



Université Paris Descartes
UFR de Mathématiques et Informatique
Master 1 Informatique

Projet de Complexité Algorithmique

Thème

Implémentation d'un interpréteur de machines de Turing

Réalisé par:

| | |
|--------------------|----------|
| BELAHCEL Wacim | 21911555 |
| BEN ABDELATIF Lina | 21912117 |
| BOURAÏ Assia | 21907038 |

Chargé de cours : Jean-Guy Mailly

2019-2020

1. Table des matières

| | |
|---|----|
| 1. Table des matières | 2 |
| 2. Table des figures | 2 |
| 3. Introduction | 3 |
| 4. Objectif du projet..... | 3 |
| 5. Planning et répartition des tâches | 4 |
| 6. Environnement de compilation et d'exécution | 4 |
| 7. Fonctionnalités de l'interpréteur | 5 |
| 7.1. Lecture du fichier de description de la machine de Turing..... | 5 |
| 7.2. Affichage du résultat de l'exécution | 5 |
| 7.3. Gestion des erreurs | 6 |
| 8. Description des classes implémentées | 7 |
| 8.1. Classe Transition | 7 |
| 8.2. Classe Tape | 7 |
| 8.3. Classe State | 8 |
| 8.4. Classe TuringMachine..... | 8 |
| 9. Fonctionnement du programme..... | 10 |
| 9.1. Instanciation de la classe TuringMachine | 10 |
| 9.2. Simulation de la machine sur l'entrée | 11 |
| 10. Conclusion | 13 |

2. Table des figures

| | |
|---|----|
| Figure 1 - Diagramme de Gantt | 4 |
| Figure 2 - Exemple d'exécution du programme..... | 5 |
| Figure 3 - Exemple de gestion d'erreurs | 6 |
| Figure 4 - Diagramme de classes | 7 |
| Figure 5 - Diagramme de séquence étape d'instanciation..... | 10 |
| Figure 6 - Diagramme de séquence étape de simulation | 11 |

3. Introduction

L'acquisition de connaissances solides dans n'importe quel domaine scientifique ne peut se faire que par une phase de recherche, d'étude, et d'application rigoureuse, ainsi le projet de mise en place d'une machine de Turing universelle fut pour nous une opportunité d'en apprendre davantage sur le sujet, ainsi que d'appliquer les connaissances acquises durant nos cours.

Dans ce court rapport, nous tacherons d'expliquer le travail que nous avons eu à effectuer, en passant par l'organisation du projet, les outils que nous avons utilisés, nous présenterons ensuite les différentes fonctionnalités mise en place ainsi que le fonctionnement interne du programme pour finir avec une courte conclusion.

4. Objectif du projet

L'objectif de ce projet est de mettre en œuvre un interpréteur de machine de Turing déterministe, i.e. un programme permettant de lire la description d'une machine de Turing déterministe à partir d'un fichier et d'exécuter cette machine sur un mot en entrée.

