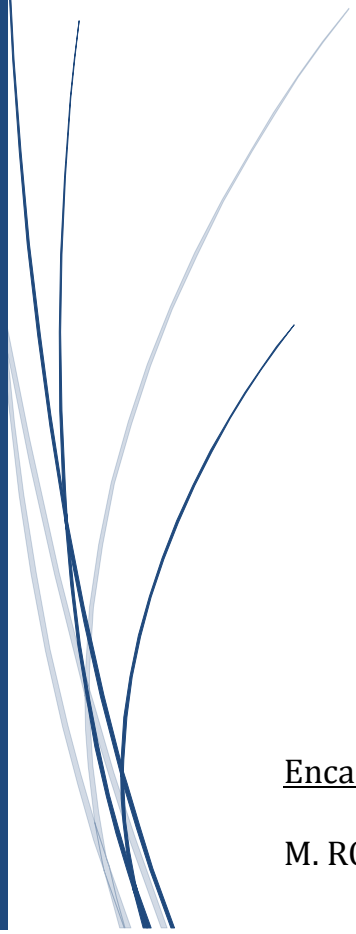
2014-2015

Projet Enigma

Compte-Rendu

DUT Informatique (S4)

TROUVE Robin
KISSI Naïm
DE LIMA Marianna



Encadré par:

M. ROY

Sommaire

Introduction

Dans le cadre de notre DUT Informatique, en semestre 4, nous sommes amenés à réaliser un projet alliant mathématiques et informatique afin d'appliquer nos connaissances et la méthodologie que nous avons pu acquérir au cours de notre formation.

Le sujet de ce projet est la machine Enigma, machine de cryptage la plus connue et utilisée lors de la Seconde Guerre Mondiale. Il s'agit donc ici de comprendre son fonctionnement, ses mécanismes et de pouvoir reproduire son comportement de manière informatisée.

Ce compte-rendu détaille notre démarche pour aborder le sujet, le comprendre et enfin réaliser techniquement le projet en s'organisant au sein du trinôme, en utilisant des outils et les méthodes appris en cours.

Nous allons donc voir dans une première partie ce qu'est la machine Enigma, son rôle, son fonctionnement et nous nous intéresserons aussi à sa cryptanalyse par Alan Turing pour comprendre ses failles. Nous verrons ensuite comment nous nous sommes organisés pour mener le projet à bien en mettant toutes nos connaissances en œuvre et nous verrons aussi les difficultés nous avons rencontrés et comment nous les avons surmontées.

1. Enigma : Machine de cryptage

1.1. Création d'Enigma

1.1.1. Remise en contexte

Enigma est une machine électromécanique permettant de crypter et de décrypter un message.
Cette machine est essentiellement composée de trois rotors, d'un réflecteur, d'un plugboard, d'un clavier et d'un afficheur de lettre cryptée mais nous détailleront plus en profondeur tous ces composants dans la partie suivante et nous expliqueront les fonctionnalités de chacun.

Arthur Scherbius

Enigma a été créée par un ingénieur en électricité allemand du nom d'Arthur Scherbius, né en 1878, qui a étudié à Munich puis à Hanovre.
Initialement il n'avait pas pour ambition de créer une machine de chiffrement d'ailleurs à la fin de ses études en 1903 il consacre sa thèse à des systèmes de turbines à eau.
C'est dans les années 1910 plus précisément à partir de 1918 qu'Arthur Scherbius commence à s'intéresser aux systèmes de chiffrement, il décide alors d'entamer la création d'un système de cryptage révolutionnaire pour l'époque du nom d'Enigma.

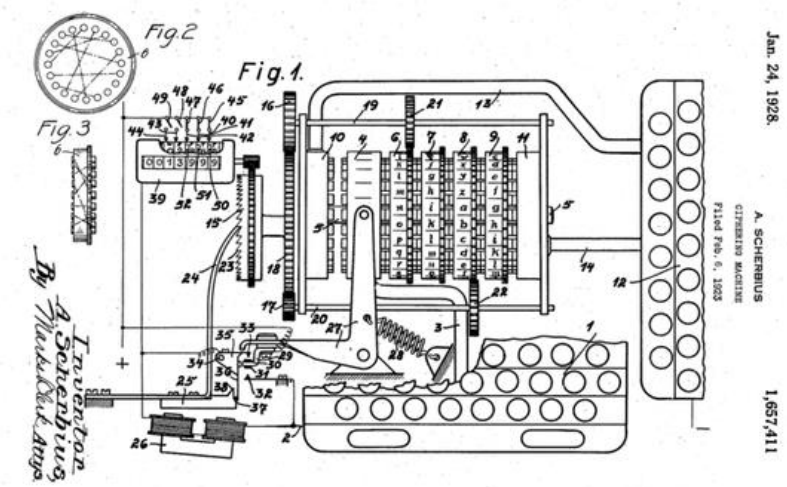
Enigma était le seul système de cryptage possédant des rotors désynchronisés et un plugboard, permettant de décupler le nombre de combinaison possible et de dépasser de loin tous les systèmes de chiffrement de l'époque en terme de sécurité (ex : le chiffre de Vigenère).

Crée par Vigenère lui-même en 1586, la méthode du chiffre de Vigenère consistait à substituer une lettre par une autre avec un chiffrement polyalphabétique (il utilisait 10 alphabets différents). Cette méthode de chiffrement a résisté trois siècles aux cryptanalystes et était donc une des méthodes de chiffrement les plus sûres. Cependant un major prussien du nom de Friedrich Kasiski a réussi à briser le mystère de ce système en 1863, le rendant ainsi obsolète.

Vigenère

La machine Enigma, de par son fonctionnement offrant un grand nombre de possibilités de combinaison, fut une véritable révolution à l'époque, elle était considérée comme inviolable car il fallait avoir la même machine que celle qui a crypté le message pour avoir une chance de le décrypter. De plus, il était impossible de tester toute les possibilités de combinaison d'Enigma sans connaitre la combinaison initiale des rotors.

Afin de protéger sa création, Arthur Scherbius a décidé de breveté son œuvre, et d'autres talentueux inventeurs Koch firent de même comme le hollandais en déposant un brevet en 1919. Ce dernier fut racheté par la société de Scherbius afin d'éviter que cela ne leur porte préjudice dans le futur.



Brevet déposé par Arthur Scherbius

1.1.2. Utilisation d'Enigma

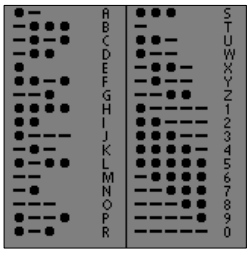
Enigma-A

Après avoir breveté son invention, Arthur Scherbius décida de commercialiser sa machine l'Enigma-A or son invention n'eut pas le succès escompté. En effet c'est un échec commercial car sa machine est considérée comme « trop chère » avoisinant les 30 000 euros, pourtant plus tard trois autres versions commerciales verront le jour.

Ici nous n'allons pas nous intéresser à la version commercial mais plutôt à la version militaire qui a été surnommée l'Enigma-D et qui a été adoptée par l'armée allemande dès 1926 plus précisément par la Marine puis par l'armée de terre en 1929. Par la suite, la machine a été exploitée par les nazis qui l'ont renommée « machine M ».

Enigma-D

Enigma fut l'un des plus grands atouts de communication et était très utilisée par l'Allemagne nazie et ses alliés pendant la seconde guerre mondiale. Ils l'utilisaient pour les bulletins météo mais surtout pour les informations sensibles notamment sur les tactiques militaires et les stratégies à adopter face à l'ennemi. Les nazis avaient donc une totale confiance dans le système d'Arthur Scherbius. Ils changeaient tout de même régulièrement la configuration des rotors pour plus de sécurité et notaient sur un livre les positions du jour de chaque rotor (cf annexes).
Les messages étaient cryptés par la machine Enigma avec les positions du jour et au début du message crypté, la clé était incluse pour permettre au receveur de trouver la bonne position afin de décrypter le message. Evidemment les deux machines devaient être les mêmes pour avoir un réglage de machine cohérent entre le cryptage et le décryptage. Le message crypté était envoyé un morse par onde radio au destinataire.



Le morse est un langage qui permet de communiquer à distance via onde radio, ce langage est composé de tiret et de point et est accompagné d'un dictionnaire permettant de traduire les tirets et les points en lettre. On utilise ce langage par le biais d'un télégraphe Morse, cet outil ce compose essentiellement d'un manipulateur, d'un relais et d'un récepteur.
Le manipulateur est composé d'un socle de bois qui possède deux bornes et un levier, qui peut osciller verticalement.
Pour envoyer un message, il faut appuyer sur la poignée du levier pour que le courant puisse passer, puis dès que l'on relâche le levier le courant s'interrompe.
C'est de cette manière que la manipulation est transmisse au récepteur via onde radio c'est à dire entre 9 kHz à 3 000 GHz.
Le récepteur utilise un électro-aimant qui est connecté à la Terre et au fil de ligne.
Dès que le courant passe dans le récepteur la plaque de fer de l'électro-aimant est attirée et repoussée grâce au ressort de rappel due au passage ou à l'interruption du courant.
Grâce à ce mécanisme le levier pouvait ainsi osciller de haut en bas, en bas dès que le courant passé dans le récepteur et donc le levier munie d'une pointe pouvait laisser des traces sur le papier et le levier était en position haut des que le courant était interrompue ainsi le résultat était récupéré sur le papier et pouvait être traduit en lettre via le dictionnaire Morse.

Le relais quant à lui été uniquement utilisé pour augmenter la puissance du courant électrique, en effet le courant de ligne à lui seul n'était pas suffisant pour laisser des traces sur le papier du coup le relais lui, est composé d'une pile et il pouvait ainsi ajouter sont courant à celui du courant de ligne pour amplifier le courant total.

1.2. Fonctionnement d'Enigma

1.2.1. Les rotors

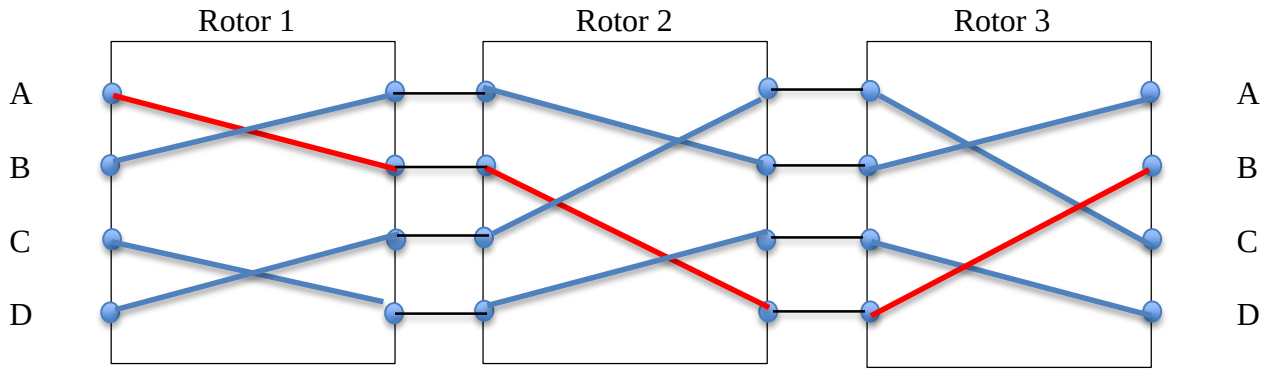
Nous allons à présent voir en détail comment Enigma fonctionne, quelles sont ses particularités qui l'ont rendu si difficile à « casser ».

La puissance d'Enigma réside dans le fait qu'elle est imprévisible en terme de chiffrement des mots rentrés et cela est directement permis par sa structure interne. C'est la combinaison de ses composants qui la rend si performante mais quels sont-ils ? En tant que machine électromécanique, Enigma est composée de rotors fonctionnant avec des contacts électriques permettant la subsitution poly-alphabétique, c'est d'ailleurs une de ses particularités.

