cette balise a les mêmes contraintes que l'alphabet précédent.

- **transitions :**

La liste des transitions du programme. Elles sont écrites de la forme :
[état courant] [symbole lu] [état suivant] [symbole écrit] [direction]
et sont séparées par une virgule.

- **init_state :**

C'est l'état initial de la machine. Il doit évidemment se trouver dans la liste des états indiqué précédemment.

- **accept_state :**

C'est l'état acceptant de la machine. Il ne se trouve pas dans la liste d'état et doit obligatoirement se trouver dans la liste de transitions en tant qu'état suivant.

Exemple de configuration

```
states:

 q0 S,

:machine_alphabet:

 0 1

:tape_alphabet:

 0 1 _

:transitions:

q0 0 q0 _ >,
q0 1 q0 _ >,
q0 _ acc _ >,

:init_state:

  q0

:accept_state:

  acc

:reject_state:

:end
```

- **reject_state :**

C'est l'état rejetant de la machine. Il ne se trouve pas dans la liste d'état et peut ne pas être inscrit dans la liste de transitions.

- **end :** (Attention ! Cette balise n'inscrit pas de « : » après, étant la dernière balise)

Cette balise indique la fin du fichier.

b) Résultat

Les résultats sont affichés dans un fichier *Configurations.txt*, créé par défaut dans le répertoire parent de l'application (Le répertoire de sauvegarde est configurable dans les paramètres). Il inscrit, ligne par ligne, l'ensemble de toutes les configurations par laquelle la machine est passée.

Une ArrayList<String> de configuration est formée pendant l'exécution du programme. Celle-ci est alors écrite dans le fichier à la fin de l'exécution complète (arrivée dans un état acceptant ou rejetant).

Elles sont écrites de la forme suivante :

[Mot à gauche de la tête] [État courant] [Symbole courant + mot à droite de la tête]

Exemple de configuration

```
Le 05/03/15 19:13
Configurations du
programme FoncTrans.txt :

q0 0 1 _
b q2 1 _
b 1 q2 _
b 1 # q5 _
b 1 q6 # 0 _
b q6 1 # 0 _
q6 b 1 # 0 _
0 q7 1 # 0 _
0 a q3 # 0 _
0 a # q3 0 _
0 a # 0 q3 _
0 a # q4 0 1 _
0 a q4 # 0 1 _
0 q4 a # 0 1 _
0 1 q7 # 0 1 _
```

4) Fonctionnement des différents modes

Tous les modes de fonctionnement sont basés sur la seule méthode d'exécution de transition. Cette méthode met à jour données et interface, en plus de faire des vérifications d'arrêts de la machine. Chaque mode initialise le ruban si ce n'est pas déjà fait.

Plusieurs booléen indiquent à chaque transition effectuée l'état de la machine :

- **started :**

  Indique si le programme a déjà commencé ou non. La machine n'a donc pas à s'initialiser au lancement d'un mode, elle continue là où elle s'était arrêtée.

- **ended :**
  Indique si le programme est arrivé à un état final ou non. Empêche d'effectuer une transition alors que le programme ne peut plus avancer.
- **running :**
  Indique si le programme est en cours d'exécution. Uniquement utilisé pour le mode ininterrompu.
- **stop :**
  Indique si le programme doit être lancé en mode « stop » ou non. Uniquement utilisé pour les modes ininterrompu et stoppant.

a) La gestion démarrage ininterrompu

Le démarrage ininterrompu du programme (lancé depuis le bouton « démarrer ») effectue toutes les transitions jusqu'à un état final, ou un arrêt provoqué par l'utilisateur.

Afin de le rendre plus fluide, ce mode est exécuté dans un Thread, qui boucle tant que le programme doit continuer (à l'aide des booléens présentés ci-dessus).

Un délai (Thread.sleep) est placé entre deux transitions, permettant ainsi de régler le temps d'exécution du programme et le rendre plus agréable à l'œil.

b) La gestion transition par transition

Lors du clic sur le bouton « état par état », le programme effectue alors une seule transition (exécution de la méthode 'doTransition'). La méthode gère elle-même si le programme peut continuer ou non.

c) La gestion étape par étape

Dans le fichier de configuration, des états peuvent être marqués comme étant des états « stoppant ». Ces états, lors d'un lancement de la machine en mode « Etape par étape » arrêtent l'exécution du programme (pause).

