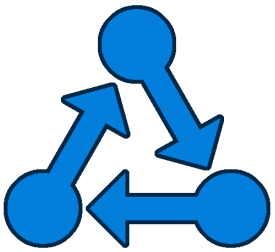
Rapport Final Projet S3

Interface pour automates et machine de Turing



Réalisé par
Emerick BIRON
Lénaïs DESBOS
Albin MORET
Alexandre ROUSSEL

Sous la direction de
Matthieu ROSENFELD

Pour l'obtention du DUT Informatique

Année universitaire **2021-2022**

Remerciements

Nous souhaitons remercier tout particulièrement notre tuteur de projet Matthieu Rosenfeld, qui nous a accompagnés et conseillés tout au long du projet. Il a su nous guider à travers les difficultés techniques et a pris le temps d'apporter des critiques constructives à notre travail et de répondre à nos nombreuses questions. Grâce au temps qu'il nous a accordé chaque semaine nous avons acquis un rythme de travail satisfaisant pour la réalisation de ce projet.
Nous remercions aussi M. Valicov qui nous a apporté son aide sur l'utilisation des classes génériques dans notre application.

Résumé

L’objectif du projet “interface pour automates et machines de Turing” est de réaliser un logiciel pour l’utilisation d’automates et de machines de Turing.
Cette interface permet de réaliser, ou d’importer, un automate ou une machine de Turing afin de visualiser leur fonctionnement et de vérifier s’il reconnaît un mot donné par un utilisateur.
Le logiciel supporte deux formats de fichiers qui lui sont propres, .atmt et .mt qui contiennent des données relatives aux machines comme les états, les transitions et leur position. Ces formats de fichiers permettent d’importer et de sauvegarder les créations des utilisateurs.
La mise en œuvre graphique de ce projet a nécessité d'utiliser le langage java avec l'utilisation du framework javafx pour la partie interface.

Mots clés : Logiciel, interface, automates, Turing, java, javafx

Summary

The aim of the project “interface for automata and Turing machines” is to produce software in order to use automata and Turing machines.
This interface will allow to realize or import an automata or a Turing machine so as to visualize their functioning, as well as test if it recognizes a word given by the user.
Our software supports two unique file formats, .atmt and .mt which contain machine data such as states, transitions and their position. These file formats that we have created allow us to import and save user creations.
To create this project, we chose to use the java language with the framework javafx for the interface.

Key words : Software, interface, automata, Turing, java, javafx

Sommaire

Table de figures

Glossaire

Les termes définis dans ce glossaire sont identifiables dans le corps du texte au moyen d'un
astérisque (*).

Java : Java est un langage de programmation utilisé à grande échelle dans le monde.

Framework : Un framework est une librairie composée d'objets aidant à la création d'autres logiciels.

Interface graphique : Une interface graphique est la représentation pour l'utilisateur du fonctionnement d'un logiciel.

JavaFX : Framework permettant de développer des interfaces graphiques en Java.

Automate : Un automate est un objet mathématique, composé d'états et de transitions, qui réalise une suite d'actions programmées par l'homme.

Machine de Turing : Une machine de Turing est une machine inventée par Alan Turing et qui se rapproche actuellement du fonctionnement de calcul des ordinateurs, une machine de Turing est un modèle du fonctionnement des appareils de calculs

Introduction

La calculabilité est une branche de la logique et de l'informatique théorique permettant de comprendre la limite de ce qui est calculable. Cette branche cherche à faire le lien entre les deux sujets cités ci-dessus en cherchant des problèmes solubles par l'utilisation d'algorithmes et en les appliquant à des problèmes mathématiques.

Afin d'aider à cette recherche, de nombreux modèles de calcul ont été développés, parmi eux nous pouvons, par exemple retrouver :

- La machine de Turing*, qui est un modèle abstrait du fonctionnement des appareils mécaniques de calcul, tel un ordinateur.
- Les automates* finis qui, quant à eux, peuvent modéliser de nombreux concepts et de nombreux problèmes tels que l'analyse et la reconnaissance de langues ou encore la manipulation de textes.

Ces modèles permettent de démontrer si un problème donné est calculable ou non, en temps fini ou infini. En somme, ces modèles ont été indispensables au développement de machines, comme l'ordinateur par exemple.

Le projet qui nous a été confié consiste donc à réaliser un logiciel avec une interface permettant de créer facilement des automates finis et des machines de Turing et de simuler le fonctionnement de ces derniers.

Dans un premier temps, nous allons donc aborder le cahier des charges relatif à la réalisation de ce projet.
Puis nous allons procéder au rapport technique ainsi qu'aux résultats de notre projet.
Enfin nous allons nous intéresser à la notion de gestion qui a enveloppé notre projet et finir par notre conclusion à la suite de laquelle vous pourrez retrouver notre bibliographie.

1. Cahier des charges

Dans cette partie nous allons présenter et analyser notre sujet, en commençant par l'analyse du contexte dans lequel il se pose et une analyse de l'existant, puis l'analyse des besoins fonctionnels et non fonctionnels.

1.1. Présentation du sujet et analyse du contexte

Après avoir analysé le sujet, nous procéderons à une analyse de l'existant et du contexte.

Nous avons eu pour sujet "Interface pour automates et machines de Turing".
Le but de ce projet est de développer une interface graphique permettant de créer facilement un automate (et une machine de Turing si le temps le permet), puis de l'exécuter.

La première partie de notre sujet est les automates.
Il est tout d'abord important de savoir qu'il existe différents types d'automates (automates finis déterministes, automates finis non-déterministes, automates à pile, …).
Nous avons trouvé une description informelle de ce qu'est globalement un automate:

> " Un automate *s'exécute* lorsqu'il reçoit une séquence d'*entrées* en *pas* ou pas de *temps* discrets (individuels). Un automate traite une entrée choisie dans un ensemble de *symboles* ou de *lettres*, qui est appelé un *alphabet d'entrée* . Les symboles reçus par l'automate en entrée à n'importe quelle étape sont une séquence de symboles appelés *mots* . Un automate a un ensemble d'*états*. A chaque instant pendant une course de l'automate, l'automate est *dans l'*un de ses états. Lorsque l'automate reçoit une nouvelle entrée, il passe à un autre état (ou *transitions*) basé sur une *fonction de transition* qui prend l'état précédent et le symbole d'entrée actuel comme paramètres. En même temps, une autre fonction appelée fonction de *sortie* produit des symboles à partir de l' *alphabet de sortie*, également en fonction de l'état précédent et du symbole d'entrée actuel. L'automate lit les symboles du mot d'entrée et effectue une transition entre les états jusqu'à ce que le mot soit lu complètement, s'il est de longueur finie, moment auquel l'automate *s'arrête* . Un état dans lequel l'automate s'arrête est appelé l'*état final* .
>
> Pour étudier les séquences d'état / d'entrée / de sortie possibles dans un automate en utilisant la théorie du langage formel, une machine peut se voir attribuer un *état de départ* et un ensemble d' *états d'acceptation* . Ensuite,

> selon qu'une exécution à partir de l'état de départ se termine dans un état d'acceptation, on peut dire que l'automate *accepte* ou *rejette* une séquence d'entrée. L'ensemble de tous les mots acceptés par un automate est appelé le *langage reconnu par l'automate*. Un exemple familier d'une machine reconnaissant une langue est une serrure électronique qui accepte ou rejette les tentatives de saisie du code correct. ” [14]

Le sujet ne précisant pas quel type d’automates nous devions modéliser, notre tuteur nous a expliqué plus en détail ses attentes et que le premier automate à modéliser devait être l’automate fini non-déterministe. Nous avons donc commencé par les automates finis déterministes, et une fois que nous avions réussi, nous avons étendu jusqu’aux automates finis non-déterministes.

La deuxième partie importante du sujet est les machines de Turing.
Une machine de Turing est principalement une machine de calcul. Nous avons trouvé une définition qui décrit assez simplement de quoi est composée une machine de Turing :

> “ La mise en œuvre concrète d'une machine de Turing est réalisée avec les éléments suivants :
>
> 1. Un « ruban » divisé en cases consécutives. Chaque case contient un symbole parmi un alphabet fini. L'alphabet contient un symbole spécial « blanc », et un ou plusieurs autres symboles. Le ruban est supposé être de longueur infinie vers la gauche ou vers la droite, en d'autres termes la machine doit toujours avoir assez de longueur de ruban pour son exécution. On considère que les cases non encore écrites du ruban contiennent le symbole « blanc ».
> 2. Une « tête de lecture/écriture » qui peut lire et écrire les symboles sur le ruban, et se déplacer vers la gauche ou vers la droite du ruban.
> 3. Un « registre d'état » qui mémorise l'état courant de la machine de Turing. Le nombre d'états possibles est toujours fini, et il existe un état spécial appelé « état de départ » qui est l'état initial de la machine avant son exécution.
> 4. Une « table d'actions » qui indique à la machine quel symbole écrire, comment déplacer la tête de lecture, et quel est le nouvel état, en fonction du symbole lu sur le ruban et de l'état courant de la machine. Si aucune action n'existe pour une combinaison donnée d'un symbole lu et d'un état courant, la machine s'arrête. ” [1]

Nous nous étions renseignés sur les machines de Turing avant la réunion, mais nous n'avions pas vraiment compris son fonctionnement. Notre tuteur nous a donc expliqué étape par étape le mode de fonctionnement d’une machine de Turing et a pu nous donner en même temps les attentes qu’il avait sur cette modélisation.