Les rotors servent de connexion électriques, ils sont au nombre de 3 dans la machine et pouvaient être choisis parmi 5 rotors en tout. Leur particularité vient du fait qu'ils sont cylindriques et fixés sur un axe autour duquel ils peuvent tourner. A chaque lettre tapée, le rotor effectue une rotation qui change complétement la donne pour la lettre suivante car du coup, la permutation sera différente pour chaque lettre même si celle-ci est retapée.

C'est un des points forts d'Enigma car de ce fait, l'analyse par fréquence qui consiste à repérer les lettres qui reviennent fréquemment dans un message crypté est impossible. Mais comme signalé, il y a 3 rotors en tout. Chaque rotor représente les lettres/chiffres que l'on veut crypter. Par exemple, si on prend l'alphabet de 26 lettres, chaque rotor aura 26 positions. Le 1er rotor tourne d'un cran à chaque fois qu'une lettre est tapée, de sa 26^{e} position à sa position initiale, il déclenche le 2^{e} rotor qui lui aussi tourne d'un cran. Lorsque le 2^{e} rotor effectue sa rotation de sa 26^{e} lettre à sa position initiale, il déclenche à son tour le 3^{e} rotor. Les rotors reviennent à leur position initiale lorsqu'ils ont tous parcouru leur 26 positions. Chaque rotor pouvait être positionné de 1 à 26 avant de commencer à taper le message.

Nous avons schématisé les 3 rotors avec 4 lettres selon un circuit électrique de cette manière :



On peut voir ici que si on tape lettre « A », elle sera permutée en « B » à l'issu du rotor 1 puis en « D » et enfin en « B ». Le rotor 1 tournera d'un cran et ses sorties seront décalées ce qui permettra d'avoir une lettre cryptée différente même si on retape à nouveau « A ».

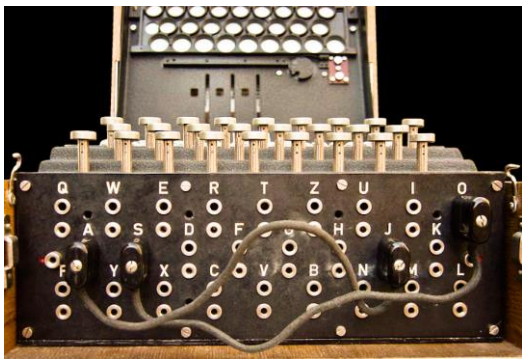
Si on calcule le nombre de possibilités de cryptage qu'offrent les rotors, nous avons :

$$5 * 4 * 3 = 60$$
$$26^3 = 17\,576$$
$$\text{Soit : } 60 * 17\,576 = 1\,054\,560$$

Jusqu'ici nous avons donc 60 possibilités de choix de rotors (3 parmis 5) et 17 576 (pour un alphabet de 26 lettres) positions possibles des rotors car chaque rotor à 26 positions donc 26*26*26 soit un total de 1 054 560 possibilités de cryptage. Si l'alphabet comporte plus de caractères, cela augmentera le nombre de positions possibles des rotors et donc le nombre de possibilités total.

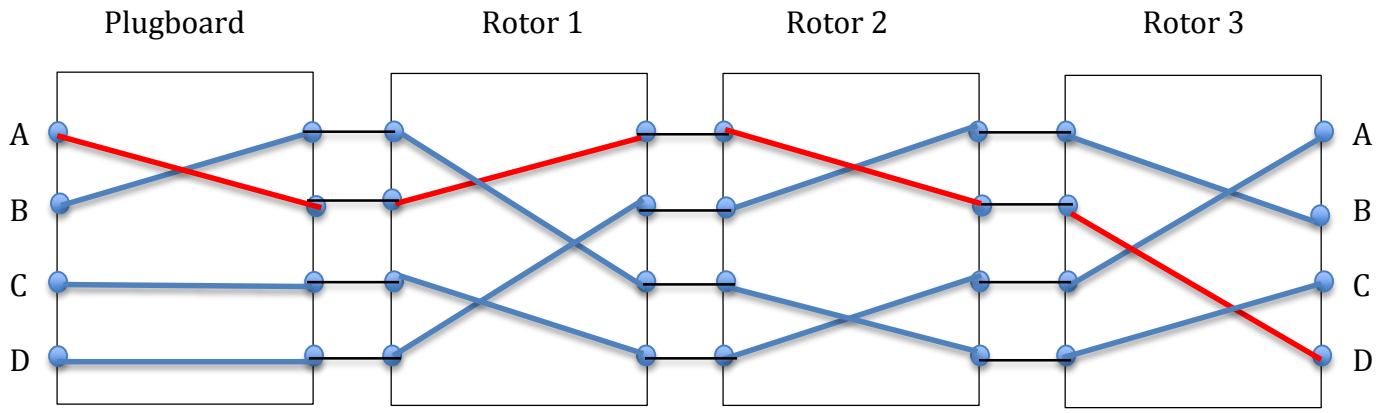
1.2.2. Le plugboard

Mais Enigma ne s'arrête pas à ces trois rotors tournant, en effet, un des composants central de cette machine est le tableau de connexion (plugboard), situé devant la machine et permettant d'effectuer 10 paires de permutation. Le plugboard était spécialement conçu pour les machines utilisées à des fins militaires, les versions commerciales en étaient dépourvues.



Les permutations sont commutatives c'est-à-dire que si la lettre « A » est reliée à la lettre « B » alors la lettre « A » sera permutée avec la lettre « B » mais la lettre « B » sera permutée avec la lettre « A ».
Il y a donc 20 lettres permutées et 6 inchangées dans un alphabet de 26 lettres.
Le but du plugboard était de brouiller les pistes car une lettre tapée était d'abord permutée suivant le tableau de connexion puis codée.

Concrètement, cela signifie que si « A » était permutée avec « B », c'est la lettre « B » qui serait effectivement cryptée en passant dans les rotors.
Nous avons donc enrichi notre schéma (cf rotors ci-dessus) avec le plugboard ce qui nous donne sur un alphabet de 4 lettres :



Dans ce cas-ci, les lettres « A » et « B » sont permutées alors que les lettres « C » et « D » restent inchangées. Si on veut crypter la lettre « A », c'est donc en fait la lettre « B » qui va être cryptée et on obtiendra donc en permutations successives: A->B, B->A, A->B, B->D.

Si nous calculons les possibilités qu'offrent les branchements frontaux pour un alphabet de 26 lettres nous avons :

$$\frac{26\,!}{6\,!10\,!2^{10}} = 150\,738\,274\,937\,250$$

En effet, nous avons 26 combinaisons de lettres (26x25x24…x1) ce qui justifie le « 26 ! ». Cependant, nous ne voulons faire que 10 paires de lettres donc 6 lettres restent non permutées. Etant donné que l'ordre des combinaisons n'importe pas, nous pouvons diviser par « 6 ! » et multiplier par le nombre de combinaison possible pour les 10 paires soit « 10 ! ». Enfin les paires sont constituées de deux lettres interchangeables, on peut alors diviser par « 2 » et comme il y a 10 paires de lettres cela nous donne « 2^{10}»
Le plugboard est donc un élément central car c'est lui qui augmente considérablement le nombre de possibilités.
Si on rajoute les possibilités qu'offraient les rotors nous avons :

$$150\,738\,274\,937\,250 * 1\,054\,560 = 158\,962\,555\,217\,826\,360\,000$$

Voilà donc le nombre de possibilités de cryptage que permettait Enigma grâce à la combinaison de ses composants.

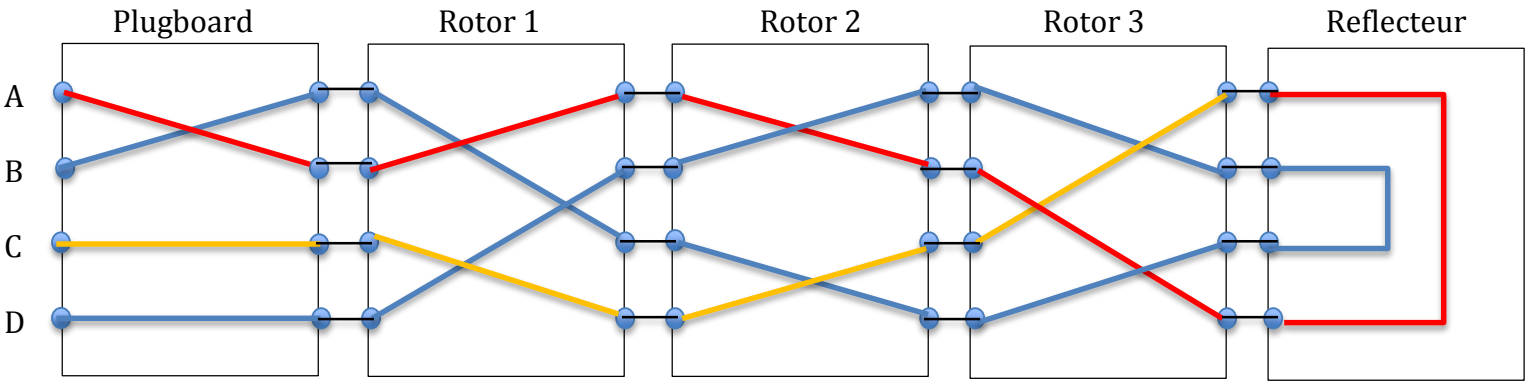
1.2.3. Le réflecteur

Cependant, à ce stade, la machine est capable de crypter mais pas de décrypter. Il faudrait donc une machine pour crypter et créer une seconde machine pour décrypter. Pour pallier à ce problème, Enigma s'est dotée d'un réflecteur, une sorte de rotor fixe qui a pour but de faire une ultime permutation et de renvoyer le courant dans l'autre sens. De ce fait, il inverse par paires les lettres et permet de lier les lettres de façon à pouvoir rendre le cryptage réversible. Concrètement, si on tape la lettre « A » et qu'elle est cryptée en « D » alors lorsqu'on décryptera la lettre « D » elle renverra « A » (la machine doit avoir les mêmes paramétrages pour le cryptage et le décryptage).

Il existait plusieurs versions de réflecteur. La version B était la plus couramment utilisée mais il a aussi existé les versions C et D. Certaines de ces versions permettant de positionner le réflecteur de différentes manières (différentes positions un peu comme les rotors).

Le principe même du réflecteur empêchait toute lettre d'être codée par elle-même ce qui sera largement exploité lors de la cryptanalyse de Turing que nous verrons dans la prochaine partie.

Nous pouvons enrichir une dernière fois notre schéma pour montrer une version complète du fonctionnement d'Enigma :



Légende : Cryptage : chemin aller
chemin retour
Decryptage : chemin aller
chemin retour

Reprenons notre exemple pour crypter la lettre « A ».

ALLER	REFLECTEUR	RETOUR
Plugboard : « A » devient « B »	Réflecteur : « D » devient « A »	Rotor 3 : « A » devient « C »
Rotor 1 : « B » devient « A »		Rotor 2 : « C » devient « D »
Rotor 2 : « A » devient « B »		Rotor 1 : « D » devient « C »
Rotor 3 : « B » devient « D »		Plugboard : « C » reste « C »

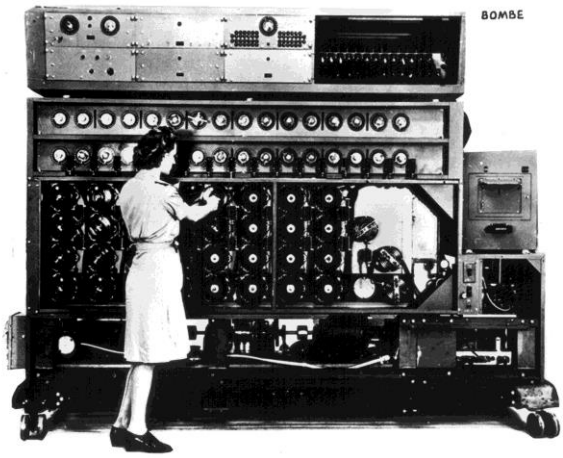
« A » est donc cryptée en « C » et si on veut décrypter « C », on remonte le chemin retour jusqu'au début du chemin aller et on obtient bien « A ». Lorsque la lettre est passée par tous les composants, une pile électrique, alimentant Enigma, permet d'allumer la lampe correspond à la lettre cryptée/décryptée que l'on doit alors noter.

