Convertisseur Numérique c#



(Une image originale représentant le projet)

Pittier Stéphane – Cin4B

ETML

Environ 110H

Chef de projet : Gilbert Gruaz

Expert 1 : Xavier Carrel

Expert 2 : Jean Zahn

Table des matières

[1 Spécifications 4](#_Toc484183219)

[1.1 Titre 4](#_Toc484183220)

[1.2 Description 4](#_Toc484183221)

[1.3 Matériel et logiciels à disposition 4](#_Toc484183222)

[1.4 Prérequis 4](#_Toc484183223)

[1.5 Cahier des charges 4](#_Toc484183224)

[2 Planification Initiale 5](#_Toc484183225)

[3 Analyse 6](#_Toc484183226)

[3.1 Opportunités 6](#_Toc484183227)

[3.2 Document d’analyse et conception 6](#_Toc484183228)

[3.2.1 Analyse concurrentielle 6](#_Toc484183229)

[3.2.2 Structure graphique 7](#_Toc484183230)

[3.2.3 Maquette 9](#_Toc484183231)

[3.2.4 Structogrammes 10](#_Toc484183232)

[3.2.5 Développement 14](#_Toc484183233)

[3.3 Conception des tests 14](#_Toc484183234)

[3.4 Planification détaillée 14](#_Toc484183235)

[4 Réalisation 14](#_Toc484183236)

[4.1 Dossier de Réalisation 14](#_Toc484183237)

[4.1.1 Listes des outils 14](#_Toc484183238)

[4.1.2 Programme 14](#_Toc484183239)

[4.1.3 Nombre à virgule 14](#_Toc484183240)

[4.1.4 Fonction Décimal -> BCD 15](#_Toc484183241)

[4.1.5 Fonction Binaire -> GRAY 16](#_Toc484183242)

[4.1.6 Fonction convertButton\_Click 16](#_Toc484183243)

[4.1.7 Fonction convertToAll 17](#_Toc484183244)

[4.1.8 Méthode des Calcules 18](#_Toc484183245)

[4.1.9 Initialisation des Opérations 18](#_Toc484183246)

[4.1.10 Fonction addValue 19](#_Toc484183247)

[4.1.11 Fonction substractValue 20](#_Toc484183248)

[4.1.12 Fonction multiplicateValue 21](#_Toc484183249)

[4.1.13 Fonction d’affichage 22](#_Toc484183250)

[4.2 Modifications 22](#_Toc484183251)

[4.2.1 Conversion de nombre à virgule 22](#_Toc484183252)

[4.2.2 Les retenues des opérations 23](#_Toc484183253)

[4.2.3 Division détaillée 23](#_Toc484183254)

[4.2.4 Améliorations possible 23](#_Toc484183255)

[5 Tests 24](#_Toc484183256)

[5.1 Grille de tests 24](#_Toc484183257)

[5.2 Dossier des tests 24](#_Toc484183258)

[5.2.1 Bilan des tests 24](#_Toc484183259)

[5.2.2 Découverte des bugs 24](#_Toc484183260)

[6 Conclusion 25](#_Toc484183261)

[6.1 Bilan des fonctionnalités demandées 25](#_Toc484183262)

[6.2 Bilan de la planification 25](#_Toc484183263)

[6.3 Bilan personnel 26](#_Toc484183264)

[7 Divers 28](#_Toc484183265)

[7.1 Journal de travail 28](#_Toc484183266)

[7.2 Webographie 28](#_Toc484183267)

[8 Annexes 29](#_Toc484183268)

[8.1 Cahier des Charges 29](#_Toc484183269)

[8.2 Manuel Utilisateur 31](#_Toc484183270)

[8.3 Normes de codage ETML 31](#_Toc484183271)

[8.4 Code source 31](#_Toc484183272)

[8.4.1 FormConverter 31](#_Toc484183273)

[8.4.2 FormCalculate 31](#_Toc484183274)

[8.5 Journal de travail 31](#_Toc484183275)

[8.6 Planification détaillée 31](#_Toc484183276)

# Spécifications

## Titre

Réalisation d’une application de contrôle d’exercices en électronique numérique

## Description

Implémenter une application qui va permettre aux utilisateurs (élèves ou enseignants), de contrôler les résultats des exercices de conversions de données numérique en binaire, octal, décimal, hexadécimal, BCD nombres réels à virgule flottante, ainsi que des opérations élémentaires comme des compléments à deux, addition et soustraction.

## Matériel et logiciels à disposition

* 1 ordinateur standard ETML
* Visual Studio 2015
* Suite Office 2016

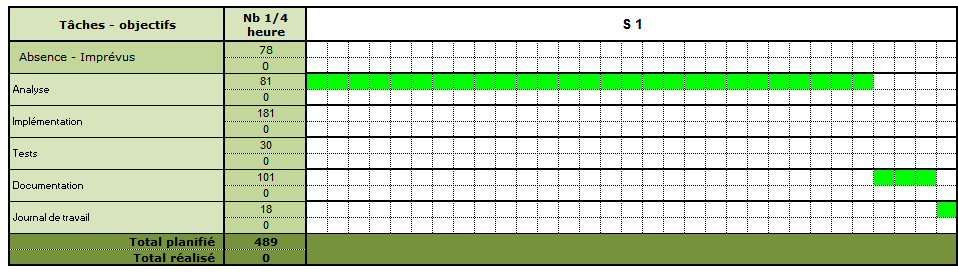
## Prérequis

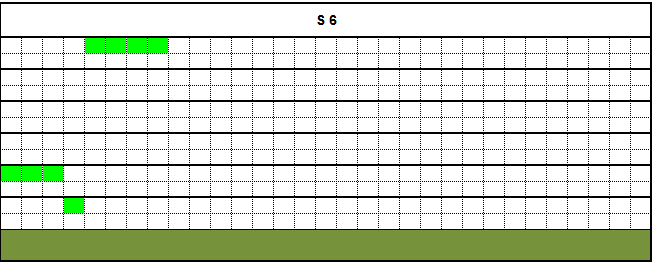
Avoir suivi les modules ICH à l’ETML, les projets et effectué des stages.

## Cahier des charges

[Lien vers le cahier des charges (Annexe 8.1)](#_Cahier_des_Charges)

# Planification Initiale





# Analyse

## Opportunités

Ce projet va me permettre de m’améliorer dans les points suivant :

* Code c#
* Gestion de projet
* Optimisation de code

C’est aussi un bon défi pour voir si j’arrive à gérer le stress lié à un projet sur six semaines.

## Document d’analyse et conception

Comme écrit dans les opportunités, le programme sera développé en C#. Il utilisera le type Windows Form Project, plus simple d’utilisation pour l’utilisateur que la console. Il sera aussi séparé en 2 fenêtre. Le convertisseur et le calculateur.

### Analyse concurrentielle

Pour cette partie, j’ai trouvé deux sites permettant de faire presque pareil que mon application. Le premier contient plus de types que je veux, mais la conversion en ternaire et en quintal n’est pas une priorité. Pour le deuxième, il ne peut convertir que le décimal, hexadécimal et binaire.