1.1.1 Formalisation mathématique du sujet

Pour un automate, les données du projet sont caractérisées par :

- un ensemble de n états {e1, .., en}
- un ensemble de d états initiaux {i1, .., id} ∈ {e1, .., en} (automates)
- un état initial {ei} ∈ {e1, .., en} (machines de Turing)
- un ensemble de t états finaux {f1, .., ft} ∈ {e1, .., en}
- un ensemble de k transitions {t1, ..., tk} composé d'un état de départ {d} ∈ {e1, .., en}, d'un état d'arrivé {a} ∈ {e1, .., en}, et d'une étiquette {q}
- m : le mot à analyser composé d'un ensemble de x lettres {l1,...,lx}

Pour une machine de Turing, les données du projet sont caractérisées par :

- un ensemble de n états {e1, .., en}
- un état initial {ei} ∈ {e1, .., en}
- un ensemble de t états finaux {f1, .., ft} ∈ {e1, .., en}
- un ensemble de k transitions {t1, ..., tk} composé d'un état de départ {d} ∈ {e1, .., en}, d'un état d'arrivé {a} ∈ {e1, .., en}, et d'une étiquette de base {qb}, d'une nouvelle étiquette {qn} et d'un mouvement (gauche ou droite)
- m : le mot en binaire ou en décimal à analyser composé d'un ensemble de x chiffres {c1,...,cx}
- un curseur indiquant où nous somme sur le mot

Problème 1 : l'automate analyse un mot

- entrée : n états, d états initiaux (avec 0≤d≤n), t états finaux (avec t≤n), k transitions et m (n, d, t, k et m définis par l'utilisateur)
- sortie : analyser le mot m lettre par lettre avec li la lettre analysée
  - si li = q d'une transition partant de l'état actif, alors on désactive d et on active a
  - sinon on désactive l'état actif

Une fois m complètement analysé, on regarde les états finaux :

- si au moins l'un d'eux est actif, le mot est reconnu par l'automate
- sinon l'automate ne reconnaît pas le mot

Problème 2 : la machine de Turing analyse un mot

- entrée : n états, un ou zéro état initial, t états finaux (avec t≤n), k transitions et m (n, t, k et m définis par l'utilisateur)
- sortie : analyser le mot m chiffre par chiffre avec ci le chiffre analysé
  - si ci = qb d'une transition partant de l'état actif, alors on remplace ci par qn dans m, on déplace le curseur dans la direction indiqué par la transition, puis on désactive d et on active a
  - sinon on passe au chiffre suivant sans rien modifier

Une fois m complètement analysé, on affiche le nouveau mot obtenu.

1.1.2 Analyse de l'existant

Afin de nous donner une idée de comment devrait fonctionner et à quoi devrait ressembler notre logiciel, nous avons cherché s'il existait déjà des logiciels ou sites internet ou tout autre élément que nous pouvions trouver qui avait les mêmes fonctions que le logiciel que nous devions créer. Puis nous avons procédé à l'analyse de l'existant à l'aide de nos trouvailles.

Nous avons trouvé des sites internet et des logiciels téléchargeables permettant de simuler des automates et/ou des machines de Turing. Il existe également des machines de Turing programmables qu'il est possible d'acheter.

Nous avons analysé les sites et logiciels en fonction de leurs visuels, et des fonctionnalités qu'ils proposent. Vous pouvez retrouver nos critères d'analyse dans le tableau ci-dessous.

	Automaton Simulator: site internet [2]	Automaton Simulator: logiciel [3]	Turing Machine Simulator [15]	Online Turing Machine [11]
Accessibilité	Pas très compréhensible, aucune aide	Création d'un automate assez intuitive et fichier d'aide décrivant le fonctionnement	Manuel d'utilisation tout en haut de la page, autres éléments informatifs en dessous de l'interface machine	Assez intuitif, différents tutoriels d'utilisations disponibles
Exemples	3 exemples disponibles, mais on ne sait pas à quoi ils correspondent	Pas d'exemple disponible	10 exemples disponibles avec un nom clair (on sait ce que la machine simule)	9 exemples disponibles avec un nom clair (on sait ce que la machine simule)
Mots compris	Tout l'alphabet	Uniquement a, b, c, d ou else	Nombres binaires uniquement	Nombres décimaux ou binaires (en fonction de l'exemple choisi)
Résultat	Deux tableaux contenant les mots qui devraient être reconnus et	On lance la machine sans lui avoir donné de mot. On ajoute les	Affiche un smiley :) si le mot est reconnu et :(s'il ne l'est pas	"Accepted" ou "rejected" s'affiche si le mot est reconnu ou pas.

	ceux qui ne le devraient pas, et un dernier tableau indiquant pour chaque mot des deux tableaux précédents s'ils sont reconnus ou pas. Si les mots du tableau "accepted" ne sont pas reconnus, il affiche "fail" en rouge sinon "pass" en vert, et l'inverse pour le tableau des mots "rejected"	lettres au fur et à mesure et l'on voit l'avancement grâce au changement de couleur des états et les mouvements sur les transitions. Pas d'affiche de résultat, on doit regarder si un état final est allumé à la fin de notre mot.		
états	Longs avec le nom au centre, une case cochable à gauche dont on suppose servir à signifier si l'état est final ou pas. Un seul état de départ possible qu'on ne peut pas déplacer. Pour les déplacer, il suffit de rester cliqué sur l'état que l'on souhaite déplacer, et lâcher la souris quand on le souhaite. Pour le supprimer il suffit de cliquer sur la croix qui apparaît quand on passe la souris au-dessus de l'état. Les états pour les MT sont identiques.	Ronds rouges, avec une flèche arrivant dessus pour indiquer l'état initial (qui est unique mais qu'on peut choisir et changer) et un cercle blanc en bord des états finaux permet de les repérer. Pour en ajouter il suffit de sélectionner le bouton d'état et de cliquer n'importe ou sur la fenêtre (sauf sur un autre état). Pour déplacer un état il faut avoir sélectionné le bouton d'état et déplacer un état en le sélectionnant en restant enfoncé et relâcher la souris quand on	On ne voit pas les états, juste l'endroit où on se trouve sur le ruban	On ne voit pas les états, juste l'endroit où on se trouve sur le ruban

		ne souhaite plus le bouger. Pour le rendre initial, terminal ou le supprimer, il faut faire un clique droit dessus et choisir l'action souhaitée.		
Transitions	Une case sur la partie droite de l'état permet de créer des transitions en maintenant la souris enfoncée et en la déplaçant jusqu'à la case de l'état auquel on souhaite ajouter l'extrémité de la transition et une fenêtre apparaît pour la nommer. Pour faire une auto-transition il faut faire sortir la flèche de l'état puis la ramener dessus et la lâcher. Pour changer son nom ou la supprimer il suffit de cliquer dessus. Une fenêtre apparaît et on choisit l'action souhaitée. Il n'est pas possible de les bouger pour améliorer la vision de l'ensemble. Pour les transitions de MT c'est la même chose	Pour ajouter des transitions il faut sélectionner le bouton des transitions, et cliquer sur un état et faire glisser la souris en la maintenant enfoncée jusqu'à l'état souhaité. Pour l'auto transition il suffit de cliquer sur l'état souhaité. Pour pouvoir déplacer les transitions il faut avoir le bouton des transitions sélectionné. Pour nommer la transition ou changer son nom, il faut faire un clique droit sur la transition et sélectionner parmi a, b, c, d ou else. Pour supprimer une transition, il faut également faire un clique droit dessus et sélectionner "Delete".	On ne voit pas les transitions, juste le déplacement sur le ruban	On ne voit pas les transitions, juste le déplacement sur le ruban

	avec 3 cases à remplir lors de la création.			

Grâce à l'analyse de ces 4 différents supports, nous avons pu retenir quelques bonnes idées, et des éléments à ne pas reproduire. Nous les avons listés ci-dessous.

Points Négatifs	Points Positifs
● Ne fournir aucune aide, pas de fichier explicatif ● Ne pas fournir d'exemple ● Limiter les transitions à quelques lettres ● Ne pas donner le mot à analyser avant de lancer la machine ● Ne pas donner de mots du tout ● Faire des tableaux pré rempli et tout analyser d'un coup sans explication claire ● Ne pas indiquer lorsqu'un mot est reconnu ou pas ● Ne pas indiquer clairement les états initiaux et finaux ● Ne pas pouvoir créer sa propre machine de Turing (en plus de celles prédéfinie disponible) ● Ne pas pouvoir déplacer l'état initial	● Interface assez simple pour comprendre l'essentiel de son utilisation sans aide ● Possibilité d'ouvrir des exemples avec fonctionnement clair ● Pouvoir mettre tout l'alphabet et des chiffres dans les transitions ● Voir l'avancement étape par étape sur les états et transitions ● Voir l'avancement étape par étape avec le pointeur sur le ruban ● Afficher clairement si le mot a été reconnu ou pas ● Voir les états même pour les machines de Turing ● Indication visuelle pour les états initiaux et finaux ● Glisser la souris d'un état à un autre pour ajouter des transitions ● Clic simple pour auto transition ● Pouvoir modifier une transition ● Pouvoir renommer les états

1.1.3 Analyse du contexte

Nous n'avons pour l'instant pas l'objectif de rendre notre application publique et disponible sur internet ou autre. Nous n'avons donc pas de réel problème ou obligation au niveau technique pour son intégration.
Notre tuteur nous a seulement demandé de lui fournir un fichier jar chaque semaine avant notre réunion afin de voir notre avancement et pouvoir nous faire des commentaires durant la réunion.

Nous n'avions donc aucune contrainte spécifique au niveau du choix de la technologie. C'est pourquoi nous avons choisi le langage qui nous semblait le plus

logique et avec lequel nous pourrions le plus facilement avancer car nous avions déjà les connaissances du langage java.

1.2. Analyse des besoins fonctionnels

Dans cette partie, nous allons expliquer ce que notre interface doit faire.

Nous avons donc listé à partir des demandes du client les choses nécessaires aux utilisateurs. Les demandes étant à peu près identiques pour les automates et les machines de Turing, nous n'avons fait qu'un diagramme de cas d'utilisation pour les automates que vous pouvez voir sur la Figure 1.