5. Planning et répartition des tâches

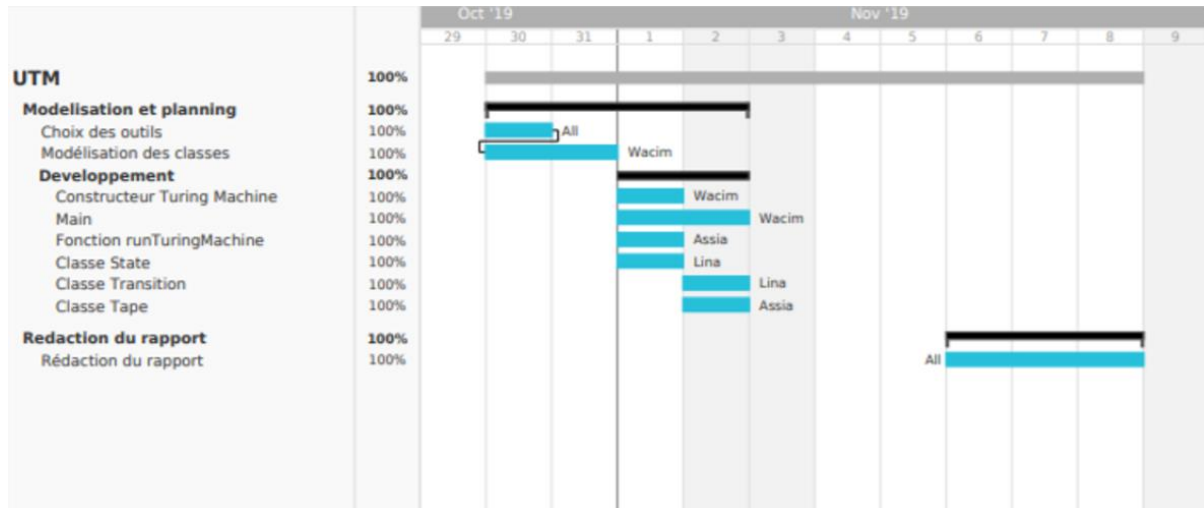


Figure 1 - Diagramme de Gantt

6. Environnement de compilation et d'exécution

Le projet a été testé et compilé sur des machines Windows 8.1, Windows 10, Ubuntu 19.04.

La compilation se fait en utilisant le compilateur g++, un script build.sh est mis à disposition afin de compiler le programme, en sortie un exécutable nommé UTM(.exe pour Windows) sera utilisé afin d'exécuter le programme.

UTM [Options]... TM_FILE_DESCRIPTOR... INPUT...

- Options : il y'a 3 types d'options possibles :
 - -steps : affichera chaque étape de transition.
 - -state : affichera l'état final.
 - -tape : affichera le contenu de la tape.
- TM_FILE_DESCRIPTOR (Obligatoire) : le nom du fichier d'extension .tm qui contiendra la description de la machine de turing à simuler.
- INPUT (Obligatoire : l'input à fournir en entrée à la machine de turing simulé.

7. Fonctionnalités de l'interpréteur

Le programme implémenté consiste en une machine de Turing universelle prenant en entrée la description d'une machine de Turing déterministe et un mot en entrée et exécutant cette machine sur ce dernier.

7.1. Lecture du fichier de description de la machine de Turing

Le fichier de description de la machine de Turing (nomfichier.tm) est un fichier texte où chaque ligne décrit un élément de la machine.

Ce fichier doit respecter un certain nombre de règles. Ces dernières sont décrites dans ce qui suit :

- La ligne **initial(XXX)**, décrit l'état initial de la machine avec **XXX**, nom de l'état initial
- Les lignes **state(XXX,Y)**, décrit le reste des états de la machine, tel que **XXX** représente l'état actuel de la machine et **Y** indique si l'état final a été atteint : **T(true)** ou pas **F(false)**.
- La ligne **Blank(B)**, donne le nom du symbole blank et les autres symboles sont décrits par les lignes **symbol(X,Y)** où **X** est le nom du symbole et **Y** représente **T(true)** ou **F(false)** et indique si **X** est un symbole appartenant au vocabulaire d'entrée Σ .
- Les lignes **transition(XXX,X,YYY,Y,M)** représentent les différentes transitions de la machine où **XXX** représente l'état courant, **X** représente le symbole lu, **YYY** représente l'état suivant et **M** indique la direction du déplacement sur le ruban i.e. (**R: right** ou **L: left**).

7.2. Affichage du résultat de l'exécution

Le programme prend en argument un certain nombre de paramètres en plus du fichier de description de la machine de Turing et du mot. Ces derniers servent à spécifier l'affichage du résultat donné lors de l'exécution.

Les arguments sont les suivants :

- **-steps** pour obtenir un affichage contenant toutes les étapes d'exécution de la machine.
- **-state** pour afficher l'état final de la machine.
- **-tape** pour afficher le contenu du ruban à la fin de l'exécution.