Le fonctionnement d'Enigma permet donc de faire une machine à crypter et décrypter très puissante, avec très peu de failles. Cependant, la seule faiblesse apparente qu'elle a (une lettre ne peut être cryptée par elle-même) va être le point de départ de la cryptanalyse d'Alan Turing.

1.3. Cryptanalyse par Alan Turing

Les nazis considéraient le système de chiffrement d'Enigma inviolable mais un homme a réussi l'exploit, déchiffrer Enigma, cet homme c'est Alan Mathison Turing dit Alan Turing qui était mathématicien, cryptologue et informaticien britannique, il est né le 23 juin 1912 et mort le 7 juin 1954.

Alan Turing crée une machine surnommée « Bomb » qui permet de tester des combinaisons possible d'Enigma mais pas toutes car cela serait impossible dans des délais humain.
Il réduit donc les combinaisons à tester grâce à la méthode qu'il a trouvé. Il se sert d'une faille du système Enigma, en effet une lettre à cryptée n'est jamais substituée à elle-même du fait du réflecteur donc un N ne pourra jamais donner un N.
De plus, l'autre faille venait du système allemand en lui-même, c'est à dire qu'ils avaient l'habitude d'envoyer un bulletin météo tous les jours à 6h et ce bulletin météo avait toujours la même forme. Il contenait notamment le mot « wetterbericht » qui voulait dire bulletin météo en allemand et se finissait toujours par « Hitler ».



Avec ces informations, Alan Turing comparait les mots clés allemands avec le message codé en vérifiant que chaque lettre du mot clé soit différente du message. Si ce n'était pas le cas il déplaçait le mot clé jusqu'à que toute les lettres soient différentes, cela signifiait alors que le mot clé pouvait peut-être se situer à cet emplacement dans le message codé.
A partir de cela, la machine Bomb testait les positions des rotors en fonction des couples du plugboard. C'est-à-dire qu'on positionnait le Rotor 1 à 1 par exemple et on faisait la supposition que dans le plugboard deux lettres étaient liées. La machine vérifiait que la supposition était exacte en lisant le message codé et le mot-clé. Si on tombait sur une contradiction, par exemple on suppose A-D branchés dans le plugboard et au fil de la lecture on a un autre couple A-Z alors qu'une mettre lettre ne peut pas être branchée deux fois dans le plugboard, alors la supposition de départ était fausse et la position supposée des rotors aussi et on essayait alors une autre position. Pour accélérer les choses, Turing a émis l'idée que si il y avait une contradiction alors tous les autres couples trouvés qui découlaient de la supposition initiale fausse étaient faux aussi et n'avaient alors pas besoin d'être revérifier.
Finalement, la machine Bomb était capable de trouver les couples du plugboard et les positions de rotors utilisées pour crypter le message en 20 minutes. Il ne restait alors plus qu'à décrypter le code avec les bons réglages.

Cette phase d'analyse nous a véritablement permis de cerner le sujet et de mieux comprendre le rôle de chaque composant de la machine Enigma. De plus, cela nous a aussi permis d'avoir des pistes notamment pour le décryptage en s'appuyant de la cryptanalyse faite par Alan Turing.

2. Enigma : Simulation informatique

2.1. Organisation du projet

Avant de nous lancer dans le développement du projet, nous avons dû dans un premier temps nous organiser sur différents points.

2.1.1. Choix et contraintes

Nous avons dû effectuer certains choix avant de commencer à développer et même avant de pouvoir conceptualiser le projet.
Nous nous sommes tout d'abord posé la question du choix du langage de programmation. Celui-ci étant complétement libre avec pour seule obligation, avoir une interface graphique, nous pouvions partir sur du développement logiciel (C, C++, Java) mais aussi sur du développement web (html, css, javascript). Parmi les langages que nous connaissons, celui que nous avons le plus utilisé et où nous nous sentons le plus à l'aise reste le langage Java.

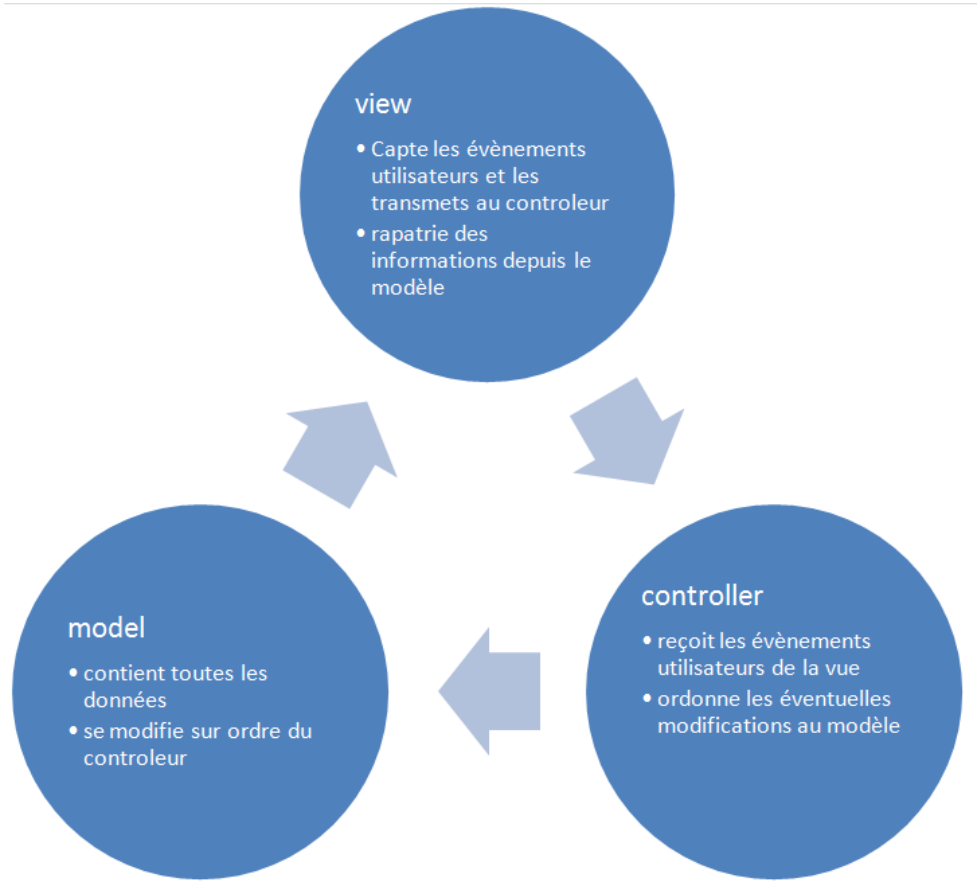
C'est donc le langage de programmation Java, langage orienté objet, que nous avons retenu étant donné que nous avons déjà eu un cours d'interface Homme Machine facilitant le développement de l'interface graphique et que nous sommes à l'aise avec la notion d'objet qui nous a paru cohérent avec les composants de la machine Enigma (nous détaillerons ce point dans une prochaine partie).

Nous avons eu, au semestre dernier, un cours sur les principaux design pattern que l'on peut utiliser afin d'avoir une bonne conception d'un projet permettant de décrire une solution standard à un problème de conception logiciel qui peut être par la suite réutilisé sans avoir à modifier le code existant (dans le cas de rajout de classes par exemples). Les design pattern permettent de respecter un certain nombre de bonnes pratiques permettant une conception optimisée et de ce fait un code clair et organisé.

Dans le cadre de notre projet, nous avons décidé d'utilisé le design pattern MVC (Modèle Vue Contrôleur) permettant de faire communiquer les différentes couches de notre projet de façon évènementielle c'est-à-dire qu'une action de l'utilisateur via l'interface graphique (la vue) déclenche automatiquement un évènement dans le contrôleur qui lui-même se chargera de notifier la couche modèle (c'est le cœur du projet, elle contient toutes les données principales et se charge des principaux calcul).

Afin de faciliter cette programmation évènementielle, nous avons implémenté les classes observer et observable définie par défaut en Java et pour créer l'interface graphique, nous avons utilisé la bibliothèque Swing de Java (dans notre première version de l'interface). Nous avons ainsi défini l'architecture globale de notre projet et commencé la phase de conception.

Voici un petit schéma explicatif du design pattern MVC:



Source : http://raphael-waeselynck.developpez.com/tutoriels/java/java-me/mvc/

2.1.2. Outils de travail

Ces choix nous ont permis de défini les outils de travail que nous allions utiliser pour partager notre travail et communiquer efficacement.

Comme appris en cours de méthodologie, nous avons utilisé Github (cf annexe 2), une plateforme de développement collaboratif basée sur git, un outil de versionning. Cet outil permet de mettre en ligne notre projet et les évolutions apportées de façon à ce que tous les membres de l'équipe aient la dernière version du projet disponible et puisse rendre accessible instantanément toutes les modifications effectuées. Cela facilite grandement la création de projet et nous permet d'être très réactifs quant aux erreurs qui pourraient y avoir sur le projet pour les corriger rapidement et ne pas continuer de développer sur une mauvaise base.

Concernant le développement, nous avons décidé d'utiliser Eclipse dû au choix du langage de programmation qui est un IDE (Integrated Developement Environment) permettant de développer des logiciels en langage Java et de faciliter les tests fonctionnels. C'est un des IDE les plus utilisé pour le développement Java (avec netbeans) et c'est aussi celui que nous connaissons le mieux car nous l'avons déjà utilisé lors d'un projet ultérieur et lors des séances de programmation à l'IUT.

En combinaison, nous avons utilisé Scene Builder (cf annexe 3) pour créer la deuxième version de l'interface graphique. Ce logiciel permet de créer simplement une application en utilisant le « drag and drop » pour positionner les éléments. Il génère automatiquement le code du fichier contenant la vue selon la position des éléments. Ce logiciel fonctionne avec Java FX (une API Java) permettant de remplacer la bibliothèque Swing utilisée pour créer une interface graphique.

Nous avons aussi été amenés à utiliser le logiciel Mumble, un logiciel VoIP, afin de communiquer oralement en équipe lorsque nous ne pouvions pas nous voir à l'IUT.
Cela nous a permis de communiquer efficacement, de réfléchir aux éventuelles difficultés et de proposer des solutions facilement sans avoir à attendre de se retrouver à l'IUT pour en parler.

Ces outils ont été une aide pour avoir une meilleure gestion de l'équipe mais celle-ci s'est aussi faite autour d'autres points que nous allons détailler.

2.1.3. Gestion de l'équipe

Dans un projet, il est important d'avoir une équipe qui sait communiquer, qui s'investi et il est aussi important de répartir les tâches à chaque membre afin d'optimiser le développement du projet.