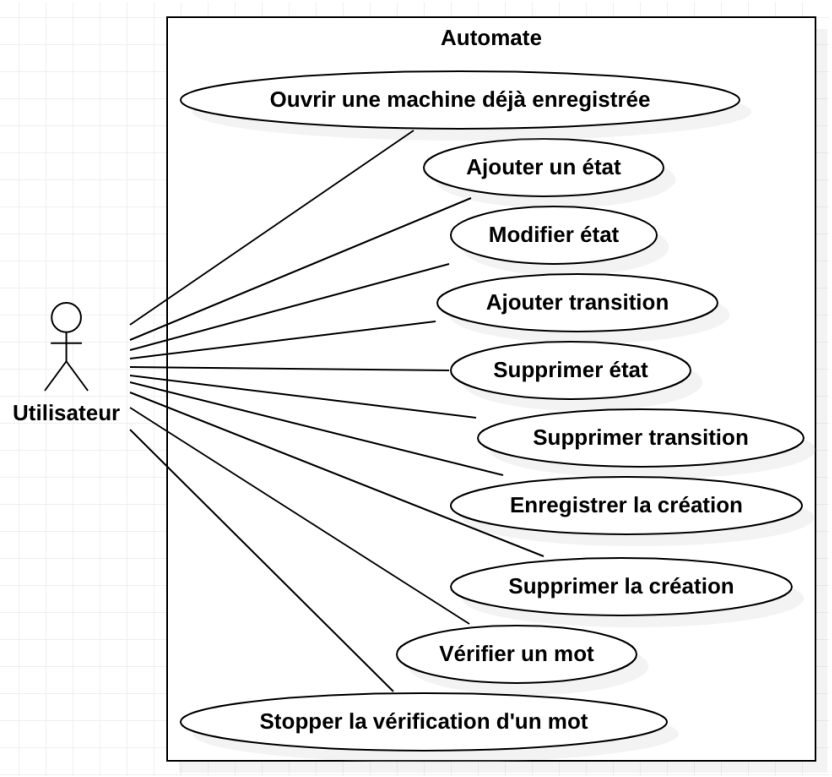


Figure 1 - Diagramme de cas d'utilisation d'un automate

La différence entre les automates et les machines de Turing réside dans la vérification du mot.

Une machine de Turing ne vérifie pas un mot mais fait un calcul à partir d'un mot. Le diagramme de cas d'utilisation d'une machine de Turing serait donc identique à celui de l'automate (cf. Figure 1), sauf que "vérifier un mot" serait remplacé par "calculer le nouveau mot" et "stopper la vérification d'un mot" serait remplacé par "stopper le calcul".

1.3. Analyse des besoins non-fonctionnels

Dans cette partie nous allons parler du comportement et de la performance que l'interface doit avoir.
Nous allons spécifier les demandes du client aux niveaux technique et ergonomique.

1.3.1 Spécifications techniques

Nous allons ici présenter les contraintes techniques imposées par notre tuteur.

Lorsque nous avons ajouté l'enregistrement et l'ouverture des créations, notre tuteur nous a demandé de créer un type spécifique pour nos fichiers. Nous avons donc fait des fichiers "nom.atmt" pour "automate (atmt)" et "nom.mt" pour "machine de Turing (mt)".

Au niveau des états, il voulait que l'on puisse changer la spécificité d'un état (initial, terminal ou pas) non seulement au moment de la création de l'état mais également après l'avoir créé.

Il nous a demandé de faire en sorte que l'utilisation soit simplifiée grâce à des explications, soit en les faisant apparaître lorsque la souris passe sur un bouton, soit en faisant un fichier explicatif. Nous avons opté pour le manuel d'utilisation qui est plus simple à modifier au fur et à mesure de nos ajouts et modifications.

Nous devions également créer un menu sur lequel il devait être possible de sélectionner si l'on voulait faire un automate ou une machine de Turing, et avoir également la possibilité de choisir un automate ou une machine déjà créée.

Une autre demande qui est venue plus tard dans la réalisation de notre projet a été de proposer des automates et machines de Turing déjà prêts que l'utilisateur pouvait sélectionner et ouvrir pour commencer.

Lors de la dernière réunion, notre tuteur nous a lancé un défi : il voulait que nous ajoutions aux exemples disponibles une machine de Turing qui prend un nombre en binaire, un symbole et un autre nombre en binaire et que la machine fasse la somme des deux.

1.3.2 Contraintes ergonomiques

Nous allons ici présenter les contraintes ergonomiques imposées par notre tuteur.

Notre tuteur nous a laissé beaucoup de liberté au niveau de l'affichage. Nous allons donc revenir sur les points qu'il nous a spécifiés et comment nous avons répondu à ses demandes.

Etats

La demande était de pouvoir différencier facilement les états initiaux et terminaux. Nous avons donc choisi d'ajouter un - en haut à gauche des états initiaux, et un + en haut à droite des états terminaux.

Transitions

Il fallait pouvoir différencier les transitions lorsqu'il y en avait plusieurs entre deux mêmes états. Un seul trait droit est visible entre deux états, peu importe le nombre de transition. Pour afficher la direction d'une transition, on met une flèche pointant vers l'état d'arriver à 2 tiers après l'état de départ. Ainsi, s'il y a des transitions dans les deux sens, les flèches ne se chevauchent pas. L'étiquette de la transition s'affiche au niveau de la flèche correspondante sur la transition. S'il y a plusieurs transitions dans la même direction, les lettres se mettent les unes à la suite des autres dans l'ordre d'ajout. Pour les auto-transitions, nous avons ajouté l'image d'une flèche revenant sur elle-même en bas à gauche de l'état.

Sélection

La demande était de rendre visible lorsqu'un état ou une transition était sélectionné. Nous avons donc ajouté un cercle bleu foncé autour des états sélectionnés et rendu le trait des transitions sélectionnées bleu foncé.

Lancement d'un mot

La demande était de rendre visible les changements d'état tout au long de la lecture du mot. Nous avons donc commencé par donner des couleurs aux états. Au départ, les états sont bleus, ce qui veut dire qu'ils ne sont pas actifs. Lorsque l'on lance la lecture du mot, les états initiaux deviennent verts, ce qui veut dire qu'ils sont actifs. Puis les états passent vert/bleu en fonction du mot et des transitions qui les rendent actifs ou non.

Cette première version lui a plu mais n'était pas suffisante car il nous a fait remarquer que pour les automates, lorsque la lecture d'un mot n'est pas finie mais que pour une raison quelconque un seul état reste actif jusqu'à la fin, on ne voit aucun mouvement. Il nous a donc demandé d'afficher le mot et l'avancement. Nous avons donc mis une barre de chargement qui avance à chaque fois qu'une nouvelle lettre du mot est lue et nous affichons également le mot, d'abord en gris, puis chaque fois qu'une lettre est lue, elle devient bleue. Ainsi le suivi de la lecture du mot est devenu plus clair.

Pour les machines de Turing, la seule différence est au niveau de l'affichage du mot. Seule la lettre analysée est en bleu dans le mot, car la lecture ne se fait pas obligatoirement de gauche à droite, et cela permet de voir clairement les changements effectués sur le mot.

Menu

Sur le menu, il nous a demandé d'afficher le nom du sujet de notre projet, les noms des membres du groupe et le nom de notre tuteur.
Pour les exemples d'automates et de machines de Turing, nous les avons ajoutés sous forme de menu déroulant en dessous du bouton correspondant.

Affichage final

La seule demande était de montrer que la lecture s'était arrêtée et d'afficher le résultat obtenu. Nous avons donc opté pour l'affichage d'une pop-up.
Pour les automates, la pop-up apparaît lorsque la lecture du mot est finie et elle indique simplement si le mot est reconnu ou non.
Pour les machines de Turing, la pop-up apparaît lorsque un état final s'active. Elle indique que le calcul est terminé et affiche également le résultat obtenu (le nouveau mot).

2. Rapport technique

Cette partie explique le processus de conception de notre application que nous avons réalisé ainsi que la façon dont nous avons réalisé notre application.

2.1. Conception

Les différents diagrammes de classes présents dans les parties suivantes ont été simplifiés afin de permettre une meilleure compréhension pour le lecteur. En particulier les attributs privés ne possédant pas de getter, les méthodes privées ainsi que les getters et les setters n'ont pas été représentés. L'interface de notre application possède de nombreux composants (Buttons, HBox, TextField, ScrollBar, …), ces derniers ne sont donc pas tous représentés.

2.1.1 Nos choix technologiques

Pour la réalisation de ce projet de nombreux langages et frameworks étaient envisageables. Nous aurions pu choisir l'utilisation du langage python avec la librairie TKinter, le C++ avec l'API Qt ou encore du JavaScript couplé à du HTML et CSS comme nous l'a suggéré notre tuteur de projet. Cependant, ces langages n'ont pas ou peu été abordés lors de notre formation, contrairement au langage Java qui a été étudié tout au long de notre formation à l'IUT informatique, notamment grâce à de divers TD et projets. C'est donc tout naturellement que notre choix s'est porté sur le langage de programmation Java.

Pour le framework nous avons choisi JavaFX. En effet, ce framework a été développé par Oracle lui-même ainsi que la communauté OpenJFX ce qui garantit une très bonne intégration avec le langage Java. Actuellement, JavaFX est l'outil de création d'interface graphique officielle du langage Java. Cette bibliothèque vient remplacer Swing et AWT, pour résoudre les défauts de ces derniers et ajouter de nouvelles fonctionnalités. Nous pouvons notamment souligner que JavaFX est plus performant, plus configurable, plus puissant et plus souple que ses prédécesseurs. [9] [13]

2.1.2 Conception de la logique de notre application.

Il est important de souligner que notre projet a été découpé en deux phases.
La première phase de notre projet a été la conception et la réalisation d'un logiciel permettant la création et la simulation d'automates finis non-déterministes.