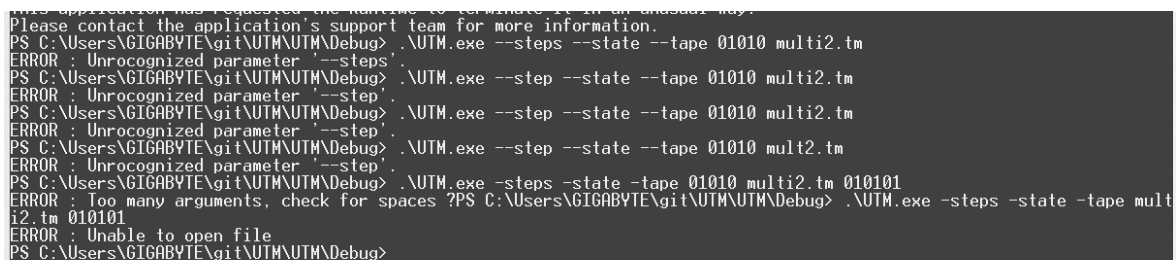
```
ERROR : unrecognized parameter -step :
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe -state -steps -tape .\mult2.tm 101010
(In Progress, 1) -> (In Progress, 1, R)
(In Progress, 0) -> (In Progress, 0, R)
(In Progress, 1) -> (In Progress, 1, R)
(In Progress, 0) -> (In Progress, 0, R)
(In Progress, 1) -> (In Progress, 1, R)
(In Progress, 0) -> (In Progress, 0, R)
(In Progress, 0) -> (In Progress, 0, R)
(In Progress, 0) -> (Done, 0, R)
Final State: Done
Final Tape : 1010100
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug>
```

Figure 2 - Exemple d'exécution du programme

7.3. Gestion des erreurs

L'interpréteur implémenté effectue des vérifications de syntaxe et renvoie des exceptions si des erreurs ont été trouvées.

- Vérification des arguments passés en paramètres.
- Vérifier si le fichier de description passé en paramètre peut être lu et s'il respecte bien toutes les règles décrites dans la section 1.1, à savoir contenir l'état initial, le symbole blanc et le reste des symboles de l'alphabet d'entrée, les transitions et les différents états.
- Vérifier que le fichier décrit bien une machine de Turing déterministe, c'est à dire que pour un état donné il ne peut y avoir qu'une seule transition possible vers un autre état.
- Vérifier que l'input contient bien les symboles du vocabulaire d'entrée.
- Vérifier que pour l'input donné en paramètre, il existe bien une transition permettant de passer d'un état à un autre.
- Vérifier si la direction du déplacement est bien définie, i.e. soit L soit R (Left ou Right).



```

Please contact the application's support team for more information.
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe --steps --state --tape 01010 multi2.tm
ERROR : Unrecognized parameter '--steps'
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe --step --state --tape 01010 multi2.tm
ERROR : Unrecognized parameter '--step'
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe --step --state --tape 01010 multi2.tm
ERROR : Unrecognized parameter '--step'
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe --step --state --tape 01010 mult2.tm
ERROR : Unrecognized parameter '--step'
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe -steps -state -tape 01010 multi2.tm 010101
ERROR : Too many arguments, check for spaces ?PS C:\Users\GIGABYTE\git\UTM\UTM\Debug> .\UTM.exe -steps -state -tape mult
i2.tm 010101
ERROR : Unable to open file
PS C:\Users\GIGABYTE\git\UTM\UTM\Debug>

```

Figure 3 - Exemple de gestion d'erreurs

8. Description des classes implémentées

La figure ci-dessous illustre le diagramme de classes de notre interpréteur.