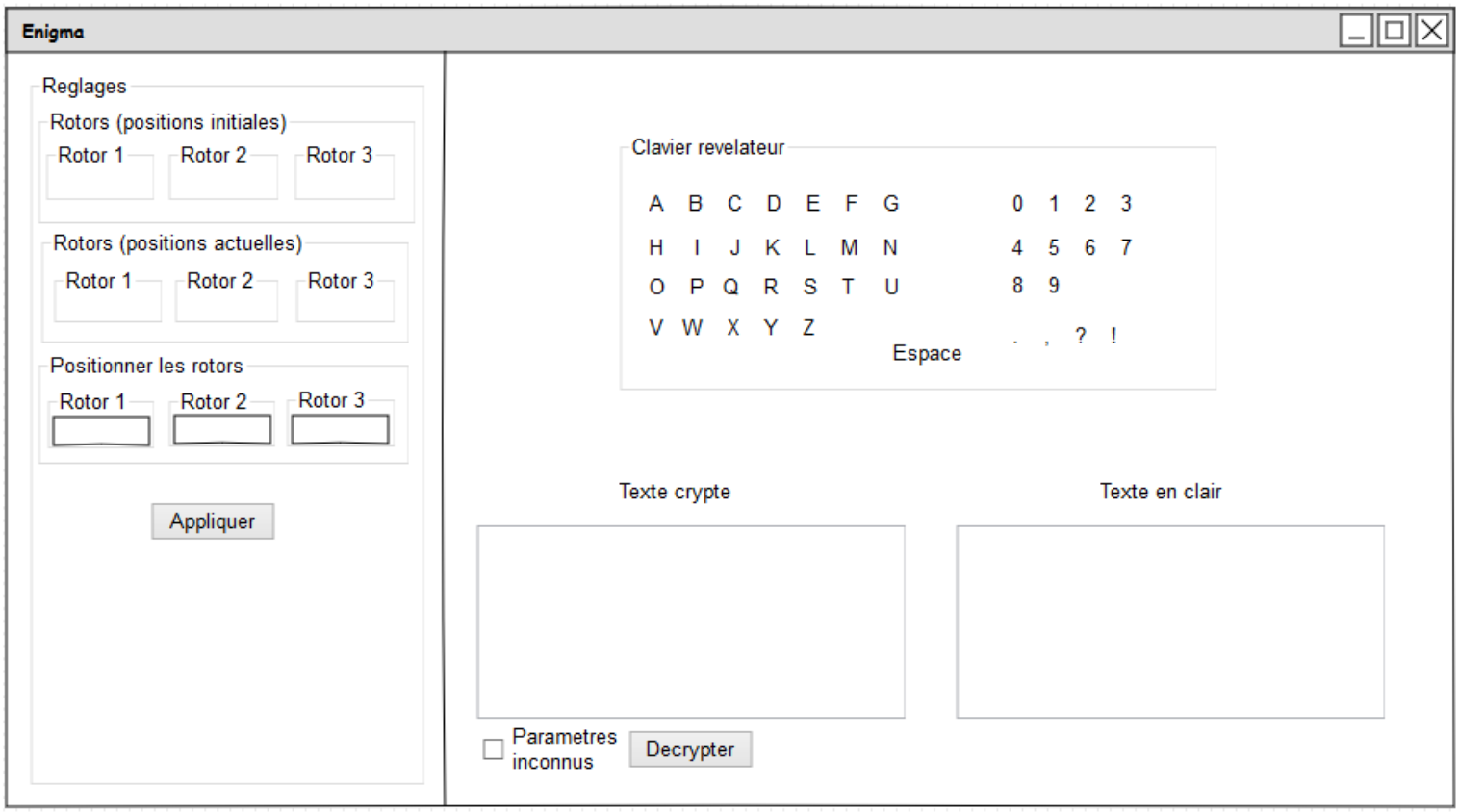
Dans un premier temps, il était important de se mettre au point sur les connaissances que nous avions acquises sur la machine Enigma lors de nos recherches afin d'avoir une vue générale et cohérente du projet par tous les membres de l'équipe. Pour ce faire, nous avons organisé des petites réunions où nous avons expliqué à chacun ce que nous avions compris ou ce que nous n'avions pas compris pour que l'on puisse lever les zones d'ombres restantes qui nuisent au développement du projet. Ces réunions ont été l'occasion de

schématiser les problèmes et d'apporter des solutions de conception. Les avis sur les différentes possibilités de conception nous ont permis d'enrichir le projet et de réfléchir sur des solutions plus optimisée que ce qui était proposé initialement. Cela nous a permis d'être d'un commun accord sur la façon dont nous allions procéder pour développer le projet. Ces petites réunions ont été la base d'un travail d'équipe solide et efficace.

Par la suite il a été nécessaire de diviser le projet en plusieurs tâches notamment dû au choix de conception que nous avions fait (architecture MVC) et qui nous a permis de développer indépendant et parallèlement chaque partie. De ce fait chaque membre de l'équipe s'est occupé d'une des couches de l'architecture 3-tiers c'est-à-dire : la partie métier, la partie interface et la partie contrôleur. Pour combler les dépendances entre ces couches nous avons mis en place un « Mock » qui permet de simuler une classe même si celle-ci n'est pas encore opérationnelle. Plus concrètement, il s'agit d'une classe simplifiée qui garde le squelette de la classe réelle mais qui renverra juste les valeurs attendues sans effectuer de calcul (contrairement à la classe réelle).

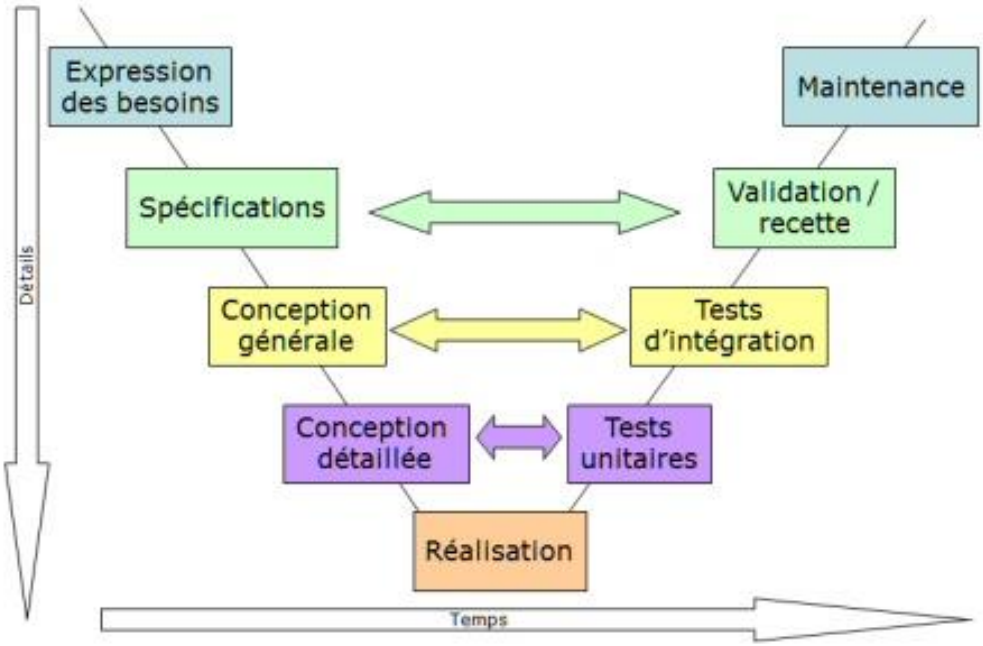
De plus, nous avons constitué une maquette de l'interface (version 1) pour que la personne en charge de la partie Vue puisse suivre un modèle où nous nous étions mis d'accord.

Celle-ci se présente comme ceci :



Afin de respecter les délais, et d'avoir un point de repère temporel sur le travail effectué, nous avons conçu un diagramme de Gantt prévisionnel nous permettant de mieux nous situer dans le temps et d'améliorer notre efficacité (voir annexe 5).

Nos méthodes de travail se sont apparentés aux méthodes agiles pour la gestion de projet et plus précisément Scrum du fait de nos réunions, de nos découpage de tâches mais aussi parce que lorsque nous avions terminés une micro-tâche (ex : une fonction complexe d'une classe), nous la testions et nous étions susceptible de modifier pour l'adapter. Plus précisément, nous n'avons pas suivi un cycle en V :



Source : http://www.methodesagiles.info/Agilite.php

Nous avons réalisé chaque fonctions puis effectués des tests fonctionnels et corriger/adapter au besoin.
Cette méthode à l'avantage d'être moins rigide que le cycle en V et de permettre une évaluation quotidienne des fonctionnalités.

Grâce à cela, la gestion de l'équipe a été grandement facilitée et le travail d'équipe a été plus efficace ainsi que la mise en place du projet.

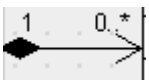
2.2. Représentation d'Enigma

Une part importante dans la simulation d'Enigma a été de modéliser la machine pour mettre en valeur les composants et les fonctionnalités primordiales de la machine.

2.2.1. Conception UML

L'UML (Unified Modeling Language) est un langage de modélisation qui subvient durant la phase d'analyse d'un projet (pour notre cas, juste après l'analyse de la machine Enigma). Il permet d'avoir une vue globale du projet avec une possible manière de fonctionner. La conception UML permet donc de faire le squelette du programme et oblige à se poser les bonnes questions au niveau des fonctionnalités et de la logique de fonctionnement avant de commencer à programmer. Il y a plusieurs types de diagrammes réalisables : diagrammes de cas d'utilisation, diagrammes de séquence, diagramme d'activité…Nous avons choisi d'effectuer uniquement le diagramme de classe qui permettait répondre à nos principales interrogations. Il consiste à représenter les principales classes qui seront présentes dans le programme et leurs liens avec les autres classes.

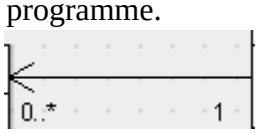
Nous avons gardé tous les composants de la machine car ils jouaient tous un rôle important dans le fonctionnement d'Enigma. Ainsi, dans notre diagramme UML que nous avons établi après l'étude d'Enigma, nous retrouvons les rotors, le plugboard et le réflecteur. Cette partie est la partie « Métier » et le cœur même du programme. C'est là où les calculs seront effectués, et c'est cette partie qui représente Enigma en soit.



Flèche de composition

Nous pouvons y voir la classe Machine qui peut être vue comme la boite d'Enigma. A l'intérieure de cette boîte nous avons les rotors, le plugboard et le reflecteur. Notre UML montre qu'Enigma est « composée » de ces éléments. La machine entière est capable de crypter, décrypter grâce à l'interaction de ses composants. C'est la classe Machine qui se chargera de faire le lien entre eux. Par exemple, elle ira chercher la correspondance d'une lettre dans le plugboard puis la transmettra aux rotors puis au reflecteur et indiquera le chemin inverse.
Les classes Rotor, Reflecteur, et Plugboard ne sont autre qu'une reproduction de ce que nous avons compris sur leur fonctionnement : les rotors lient deux lettres ensembles et tournes, le plugboard fait 10 couples de lettres et le reflecteur effectue une ultime permutation.

Nous avons ensuite la classe Vue qui concerne tout ce qui est visuel et saisie utilisateur: champ de texte, affichage des messages cryptés et décryptés. C'est une interface qui fait le lien entre l'utilisateur et le programme.



Flèche d'association

La classe Vue récupère automatiquement les modifications qui s'effectuent via la classe Machine et se met à jour. Pour se faire, la classe Vue est donc « associée » à la classe Machine.

La classe Vue et la classe Machine sont finalement associées à la classe Controleur qui permet d'harmoniser le tout. La classe Controleur est déclenchée à chaque action de l'utilisateur, elle vérifie les informations saisies avant de les envoyer à la classe Machine. Cela permet d'éviter tout plantage du programme à cause d'informations qui ne seraient pas attendues . Elle peut aussi mettre à jour la Vue (affichage d'un message d'erreur par exemple).

Même si le diagramme de classe a constitué la base de notre code, nous avons dû effectuer quelques modifications au cours du projet afin de s'adapter à de nouveaux besoins apparus en programmant (comme la classe Decrypte et les threads). Sans conception, ces modifications auraient été bien plus nombreuses et le risque d'un mauvais fonctionnement entre toutes les classes aurait été plus élevé.