Ce mode ne fait que lancer le mode ininterrompu (« Démarrer ») avec une condition supplémentaire (booléen « stop » à true). Encore une fois, c'est la méthode 'doTransition' qui vérifie si le programme doit s'arrêter ou non, à l'aide du booléen l'indiquant.

d) La gestion de l'arrêt en cours d'exécution

Le bouton d'arrêt ne fait que placer le booléen « running » à faux. Le Thread d'exécution s'arrête alors de boucler (Valable pour le mode ininterrompu et le mode étape par étape).

5) La lecture du fichier de configuration

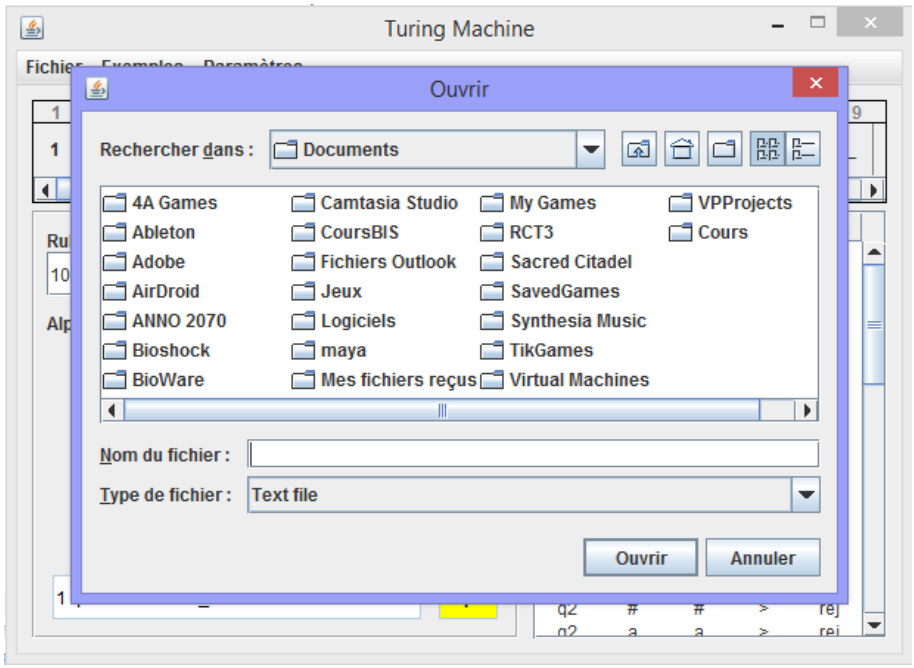
Le fichier de configuration est entièrement parcouru et vérifié. La classe *'TuringSyntaxe'* lit le fichier, et lance une exception correspondant à chaque type d'erreur possible. Un pop-up est alors affiché sur l'interface, indiquant qu'elle erreur est présente dans le fichier chargé.

Cette vérification s'effectue à grand coups de regex et de conditions.

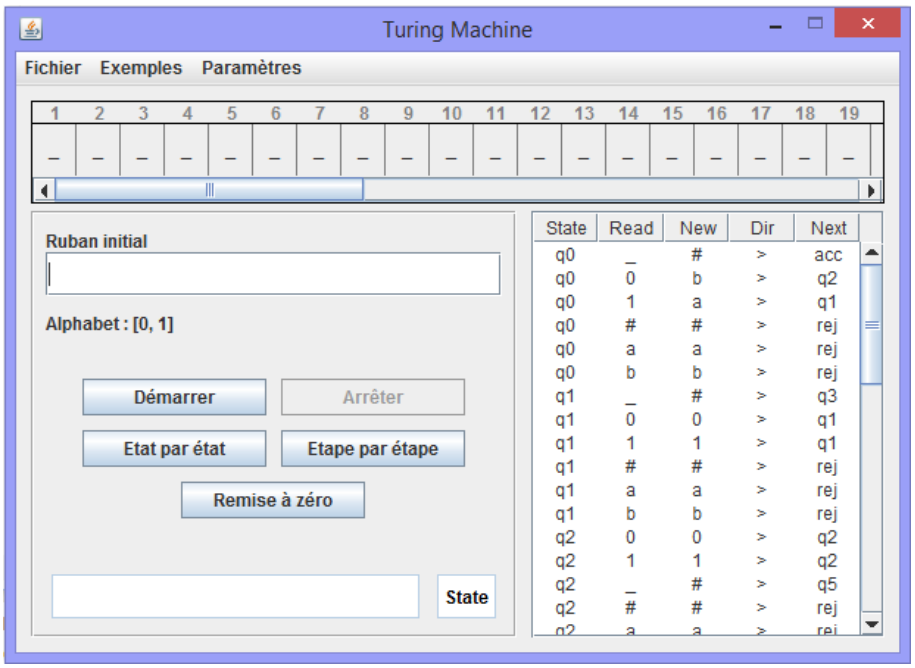
La classe *'TuringIO'* permet de récupérer le fichier de configurations, et de sauvegarder le fichier résultant de l'exécution d'un programme. Ce fichier est écrasé à chaque exécution.

