

Projet Tutoré 2016/2017 - Groupe M

**Dossier de Réalisation**

**Réalisé par :**

Guillaume Robert

Yoann Gathignol

Titouan Bouete-Giraud

**Destinataire :**

Isabelle Clavel

**Superviseur :**

Laurence Redon

I Hear, I Say

Table des matières

[Introduction ( recontextualisation ) 3](#_Toc477469889)

[A) Traitements implémentés 4](#_Toc477469890)

[B) Diagrammes séquences système 5](#_Toc477469891)

[Diagramme de séquence système pour le UC « Créer une grille » : 5](#_Toc477469892)

[Diagramme de séquence système pour le UC « Importer une grille » : 6](#_Toc477469893)

[C) Diagramme séquence 7](#_Toc477469894)

[D) Diagramme de Classe 8](#_Toc477469895)

[E) Point technique : Génération de la grille 10](#_Toc477469896)

[Fonctionnement de la grille 10](#_Toc477469897)

[La génération par choix des dimensions de la grille 11](#_Toc477469898)

[La génération par saisie d’une chaine de texte 11](#_Toc477469899)

[F) Point technique : L’écriture au sein du fichier CSV 13](#_Toc477469900)

[Fonctionnement de l’écriture 13](#_Toc477469901)

[L’implémentation du patron Strategy 13](#_Toc477469902)

[Bilan 15](#_Toc477469903)

[Organisation du travail 15](#_Toc477469904)

[Organisation de la communication 16](#_Toc477469905)

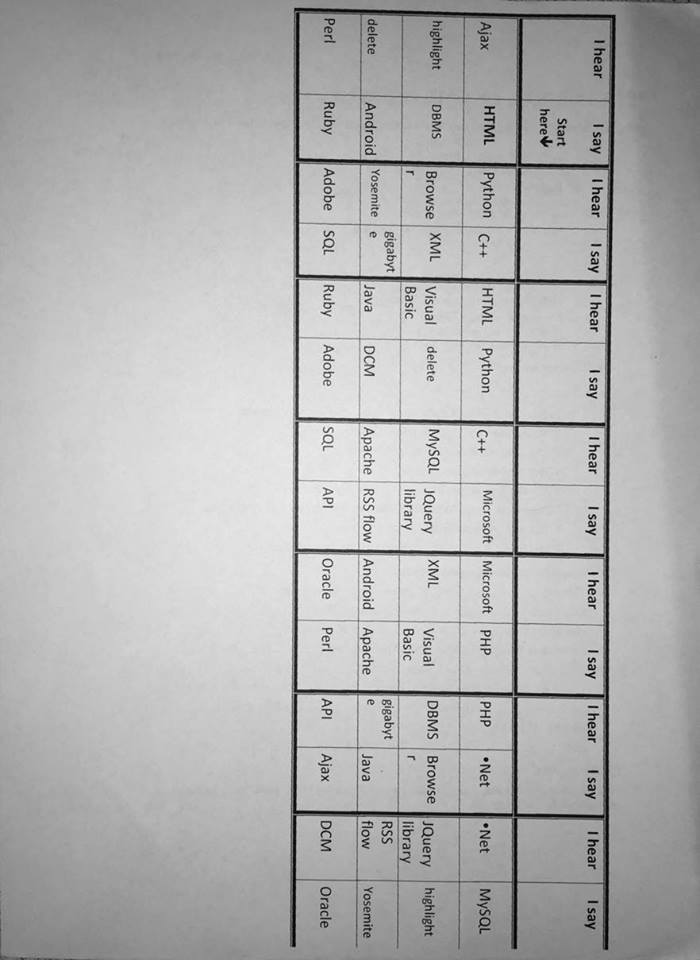
# Introduction ( recontextualisation )

Ce projet tutoré est une demande d’Isabelle Clavel, professeure d’anglais au département Informatique de l’Institut Universitaire Technologique de Blagnac. Laurence Redon, également enseignante à l’Institut, supervise ce projet.