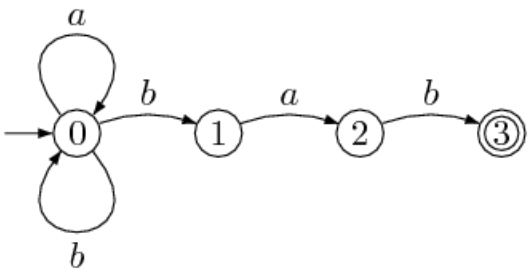


Figure 2 - Représentation d'un automate fini

Lors de la conception de notre projet (pour la partie automate) nous avons identifié 3 classes principales. Une classe Transition qui représente une transition entre 2 états. Une classe Etat qui représente un état d'un Automate et l'Automate en lui-même. Grâce aux connaissances que nous avons acquises en mathématiques sur les automates et comme il est possible de le voir sur la figure 2 et sur notre UseCase (cf. Figure 1), nous avons pu déterminer plusieurs choses :

- Une transition possède une étiquette, son état de départ ainsi que son état d'arrivée.
- Un état peut être initial ou terminal, il possède la liste de ses transitions sortantes et il peut être actif lors de l'exécution de l'automate.
- L'automate possède l'ensemble des états qui le composent ainsi que le mot qui est initialisé au lancement de la machine.
- Des transitions peuvent être ajoutées et supprimées à un état.
- Il est possible d'obtenir la liste des états qui peuvent être atteints avec une certaine lettre.
- Des états peuvent être ajoutés et supprimés à un automate.
- Un automate peut être chargé et sauvegardé à partir d'un fichier.
- Et bien entendu, un automate doit pouvoir être lancé à partir d'un certain mot.

À partir de ces informations nous avons produit le diagramme de classes de la partie logique d'un automate (cf. figure 3).

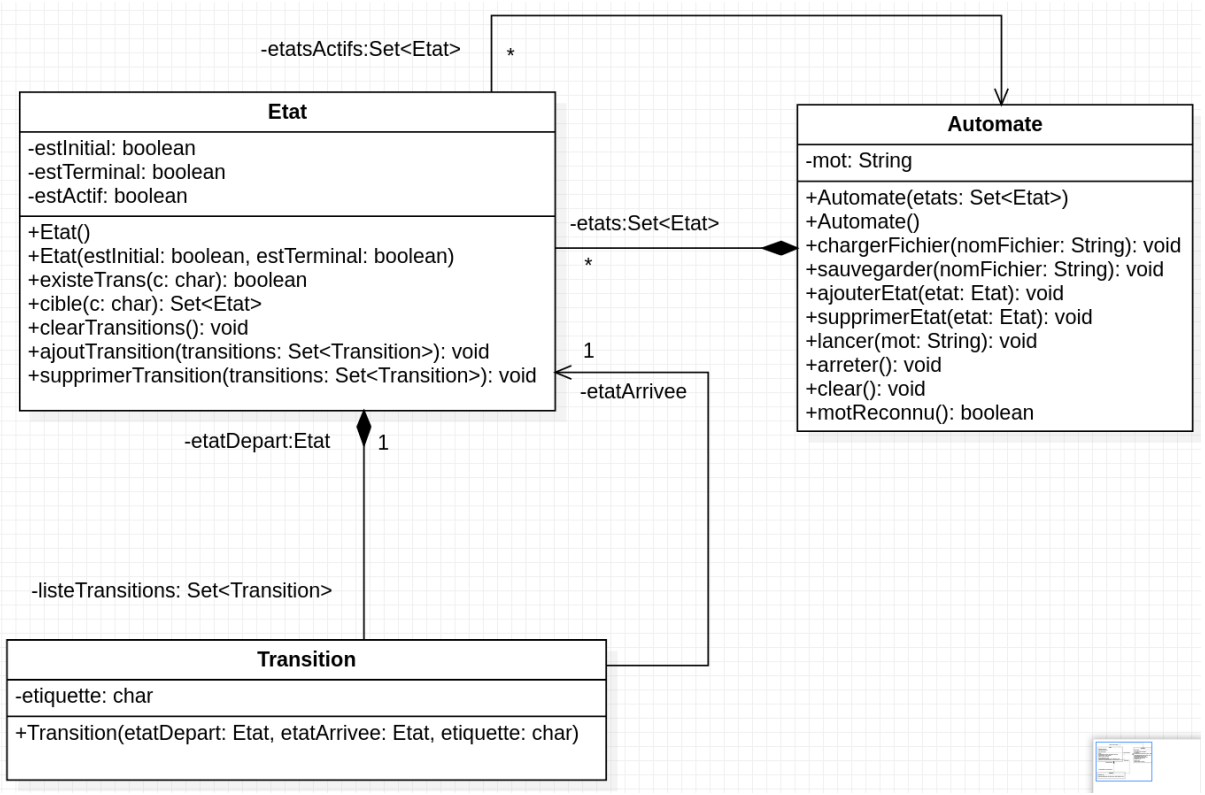


Figure 3 - Diagramme de classes logique automate

La deuxième phase de notre projet consistait en l'implémentation des machines de Turing.

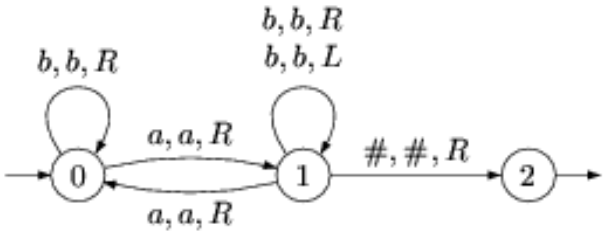


Figure 4 - Représentation d'une machine de Turing

Un automate fini et une machine de Turing possèdent des différences mais ont énormément de choses en commun. Nous avons donc extrait une grande partie des attributs et des méthodes de la classe Automate vers une classe mère abstraite Machine dont héritent les classes concrètes Automate et MT (machine de Turing). De la même manière les classes TransitionAtmt (les transitions des automates) et TransitionMT (les transitions des machines de Turing) héritent toutes deux de la classe abstraite Transition.
Dans notre application un état d'un automate est identique à un état d'une machine de Turing. La seule différence est le type de transition qu'il possède, c'est pourquoi

la classe Etat est paramétré par un Type de Transition (public class Etat<T extends Transition<T>>).
Nous n'avons pas représenté cela sur la figure 5 pour une meilleure lisibilité.
Nous avons identifié différentes choses pour les machines de Turing :

- Une machine de Turing possède un ruban ainsi qu'une tête de lecture.
- Une transition d'une machine de Turing possède en plus une lettre d'écriture ainsi qu'un mouvement (gauche ou droit).

À partir de ces informations nous avons pu produire le diagramme de classes de la logique de notre application (cf. figure 5).

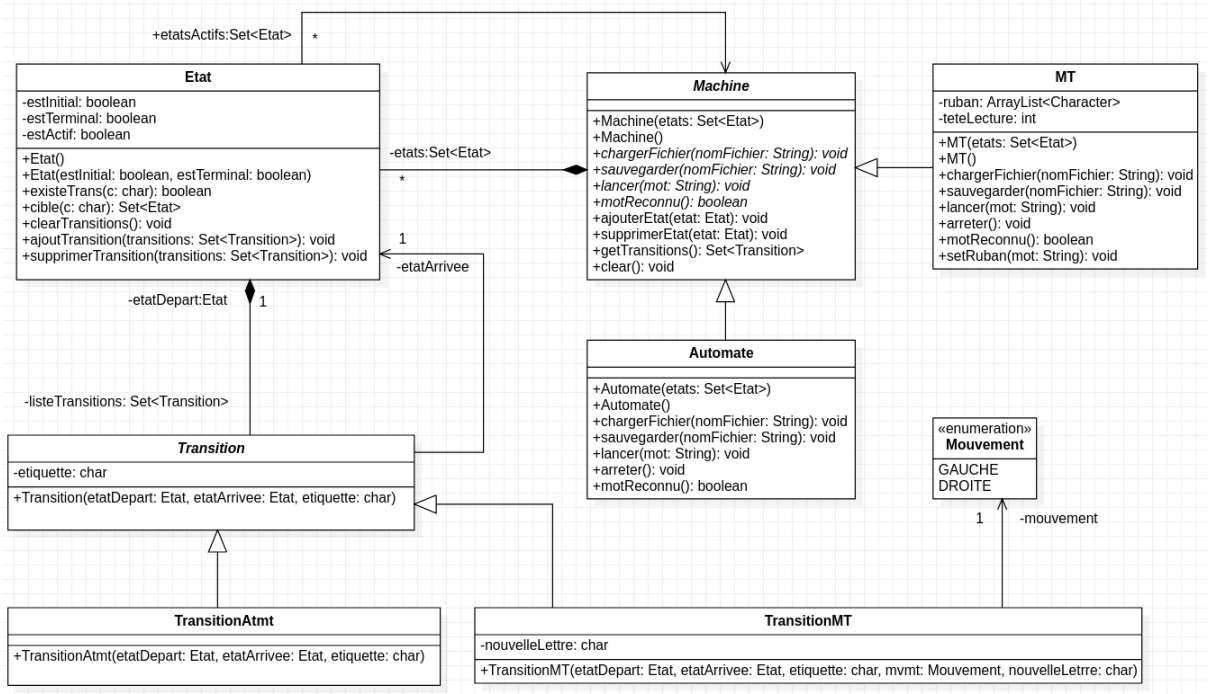


Figure 5 - Diagramme de classes de la partie logique

2.1.3 Conception de l'interface graphique.

L'interface graphique de notre application fonctionne avec des vues. Chaque classe de la partie logique de notre application possède une vue associée : Etat -> VueEtat, Transition -> VueTransition, Machine -> VueMachine. De plus, notre application possède une classe VuePrincipale qui contient tout le menu, les boutons et tout autre composant nécessaire pour le fonctionnement de notre application. Par exemple, le bouton "Lancer". Comme expliqué précédemment, un automate fini et une machine de Turing possèdent des différences mais ont énormément de choses en commun. Leurs vues associées (VueAutomate / VueMT et VueTransitionAtmt / VueTransitionMT) héritent donc des vues abstraites VueMachine et VueTransition. Il en est de même pour les classes VuePrincipaleAtmt et VuePrincipaleMt qui héritent de la classe abstraite VuePrincipale.

Nous avons identifié différentes choses pour la partie graphique de notre application:

- Les vues état et transition peuvent être sélectionnées et désélectionnées.
- Il est possible d'ajouter et de supprimer des vues état et transition à la vue Machine.
- La vue principale et la vue machine doivent posséder toutes les méthodes exécutées par les différents composants. Comme par exemple la méthode lancer pour le bouton “Lancer”.

À partir de ces informations nous avons pu produire le diagramme de classes de la partie graphique de notre application (cf. figure 6).