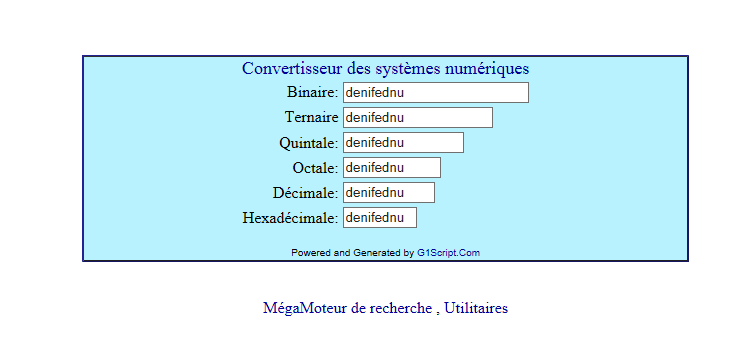
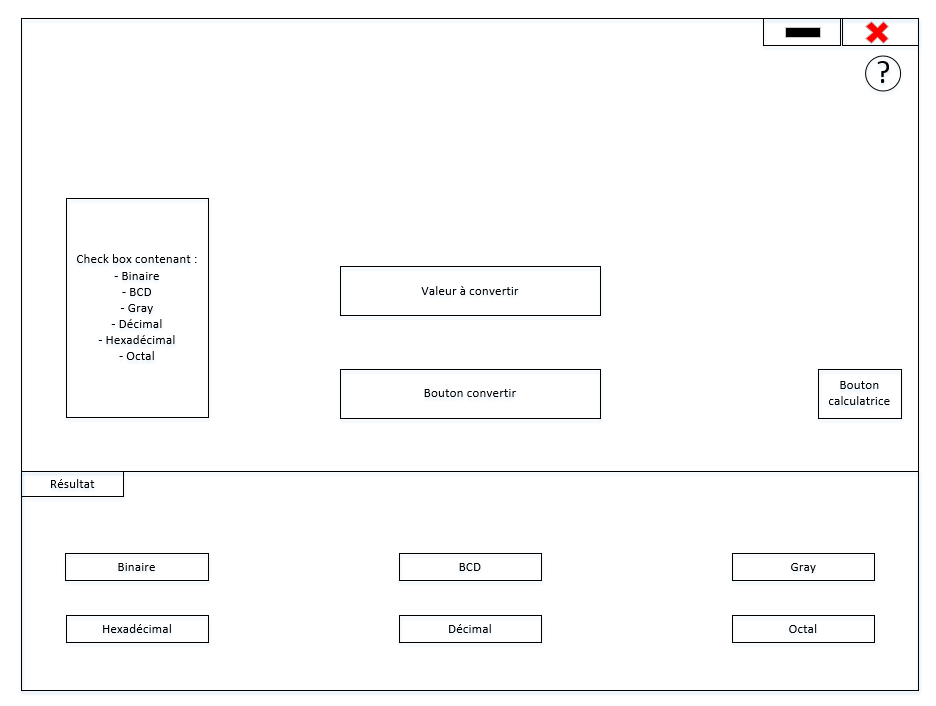
La différence se trouve sur le fait qu’aucun des deux sites ne puissent convertir des valeurs à virgule, et que le premier ne puisse pas convertir une valeur négative.

Figure 0.1 Exemple de conversion d'une valeur négative sur le site 1

### Structure graphique

Figure 0.2 Page d'accueil du site 2



6

5

4

1

2

3

Figure Design graphique de la FormConvert

La page principale du programme contient la partie convertisseur. Elle contient un nombre de petites informations diverses :

1. L’utilisateur doit choisir son type de base. Une série de check box seront à sa disposition pour informer le programme de quelle base de chiffre il doit partir pour effectuer la conversion.
2. Après avoir choisi son type, l’utilisateur doit entrer la valeur à convertir. Après avoir cliqué sur le bouton convertir, le programme va vérifier que toutes les informations sont correctes et lancer la méthode « Conversion ». Je décide de mettre ces options car dans le cas d’une recherche automatique du type, si l’utilisateur converti le nombre « 3654 » comment savoir s’il est question d’un nombre décimal, octal ou hexadécimal.
3. Une fois la conversion effectuée, le résultat va s’afficher. Pour gagner du temps, le résultat comprendra toutes les possibilités du programme (Binaire, BCD, Gray, Décimal, Octal et Hexadécimal).
4. Le bouton permettant d’afficher la fenêtre pour les opérations
5. Ce bouton permet d’afficher les informations relatives aux informations et au fonctionnement du logiciel
6. Bouton pour réduire et fermer le programme

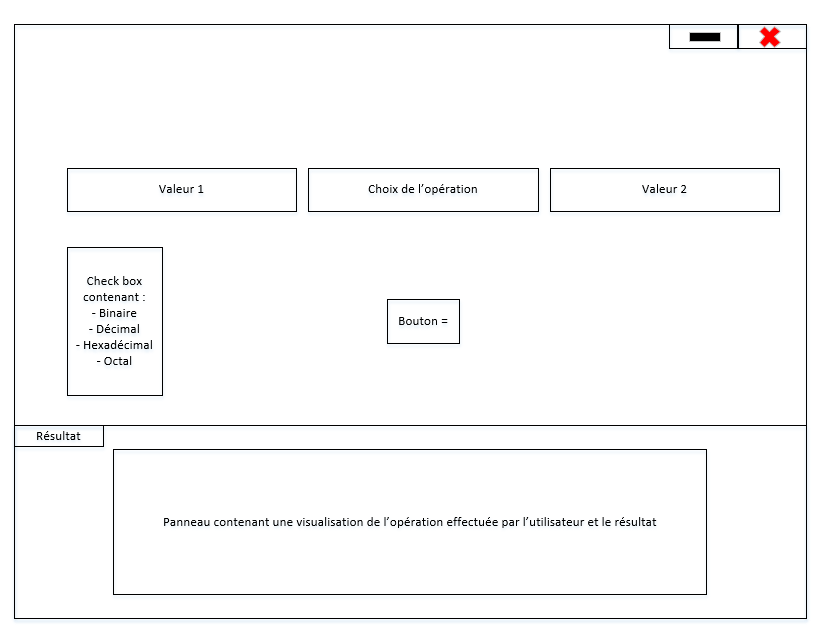
La fenêtre pour les opérations ressemble beaucoup à celle de la conversion. Cette page s’ouvre à côté de l’autre et est indépendant de la conversion.

Figure Design graphique de la FormCalculator

5

1

3

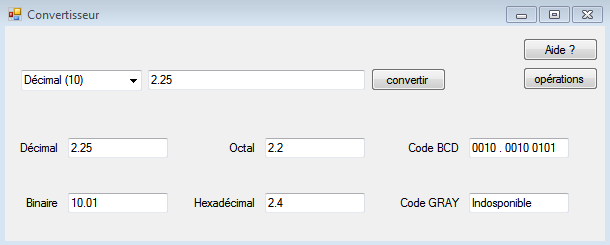
2

4

1. L’utilisateur entre les valeurs et choisi l’opération à effectuer (addition, soustraction et multiplication).
2. L’utilisateur choisi le type des valeurs qu’il a entré au point 1. Comme pour la fenêtre de conversion, le choix est indispensable pour éviter des problèmes dans le futur développement.
3. Une fois les valeurs entrées et le type choisi, l’utilisateur va cliquer sur le bouton « = ». Le programme va alors vérifier les valeurs et les envoyer dans la méthode « Calcul ».

1. Le panneau de résultat va afficher le résultat mais aussi un développement de l’opération pour que l’utilisateur puisse comprendre, de manière graphique, comment l’opération s’effectue.
2. Bouton pour réduire ou fermer la fenêtre de calcul. Entant donné qu’elle est indépendante par rapport à la conversion, si elle est fermée elle ne quitte pas le logiciel.

### Maquette

La maquette du site ressemble beaucoup à la version abordée au point 3.2.2. Cependant plusieurs modifications, discutée avec le chef de projet, viennent améliorer le programme.