Dans le cadre de son enseignement, Mme Clavel utilise des grilles d’exercices de prononciation qu’elle crée elle-même à l’aide de logiciel de tableur tel que Microsoft Office Excel. Bien que très complet, ce type de logiciel n’a pas pour but premier ce type d’exercice, et peut donc présenter diverses contraintes pour l’utilisateur :

* De temps : l’absence de format standard force l’utilisateur à dédier une partie non négligeable de son temps à gérer la forme de la grille.
* De suivis : La modification de grille peut s’avérer très fastidieuse à cause des contraintes sur les données saisies, à tel point qu’il peut s’avérer préférable de commencer une nouvelle grille.

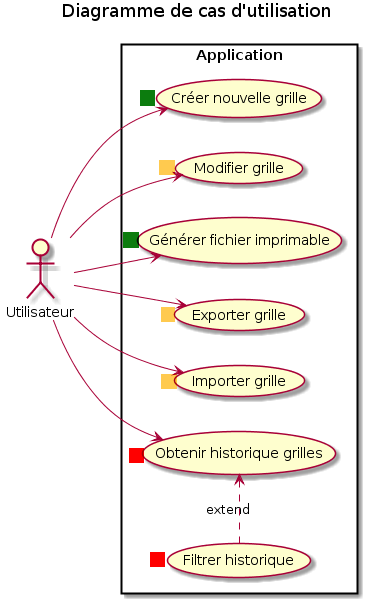
Le client a donc demandé une solution alternative qui lui permettrait un gain de temps non seulement à la réalisation, mais aussi lors de la modification éventuelle de ces grilles. Pour de plus amples informations, nous invitons le lecteur à consulter le Cahier Des Charges Utilisateurs (CDCU) fournis avec ce dossier.



*Figure 1 : Exemple de résultat*

Sur cette figure on peut voir ce qui est attendu au final. Ce que nous devons permettre au client de générer grâce à notre application. En effet l’activité consiste à prononcer des mots de la colonne « I Say », celui qui a ce mot dans la colonne « I Hear » devra dire le mot correspondant et ainsi de suite. Cela forme une chaine jusqu’à ce que tous les mots aient été prononcés.

# A) Traitements implémentés



**Méthode MoSCoW**

Must have

Should have

Could have

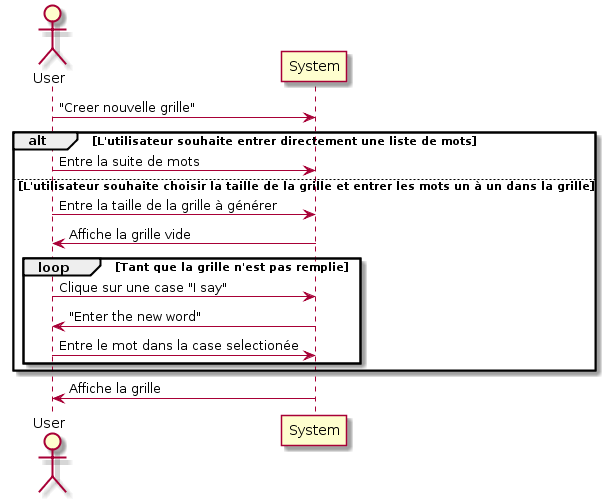
Won’t have

Ce diagramme des Cas d’Utilisation a été créé après l’analyse des besoin du clients qui a ensuite validé ce diagramme. Pour l’organisation de la phase de développement, nous avons utilisé la méthode « MoSCoW » qui sert à hiérarchiser les besoin pour avoir le plus rapidement possible un prototype fonctionnel et pour un déroulement général efficace du projet.

A la fin du projet, tous les cas d’utilisations représentés en vert et jaune ont été réalisés et implémentés. Les cas d’utilisations en rouge n’ont pas pu être réalisés par cause de manque de temps. On peut cependant estimer que le projet est fidèle à la demande du client car les cas d’utilisation non réalisés ne faisaient pas parti des besoins primaires du projet mais constituaient les évolutions et amélioration possibles de celui-ci.