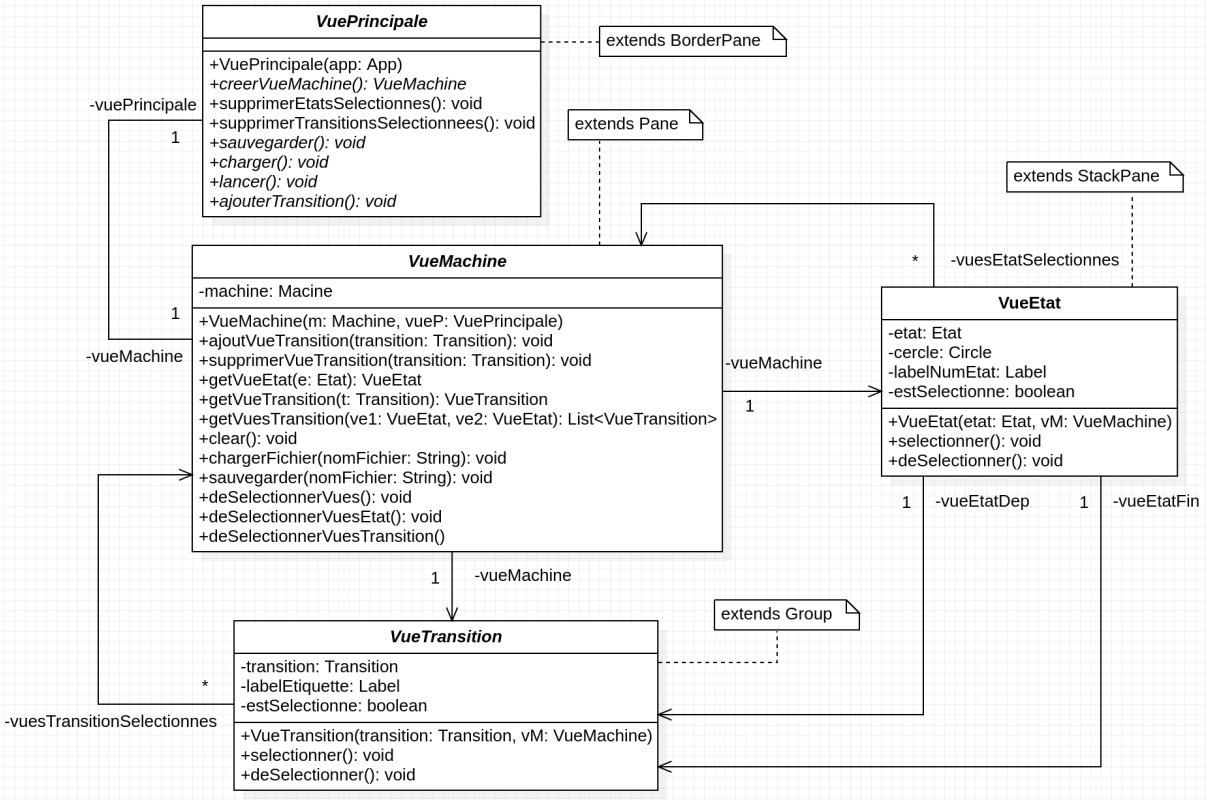


Figure 6 - Diagramme de classes de la partie interface graphique

2.2. Réalisation

2.2.1 L’architecture de notre programme

Pour notre application nous avons choisi d’utiliser Maven. Maven est un outil de construction de projets. Il permet de faciliter et d'automatiser certaines tâches de la gestion d'un projet Java [8]. Cet outil nous a notamment permis de faciliter l’intégration des librairies JavaFx dans notre projet.
En ce qui concerne l’arborescence des fichiers de notre application, le package machines contient l’ensemble du code. Nous avons ensuite, dans ce package,

séparé la logique de notre application et la partie graphique en deux packages (respectivement *logique* et *gui*). Ces packages contiennent l'ensemble des classes communes aux machines de Turing et aux automates ainsi que les packages *automates* et *mt* qui, eux, contiennent les classes spécifiques aux automates ou aux machines de Turing (cf. figure 7).

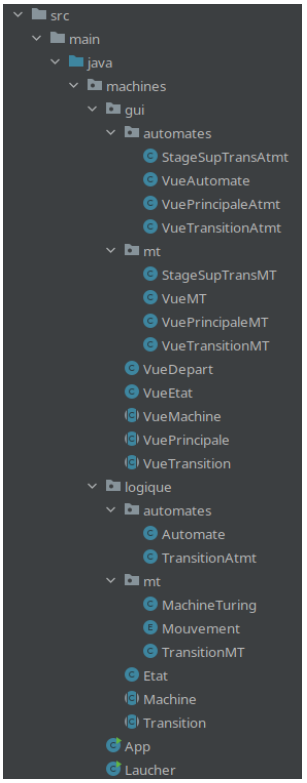


Figure 7 - Arborescence des fichiers de notre application

2.2.2 Les fonctionnalités de notre application

Suite au cahier des charges établi avec M. Rosenfeld, nous avons donc réalisé un logiciel avec une interface permettant de créer facilement des automates finis et des machines de Turing et de simuler le fonctionnement de ces derniers. Notre application permet donc, pour les automates et les machines de Turing, d'ajouter des états et de relier ces derniers par des transitions. Il est aussi possible de supprimer des états ainsi que des transitions, de définir si un état est initial et/ou terminal, de sauvegarder les automates et les machines de Turing créés sous forme de fichiers et de charger ces fichiers. Une fois qu'un automate ou une machine de Turing sont créés il est possible de les lancer afin d'observer l'exécution de ces derniers.

2.2.3 Le lancement d'un automate

Le lancement d'un automate s'effectue en appuyant sur le bouton *Lancer*. L'appuie sur ce bouton va appeler la méthode *lancer()* de la classe *VuePrincipaleAtmt*. Cette méthode va principalement récupérer le mot, ainsi que la vitesse d'exécution, saisies par l'utilisateur puis appeler la méthode *lancer()* de la classe *Automate* en donnant ces paramètres.
L'utilisation d'une pause entre chaque étape de l'exécution de l'automate (entre chaque lecture des lettres du mot donné) nécessite l'utilisation de la méthode *sleep()* de la classe Thread. Cette méthode va effectuer un arrêt temporaire de l'exécution du programme. Il faut savoir que l'utilisation de cette méthode dans une application JavaFx va bloquer la mise à jour des éléments graphiques de cette application. Il est donc nécessaire de lancer un nouveau Thread dans lequel les instructions à réaliser pour le lancement de l'automate seront exécutées. Ces instructions seront définies dans une Task qui sera fournie au constructeur du nouveau Thread (cf. figure 8).

```
public void lancer(String mot, long dellayMillis) {
    this.mot = mot;
    if (taskLancer == null) {
        initTaskLancer(mot, dellayMillis);
    } else if (taskLancer.isRunning()) {
        taskLancer.cancel();
        initTaskLancer(mot, dellayMillis);
    } else {
        initTaskLancer(mot, dellayMillis);
    }
    taskLancer.setOnCancelled(getOnCancelled());
    taskLancer.setOnRunning(getOnRunning());
    taskLancer.setOnSucceeded(getOnSucceeded());
    if (listenerValueTaskLancer != null)
        taskLancer.valueProperty().addListener(listenerValueTaskLancer);
    new Thread(taskLancer).start();
}
```

Figure 8 - Méthode lancer() de la classe Automate

Les instructions définies dans cette Task sont les suivantes : les états initiaux sont activés puis tant qui reste des états actifs ou que le mot n'a pas été totalement parcouru, une étape de l'exécution (via la méthode *step()*) est effectué sur la lettre courante (cf. figure 9).

```
private void initTaskLancer(String mot, long dellayMillis) {
   taskLancer = (Task) () -> {
           getEtats().forEach(Etat::desactive);
           getEtatsInitiaux().forEach(Etat::active);
           progress.set(0);
           int i = 0;
           while (getEtatsActifs().size() > 0 && i < mot.length()) {
               Thread.sleep(dellayMillis);
               char lettre = mot.charAt(i);
               step(lettre);
               updateValue(i);
               progress.set((double) (i + 1) / mot.length());
               i++;
           }
           return i;
       }
   };
}
```

Figure 9 - Création de la Task pour un Automate

A chaque étape de l'exécution les anciens états actifs sont désactivés puis tous les états atteignables par les transitions portant la lettre courante comme étiquette sont activés (cf figure 10).

```
private void step(char lettre) {
   Set<Etat<TransitionAtmt>> nouveauxActifs = new HashSet<>();
   getEtatsActifs().forEach(e -> nouveauxActifs.addAll(e.cible(lettre)));
   getEtatsActifs().forEach(Etat::desactive);
   nouveauxActifs.forEach(Etat::active);
}
```

Figure 10 - Méthode step() de la classe Automate

Les automates possèdent des attributs WorkerStateEvent qui sont des actions à exécuter à des moments donnés. Les trois WorkerStateEvent sont *onRunning*, *onCancelled* et *onSucceeded*. Ces WorkerStateEvent sont définis dans la classe *VuePrincipaleAtmt* et vont notamment permettre d'afficher si le mot a été reconnu à la fin de l'exécution ou encore d'afficher la lettre courante à chaque étape de l'exécution de l'automate.

2.2.4 Le lancement d'une machine de Turing

Le lancement d'une machine de Turing repose sur le même principe que le lancement d'un automate (cf. 2.2.3). Là aussi le lancement s'effectue en appuyant sur le bouton *Lancer*. L'appuie sur ce bouton va appeler la méthode *lancer()* de la classe *VuePrincipaleMt*. Cette méthode va principalement récupérer le ruban initial, ainsi que la vitesse d'exécution saisies par l'utilisateur, puis appeler la méthode *lancer()* de la classe *MachineTuring* en donnant ces paramètres. La méthode *lancer()* de la classe *MachineTuring* et identique à celle de la classe *Automate* (cf. figure 8).