2

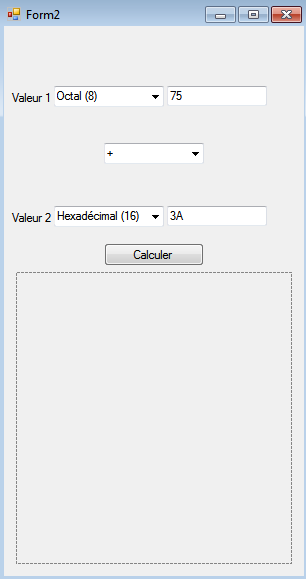
1

Figure Maquette de la FormConvert

Un remaniement de l’espace a été envisagé pour permettre une meilleure lisibilité.

1. Les checkboxs de la structure graphique, ont été remplacées par une liste déroulante plus pratique et plus petites à placer sur la fenêtre
2. Petite précision pour le code BCD et GRAY. Tous les formats peuvent être converti en code BCD et inversement.

Le code GRAY peut être converti uniquement en code BCD, et s’obtient uniquement à partir du BCD.

Pour la maquette de la calculatrice, certaines différences font aussi leurs apparitions.

1. Les 2 valeurs possèdent maintenant aussi une liste avec les types pour permettre les opérations avec des valeurs de types différentes.

1

2

Les types disponibles dans ces opérations sont le binaire, le décimal, l’octal et l’hexadécimal.

1. Les opérations sont aussi sous format de liste déroulante pour la même raison que les types des valeurs.

Figure Maquette de la FormCalculator

3

3A (16) = 72 (8)

1

75

+ 72

167 (8)

75 (8) = 3D (16)

1

3D

+ 3A

77 (16)

1. Pour les résultats étant donné qu’il faut une base commune pour effectuer les opérations, j’ai choisi de faire les 2 bases. La première version de l’opération sera toujours dans la base de la valeur 1. Et la deuxième dans la base de la valeur 2. Ensuite le programme prendra les deux valeurs pour effectuer l’opération demandée et afficher le détail sous forme d’opération en colonne avec les retenues.

### Structogrammes

Le programme sera décomposé en 3 parties majeurs. La conversion, le calcul et l’affichage.

On peut aussi séparer les grandes méthodes en plus petites mais faire un structogramme pour ces étapes intermédiaires ne servirait pas à grand-chose.

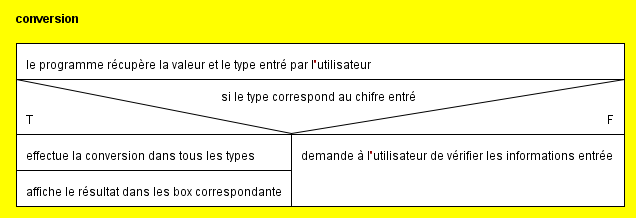
1. La méthode Conversion

Figure Structogramme conversion

Ce structogramme va me servir pour la conversion binaire, décimal, octal et hexadécimal. Les fonctions fournies par c# suffiront amplement à remplir ce travail.

1. BCD et GRAY

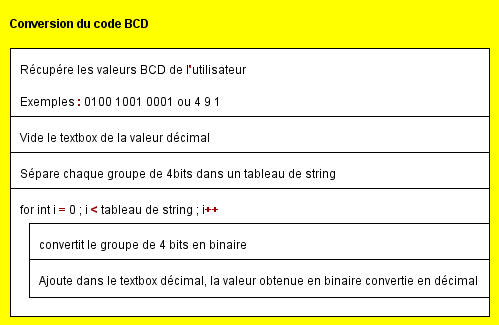
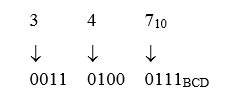
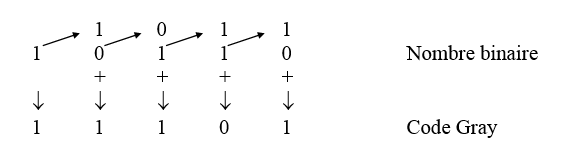
Pour le BCD et le code gray les choses se compliques un peu. Pour le code BCD il faut récupérer les valeurs de chaque « segments » de 4 bits afin de les convertir dans les bons formats.

Figure Structogramme de la fonction de conversion du BCD

Ce qui suis la méthode présentée dans le document d’ELEOC donné aux élèves.

Pour le code GRAY, Je garde aussi le principe donné en cours d’ELEOC. Je convertis la valeur entrée par l’utilisateur en binaire. Ensuite je fais une addition logique entre le bit actuel et le bit précédent. Ce qui suis le schéma suivant.

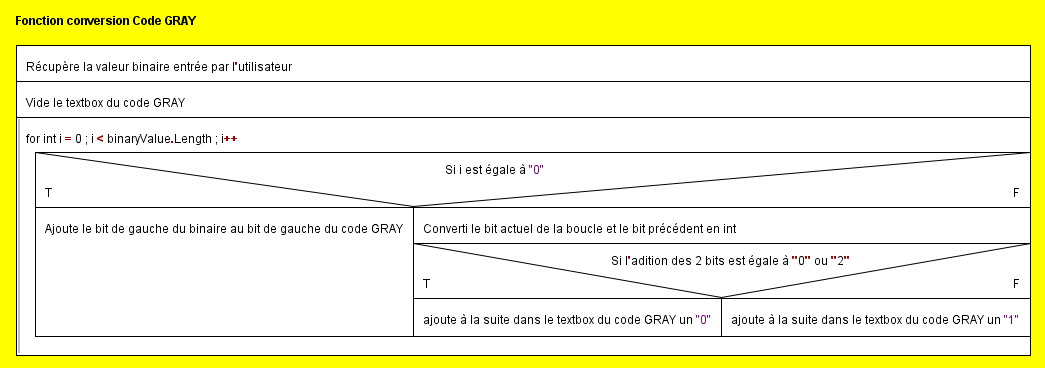
Ce qui donne le structogramme suivant pour la fonction binaireToGray

Figure Structogramme de la fonction de conversion du GRAY

1. La méthode Calcul

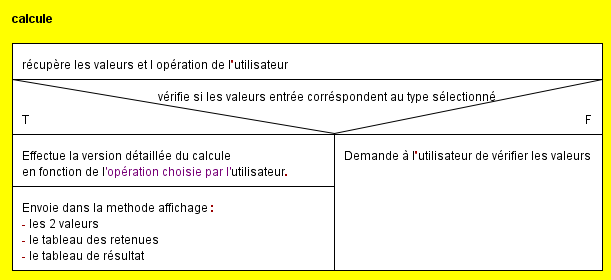
Comme marqué dans le chapitre 4.2.1, j’ai complétement remanié le système de calcul et d’affichage.

Figure Structogramme de la fonction des opérations

Pour montrer comment fonctionne la version détaillée du calcul, voici le structogramme de la fonction addition.

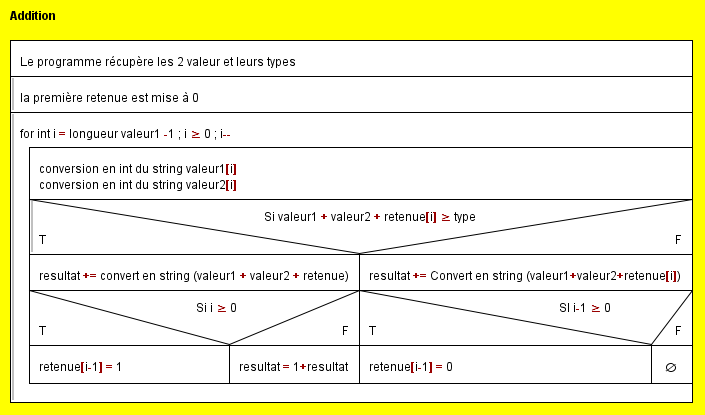


Figure Structogramme de la fonction de l'addition

Toutes les fonctions pour les opérations suivront le même schéma. En effectuant directement la version détaillée du calcul, j’économise du temps et des lignes de codes.