# B) Diagrammes séquences système

## Diagramme de séquence système pour le UC « Créer une grille » :



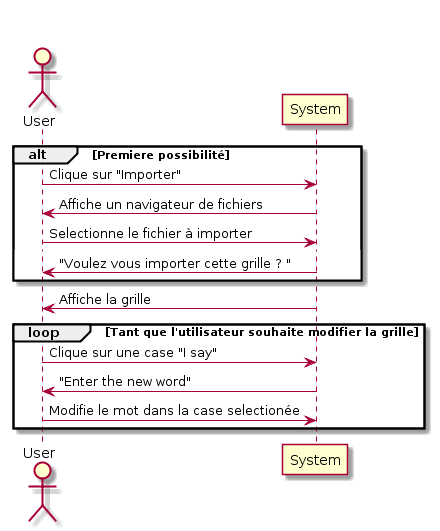
Ceci est le diagramme de séquence système dans le cas où l’utilisateur souhaiterait créer une tout nouvelle grille.

L’utilisateur a deux possibilités (exigés par le client) :

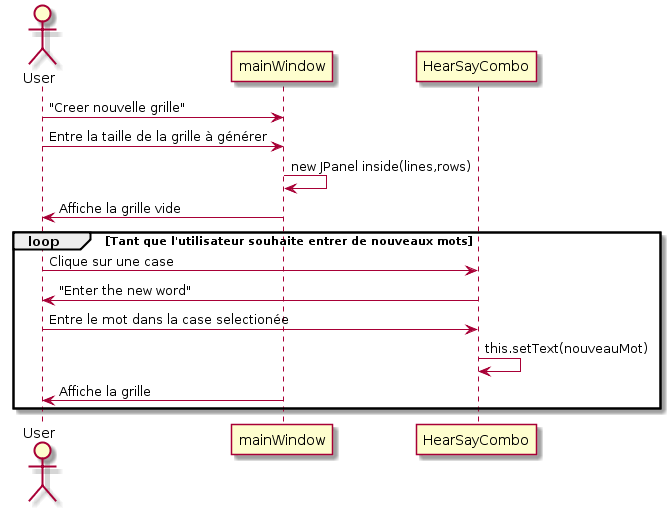
- la première étant de simplement rentrer en une seule fois toute la liste des mots qu’ils souhaitent retrouver dans la grille, et cette même grille se génèrera et se mélangera automatiquement. L’ordre de la chaine est aléatoire.

- la deuxième étant de choisir initialement une taille (nombre de lignes et nombre de colonnes) et de rentrer les mots un à un dans chaque case « I say ». La chaine se créera toute seule, et encore une fois, avec un ordre aléatoire.

## Diagramme de séquence système pour le UC « Importer une grille » :



# C) Diagramme séquence

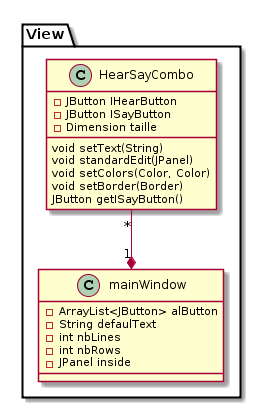


Ceci est le diagramme de séquence pour le cas d’utilisation « Créer nouvelle grille » .

Le fonctionnement est le suivant : lors de la création d’une nouvelle grille, l’utilisateur devra entrer la taille de la grille à générer, la classe de la fenêtre principale (mainWindow) créera un nouveau JPanel qu’il remplira de plusieurs HearSayCombo (qui sont constitué notamment de deux JButton) qui constitueront les éléments du tableau (new JPanel inside(...)).