Ces deux classes de lectures sont des singletons.

III) Copies d'écran



Ouverture d'un pop-up pour sélectionner le fichier de configuration.



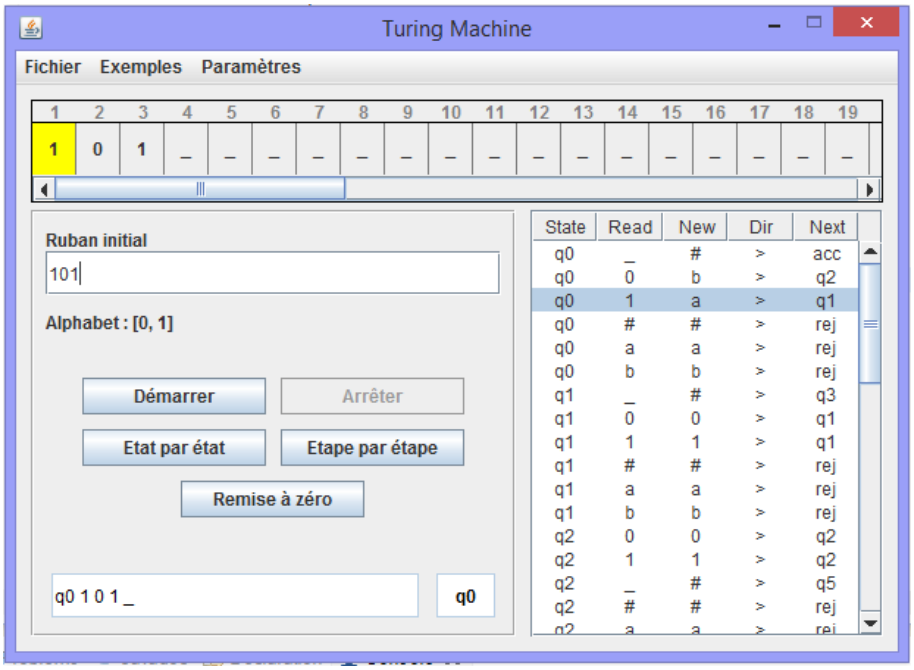
Lancement de l'application.

Le fichier a été chargé et vérifié. Il ne reste plus qu'à entrer le ruban initial !

Mehdi Haddad – Florent Catiau-Tristant

Yoann Boyère – Romain Ringenbach

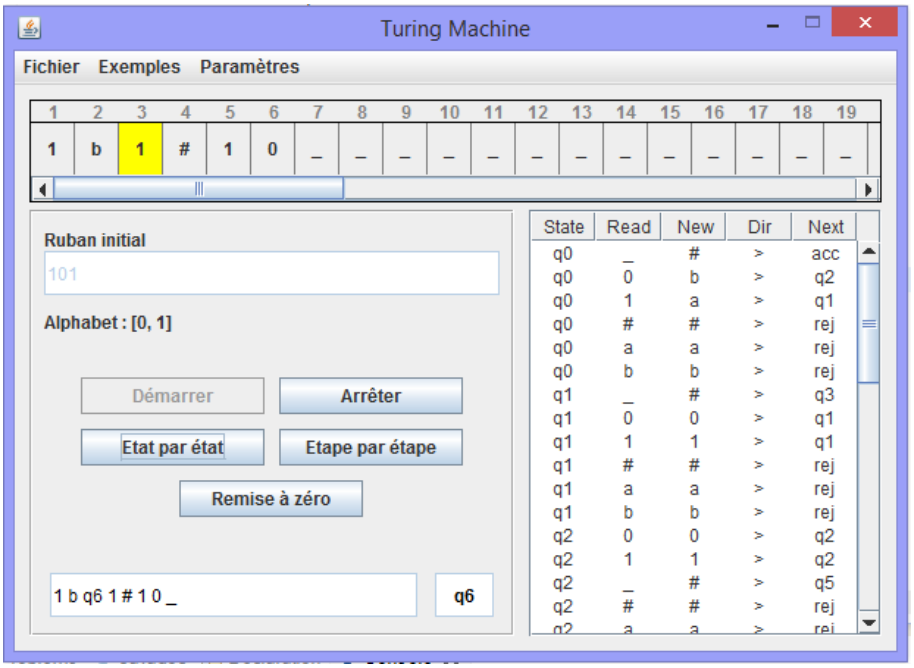
Le 6 mars 2015



Le ruban est initialisé avec le mot entré.