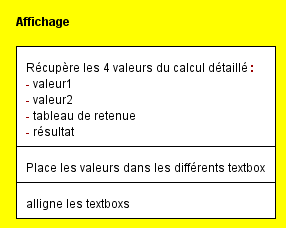
1. La méthode Affichage

Figure Structogramme de la fonction d'affichage

Pour la méthode d’affichage, je vais me baser sur le document utilisé en ELEOC. Il arrive à montrer de manière précise comment sont effectuées les opérations.

Pour le placement se sera surement des textbox qui seront utilisées. Mais si une meilleure alternative est trouvée, il en sera fait mention dans le chapitre **4.1.13.**

### Développement

La plus grosse partie de ce TPI réside dans la partie développement du programme. Pour ce faire, je vais utiliser l’ide Visual Studio 2015. Il sera développé en langage c# et suivra les normes de l’ETML.

## Conception des tests

Les tests suivront une grille prédéfinie, et le tout sera détaillé dans le point 5 du rapport.

## Planification détaillée

[La planification détaillée du projet se trouve dans les annexes au chapitre 8.6](#_Planification_détaillée)

# Réalisation

## Dossier de Réalisation

### Listes des outils

* Visual Studio 2015
* ConventionsDeCodageV2.3.3 ETML

### Programme

Les fichiers contenant le code sources sont disponible sur le git ou dans les annexes. Dans cette partie je vais reprendre les méthodes les plus intéressantes et les décrire dans le détail.

### Nombre à virgule

Pour le moment les nombres à virgule flottante ne fonctionne pas. La fonction de base de c# ne supporte pas les variables de format « Float » ou « Double ».

Cependant une alternative à l’aide d’un tableau de bits est envisageable. Cela rendrait Le développement un peu plus long. En effectuant des recherches, j’ai trouvé une méthode qui permet de passer d’un nombre à virgule décimal au tableau de bits, mais impossible de le transformer en un autre format.

Une fois l’affichage terminé, je me pencherais plus en profondeur sur ce problème.

### Fonction Décimal -> BCD

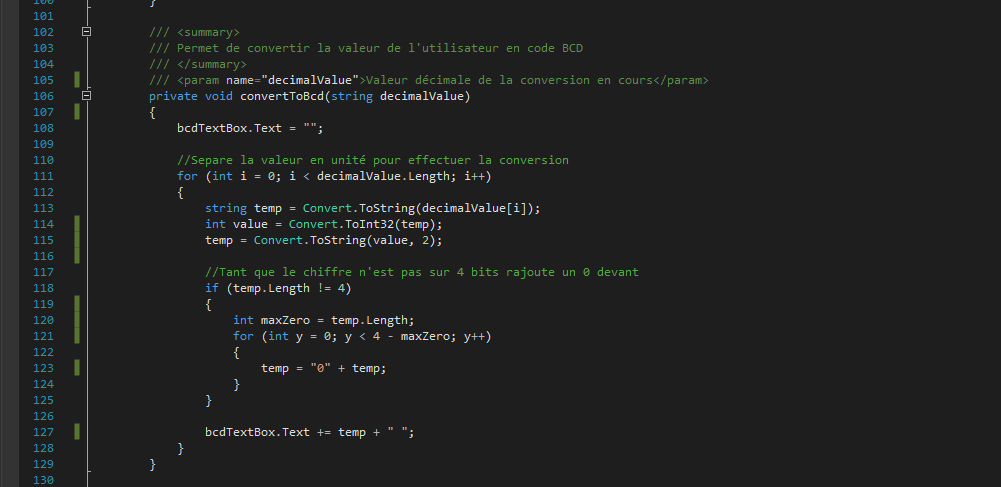
La fonction pour convertir un nombre décimal en un nombre en code BCD me semble un bon exemple pour commencer.

Figure Fonction convertToBcd

La valeur entrée par l’utilisateur, est envoyée dans la méthode sous forme de tableau « String ». La première boucle commence, elle a pour effet de prendre chaque chiffre de la valeur pour ensuite la convertir en binaire.

Une fois la conversion en binaire effectuée, cette nouvelle valeur est stockée dans une variable temporaire (appelé « temp »). Le code BCD nécessite 4 bit et dépendant de la valeur il se peut que ce maximum ne soit pas atteint.

C’est alors que la deuxième boucle entre en jeu. Son principe est de calculer la longueur de notre string et de rajoute le bon nombre de 0 devant pour arriver à quatre bits. Une fois ceci effectué, la variable temp est placée dans la textbox de la valeur BCD.

### Fonction Binaire -> GRAY

Figure Fonction convertToGray

Pour cette fonction, le principe suit le même cheminement que pour le « Décimal -> BCD ».

Une valeur en tableau « String » est envoyée dans la fonction. Elle va ensuite prendre la longueur de ce tableau. Comme le bit de gauche est le même en binaire et en code gray, il va directement le placer dans la textbox. Pour les autres bits, il va sélectionner le bit actuel de la boucle plus celui à la position « i-1 », afin d’effectuer le complément à deux. Pour être sûre que le programme ne fasse pas d’erreur il va vérifier le résultat de ce complément et attribuer la valeur 1 ou 0 à la suite du textbox.

### Fonction convertButton\_Click

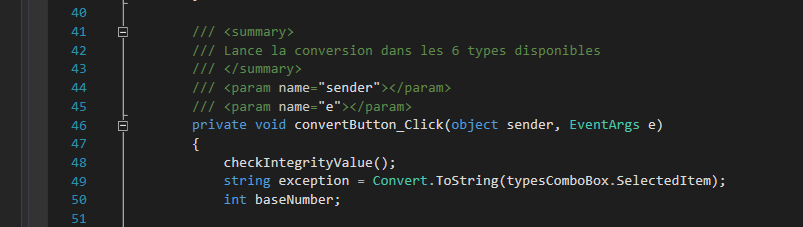
Avant de convertir dans les 6 types, le programme va préparer les informations.

Figure Fonction convertButton\_Click

Il va tout d’abord récupérer la valeur de l’item sélectionné dans la liste déroulante et initialiser la variable baseNumber.

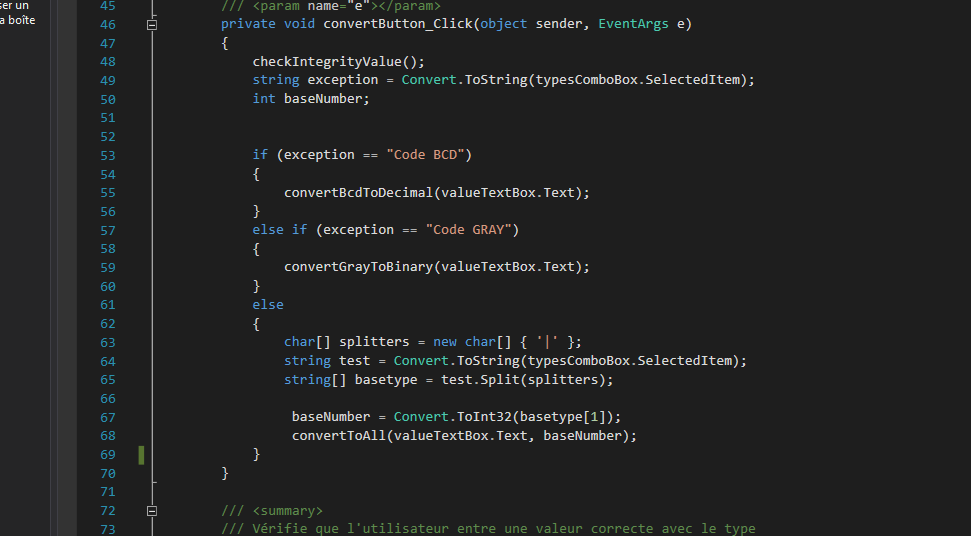


Figure Séparation des types de bases

Ensuite, grâce à la variable « exception », le programme va définir la base de la valeur entrée par l’utilisateur. Si elle est égale à BCD ou GRAY, il va d’abord lancer la fonction de conversion adéquate.