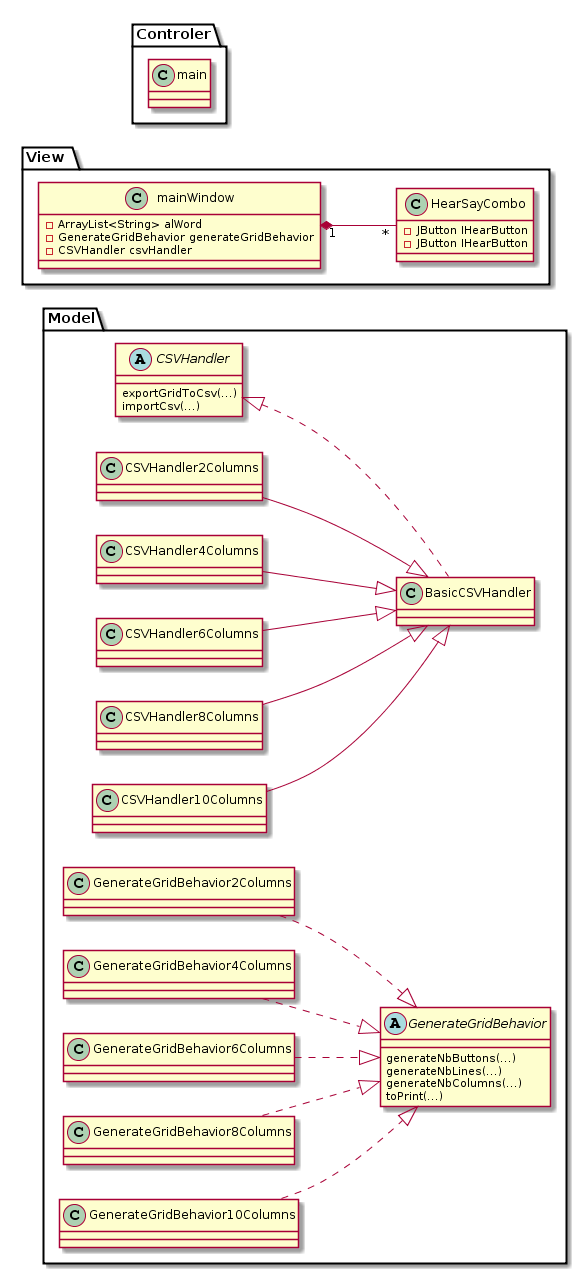
L'utilisateur devra cliquer sur les cases « I say » (de type JButton) qui ouvriront une fenêtre demandant d’entrer le nouveau mot, et la case se modifiera (this.setText(…)).

Diagramme de classes participantes :



Ici nous avons décrites les deux classes principales de l’application avec les méthodes appelées dans le diagramme de séquence ci-dessus.

# D) Diagramme de Classe



Ce diagramme de classe représente la structure du code de l’application du projet. Nous avons choisi d’appliquer une architecture MVC à nos classes afin de simplifier la maintenance de celles-ci.

Le premier package « Controler » ne contient que la classe « main » qui est celle qui est appelée lors du démarrage de l’application.

Le deuxième package « view » contient les classes générant les éléments graphiques de l’application (l’interface). La classe « mainWindow » représente la fenêtre principale, avec laquelle interagira l’utilisateur. La classe « HearSayCombo » représente les duos de mots avec lesquels interagit l’utilisateur lorsque qu’il souhaite entrer un mot dans la grille.

Le troisième et dernier package est le package « Model » et contient toute les classes représentant les structures de données manipulées par l’application. L’utilisateur n’interagit jamais directement avec celles-ci. Elles constituent le cœur de l’application, et leurs utilisations seront décrites plus en détails dans la partie ci-dessous.

# E) Point technique : Génération de la grille

## Fonctionnement de la grille

La grille est représentée par l’attribut JPanel grid dans la classe mainWindow. Ce panel, organisé en GridLayout, va symboliser la grille de cases. Afin de permettre à l’utilisateur de modifier ces boutons de manière intuitive, il a été choisis d’employer des boutons de type JButton générés par la classe HearSayCombo afin de symboliser le système de paires. Le nombre de JButton contenus dans la grille suit certaines règles. Tout d’abord, il se doit d’être pair afin de ne pas casser de paires. De plus, on choisira, pour chaque grille, d’ajouter 2 faux boutons Start et End pour un rendu final plus lisible.