En ce qui concerne les instructions de la Task de lancement, cette dernière va activer l'état initial et initialiser le ruban de départ. Puis tant qu'il reste un état actif et qu'un état terminal n'a pas été atteint, une étape de l'exécution (via la méthode step()) est effectué sur le caractère pointé par la tête de lecture (cf. figure 11).

```
private void initTaskLancer(String mot, long dellayMillis) {
   taskLancer = (Task) () → {
            getEtats().forEach(Etat::desactive);
            getEtatInitial().active();
            setRuban(mot);
            int compteur = 0;
            while (getEtatActif() ≠ null && !getEtatActif().estTerminal()) {
                updateValue(compteur);
                Thread.sleep(dellayMillis);
                //lire() récupère le caractère pointé par la tête de lecture
                step(lire());
                compteur++;
            }
            updateValue(compteur);
            return compteur;
        }
   };
}
```

Figure 11 - Création de la Task pour une machine de Turing

A chaque étape de l'exécution, l'état atteignable par la transition portant la lettre courante comme étiquette est activé et l'ancien état actif est désactivé. Ensuite la tête de lecture va écrire un caractère sur le ruban et se déplacer selon les instructions de la transition (cf figure 12).

```
private void step(char lettre) {
   Etat<TransitionMT> etatActif = getEtatActif();
   if (etatActif ≠ null) {
       TransitionMT transEtape = null;
       for (TransitionMT trans : etatActif.getListeTransitions()) {
           if (trans.getEtiquette() == lettre) transEtape = trans;
       }
       if (transEtape ≠ null) {
           ecrire(transEtape.getNouvelleLettre(), transEtape.getMouvement());
           transEtape.getEtatDepart().desactive();
           transEtape.getEtatArrivee().active();
       } else {
           etatActif.desactive();
       }
   }
}
```

Figure 12 - Méthode step() de la classe MachineTuring

Comme pour les automates, les machines de Turing possèdent des attributs WorkerStateEvent qui sont des actions à exécuter à des moments donnés. Les trois WorkerStateEvent sont *onRunning*, *onCancelled* et *onSucceeded*. Ces

WorkerStateEvent sont définis dans la classe *VuePrincipaleMt* et vont notamment permettre d'afficher le résultat obtenu et la validité du calcul à la fin de l'exécution ou encore d'afficher la lettre courante à chaque étape de l'exécution de l'automate.

3. Résultats

Cette partie rend compte de l'aspect final du logiciel.
Ici vous pourrez trouver les informations relatives à l'installation du programme, aux différents tests réalisés ainsi que le manuel d'utilisation.

Actuellement deux modèles sont disponibles, automates et machines de Turing. Toutes les fonctionnalités nécessaires ont été implémentées.

3.1. Installation

Vous pourrez retrouver notre application à télécharger sur ce lien.
Ensuite, si Java est installé sur votre ordinateur vous pourrez lancer le jar pour accéder au logiciel (java -jar projetS3.jar).

3.2. Tests

Notre programme a été testé tout au long de la programmation par l'équipe de projet et notre tuteur qui, à la fin de la période de programmation, a validé notre programme.
Notre application ne contient pas de tests automatiques.

3.3. Manuel d'utilisation

Le manuel d'utilisation de notre application peut aussi être retrouvé dans le répertoire de l'application à télécharger. En particulier les images .gif du lancement des différentes machines risquent de mal s'afficher ici.

3.3.1 Fenêtre de départ

Lors du lancement de l'application une fenêtre s'ouvrira et vous permettra de choisir de lancer l'application de gestion d'automates en appuyant sur le bouton *Automate* (cf. Figure 13).
La liste déroulante, en dessous du bouton *Automate* permet de choisir de partir d'un nouvel automate si l'option *nouveau* est choisie ou de charger directement des fichiers *.atmt* en choisissant les noms de ces derniers. Les fichiers pouvant être chargés sont les fichiers présents dans le répertoire *default* du répertoire *fichiers_machines*.

Les fichiers .atmt doivent impérativement se situer dans un répertoire default, lui-même situé dans un répertoire fichiers_machines qui doit être au même endroit que le jar de l'application pour être proposé par la liste déroulante.

Figure 13 - Fenêtre de départ de l'application

3.3.2 Automates

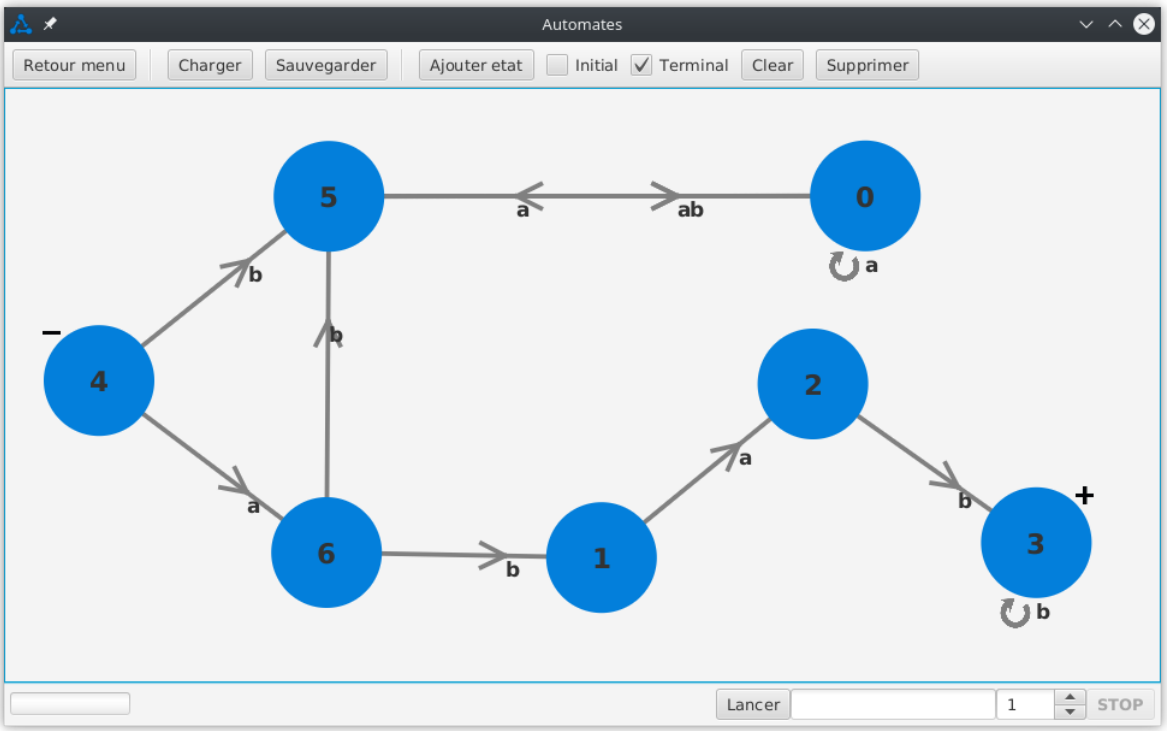


Figure 14 - Interface d'un automate

Les états

Pour ajouter un état à l'automate, il suffit de cliquer sur le bouton *ajouter état*.

Pour qu'un état soit initial ou final (représenté par les - et les + à côté des états) il suffit de cocher les cases *initial* ou *terminal* avant de créer l'état, ou après avoir sélectionné l'état que l'on souhaite modifier (si plusieurs états sont sélectionnés, seul le dernier à avoir été sélectionné sera modifié).
Pour déplacer un état, il suffit de le faire glisser avec la souris (cf. Figure 14).

Les transitions

Pour ajouter une transition, il faut **d'abord** sélectionner l'état d'arrivée **puis** l'état de départ en gardant la touche *Ctrl* enfoncée (ou seulement un état pour une auto-transition). Il faut ensuite ajouter la lettre de la transition dans le cadre qui apparaît en haut à droite puis cliquer sur le bouton *ajouter transition* (cf. Figure 15).

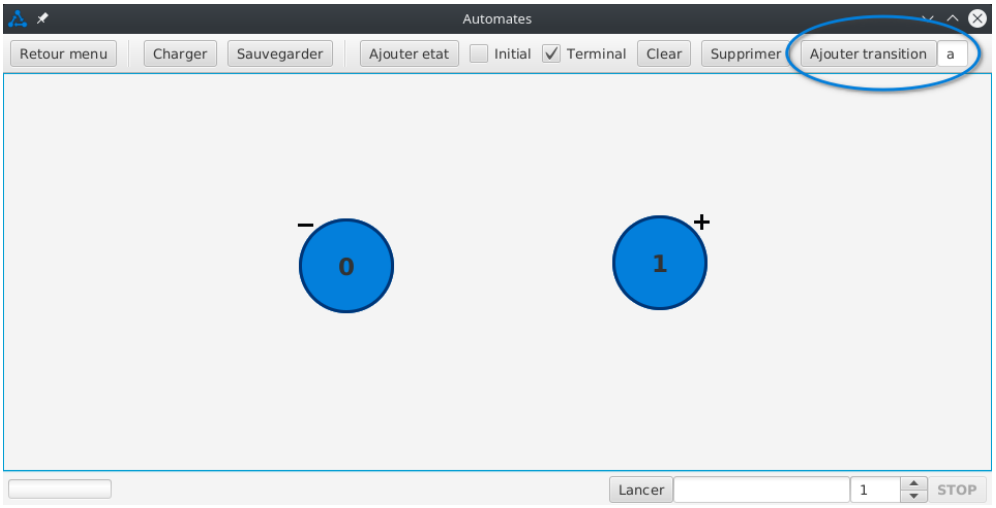


Figure 15 - Ajout d'une transition (automate)

La sélection

Pour sélectionner un état ou une transition, il suffit de cliquer dessus. Cet élément aura donc un contour bleu. Pour sélectionner plusieurs éléments, il faut maintenir la touche *Ctrl* enfoncée et cliquer sur un autre élément.

La suppression

Un appui sur le bouton *Supprimer* supprimera tous les éléments sélectionnés. Dans le cas où plusieurs transitions entre deux mêmes états seraient sélectionnées une fenêtre de dialogue s'ouvrira afin de vous permettre de choisir spécifiquement les transitions à supprimer (cf. Figure 16).
Il est aussi possible de supprimer tous les états et toutes les transitions en appuyant sur le bouton *Clear*.