L'état initial est « q0 », la transition allant s'exécuter est sélectionner dans le tableau, et la configuration initiale est affichée.

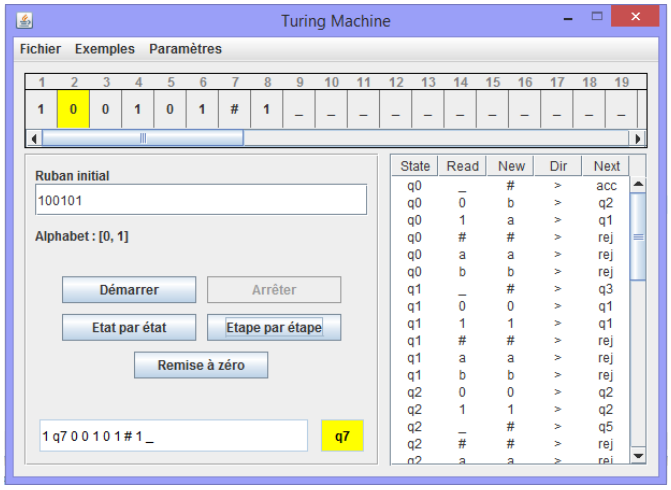
Une fois le programme lancé, la tête (ici en jaune) se déplace. La configuration change et on ne peut plus lancer « démarrer ».