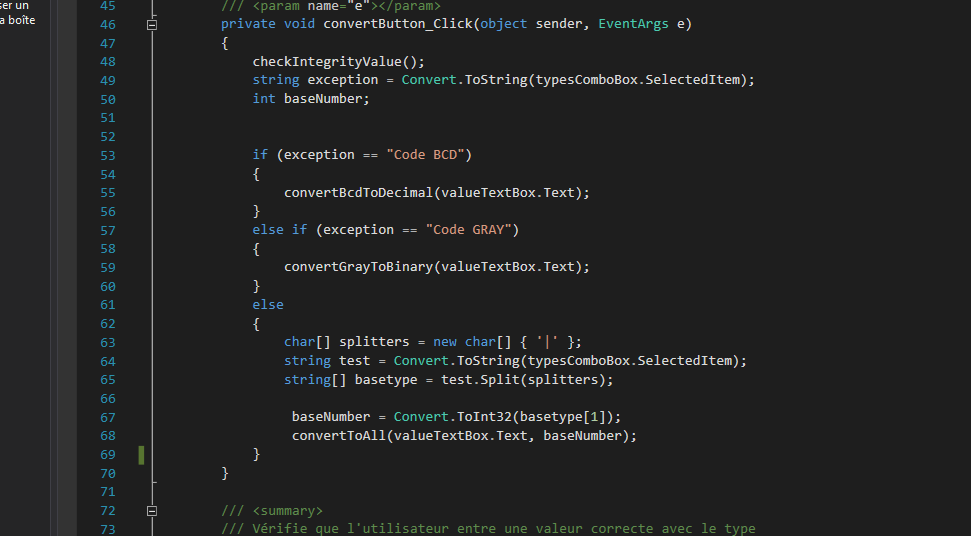


Figure Lancement du convertToAll

Sinon il va séparer la valeur au niveau du | pour ne garder que le chiffre de la base. Pour l’exemple, si l’utilisateur choisi le type « Binaire | 2 » Le programme donnera la valeur 2 à la variable userBase.

Par la suite, il lancer la fonction « convertToAll » avec comme argument :

* La valeur entrée par l’utilisateur dans le textbox
* La base de la valeur choisie par l’utilisateur

### Fonction convertToAll

Figure Fonction ConvertToAll

Pour cette fonction, le principe est simple. Les valeurs envoyée par la fonction « Convert\_click » sont ensuite transposée dans toutes les bases. Pour ce faire, j’utilise la fonction de conversion de type proposée par c#.

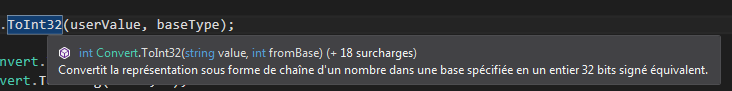


Figure Description des arguments de ConvertToInt32

L’orsque que l’on effectue une conversion en type INT ou STRING, un argument permet de venir insérer une valeur (2,8,10 ou 16) pour choisir la base de notre base.

Une fois ces opérations éfféctuée, le programme lance les fonction de conversion en BCD et en GRAY

### Méthode des Calcules

La méthode pour effectuer les opérations, se situe dans la Form Calculate. Elle se compose de quatre méthodes principales :

* Addition
* Soustraction
* Multiplication
* Division

Le fait de la séparer en quatre petites fonctions, permet de mieux contrôler les spécificités de chaque opération arithmétique. Cependant cela rajoute des lignes de codes. Pour les opérations avec des valeurs de bases différents, les fonctions retourneront le résultat avec la valeur 2 convertie avec la base de la valeur 1.

### Initialisation des Opérations

Au moment où l’utilisateur va activer le bouton opération, le programme va initialiser tous les éléments. Cette initialisation va prendre place dans 2 fonction :

* calculateButton\_Click
* calcualteValues

Un switch est mis en place pour faciliter l’aiguillage entres les diverses fonctions de calcules.

### Fonction addValue

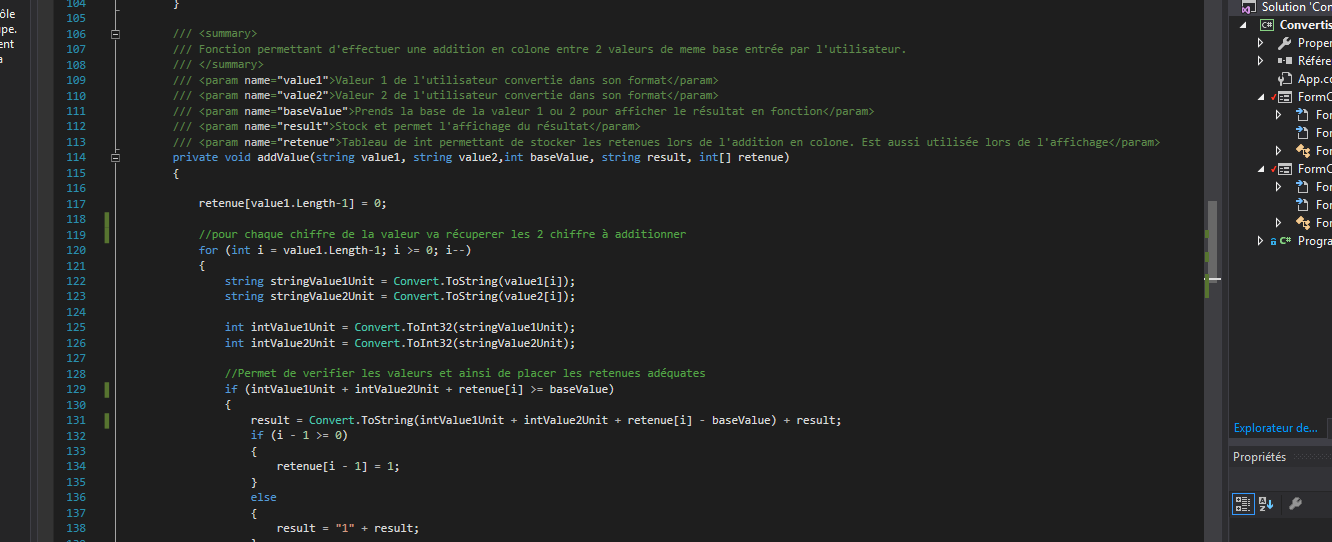
****Le principe restera pour toutes les fonctions de calcules. Voici le fonctionnement de cette fonction.

Figure Fonction addValue

Le programme va prendre chaque unité des valeurs, les unes après les autres.