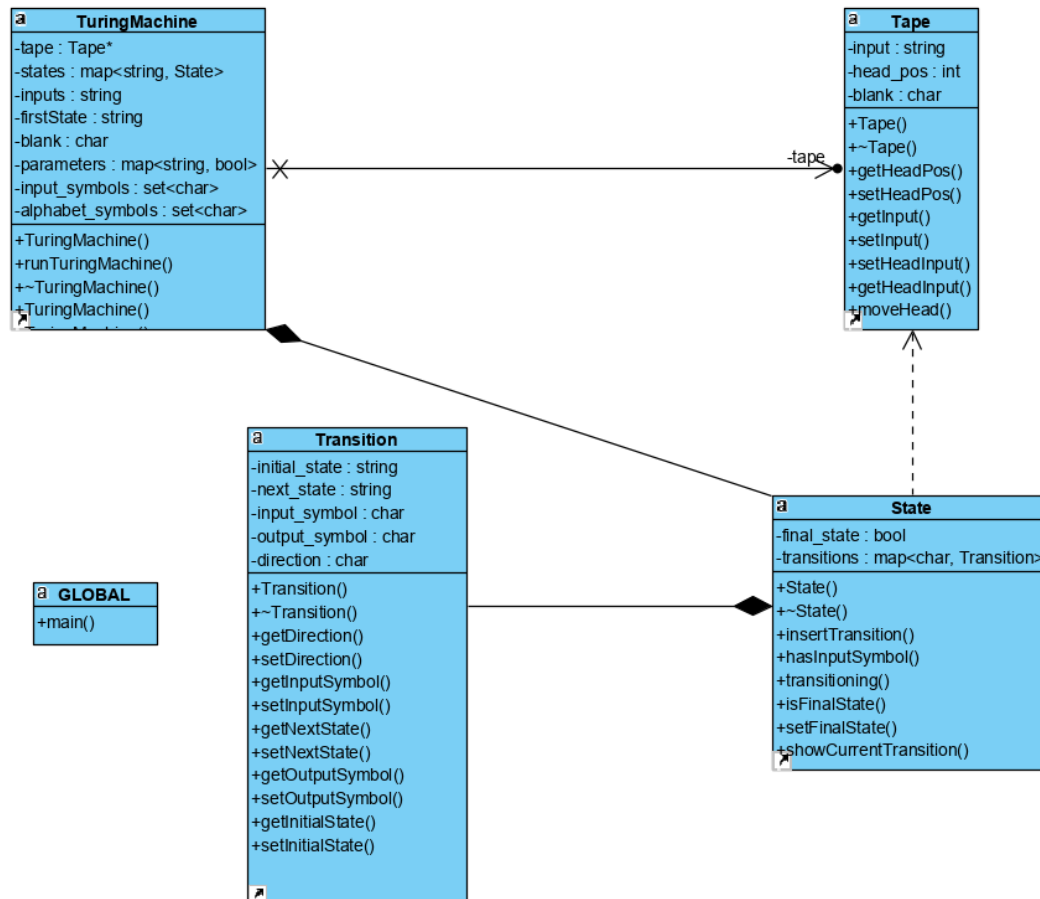


Figure 4 - Diagramme de classes

8.1. Classe Transition

La classe transition, contient comme son nom l'indique, tous les attributs d'une transition (initial_state, input_symbol, output_symbol, direction, next_state). Elle sert à stocker le contenu d'une transition en particulier, lue à partir du fichier de description de la machine de Turing. Pour chacun de ses attributs, des setters et des getters ont été définis.

8.2. Classe Tape

La classe Tape, représente le ruban de lecture/écriture de la machine de Turing. Elle comporte le contenu du ruban (input_symbol) et la position de sa tête (head_pos) tout au long de l'exécution de la machine de Turing.

Fonctions manipulées :

- Des setters et getters pour chacun des attributs input_symbol et head_pos.

- **MoveHead()** : Cette fonction permet de déplacer la tête du ruban en fonction de la direction spécifiée dans la transition.

8.3. Classe State

Chaque objet de type **State** représente un état de transition, et est composé d'une liste de transitions (objets de type **Transition**) que l'on puisse effectuer à partir de cet état.