Lorsque l’on souhaite utiliser une nouvelle grille, l’application va suivre différentes étapes :

* L’assignation d’un GenerateGridBehavior : ce concept suit la notion de patron Strategy, voir la partie « La génération par saisie d’une chaine de texte »
* Le redimensionnement du JPanel grid et la mise à jour des valeurs correspondantes à ces dimensions dans la méthode generateGrid().
* La génération puis l’assignation au JPanel des différents boutons qui vont composer cette grilles avec la méthode fillGrid(ArrayList<String>). Cette ArrayList<String> contient les textes (non doublés) des boutons. Elle ne contient pas Start et End.
* Le mélange de la grille se fait par la méthode shuffleWords(), qui va aléatoirement mélanger les textes dans l’attribut alWord (ArrayList<String>) utilisé pour la méthode fillGrid(ArrayList<String>).

Quelques détails sur les choix techniques pour ce type de génération : L’application présentant une structure assez rigide pour ce qui est de sa taille, la grille (attribut grid) possède donc une taille fixe, indépendante du nombre de boutons en son sein. Nous avons donc choisis de limiter les choix de l’utilisateur afin de conserver un rendu final ergonomique.

## La génération par choix des dimensions de la grille

Lorsque l’utilisateur choisis de créer la grille, il dispose de deux options :

Il peut générer une grille vide de la taille désirée : il lui sera alors demandé de sélectionner le nombre de lignes, puis de colonnes souhaité. L’application va alors directement récupérer ces valeurs dans la classe mainWindow, générée une grille avec ces nouvelles dimensions avec la méthode generateGrid() (grâce à l’attribut grid de type JPanel organisé en GridLayout), puis remplir celui-ci de boutons vides (de type JButton) avec la méthode fillGrid(ArrayList<String>) générés grâce à la classe HearSayCombo.

Dans le cas de la sélection du nombre de lignes, ce choix se traduit par une sélection allant de 1 à 10. Cette limitation est purement esthétique donc, et peut être modifiée sans nuire au fonctionnement de l’application.

La sélection du nombre de colonnes est également limitée par le rendu, mais aussi par le principe de la grille en elle-même : l’exercice marchant sur un système de paires de texte il faut donc que la grille comporte un nombre final de bouton pairs. Afin de pallier à ce problème, il a été choisi de limiter le nombre de colonnes possibles au sein de la grille à un nombre pair. Dans la version finale de cette application, le nombre de colonnes pouvant être sélectionné par l’utilisateur est donc 4, 6, 8 ou 10.

## La génération par saisie d’une chaine de texte

La deuxième option afin de générer la grille consiste à permettre à l’utilisateur de saisir une suite de mots, séparés par « ; » représentant le contenu des futures cases sans se soucier du nombre de cases ou de la taille de la grille.

Un problème survenait tout de même : une fois le nombre de texte compté par l’application, il fallait générer une grille appropriée qui puissent contenir un nombre suffisant de cases afin d’afficher ces textes. Cependant, le nombre de mots comptés pouvait donner suite à des grilles de différentes dimensions. Par exemple, si 7 mots ont été saisis, la grille aura un besoin de 2x7+2 soit 16 boutons (7 boutons doublés pour les paires IHear/ISay et les boutons Start/End). Cela laisse donc plusieurs possibilités concernant les dimensions de la grille :

* 2 lignes et 8 colonnes
* 3 lignes et 6 colonnes
* 4 lignes et 4 colonnes
* 6 lignes et 3 colonnes
* 8 lignes et 2 colonnes

Plutôt que de forcer un choix qui pourrait ne pas convenir aux besoins de l’utilisateur, il a été choisi de laisser celui-ci choisir uniquement le nombre de colonnes au sein de la grille (le nombre de lignes étant calculé par l’application en fonction de ce choix). C’est ici qu’intervient le patron Strategy concernant la génération de la grille

Afin de favoriser la maintenabilité de ce code, deux des fonctionnalités de l’application ont été implémentés en respectant le patron de conception (ou Design Pattern) *Strategy.* Ce patron vise à mettre en place plusieurs variantes d’une fonctionnalité, et de pouvoir modifier la version utilisée préalablement à l’exécution de cette fonctionnalité afin d’en modifier son comportement efficacement. De manière générale, ce patron est caractérisé par le diagramme UML suivant :