Mehdi Haddad – Florent Catiau-Tristant

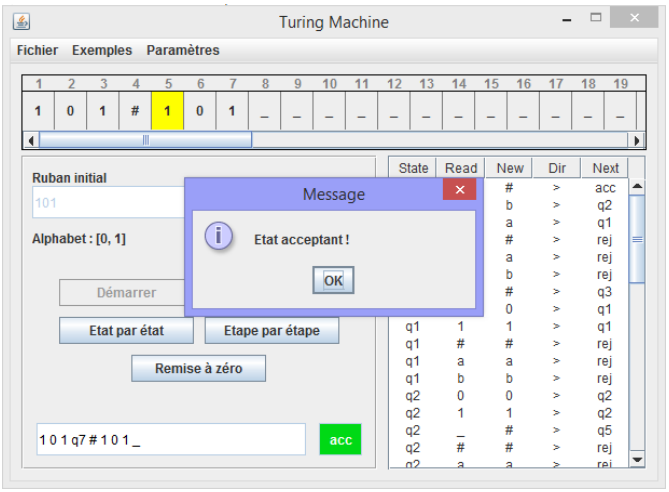
Yoann Boyère – Romain Ringenbach

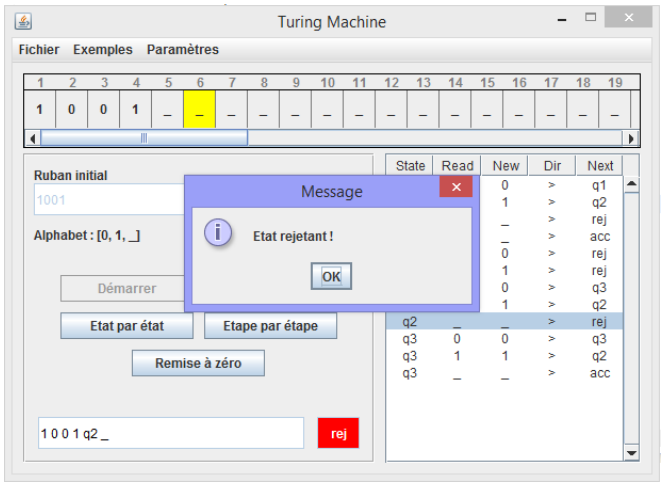
Le 6 mars 2015



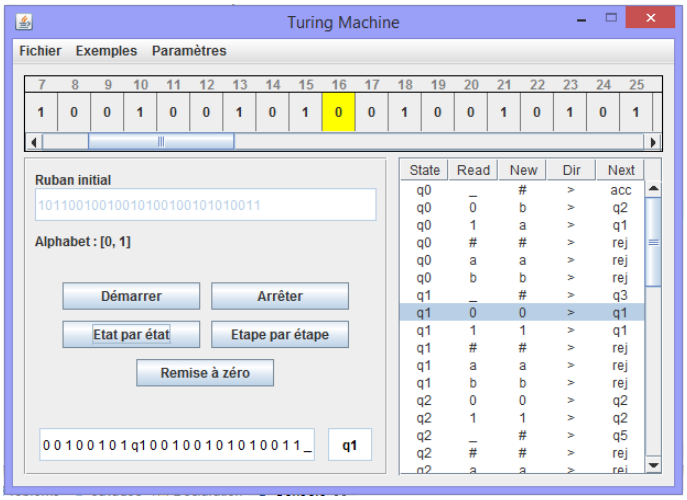
En mode « Etape par Etape », le programme s'arrête sur les états stoppant. Le label coloré en jaune indique l'état stoppant et l'arrêt de la machine.

Une fois l'état acceptant atteint, le label se colore en vert, et un pop-up indique la fin du programme.

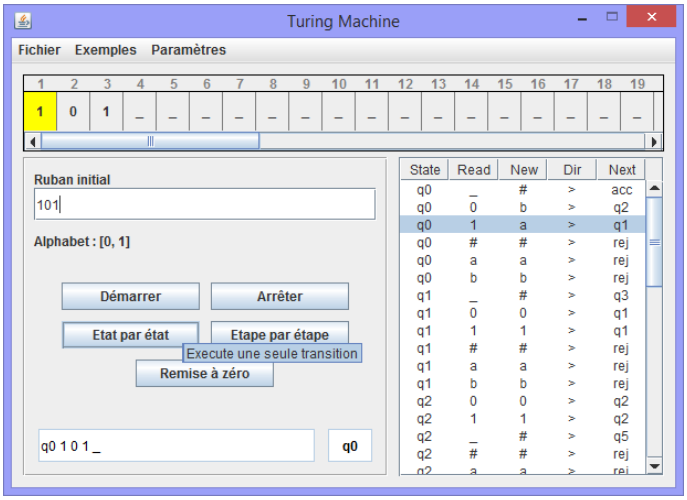




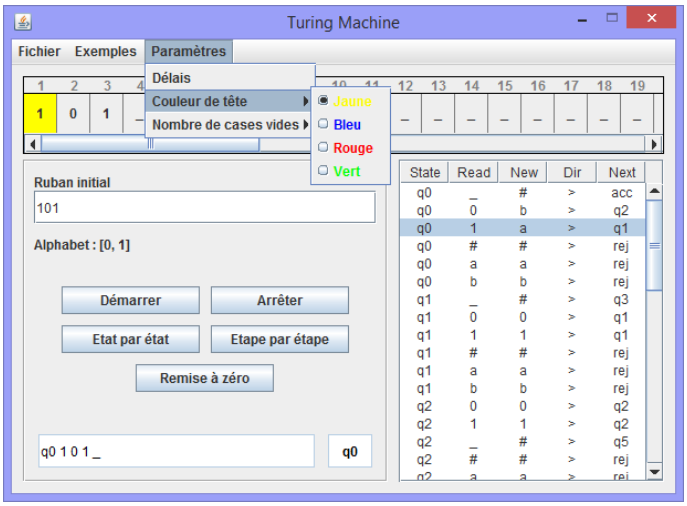
Une fois l'état rejetant atteint, le label se colore en rouge, et un pop-up indique la fin du programme.



Si la tête se déplace à droite, le panel scroll automatiquement, afin de toujours garder la tête affichée.