Fonctions manipulées :

- **InsertTransition()** : Cette fonction permet d'ajouter une transition à la liste.
- **HasInputSymbol()** : Cette fonction permet de vérifier si un mot en entrée se trouve déjà dans la liste de transitions, et donc de lancer une erreur au niveau de la fonction **InsertTransition** si le mot en entrée est présent dans la liste au moment de l'insertion de la transition.
- **Transitionning** : Cette fonction prend en paramètre un objet de type **Tape**, et renvoie le prochain état.
- **isFinalState()** : Cette fonction permet de retourner l'état final.
- **ShowCurrentTransition()** : Cette fonction permet d'afficher la transition en cours.

8.4. Classe TuringMachine

La classe **TuringMachine**, permet d'initialiser la machine de Turing selon le fichier passé en paramètre en vérifiant toutes les erreurs de syntaxes possibles.

Un objet de cette classe possède les attributs ci-dessous qui sont initialisés lors de la lecture du fichier :

-Un objet de la classe "Tape"

-Une liste d'états "states", cette liste est contenue dans un map <string, State> dans lequel chaque élément possède comme clé le nom d'un état et comme valeur un objet de la classe "State".

-Une chaîne de caractère "inputs" représentant le mot donné en paramètre et sur lequel la machine sera exécutée.

-Une chaîne de caractère "firstState", qui stocke le nom de l'état initial de la machine.

-Un caractère "blank" représentant le mot blanc.

- "parameters" qui représente les paramètres -state, -steps ou -tape donnés en arguments au programme.

-Un ensemble "input_symbols", représentant le vocabulaire d'entrée de la machine de Turing.

-Un ensemble "alphabet_symbols", représentant l'alphabet de la machine de Turing.

-Un objet de la classe "Tape".

Fonctions manipulées :

- `turingMachine()`

L'interpréteur construit la machine de Turing en lisant le contenu du fichier et en remplissant la liste "states", en prenant le soin de ne pas ajouter un élément qui existe déjà dans la liste.

Pour chaque élément de la liste "state", les transitions liées à un état sont ajoutées dans le map "transitions" de l'objet "State" manipulé, et si une clé existe déjà dans le map alors une exception est levée (car nous sommes dans le cas d'une machine déterministe).

Dans le cas où la machine ne contient aucun état ou aucune transition une exception est renvoyée.

- `runTuringMachine()`

La fonction `runTuringMachine()` prend l'input en paramètre afin de lancer l'exécution de la machine.

Ceci est fait en initialisant un nouvel objet "Tape" et en transitionnant d'un état à un autre en utilisant la fonction `transitionning()` de la classe "State".

9. Fonctionnement du programme

Le fonctionnement du programme est réparti en deux parties majeures, qui correspondent aux deux fonctions principales de TuringMachine, l'instanciation de la classe grâce au constructeur et au fichier décrivant la machine, et le lancement de la machine sur l'input envoyé par l'utilisateur.

9.1. Instanciation de la classe TuringMachine

Lors de l'instanciation de la classe TuringMachine, deux arguments sont envoyés au constructeur, le nom du fichier décrivant la machine de Turing à simuler, ainsi que les paramètres envoyés par l'utilisateur.