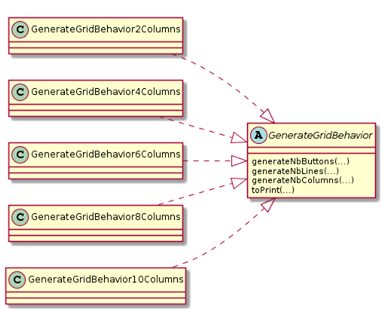


*Figure 2 : Digramme UML du patron Strategy*

Dans le cas de notre application, les deux fonctionnalités qui suivent ce patron sont :

* La taille de la grille générée lorsque que l’utilisateur choisit de saisir le texte
* L’export des données (l’écriture) dans le fichier de type CSV détaillé dans la partie «  L’écriture au sein du fichier CSV »

Dans ce cas-ci, la mise en place du patron Strategy se caractérise par la partie suivante du diagramme de classes de l’application :

**

*Figure 3 : Diagramme UML de GenerateGridBehavior*

Dans ce diagramme, on peut voir l’interface GenerateGridBehavior qui sera contenue dans la classe mainWindow. On peut également y voir les différentes versions du GenerateGridBehavior qui implémentent directement l’interface. Ce sont ces variantes qui vont mettre à jour les attributs représentant les dimensions de la grille dans mainWindow grâces aux méthodes generateNbButtons(int), generateNbLines(int) et generateNbColumns().

Chaque variante va donc pouvoir calculer un nombre de lignes différent en fonction du nombre de colonnes. Cependant, suite à l’implémentation de cette fonctionnalité, l’utilisateur sera à même de choisir le nombre de colonnes qu’il désire. Le generateGridBehavior s’occupera ensuite d’ajouter, et uniquement ajouter, des paires de boutons vides afin de conserver une forme rectangulaire pour la grille.

# F) Point technique : L’écriture au sein du fichier CSV

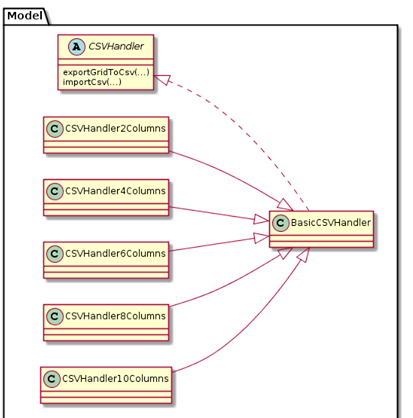
## Fonctionnement de l’écriture

L’écriture dans le CSV se fera au sein d’une classe de type BasicCSVHandler grâce à la méthode exportGridToCsv (File, ArrayList<String>). Cette méthode va tout d’abord doubler chaque mot de l’ArrayList<String> passée en paramètre (Cette opération est liée au fait que cette méthode a été pensée pour être utilisée avec mainWidow.alWord, qui ne représente pas le système de paire). Un header va ensuite être initialisé par l’écriture alternative de « I hear » et « I say » afin de correspondre au nombre de colonnes désiré. La nouvelle ArrayList<String> sera ensuite recopiée (sans oublier l’ajout de « Start » et « End ») dans le .CSVen respectant bien l’attribut nbColumns qui définit le nombre de mots par ligne. Cela permet au Csv écrit de disposer du même format que la grille générée dans l’application.