2.2.2. Logique de fonctionnement

Nous avons établi une logique de fonctionnement pour que la simulation informatique d'Enigma imite au mieux le comportement de la vraie machine. Nous avons vu les rotors comme des circuits composés d'entrées/sorties. Nous avons donc représenter ces rotors sous formes de tableaux où l'indice de la case est l'entrée et le chiffre contenu dans la case est la sortie.

Par exemple :

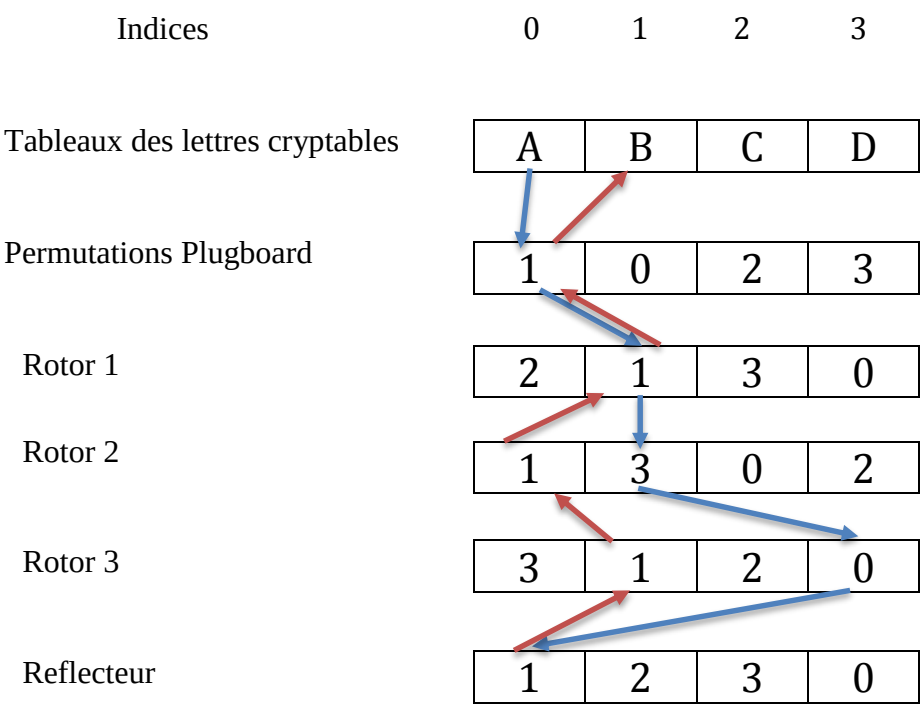
3	2	4	1
0	1	2	3

Cela signifie, si nous prenons 4 lettres A B C D que la lettre entrante en 0 (après permutation dans le plugboard) sera dirigée vers la sortie 3 donc elle sera modifiée en D (de 0 à 3) et elle entrera donc à l'indice 3 dans le rotor suivant qui lui-même déterminera quelle entrée la lettre prendra dans le rotor 3 puis dans le réflecteur avant de faire le chemin inverse. Une fois la lettre cryptée, les rotors se décalent ce qui signifie ici que c'est le contenu des cases qui va se décaler.

La création du réflécteur et du plugboard est basé sur le même principe, à l'exception que le contenu ne se décale jamais. Ces derniers se chargent juste de faire des permutations, c'est-à-dire dans notre programme, de diriger la lettre vers une autre sortie.

Nous avons dû définir de manière claire, quelles lettres, chiffres et symboles pouvaient être cryptés par la machine. Pour cela, dans la classe Machine, nous avons mis en place un tableau fixe contenant tout ce que nous voulions pouvoir crypter.

Voyons un exemple concret sur 4 lettres A B C D : On tape la lettre « A », la lettre cryptée est « B »



La logique de fonctionnement est alors uniquement basée sur les tableaux et les indices que nous pouvons considérer comme des pointeurs sur d'autres cases jusqu'à arriver à la lettre cryptée. A l'aller, il faut se poser la question «quel est le contenu de la case à l'indice x ? », le contenu deviendra l'indice du composant suivant. Au retour, il faut se demander « quel est l'indice de la case qui contient ce chiffre ? »
Et on aura ainsi l'entrée correspondante à la sortie.

C'est la classe Machine qui se charge de récupérer les indices et d'aller « demander » au composant suivant quel indice il faut suivre pour arriver jusqu'au bout.

2.3. Difficultés rencontrées et solutions retenues

Lors de la réalisation du projet, nous avons été confrontés à certains problèmes ou certaines problématiques qui ont attiré notre attention.

La première problématique qui est apparue est celle de la complexité algorithmique de notre programme, c'est-à-dire le temps de calcul de l'ordinateur pour effectuer une tâche. Nous avons été sensibilisés à cette question lors de nos cours d'algorithmique et la logique de fonctionnement de notre programme (avec des tableaux) impliquait inévitablement d'avoir des complexités à vérifier. En effet, le chemin d'aller du cryptage est simple et suit une complexité en O(1) : nous avons l'indice, nous nous rendons à ce même indice du tableau suivant puis nous récupérons l'indice suivant dans la case etc…

Cependant, le chemin du retour est plus complexe. En Java, il est impossible d'utiliser la méthode « contains() » sur les tableaux. Or, le chemin inverse consiste à inverser les entrées/sorties des rotors. Les indices deviennent donc les sorties et le contenu des case l'entrée pour le tableau suivant. Cette inversion est obligatoire sinon les couples entrées/sorties du chemin aller seront différents pour le chemin retour ce qui n'est pas représentatif du fonctionnement d'Enigma. Dans ce cas, nous sommes obligés de parcourir pour chaque lettre cryptée, tous les tableaux afin d'avoir la correspondance entre la sortie et l'entrée correspondante. Nous avons alors une complexité d'O(n) dans le pire des cas pour une seule lettre cryptée. Donc $O(n^2)$ pour une chaine de longueur n. Afin de résoudre ce problème, nous avons mis en place une méthode « createMirror() » qui se charge de créer un tableaux inversant les entrées sorties pour le chemin inverse. La fonction createMirror() coût O(n) à chaque appelle (dès que les rotors tournent) mais l'accès aux correspondances sorties/entrées coûte alors O(1). Nous avons donc réussi à réduire la complexité du programme.

La deuxième difficulté rencontrée était liée à l'interface graphique v1. Cette interface a été créée à l'aide de la bibliothèque Swing et tous les placements ont été décidés au pixel près. Cela a fini par poser un souci de Responsive Design, c'est-à-dire, la capacité de l'application à pouvoir s'adapter à tout type de support et donc à différents ordinateurs et différentes tailles d'écran. Il s'est avéré que l'application réagissait mal lorsque nous essayions d'agrandir la fenêtre ou de la réduire. Certains composants (boutons par exemple) n'étaient alors plus accessibles. Pour corriger ce problème, nous avons décidé de créer une deuxième version de l'interface avec Java Fx et Scene Builder permettant de placer les composants de manière plus intuitive et de gérer plus facilement le réarrangement des composants lors des changements de taille de l'application.

Enfin, le dernier problème a concerné la méthode « decrypter() » consistant à décrypter une chaine sans avoir connaissance des réglages de la machine. Nous nous sommes inspirés des travaux d'Alan Turing pour comprendre les failles d'Enigma mais les mots possibles (les plus courants) de la langue française étaient bien plus nombreux que les mots qu'Alan Turing essayait de placer pour pouvoir en déduire les réglages de la machine. Nous avions plusieurs pistes :

->Crypter chaque mot courant et essayer de les placer dans le message crypté avec tous les réglages possibles de la machine.

->Prendre le mot courant et le placer dans le message crypter en éliminant les positions possibles où il pourrait se trouver lorsqu'une lettre était cryptée par elle-même (ce qu'Enigma ne peut pas faire) et répéter l'opération pour tous les réglages possibles.

->Utiliser l'indice de coïncidence permettant de déterminer lorsque le texte est proche d'un texte français. Seulement, il faut que la taille du texte soit très conséquente pour que l'indice de coïncidence soit vraiment fiable.

Tout d'abord, contrairement à notre conception UML de départ, nous avons décidé de faire une classe Decrypte indépendant qui n'aurait pour rôle que le décryptage d'un message codé. Nous avons vu cette classe comme une représentation de la machine d'Alan Turing « Bombe ». Dans un premier temps, nous avons opté pour la première méthode. Nous avons mis en place un dictionnaire de 600 mots environs et nous cryptions chaque mot 46^3 fois afin d'en trouver un dans le texte pour ensuite déterminer la position des rotors et décrypter tout le texte. C'est un traitement long et fastidieux surtout si le seul mot présent est le 599^e (bien plus d'une heure pour tout tester). Une autre limite est que si nous n'avons pas le mot dans le dictionnaire, il nous est impossible de décrypter le texte. Nous avons alors pensé à mettre en place des threads pour optimiser le temps de calcul. Les threads sont des processus indépendant qui sont capables de fournir un travail en fond sans bloquer l'application. Nous voulions mettre en place deux threads : Le premier se serait chargé de traiter

les mots de 0 à 299 et le deuxième de 300 à 599. Cependant, là encore, le temps de calcul était long. Nous voulions aussi mettre en place des dictionnaires secondaires avec des mots moins courant pour pallier le premier dictionnaire au cas où le mot ne s'y trouverais pas. Nous n'avons pas continuer avec cette méthode car cela n'aurait que rallongé le temps de calcul.

Nous avons alors décidé de partir sur la troisième méthode, l'indice de coïncidence, et de l'implémenté. Bien qu'elle soit plus fiable sur un texte de minimum 100 caractères (chiffres et ponctuation non compris), ça n'en reste pas moins la meilleure méthode parmi celles que nous avons essayées. L'indice de coïncidence consiste à calculer la probabilité de répétition des lettres du message. Voici alors sa formule :

$$IC = \sum_{i=1}^{26} \frac{ni(ni-1)}{N(N-1)}$$

Où « ni » représente le nombre d'occurrences de chaque lettre dans le texte et N la longueur totale de la chaine (seules les lettres de A à Z sont comptées).

On sait que la moyenne de l'indice de coïncidence en français est de 0.074. Comme nous étions sûrs d'avoir un texte français crypté, nous avons alors décrypté le texte 46^3 fois et cherché l'indice de coïncidence se rapprochant le plus de 0.074. Nous avons ainsi au plus (si le texte a été crypté en position 45 45 45), 46^3 possibilités de cryptage avant de décrypter le texte. Bien plus optimisé que les versions précédentes, la position des rotors peut être alors très vite trouvée. Avec une telle méthode, il serait aussi possible d'avoir la position des rotors probables puis de chercher les couples dans le tableau de connexion (plugboard) afin de déchiffrer le texte (cette dernière partie n'est pas implémentée dans notre projet).

Cependant, comme signalé, l'indice de coïncidence à un inconvénient : il n'est pas très fiable avec des phrases très courtes ou ne contenant un seul mot car il peut être facilement faussé suivant les lettres qui apparaissent.

Bilan

Nous avons trouvé ce projet très intéressant car même si nous connaissions la machine Enigma, nous ne savions pas concrètement comment elle fonctionnait. Cela nous a alors permis d'acquérir des notions de cryptographie supplémentaires car lors de nos recherches, nous avons aussi comparé le fonctionnement d'Enigma avec d'autres méthodes de cryptage et décryptage et nous nous sommes intéressés à la cryptanalyse en générale.