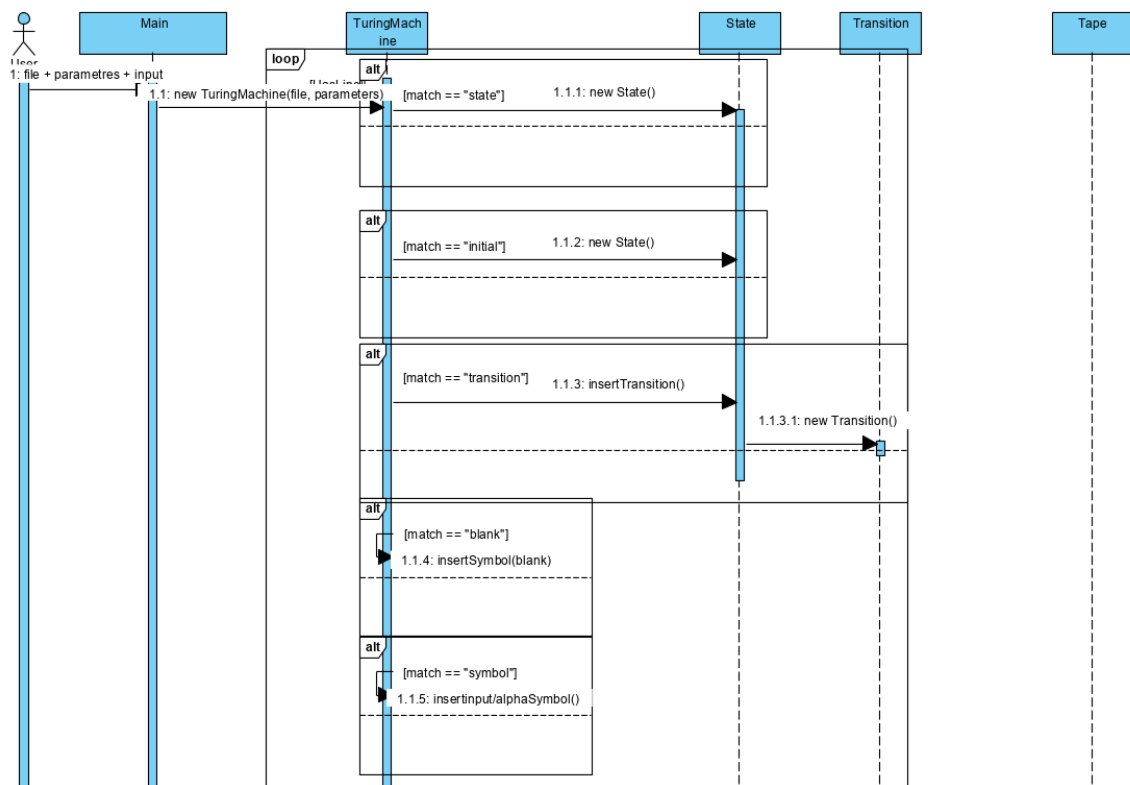


Figure 5 - Diagramme de séquence étape d'instanciation

Le constructeur quant à lui sert majoritairement tel que décrit plus haut dans la description de la classe, à lire et « dé sérialiser » le contenu du fichier décrivant la machine de Turing afin de la convertir en Objet qui pourra ensuite être utilisé par notre programme, ainsi on peut voir sur la figure ci-dessus l'ensemble des cas par lequel le constructeur pourrait passer en lisant les lignes du fichier :

- En lisant la ligne « initial » ou « state » le constructeurinstanciera un nouvel objet state qui fera partie de sa liste d'états.
- En lisant la ligne « transition » le constructeur fera appelle à la méthode insertTransition du state correspondant, qui à son tour instanciera une nouvelle transition qui sera ajouté à la liste des transitions de l'état correspondant.

- « Blank » ajoutera le symbole « vide » aux paramètres de notre machine de Turing ainsi qu'à la liste de l'alphabet.
- « symbol » ajoutera un symbole à la liste des entrées et à l'alphabet.

Plusieurs cas d'erreurs sont aussi pris en considération (cités plus haut), si aucune erreur n'est détectée durant le processus d'instanciation, le programme continuera son cours normal vers la prochaine phase.

9.2. Simulation de la machine sur l'entrée

Une fois l'instanciation terminée, le programme passe au lancement d'une simulation de la machine sur l'input envoyé en argument et cela en utilisant la fonction `runTuringMachine` de la classe `Turingmachine` en utilisant l'input envoyé en paramètre.