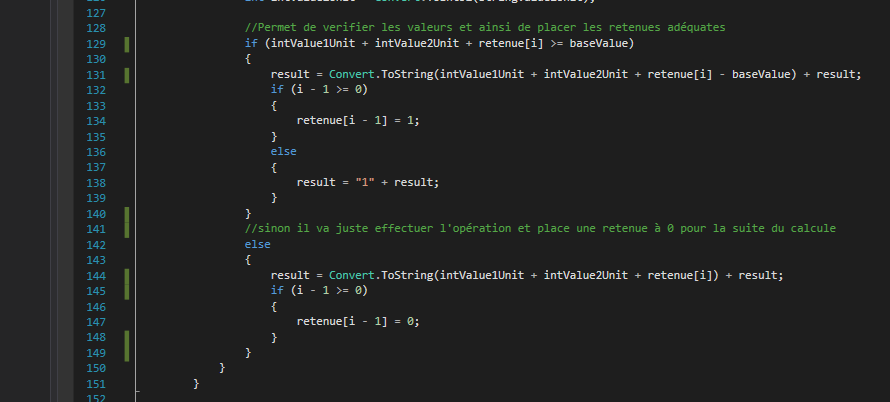


Figure Système de retenue

Ensuite, il va effectuer une addition entre le chiffre 1 et le chiffre 2. Si la valeur obtenue est plus grande que la base des 2 valeurs, le programme va placer une retenue de 1 dans le tableau.

Pour le résultat, si l’addition est plus grand que la base, il va retirer la base et placer cette valeur dans la variable de résultat.

Si la valeur obtenue est moins grande que la base, il va placer un 0 dans le tableau des retenues et placer la valeur dans la variable de résultat. Mais c’est la dernière opération, et que la retenue actuelle vaut 1, il va la placer en première position du string résultat.

### Fonction substractValue

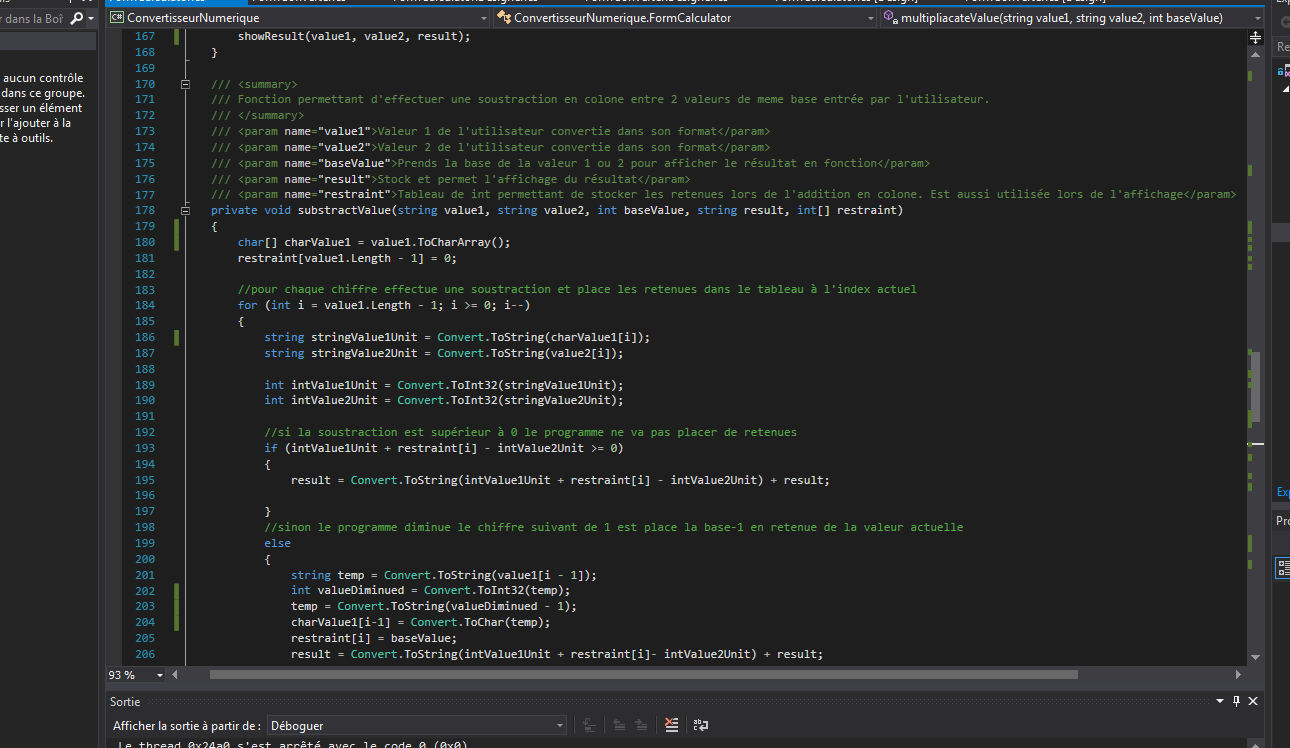
Pour la fonction de soustraction, le principe reste le même que l’addition.

Figure Initialisation de la fonction sbstractValue

Une fois les valeurs transformée et envoyée dans la fonction, le programme va soustraire à chaque unité.

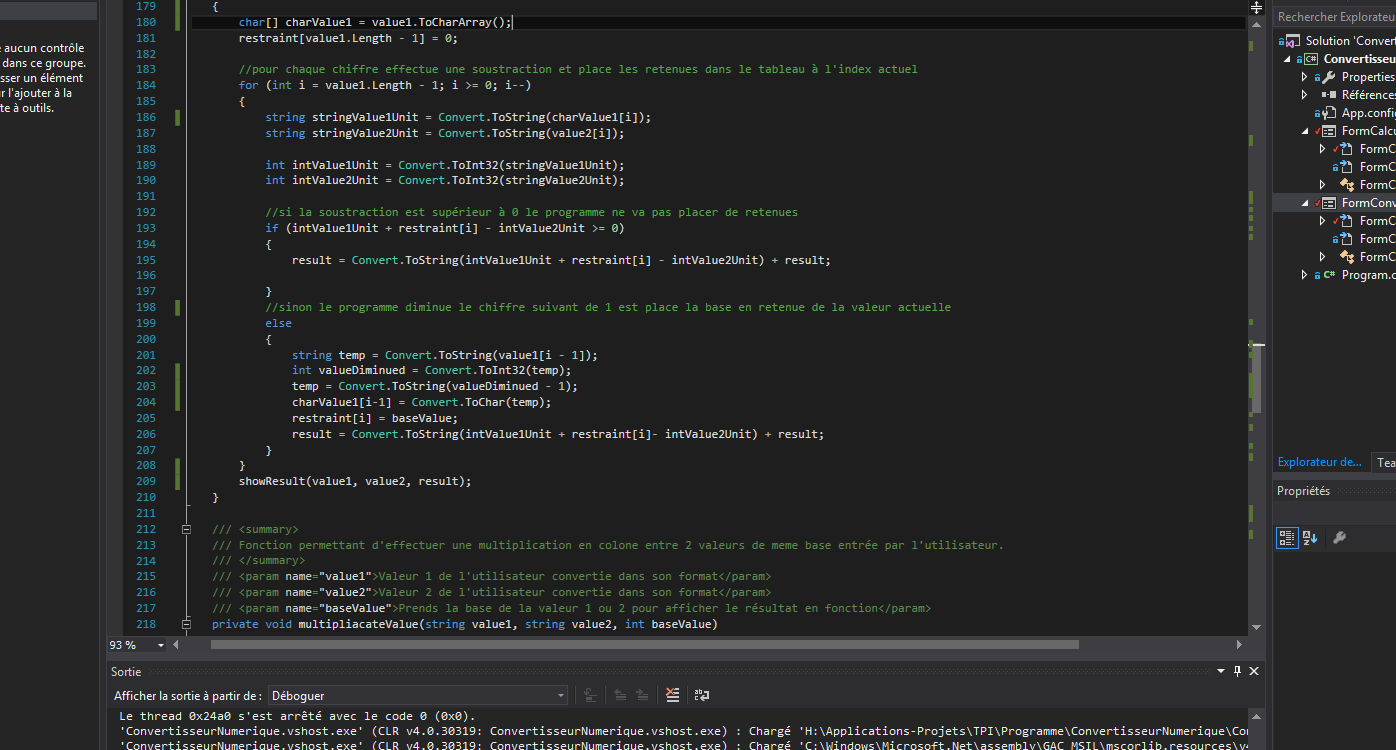


Figure Système de soustraction et des retenues

Si le résultat est plus petit que 0, la fonction va alors mettre en place la retenue. Pour ce faire elle prend le chiffre suivant et le diminue de 1. La retenue, quant à elle, vaut la valeur de la base (sauf pour le binaire ou elle vaut 1). Ensuite, elle va refaire la soustraction avec toute les nouvelle valeurs, et placer ce résultat dans la variable « result ».