Figure 16 - Fenêtre de dialogue pour la suppression de transitions (automate)

Le lancement d'un automate

Pour lancer un automate, il faut écrire le mot à tester dans la zone prévue à cet effet puis appuyer sur le bouton *Lancer* (cf. Figure 17). Vous pouvez voir la progression de la lecture du mot en bas de l'application. À la fin de l'exécution une fenêtre s'ouvrira et vous indiquera si le mot a été reconnu par l'automate.

La vitesse d'exécution de l'automate peut être modifiée via le spinner présent à la droite de la zone du mot à tester. Cette valeur représente le temps, en seconde, qui s'écoulera entre deux étapes de l'exécution de l'automate.

Il est aussi possible d'arrêter l'exécution de l'automate en appuyant sur le bouton *STOP*.

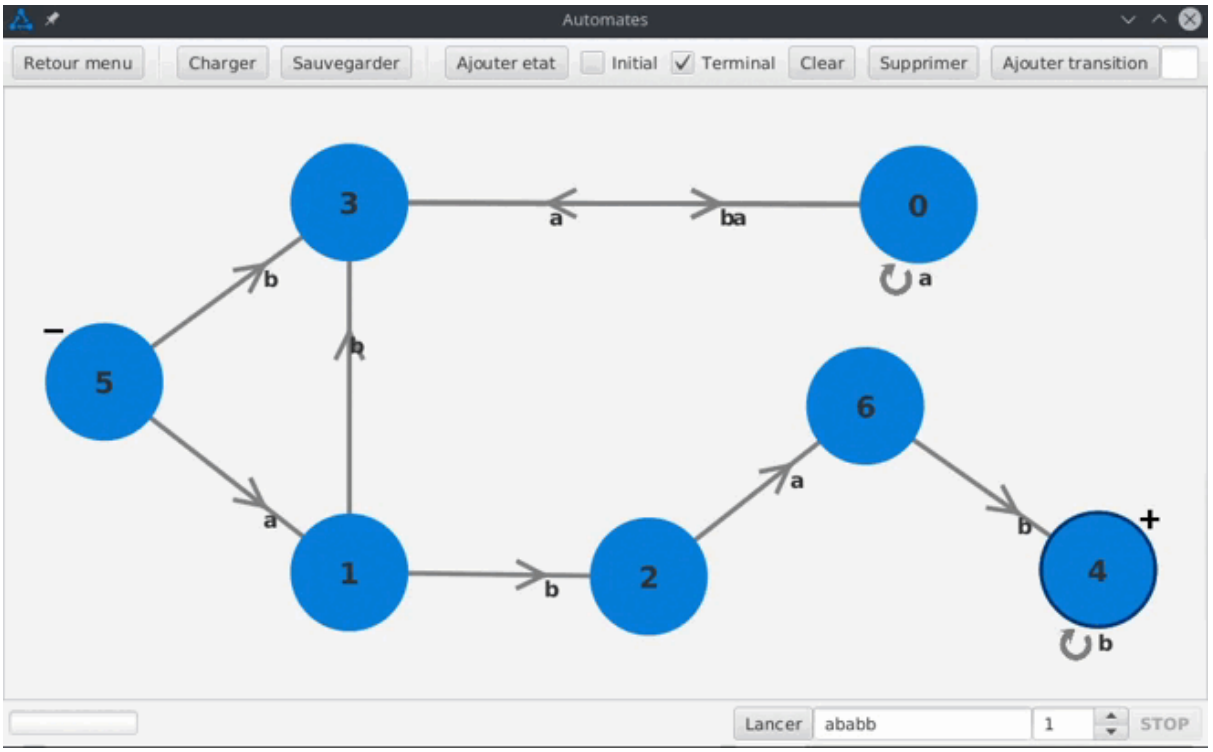


Figure 17 - Lancement d'un automate

Fichiers .atmt

Nous avons créé un type de fichier : les fichiers *.atmt*. Ces fichiers permettent de stocker les automates dans un fichier. La partie du fichier situé après les ### permet de stocker la position graphique des états.

4 initial 0 terminal 2 terminal 3 0 b 1 0 a 0 1 d 3 1 c 2 1 b 1 ### 0 79.0 179.0 0 1 343.0 176.0 1 2 563.0 97.0 3 3 555.0 326.0 2	(nombre total d'états) (état initial) (état terminal) (état terminal) (transition de l'état 0 vers l'état 1 portant la lettre b) (position de l'état 0 graphiquement et son label, ici 0)

Figure 18 - Structure d'un fichier .atmt

Attention, les numéros des états dans les fichiers *.atmt* ne correspondent pas forcément au numéro affiché. Le numéro affiché et celui spécifié après la position de l'état.

La sauvegarde et le chargement de fichiers .atmt

Il est possible de charger ou de sauvegarder un automate sous la forme de fichiers *.atmt* via les boutons *Sauvegarder* et *Charger* (cf. Figure 18).

3.3.3 Machine de Turing

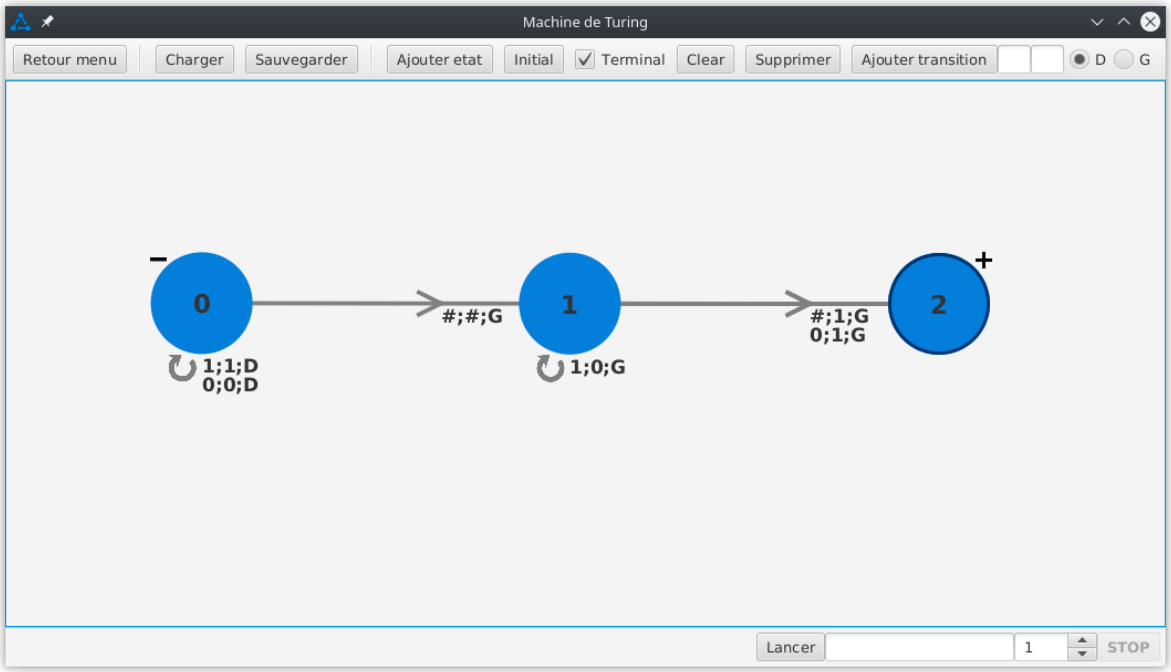


Figure 19 - Interface d'une machine de Turing

Les états

Le fonctionnement des états pour une machine de Turing est identique à celui des états pour un automate. Cependant, dans notre projet, il ne peut y avoir plusieurs états initiaux pour une machine de Turing. La case *initial* a donc été remplacée par un bouton *initial* (cf. Figure 19).

Les transitions

Pour ajouter une transition, il faut **d'abord** sélectionner l'état d'arrivée **puis** l'état de départ (ou seulement un état pour une auto-transition). Il faut ensuite ajouter une lettre de lecture, une lettre d'écriture et un mouvement (gauche ou droite), via les cadres et les boutons radios situés en haut à gauche, puis cliquer sur le bouton *ajouter transition* (cf. Figure 20).

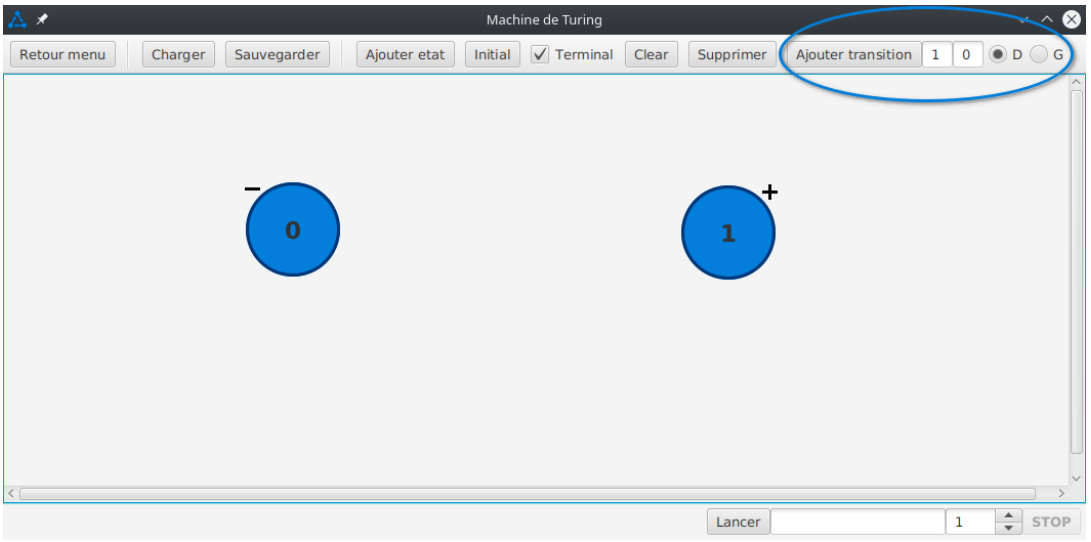


Figure 20 - Ajout d'une transition (machine de Turing)

La sélection

Pour sélectionner un état ou une transition, il suffit de cliquer dessus. Cet élément aura donc un contour bleu. Pour sélectionner plusieurs éléments, il faut maintenir la touche *Ctrl* enfoncée et cliquer sur un autre élément.