Sur le plan informatique, nous avons essayé de mettre en pratique toutes les connaissances que nous avions pu acquérir jusque-là dans notre formation et notamment pour la gestion de projet. Se faisant, nous avons eu un aperçu réaliste des phases d'analyse et de conception que nous devons dorénavant appliquer dans chaque projet car le résultat en est positivement affecté. Nous avons aussi considérablement amélioré notre communication au sein de l'équipe et nos choix d'outils de travail performants et adaptés par rapport aux autres projets que nous avions pu mener lors de notre formation. Nous avons aussi compris l'importance des tests car travaillant en équipe, il est impératif que le code de chacun soit fonctionnel pour pouvoir le mettre en commun avec les autres membres et ne pas perdre de temps sur le débogage.

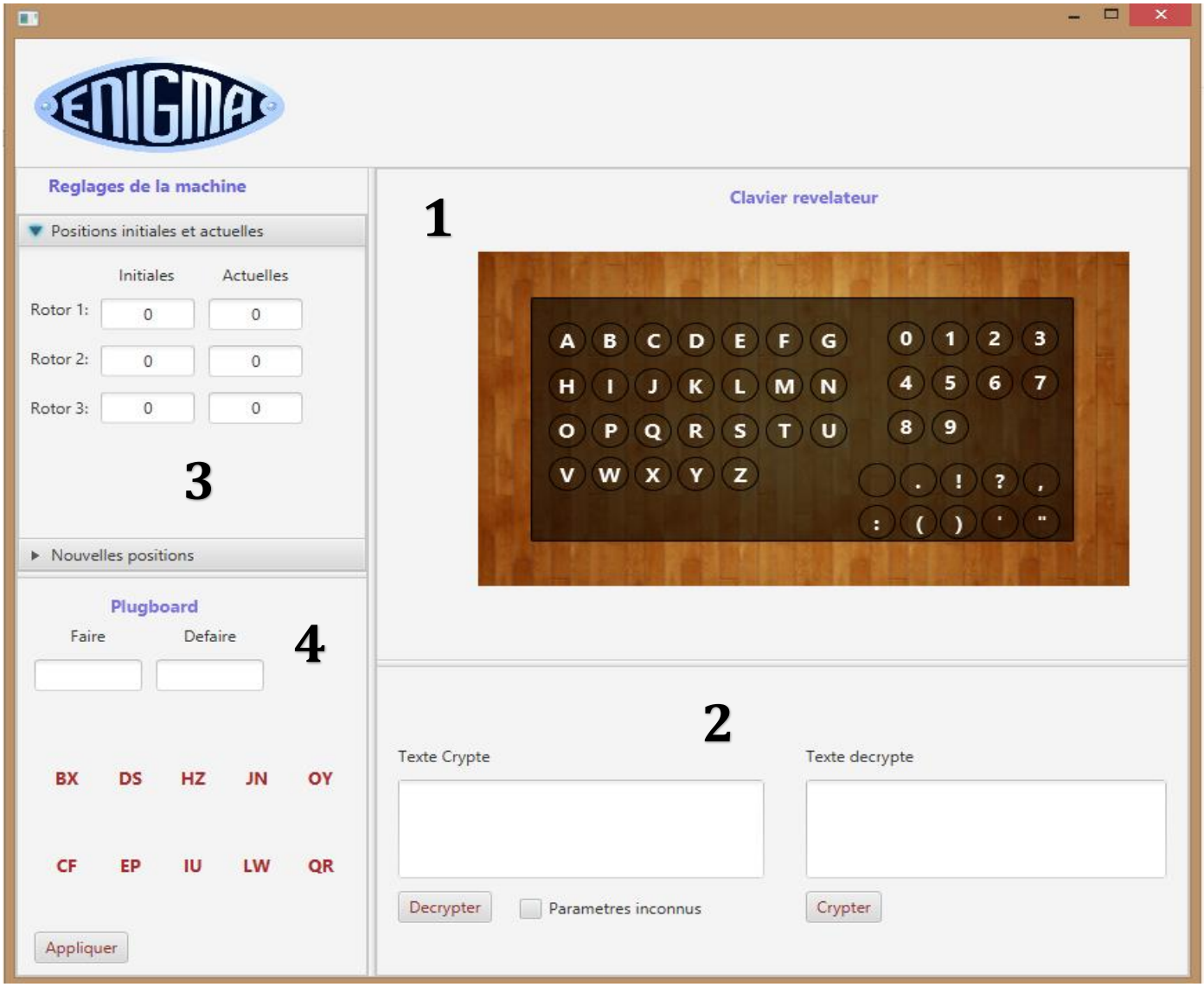
Outre le fait de mettre en pratique nos connaissances informatiques, ce projet nous a aussi donné l'occasion d'en acquérir de nouvelles et d'en consolider d'autres. En effet, certaines des notions que nous avons mis en pratique n'avaient été vues que théoriquement (architecture MVC par exemple), cela a donc été l'occasion de les appliquer et de mieux les comprendre.

De plus, nous avons été très sensibles à la notion de complexité algorithmique, une véritable problématique pour nous qui n'est pas toujours facile de résoudre. Cependant nous avons fait de notre mieux pour avoir un code optimisé, clair et cohérent ce qui représentait une vraie préoccupation.

Ce projet nous a donc beaucoup appris et nous conforte dans l'idée qu'une bonne compréhension du sujet est essentielle pour avoir une bonne conception et que notre formation nous a permis d'acquérir assez de connaissances pour mener à bien un projet de façon structurée. Il nous reste maintenant des pistes à explorer pour consolider le projet comme trouver les bonnes connexions du plugboard ou optimiser le décryptage de texte court. Nous allons sûrement continuer à améliorer notre projet.

MANUEL UTILISATEUR

Application Enigma

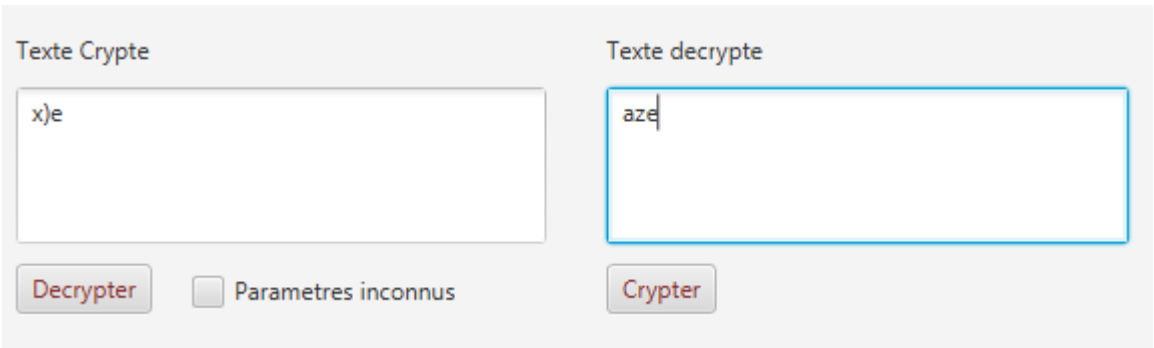


1. Clavier révélateur
2. Zone de saisie
3. Avancement des rotors
4. Réglages de la machine

Clavier révélateur

En arrivant sur l'application, l'espace le plus grand est réservé au clavier révélateur. Les lettres cryptées/décryptées sortantes sont mises en rouge lors de chaque tape de lettres comme le montre la photo ci-dessus. Il représente aussi toutes les lettres chiffrables, les autres symboles entrés ne seront donc pas pris en compte.

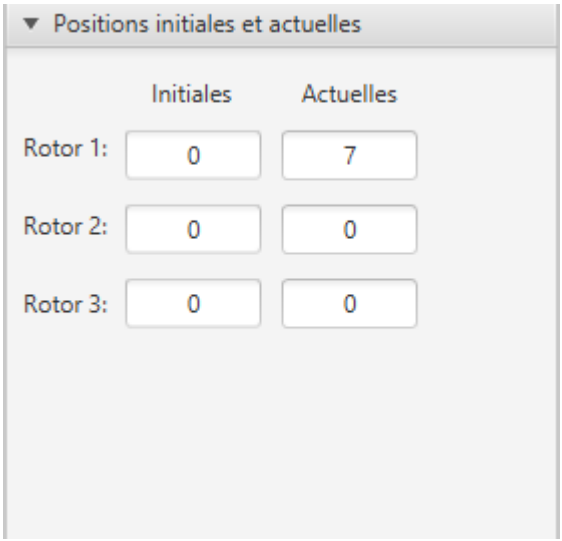
Zone de saisie



La zone de saisie se situe en dessous du clavier révélateur. Elle est composée de deux parties :

- A droite, la zone de saisie est destinée au texte que l'on veut crypter. Il y a la possibilité de crypter lettre par lettre en tapant un texte ou bien de copier/coller un texte et d'appuyer sur le bouton « crypter ».
- A gauche, le texte crypté est affiché. Si on veut décrypter un texte précédemment crypté, il est possible de décrypter lettre par lettre ou bien de copier/coller un texte et d'appuyer sur le bouton « décrypter ». A côté de ce bouton, il a une option « paramètres inconnus » qui peut être cochée si les réglages de la machine qui ont servis à crypter le message ne sont pas connus. C'est alors l'application qui se chargera de retrouver les réglages d'origine pour ensuite décrypter le texte.

Avancement des rotors



Il est possible de voir l'avancement des rotors par rapport à leur position initiale au fur et à mesure que les lettres sont tapées dans les champs de saisies. Cela peut servir à sauvegarder la position initiale des rotors et donc régler la machine correctement avant de décrypter un message.

Réglages de la machine

Enfin, la partie gauche de la machine concerne les réglages de celle-ci. Dans le haut de cette partie, il est possible de positionner les rotors sois-même en spécifiant la position pour chacun d'entre eux et en validant avec le bouton « modifier ». Les positions initiales des rotors vues précédemment seront alors remplacées par ces nouvelles valeurs.
Il est aussi possible de définir soi-même les couples du plugboard (tableau de connexion) altérant ainsi le cryptage. Ces couples sont au maximum de 10 et composés uniquement de lettres. Il faut dans un premier temps défaire un couple pour pouvoir refaire celui qu'on souhaite. Si la lettre est déjà couplée, le couple à faire sera ignoré.