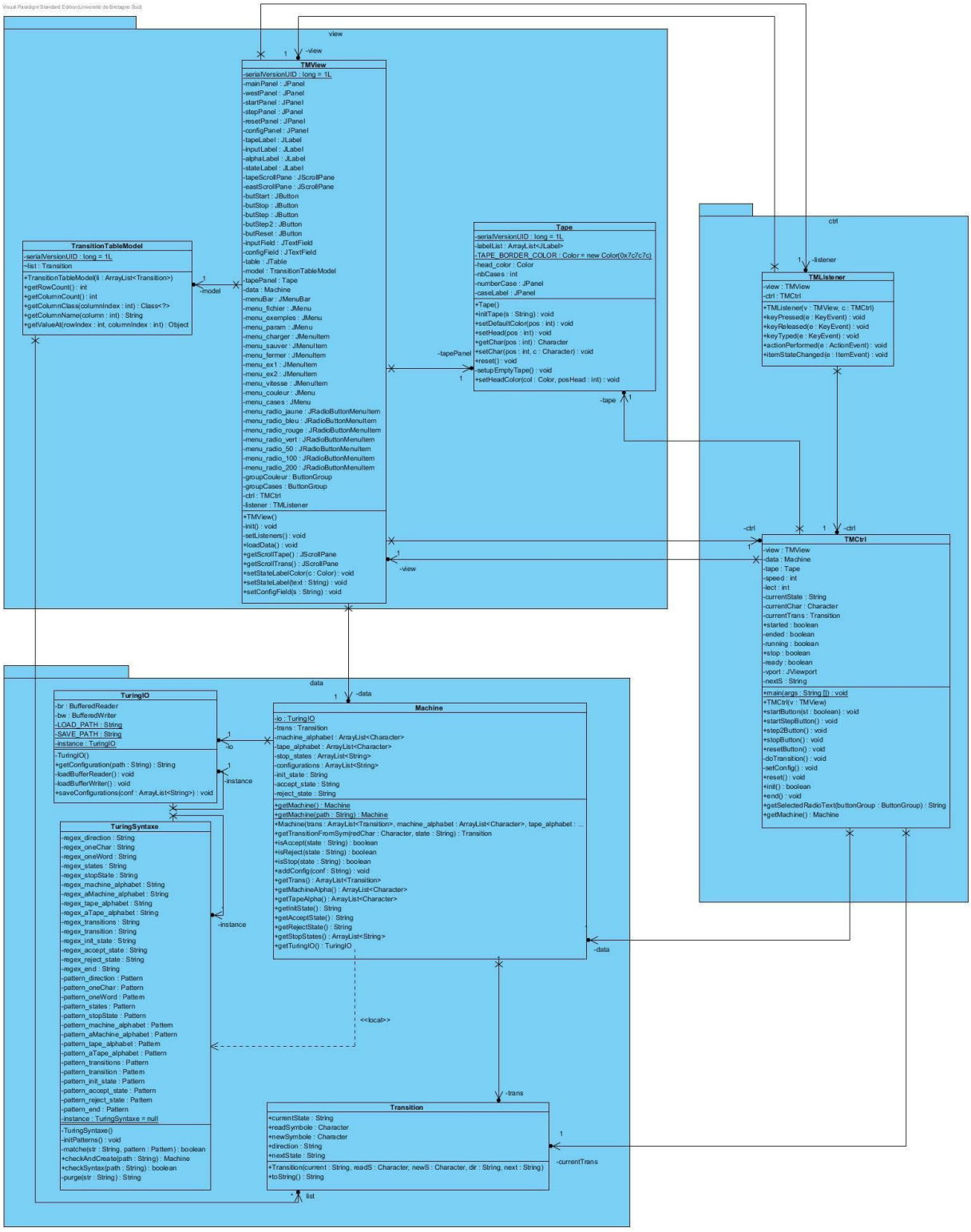
Des tooltips ont été ajoutés afin de décrire l'action de chaque composant.



Certains paramètres permettent de changer la couleur de la tête, le délai d'exécution, ou encore le nombre de cases vides à la fin du ruban.

Il est possible de charger deux exemples prédéfinis.

IV) Diagramme de classes



V) Tests effectués

Les seuls tests effectués sont sur l'algorithme de vérification de syntaxe. Ces tests s'assurent que les erreurs de syntaxe sont détectées et que les bonnes exceptions sont lancées.

VI) Erreurs restantes dans l'application

Le programme contient toujours quelques erreurs :

- La bordure gauche de la première case disparait après un déroulement du ruban.
- Selon la vitesse d'exécution, le bouton d'arrêt met un certain temps à réagir. (Décalage d'une transition)
- Le chargement d'un fichier de configuration prendre entre 0,5 et 2 secondes, laissant l'utilisateur sans indications du fonctionnement du programme.
- Les exemples ne fonctionnent que si les fichiers liés se trouvent dans le même répertoire que le .jar.
- Le tableau de transition ne se déroule pas automatiquement, contrairement au ruban, lors de la sélection de la transition suivante. En effet, celui-ci ne fonctionne pas lors de l'exécution du programme dans un Thread (Fonctionne en mode « transition par transition »).
- Le texte « état par état » est incorrecte, il s'agit d'une exécution « transition par transition »
- Le raccourci clavier Ctrl + C (pop-up de chargement) ne fonctionne pas si le champ principal a le focus. Cependant, le raccourci Ctrl + S fonctionne malgré le focus.