La suppression

Un appui sur le bouton *Supprimer* supprimera tous les éléments sélectionnés. Dans le cas où plusieurs transitions entre deux mêmes états seraient sélectionnées une fenêtre de dialogue s'ouvrira afin de vous permettre de choisir spécifiquement les transitions à supprimer (cf. Figure 21).

Il est aussi possible de supprimer tous les états et toutes les transitions en appuyant sur le bouton *Clear*.

Figure 21 - Fenêtre de dialogue pour la suppression de transitions (M.T.)

Le lancement d'une machine de Turing

Pour lancer une machine de Turing, il faut écrire le ruban initial à tester dans la zone prévue à cet effet puis appuyer sur le bouton *Lancer* (cf. Figure 22). Vous pouvez suivre l'évolution du ruban de la machine en bas de l'écran. À la fin de l'exécution une fenêtre s'ouvrira et vous indiquera le résultat du calcul et si ce dernier est valide.
La vitesse d'exécution de l'automate peut être modifiée via le spinner présent à la droite de la zone du mot à tester. Cette valeur représente le temps, en seconde, qui s'écoulera entre deux étapes de l'exécution de l'automate.
Il est aussi possible d'arrêter l'exécution de l'automate en appuyant sur le bouton *STOP*.

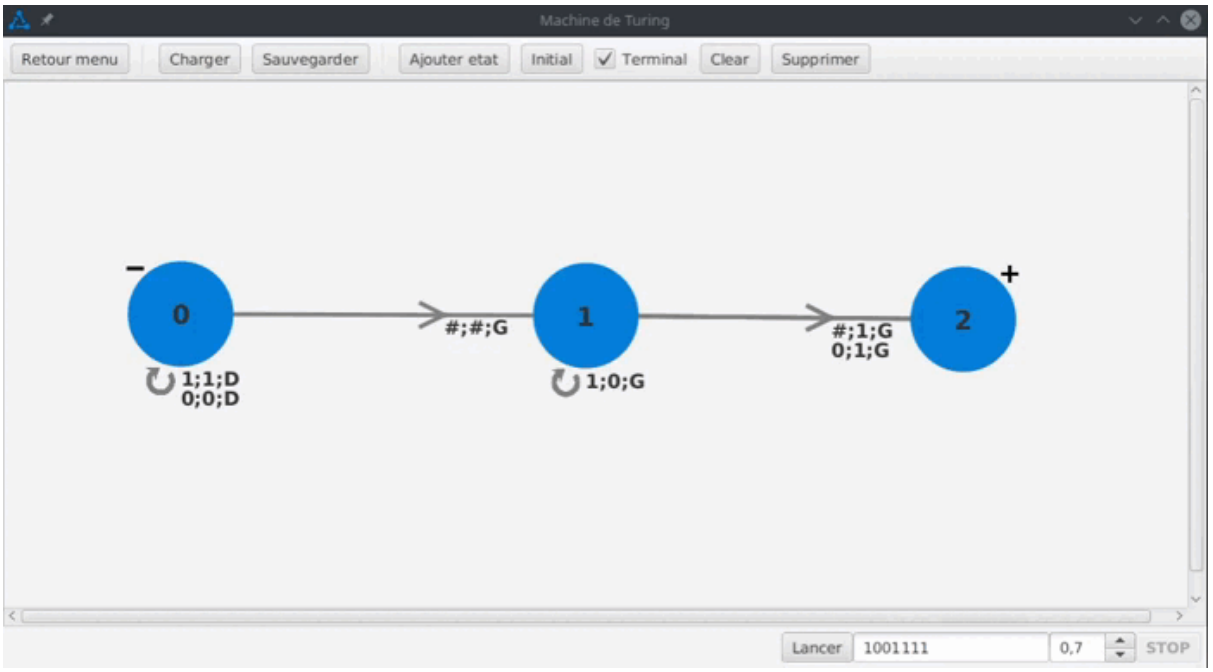


Figure 22 - Lancement d'une machine de Turing

Fichiers *.mt*

Nous avons créé un deuxième type de fichier : les fichiers *.mt*. Ces fichiers permettent de stocker les machines de Turing dans un fichier. La partie du fichier situé après les ### permet de stocker la position graphique des états.

3 initial 0 terminal 1 0 # 2 # GAUCHE 0 1 0 1 DROITE 0 0 0 0 DROITE 2 # 1 1 GAUCHE 2 1 2 0 GAUCHE 2 0 1 1 GAUCHE ### 0 187.0 158.0 0 1 854.0 157.0 2 2 521.4 156.8 1	(nombre total d'états) (état initial) (état terminal) (transition de l'état 0 à l'état 2 ayant # en lecture, # en écriture et ayant le mouvement gauche) (position de l'état 0 graphiquement ainsi que son label)

Figure 23 - Structure d'un fichier *.mt*

Attention, les numéros des états dans les fichiers .mt ne correspondent pas forcément au numéro affiché. Le numéro affiché et celui spécifié après la position de l'état.

La sauvegarde et le chargement de fichiers *.mt*

Il est possible de charger ou de sauvegarder un automate sous la forme de fichiers *.mt* via les boutons *Sauvegarder* et *Charger* (cf. Figure 23).

4. Gestion de projet

Dans cette partie, vous pouvez retrouver une courte description de l'aspect gestion que nous avons appliqué tout au long du déroulé de notre projet.

4.1. Méthode de développement et outils

Des réunions hebdomadaires ont été réalisées chaque lundi avec notre tuteur de projet afin de rendre compte de l'avancement de notre projet et de définir les différentes tâches à réaliser pour la semaine.
De plus, notre équipe réalisait des appels via l'outil discord à raison de deux fois par semaine afin de définir la répartition du travail.

Certains membres de notre équipe avaient des rôles définis :

- Lénaïs Desbos était en charge de la communication avec notre tuteur de projet ainsi que de la rédaction des comptes rendus de réunions
- Emerick Biron était chargé de la répartition des tâches ainsi que du versionning de notre application

Git et Gitlab ont été utilisés pour le versionning et le développement en équipe. Afin de pallier aux différents problèmes pouvant être rencontrés avec Git, (problèmes de merge, …) notre équipe a préféré utiliser le plugin "code with me" de l'IDE IntelliJ qui permet de travailler sur les mêmes fichiers de code à la manière d'un google doc. La gestion du versionning a donc pu être déléguée à Emerick maîtrisant l'outil Git.

4.2. Planification des tâches

En ce qui concerne la planification des tâches, nous avons fait le choix d'utiliser la méthode AGILE. Cela nous a permis de nous concentrer sur un faible nombre de fonctionnalités à implémenter entre chaque réunion avec le tuteur et de pouvoir avoir un retour rapide sur ces fonctionnalités lors de ces réunions.

4.3. Bilan critique par rapport aux cahier des charges

Avec le temps donné, toutes les attentes pour notre projet ont été réalisées.
Si le temps nous l'avait permis, nous aurions pu aussi implémenter de nouveaux modèles car la classe machine, étant abstraite, permet la réalisation de n'importe quel type d'automates.
Il pourrait donc, au futur, être envisagé d'ajouter des automates à piles ainsi que tout autre modèle s'inscrivant dans les modèles de calcul.

Conclusion

Pour notre projet nous devions réaliser une interface pour automates et machine de Turing. Le projet a été réalisé et nous pouvons observer la capacité de l'application que nous avons produite à pouvoir évoluer pour implémenter d'autres modèles de calcul.
Techniquement, ce projet nous a apporté de nouvelles connaissances et une nouvelle expérience qui nous donne un avant-goût du travail en entreprise.
Nous avons aussi pu surmonter les difficultés de conception de notre application (ajout des machines de Turing après l'implémentation des automates).
Surmonter cela permet de donner une certaine plasticité à notre logiciel pour de potentielles mises à jour et nouvelles fonctionnalités.

Bibliographie

[1] « 🔎 Machine de Turing - Définition et Explications », *Techno-Science.net*. https://www.techno-science.net/glossaire-definition/Machine-de-Turing.html .

[2] K. Dickerson, « Automaton Simulator ». https://automatonsimulator.com/ .

[3] C. Burch, « Automaton Simulator: Download », 2006. http://www.cburch.com/proj/autosim/download.html .

[4] P. Valicov, « Bases de la programmation orientée objet Exceptions », p. 4.

[5] P. Valicov, « Bases de la programmation orientée objet Généricité et Structures de données », p. 12.

[6] P. Valicov, « Bases de la programmation orientée objet Héritage et Polymorphisme », p. 9.

[7] P. Valicov, « Bases de la programmation orientée objet Introduction », p. 14.

[8] J.-M. Doudoux, « Développons en Java - Maven », 2019. https://www.jmdoudoux.fr/java/dej/chap-maven.htm .

[9] « JavaFX », *Wikipédia*. oct. 08, 2021. [En ligne]. Disponible sur: https://fr.wikipedia.org/w/index.php?title=JavaFX&oldid=186979836

[10] KooR.fr, « KooR.fr - Apprendre à coder une classe générique - Le tutoriel sur le langage de programmation Java », 2021. https://koor.fr/Java/Tutorial/java_generics_classe.wp .

[11] M. Ugarte, « Online Turing Machine Simulator », 2017. https://turingmachinesimulator.com/ .

[12] « Overview (JavaFX 17) », 2021. https://openjfx.io/javadoc/17/ .

[13]
« [Résolu] Java FX c'est quoi au juste ? par gstrat - OpenClassrooms », févr. 2015. https://openclassrooms.com/forum/sujet/java-fx-c-est-quoi-au-juste .

[14]
« Théorie des automates ». http://stringfixer.com/fr/Automata_theory .

[15]
A. Morphett, « Turing machine simulator ». http://morphett.info/turing/ .