ANNEXES

1) Livre des positions des rotors pour chaque jour

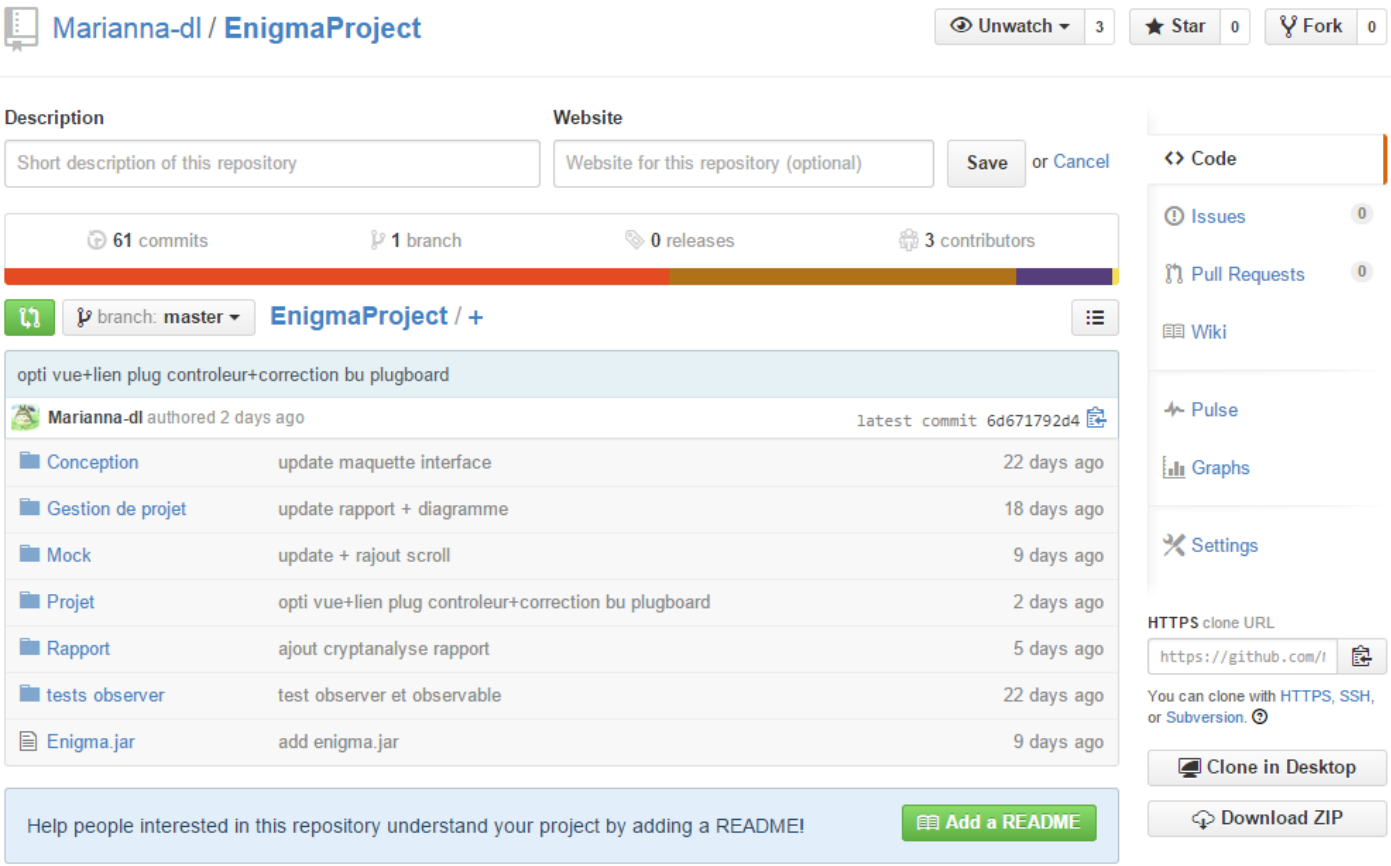
Kenngruppenheft Nr. 7
Teil A

Nr.	Kenngruppe	Spruchschlüssel	Nr.	Kenngruppe	Spruchschlüssel	Nr.	Kenngruppe	Spruchschlüssel	Nr.	Kenngruppe	Spruchschlüssel
1	D D J	A B G K	51	G J R	W O I U	101	P S J	P G B R	151	M I V	X A R G
2	L W J	D M U L	52	L T Y	Y D H G	102	R O V	B O V P	152	N Z O	B W O D
3	J E P	B O I J	53	P J Z	N S G L	103	S Y S	H R M T	153	P O E	H X V T
4	P X O	D N B S	54	S A U	C O H Y	104	U O K	L S Q M	154	Q Q H	R U Q T
5	S M F	E D H O	55	W B E	A U P E	105	W K H	K G R I	155	R Q K	W U D H
6	U P L	F C A S	56	W Y W	L R B G	106	X C B	U Z C D	156	V L H	Q N W O
7	Y O P	G D H J	57	A A C	J A H K	107	Z B D	W K P C	157	W V T	A R S J
8	A E G	G Y V F	58	D H G	R X I T	108	B E H	A T Z I	158	X Z Y	F P T J
9	F B J	H T U Z	59	F O V	Z B Q O	109	D W B	L X P Y	159	C V Z	E X N V
10	H H R	H Z N S	60	D H N	I Y F O	110	H K T	I G O E	160	A K Z	N G P O
11	K O Z	J F X K	61	G X P	B N O F	111	K W H	T V O K	161	E K Q	K O O H
12	N V K	W K U Z	62	J V Y	N C E L	112	M E X	U D F T	162	D F L	U A M N
13	R E X	E I B S	63	N F D	A H I R	113	O Q Y	I L V F	163	F U B	X I Y F
14	V J F	I V H X	64	Q W O	M O A K	114	Q J A	Y H A E	164	G V D	T H R S
15	X J H	Y D X N	65	T T O	R Z D K	115	S D X	T H O N	165	J Z K	I N L J
16	B R L	T G H P	66	T Z U	W O X U	116	T V Q	N V T X	166	K Q R	Q X Z R
17	E C J	V J H A	67	Y J K	Y M B W	117	W H F	A L W R	167	N C R	E D R Q
18	K D F	X H O N	68	Z Q S	J P B L	118	Y D E	H F O Q	168	O F V	C E T Q
19	L G T	P B V T	69	A V X	B M C R	119	B U X	A G K T	169	Q B T	P R D L
20	O W M	R J B P	70	E V B	X Q G S	120	F P N	R Z V T	170	Q Y Q	C K U T
21	Q F C	S A M L	71	G M Y	J C D K	121	D R W	X I H S	171	T R M	S X Z W
								U B Z Y	172	U L C	Q A V F

Livre tenu par les militaires allemands permettant de régler les machines Enigma au jour le jour.

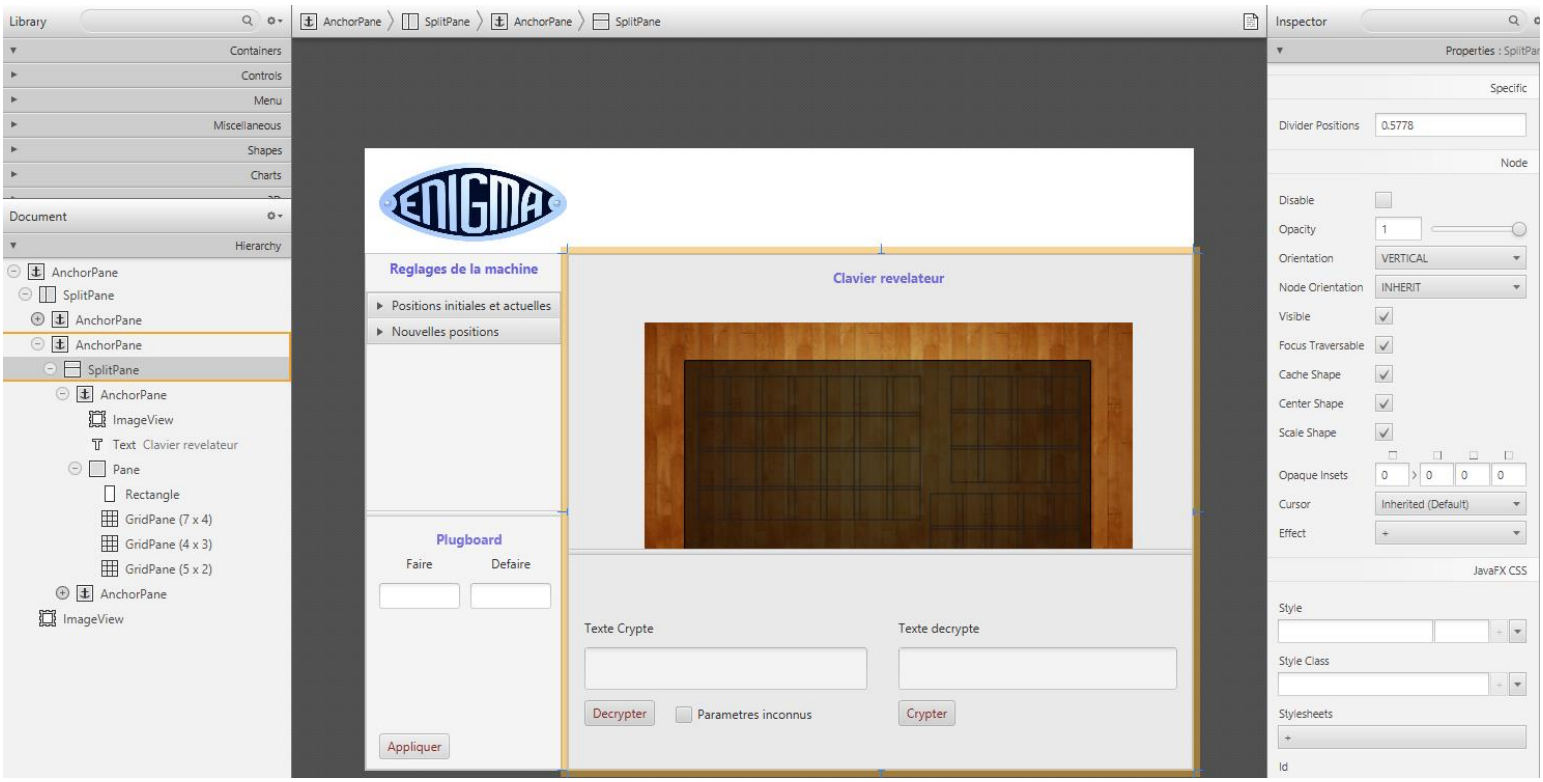
Source : http://soler7.com/IFAQ/Enigma.htm

2) Github : partage du projet entre les membres de l'équipe



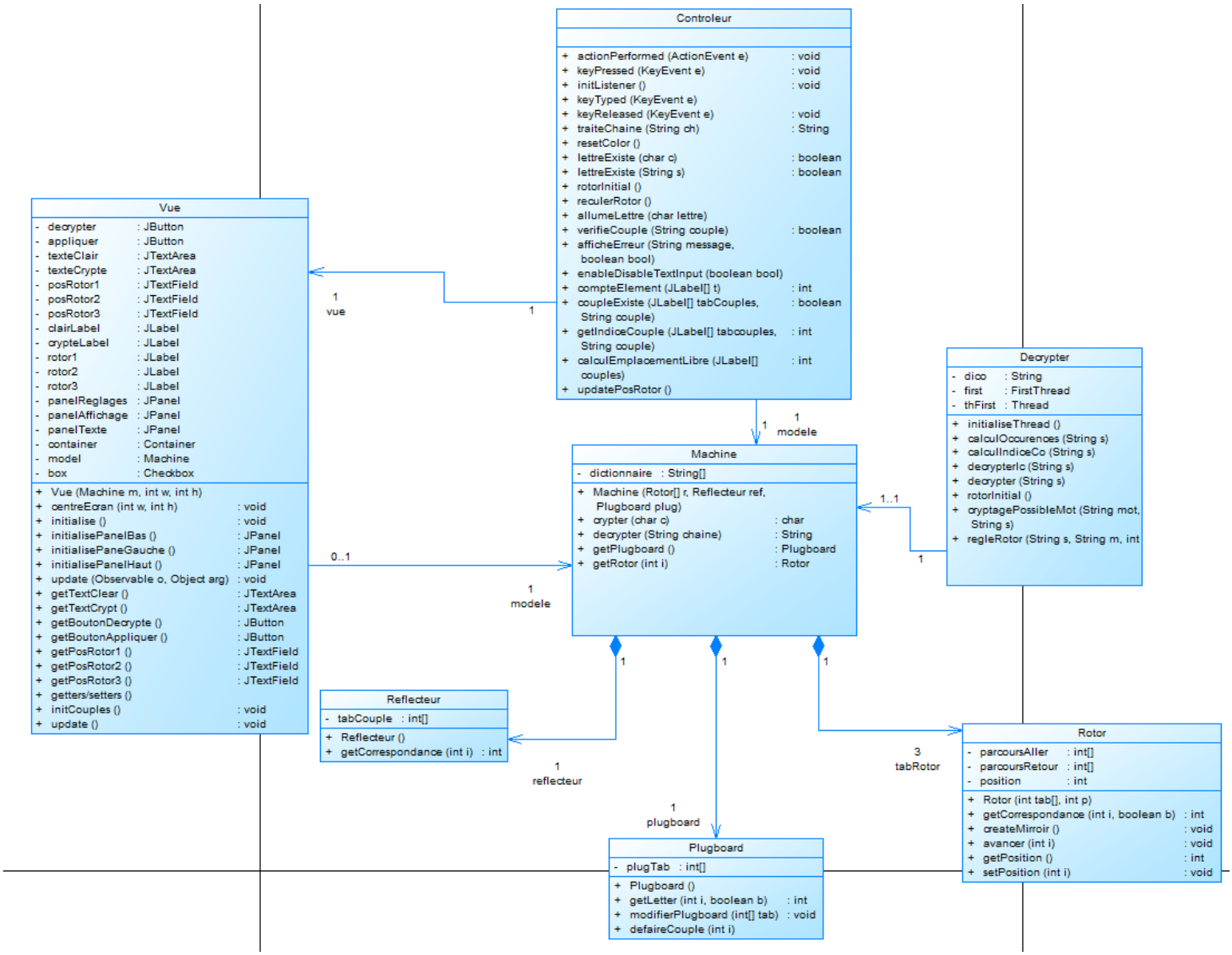
Github, la plateforme de développement collaboratif où nous avons pu partager simplement nos fichiers.