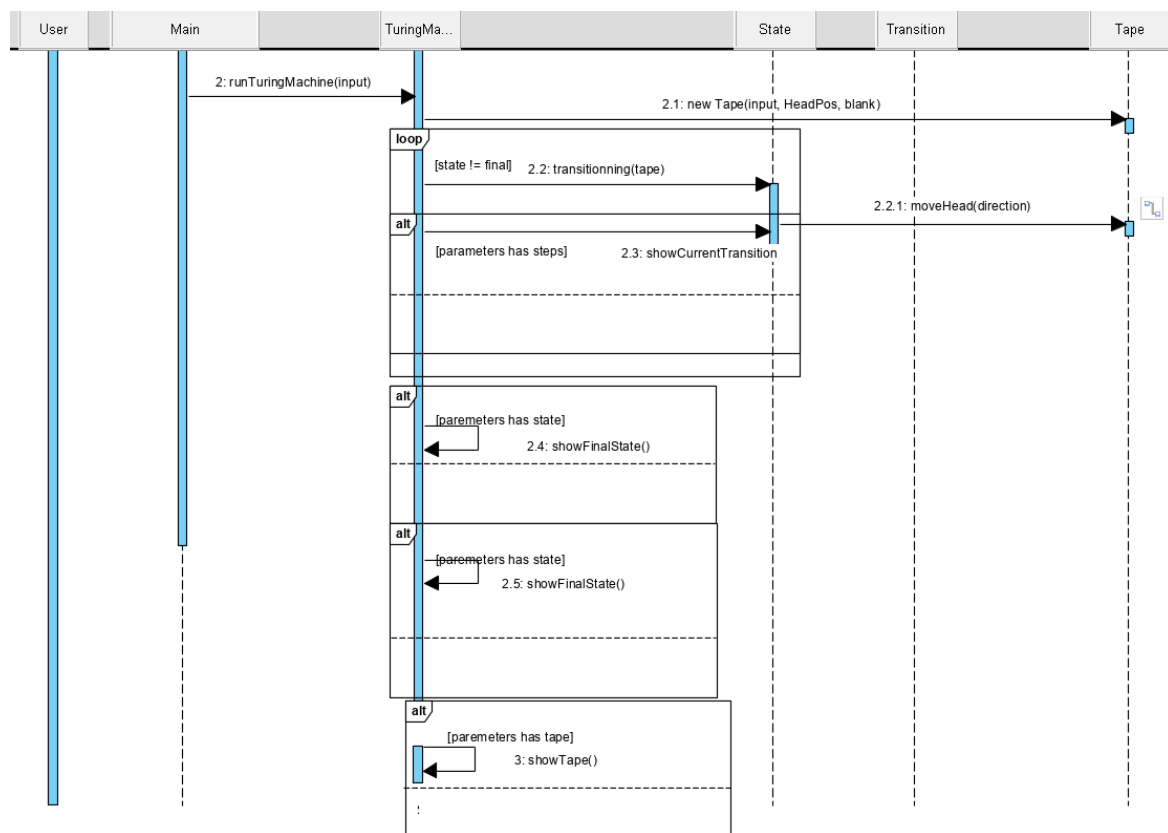


Figure 6 - Diagramme de séquence étape de simulation

Dans la fonction `turingMachine`, on commence tout d'abord par instancier un nouvel objet `Tape` dont le constructeur place la tête au début de la tape (représenté par un vecteur), l'input est ensuite copié dans le vecteur de la tape.

Ensuite, on stocke l'état actuel (initial au départ), dans une référence de type « `State` », on utilise ensuite une boucle `while`, tant que l'état actuel n'est pas un état final on continue à boucler, à l'intérieur de la boucle on utilise la fonction « `transitionning` » afin de passer au prochain état, qui fait appel à la fonction `moveHead` de la classe `Tape`.

Si le paramètre « steps » est à true, on affiche chaque étape de transition. Une fois en dehors de la boucle, si le paramètre « state » est à true, on affiche l'état final et si le paramètre « tape » est à true, on affiche le contenu de la tape.

10. Conclusion

En conclusion, ce projet a été pour nous une opportunité d'apprendre autant sur le plan technique que théorique, nous avons ainsi pu nous forger une certaine rigueur de travail qui nous permettra d'évoluer plus rapidement dans de futurs projets.

De plus, ce projet nous a permis d'appliquer des concepts que l'on a pu voir lors de nos séances de Tds, et d'élargir nos connaissances, nous permettant ainsi d'avoir de bonnes bases en complexité algorithmique.