Ensuite elle va envoyer les deux valeurs et le résultat dans la fonction d’affichage.

### Fonction multiplicateValue

Pour cette fonction, le principal problème a été de séparer les retenues du nombre à ajouter au résultat. Car si dans l’addition, une retenue est toujours égale à 1 ou 0, là il se peut que la retenue vaille 9. Pour contrer ce problème j’ai mis en place un système de tableau à 2 dimension.

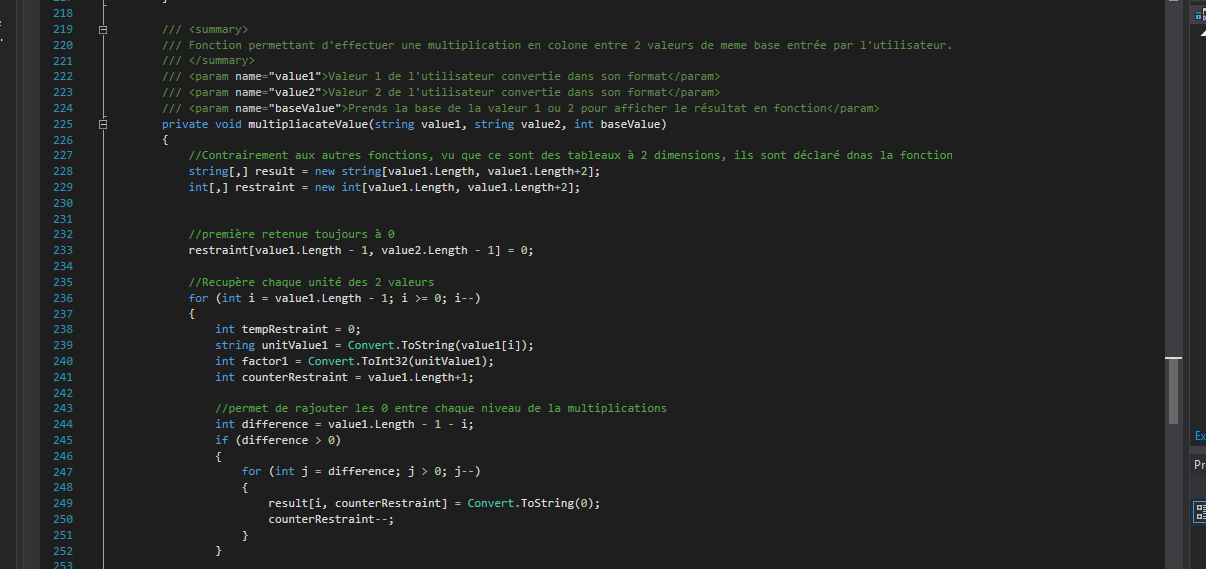


Figure Initialisation de la fonction multiplicateValue

Dans le cas d’une valeur à 2 chiffre, l’index 1 du tableau est consacré aux retenues du premier chiffre et l’index 0 aux retenues du deuxième. Les résultats sont stockés dans un tableau de string.

Pour alléger le code, j’ai mis en place le 31.05.2017, 2 variables « factor ». Elles permettent de stocker les valeurs à multiplier et permet une meilleure lisibilité du code.

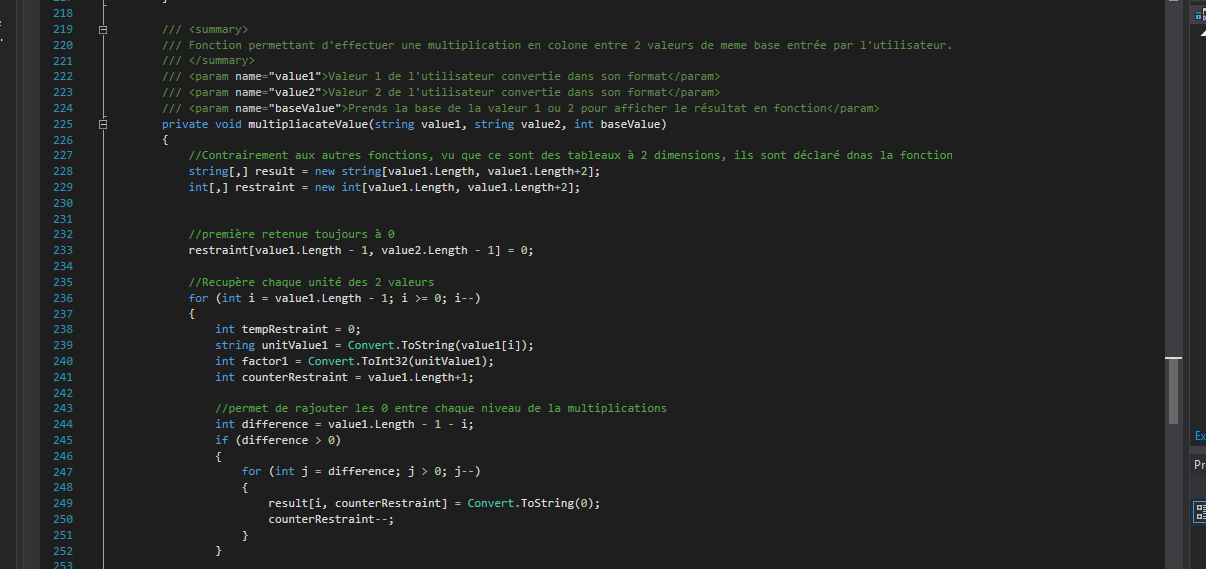
Pour le fonctionnement, je reste sur mon principe de double boucle for pour permettre une multiplication d’absolument tous les nombres.

Figure Algorithme pour les retenues

Dans la première boucle, je récupère chauqe une unité de la valeur 1. Je mets la première retenue à 0, et lance une variable de compteur permettant de placer les 0 quand on change de niveau.

La deuxieme boucle, permet de prendre chaque unité de la valeur 2. Une fois ceci récuperer, j’effectue une multiplication entre les deux, en y ajoutant la retenue d’avant stockée dans la variable « tempRestrain ». Si la valeur est en dessous de la base, j’inscrit juste la valeur dans le tableau de résultat à la suite du précedant.

Si le résultat est plus grand que la base. Je prends le premier chiffre et le stock dans le tableau des retenues et dans la variable « tempRestrain ». Pour le deuxième chiffre, je le place à la suite dans le tebleau de résultat.

Si la deuxième boucle arrive à 0 mais qu’une retenue est encore stockée dans la varible de retenue, elle sera automatiquement ajoutée à la suite dans le tableau de résultat.

### Fonction d’affichage

Pour l’affichage, la fonction consiste à placer les valeurs données par les fonctions de calcul dans les bon textboxs.