3) Scene Builder



Scene Builder est le logiciel que nous avons utilisé pour réaliser la seconde version de l'interface graphique de l'application.

4) Conception UML



[Conception UML (PDF)]

[Diagramme de Gantt (PDF)]

5) Classe Machine : Méthode « crypter »

```java
public Machine (Machine m){
    this.plugboard=m.getPlugboard();
    this.tabRotor=new Rotor[3];
    for(int i=0;i<3;i++){
        tabRotor[i]=new Rotor(m.getRotor(i));
    }
    this.reflecteur=m.getReflecteur();
}

/**
 * Crypte un caractere et avance les rotors d'un cran
 */
public char crypter (char c){
    int i= 0,nbLetter = 0;
    while(i<CONVERT.length && c!=CONVERT[i]){
        i++;
    }
    if(i<CONVERT.length){
        nbLetter=i;
    }
    else{
        return c;
    }
    int nbInter=plugboard.getLetter(nbLetter);
    for (Rotor r : tabRotor){
        nbInter=r.getCorrespondance(nbInter, true);
    }
    nbInter=reflecteur.getCorrespondance(nbInter);
    for(int j=2;j>=0;j--){
        nbInter=this.getRotor(j).getCorrespondance(nbInter, false);
    }
    tabRotor[0].avancer(1);
    if(tabRotor[0].getPosition()==0){
        tabRotor[1].avancer(1);
        if(tabRotor[1].getPosition()==0){
            tabRotor[2].avancer(1);
        }
    }
    return CONVERT[plugboard.getLetter(nbInter)];
}
```

6) Classe Rotor

```
/**
 * Donne l'entree ou la sortie correspondante
 * @param int i , boolean b
 *      true : parcours aller, on cherche la sortie qui correnspond a l'entree i
 *      false: parcours retour, on cherche l'entree qui correspond a la sortie i
 */
public int getCorrespondance(int i, boolean b){
    if(b){
        return parcoursAller[i];
    }
    else{
        return parcoursRetour[i];
    }
}
```

Permet d'obtenir la correspondance pour parcourir les rotors

```
/**
 * Avance les rotors et modifie les entrees/sorties des rotors
 */
public void avancer(int i){
    int k=0;
    while(k<i){
        int temp1,temp2;
        temp1=parcoursAller[1];
        parcoursAller[1]=parcoursAller[0];
        for(int j=1;j<parcoursAller.length-1;j++){
            temp2=parcoursAller[j+1];
            parcoursAller[j+1]=temp1;
            temp1=temp2;
        }
        parcoursAller[0]=temp1;
        k++;
        this.setPosition(this.getPosition()+1);
    }
    this.createMirror();
}
```

Avance un rotor d'un cran et met à jour les tableaux d'entrées/sorties

7) Classe Decrypt

Recherche des occurrences de chaque lettre

```
/**
 * Calcule les occurences de chaque lettres (a-z) dans une chaine
 *  @param s
 *          La chaine a analyser
 * @return int[] contenant toutes les occurences des differentes lettres
 */
public int[] calculOccurences(String s){
    ArrayList<Integer> n=new ArrayList<Integer>();
    ArrayList<Character> charVerifie=new ArrayList<Character>();
    int occurences;
    int j;
    int i=0;
    while(i<s.length()){

            occurences=0;
                j=s.indexOf(s.charAt(i));
                while(!charVerifie.contains(s.charAt(i)) && j!=-1){
                    occurences++;
                    j=s.indexOf(s.charAt(i),j+1);
                }
                if(occurences !=0){
                    n.add(occurences);
                }
                if(!charVerifie.contains(s.charAt(i))){
                    charVerifie.add(s.charAt(i));
                }
        i++;

    };


    int [] apparitions=new int[n.size()];
    for(i=0;i<n.size();i++){
        apparitions[i]=Integer.valueOf(n.get(i));
    }
    return apparitions;
}
```

Calcul de l'indice de coïncidence

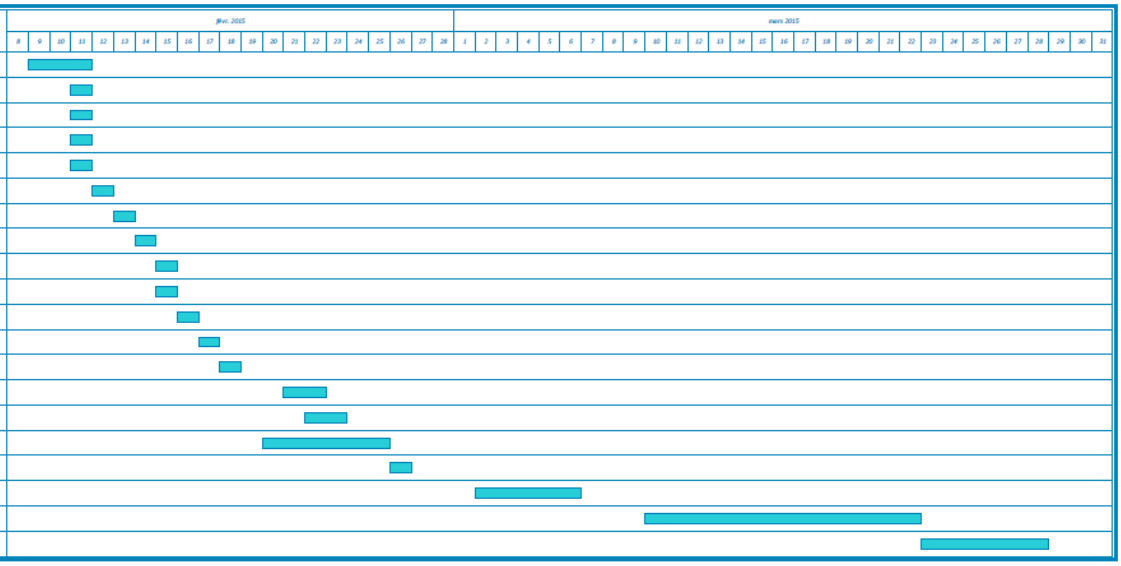
```
public float calculIndiceCo(String s){
    int [] nbApparitions=calculOccurences(s);

    float indiceCo=0.0f;
    float num;
    for(int i=0;i<nbApparitions.length;i++){
        num=(nbApparitions[i]*(nbApparitions[i]-1));
        indiceCo+=num/(s.length()*(s.length()-1));
    }
    return indiceCo;

}
```

8) Thread : Trouve l'indice de coïncidence le plus élevé

```
    //On parcourt 46^3 au plus
    while(tRotor<46 && !paused){
        sRotor=0;
        while(sRotor<46 && !paused){
            this.machine.getRotor(2).avancer(this.machine.CONVERT.length-(this.machine.getRotor(2).getPosition())+tRotor);
            this.machine.getRotor(0).avancer(this.machine.CONVERT.length-(this.machine.getRotor(0).getPosition()));
            pRotor=0;
            while(pRotor<46 && !paused){
                this.machine.getRotor(2).avancer(this.machine.CONVERT.length-(this.machine.getRotor(2).getPosition())+tRotor);
                this.machine.getRotor(1).avancer(this.machine.CONVERT.length-(this.machine.getRotor(1).getPosition())+sRotor);
                    chDecryptee=this.decryptage.getMachine().crypter(aDecrypter);
                    chDecryptee=chDecryptee.replaceAll("[^a-z]", "");
                    indice=this.decryptage.calculIndiceCo(chDecryptee);
                    if(indice<0.073 && indice>=indiceMax){//On garde l'indice de coincidence le plus eleve, ca veut dire que le texte est proche du francais
                        indiceMax=indice;
                        this.posRotors[0]=pRotor; //On sauvegarde la position des rotors pour pouvoir decrypter apres
                        this.posRotors[1]=sRotor;
                        this.posRotors[2]=tRotor;
                        System.out.println("indiceMax "+indice);

                    }
                    if(indiceMax>=0.072 && chDecryptee.length()>=100){//On peut essayer de finir la boucle plus vite. On peut supposer que si l'indice est superieur a 0.065 alors
                                                        //lorsque la longueur du texte est grande, qu'il est tres probable que ce soit bien notre texte decrypte
                         pause(); //si c'est bon, on stop le thread

                    }

                    pRotor++;
                    this.machine.getRotor(0).avancer(this.machine.CONVERT.length-(this.decryptage.getMachine().getRotor(0).getPosition())+pRotor);

            }
            this.machine.getRotor(1).avancer(this.machine.CONVERT.length-(this.decryptage.getMachine().getRotor(1).getPosition())+1);
            sRotor++;
        }
        this.machine.getRotor(2).avancer(this.machine.CONVERT.length-(this.decryptage.getMachine().getRotor(2).getPosition())+1);
        tRotor++;
    }
```

9) Diagramme de Gantt prévisionnel

ID	Nom de tâche	Début	Fin	Durée
1	Recherche et documentation sur Enigme	09/02/2015	11/02/2015	3j
2	Choix architecture MVC	11/02/2015	11/02/2015	1j
3	Modélisation UML	11/02/2015	11/02/2015	1j
4	Maquette interface v 1.0	11/02/2015	11/02/2015	1j
5	Création classe Plugboard Début classe Rotor	11/02/2015	11/02/2015	1j
6	Création classe Reflecteur	12/02/2015	12/02/2015	1j
7	Création Mock (vue, machine)	13/02/2015	13/02/2015	1j
8	Création classe controleur	14/02/2015	14/02/2015	1j
9	Création classe Vue	15/02/2015	15/02/2015	1j
10	Tests Vue/Controleur	15/02/2015	15/02/2015	1j
11	Classe Rotor	16/02/2015	16/02/2015	1j
12	Tests Machine/Rotor/Vue/Controleur	17/02/2015	17/02/2015	1j
13	Maquette interface v1.1	18/02/2015	18/02/2015	1j
14	Optimisation Classe Vue	21/02/2015	22/02/2015	2j
15	Optimisation classe Controleur	22/02/2015	23/02/2015	2j
16	Classe Machine sans décryptage	20/02/2015	25/02/2015	6j
17	Tests sans fonction décrypter	26/02/2015	26/02/2015	1j
18	Recherche pour la méthode décrypter	02/03/2015	06/03/2015	5j
19	Développement méthode décrypter	10/03/2015	22/03/2015	13j
20	Tests et validation	23/03/2015	28/03/2015	6j



Sources

https://www.youtube.com/watch?v=G2_Q9FoD-oQ par « numberphile »

https://www.youtube.com/watch?v=7dpFeXV_hqs par « e-penser »

https://interstices.info/encart.jsp?id=c_9752&encart=0&size=790,700

http://www.bibmath.net/crypto/index.php?action=affiche&quoi=debvingt/enigmafonc

http://www.dcode.fr/indice-coincidence