## L’implémentation du patron Strategy

Pour une introduction sur le patron Strategy voir la partie « la génération par saisie d’une chaine de texte. »

Pour cette fonctionnalité, le patron Strategy se représentera par le diagramme suivant :

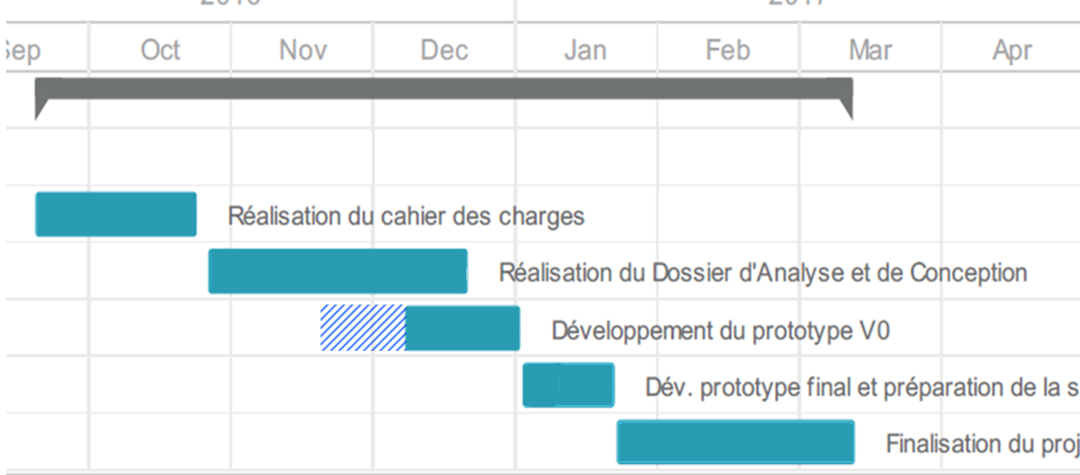


*Figure 4 : Diagramme UML de CSVHandler*

Comme expliqué en partie « Fonctionnement de l’écriture », la classe BasicCSVHandler va implémenter l’interface CSVHandler. Dans cette classe BasicCSVHandler, on va notamment trouver l’attribut nbColumns qui définit le nombre de mots par ligne dans le fichier CSV dans lequel on souhaite écrire. C’est ici qu’on peut ainsi observer la puissance du patron Strategy : les classes héritières de BasicCSVHandler on uniquement besoin de modifier cette attribut grâce à la méthode updateNbCol() pour obtenir un CSV entièrement différent. Rajouter différentes variantes est donc très simple, la seule modification à apporter au code déjà existant étant l’assignation de ce comportement dans la méthode actionBtnExport() de mainWindow.

# Bilan

## Organisation du travail



Lors de notre organisation prévisionnelle, nous avons choisi de diviser la phase d’analyse et conception en 3 sous-phases :

* Une première phase d’analyse ayant pour but de rechercher les solutions et les technologies les plus adaptées pour répondre aux besoins du client. Elle comprend aussi la mise en place des outils techniques et les outils de communication d’équipe.
* Une deuxième phase de conception pour structurer le projet, produire des diagrammes qui nous serviront à mieux organiser notre travail afin de rester efficace et pertinent. Elle servira de base et facilitera grandement la phase de développement.
* Une dernière phase de développement pour produire un prototype d’application qui suivra les choix de conception et qui répondra aux besoins spécifiés par le client dans le cahier des charges.
* Les deux dernières phases correspondent à l’affinage et la finalisation du TP

Cependant il y a eu des décalages lors de la réalisation du projet ( ici représenté par le bloc bleu barré ). En effet, nous avions anticipé que la phase de développement du premier prototype était en réalité trop courte pour pouvoir être terminée dans les temps, et avons donc décider de commencer en avance en parallèle de la réalisation du dossier d’analyse et de conception, et cela nous a donc permit une réalisation pertinente de ce dernier.

## Organisation de la communication

Concernant la communication durant ce projet, elle a été réalisée par plusieurs moyens. Tout d’abord au sein de l’équipe, grâce à plusieurs outils à savoir, Messenger, Skype, GitHub, réunion à l’IUT. Toutes nos données ont été échangés via la plateforme GitHub, notre projet est en ligne afin qu’on puisse y accéder depuis n’importe quel PC. Nous avions également organisé des réunions régulières (souvent le dimanche après-midi) pour mettre en commun notre avancement.

Dans un second temps, la communication avec le client et le superviseur à majoritairement été faite par mail et mais aussi grâce à des réunions à l’IUT. Il y a eu plusieurs réunions au cours de la période de réalisation du projet.

## 