On peut remarquer que le textbox pour le tableau des retenues à disparu. Ceci vient du fait

## Modifications

### Conversion de nombre à virgule

La conversion de nombre à virgule ne fonctionne pas. La méthode de base de c# ne supporte pas les valeurs de types « Float » ou « Double ».

Après quelques recherches je suis tombé sur un début de solution. Il consiste à transformer notre valeur en tableau de bits, et de convertir la valeur bits par bits. Cependant une erreur se produit lors de la conversion. Dès que j’essaie de convertir les bits, il revient sur une base 10.

Sinon une deuxième solution serait de passer par un parseInt. J’ai trouvé un convertisseur qui fonctionne avec cette méthode : <http://coderstoolbox.net/number/>

Le principal souci du parseInt, est que si je veux l’intégrer, je dois revoir toute l’infrastructure de mon programme. Ce qui correspondrait à tout recommencer la partie conversion.

Il me permettrait, de séparer ce qui va avant et après la virgule. Ensuite par une fonction de conversion en bits par bits, je pourrais convertir le contenu d’après virgule. Pour ce qui se trouve avant, j’utiliserais la même méthode qu’actuellement.

### Les retenues des opérations

Le principal souci des retenues, se situe au niveau de l’affichage. Le problème se trouve dans le fait de les placer correctement au-dessus des bonnes valeurs. Un moyen facile de contrer ce problème est de placer les valeurs à la fin du textboxs et non au début.

Ensuite j’ai rencontré plusieurs soucis avec les retenues des multiplications. En effet, devant effectuer un tableau à x dimensions, je me retrouve avec un dilemme lors de l’affichage. Créer plusieurs textboxs ou le dimensionner pour que toutes les valeurs entre dedans.

La meilleure solution serait de placer un textbox par ligne de retenue. Permettant une meilleure gestion du placement de ces derniers.

### Division détaillée

Pour la division détaillée, je rejoins le problème de la conversion. Pour que tout fonctionne parfaitement, je devrais passer par un parseInt et entièrement remanier le système d’opérations. Pour le moment il s’agit d’une division sommaire qui retourne un résultat entier (arrondi au-dessus).

### Améliorations possible

1. Intégrer les nombres à virgule
2. Afficher les retenues à la fin des opérations
3. Effectuer du multi threading

Ceci permettrai d’effectuer les opérations avec des valeurs de types différentes en même temps et d’afficher les deux résultats possibles. En contrepartie, ceci prends du temps à mettre en place et nécessite une bonne optimisation pour ne gagner que quelques microsecondes.

1. Intégrer un système de RegEx pour faciliter le contrôle des valeurs.
2. Modifier la fonction division pour qu’elle la fasse de manière détaillée.

# Tests

## Grille de tests

## Dossier des tests

### Bilan des tests

### Découverte des bugs

* Convertir un nombre avec une lettre dedans.
  + Date de découverte : 01.06.2017
  + Solution : Intégrer un système de REGEX dans le programme
  + Estimation du temps pour résoudre : 2h
* Lors d’une multiplication, le programme plante avant d’afficher le résultat
  + Date de découverte : 01.06.2017
  + Solution : corriger la fonction pour trouver le bug
  + Temps passé à résoudre : 1h30
  + Date d’intégration : 02.06.2017
* Sur la form « Calculator », le programme ne vérifie pas si les textbox sont bien remplis
  + Date de découverte : 01.06.2017
  + Solution : intégrer un système de vérification comme dans la form « Converter »
  + Temps passé à résoudre : 1h
  + Date d’intégration de la solution : 01.06.2017
* La conversion de nombre à virgule
  + Date de découverte : 18.05.2017
  + Solution : Une solution est disponible au point 4.2.1 du rapport.
  + Estimation du temps pour résoudre : 10h
* La fonction « multiplicateValue »
  + Date de découverte : 02.06.2017
  + Problème : Lors d’une multiplication, les opérations fonctionne normalement. Cependant il reste un souci, si on met une valeur 2 plus longue (en nombre de caractère) que la valeur 1, le programme va planter.
  + Solution : Modifier les tailles des tableaux de résultat et de retenue, pour qu’il puissent être possible de calculer toutes les valeurs.
  + Estimation du temps pour résoudre : 3h
* Opérations avec un nombre à virgule ou un nombre négatif
  + Date de découverte : 18.05.2017
  + Problème : Tout comme les conversions, les nombres à virgule ou les nombres négatif ne passent pas.
  + Solution : Remanier le programme avec la méthode ParseInt pour faire fonctionne le tout.
  + Estimation du temps pour résoudre : 4h

# Conclusion

## Bilan des fonctionnalités demandées

Pour les fonctionnalités demandées, je vais répartir de la manière suivante :

1. **Fonctionnalités terminées**

* Conversion d’un nombre entier d’un des six types en tous les autres types
* Opérations de nombre entier de n’importe quel type

1. **Fonctionnalités inachevées**

* Conversion de nombre à virgule
* Addition et soustraction de nombre à virgule ou négatif.

Toutes les informations relatives à ces deux points se trouve dans les chapitres 4.2 et 5.2 Pour expliquer pourquoi je mets une différence de temps entre la conversion et les opérations à virgule, c’est que malgré la documentation lue, il me faudrait un temps d’adaptation pour tout mettre en place.

## Bilan de la planification

La planification se trouve dans les annexes au point [8.5](#_Journal_de_travail). Mais pour expliquer les différences de temps je vais le reprendre semaine par semaine.

**Semaine 1 :**

Pour la première semaine il n’y a pas grand-chose à dire. L’analyse m’a pris moins de temps que prévu, et m’a permis de bien tout intégrer dans la documentation.

**Semaine 2 :**

A partir de là, les choses ont pris une drôle de tournure. J’ai dû refaire mes maquettes, donc cela m’as pris du temps sur l’analyse mais je compense le temps perdu par une mise en place plus rapide des deux interfaces du programme.

Ensuite vient le moment de commencer la méthode de conversion. Malgré le fait d’avoir analysé ce que je voulais faire, je prends vite du retard (je prends deux fois plus de temps que prévu) à cause des méthodes de conversion du code BCD et du code GRAY. C’est aussi à ce moment que je remarque que les nombres à virgule ne fonctionne pas. Je cherche une solution mais y passant trop de temps je documente le problème et passe à la suite.

**Semaine 3 :**

A partir de ce moment, je décide de faire mon projet en suivant au mieux ma planification. Mais tous les petits soucis rencontré auparavant me font prendre du retard. Pour être le plus économes en temps possible, je travail le plus possible sur la fonction de calcul. Au bout de la semaine, je dépasse un peu le temps prévu, mais c’est dû au fait que j’ai décidé à ce moment de fusionner les fonctions de calcul et d’affichage.

**Semaine 4 :**

A ce moment, je décide de d’intégrer dans la doc tout ce que je n’y avais pas encore mis. Après je cumule du temps sur la fonction de conversion en essayant d’intégrer les nombres à virgule.

**Semaine 5 :**

Je commence par régler quelques petits bugs repérer à la fin de la semaine précédant. Après un rapide test, je remarque que la fonction de multiplication ne fonctionne pas du tout. A ce moment je décide de mettre de côté les nombres à virgule et les négatifs, pour pouvoir finir ceci. Ce qui explique le dépassement de temps sur cette tâche.

Une fois ceci régler je rattrape mon retard en finissant ma documentation.

**Semaine 6 :**

En tenant compte du lundi de congé, cette semaine a principalement servi à corriger deux ou trois bugs de mise en page sur mon rapport et à terminer les impressions.

## Bilan personnel

En conclusion de ce bilan, je constate que malgré le fait que j’ai quand même suivi un maximum ma planification dans ma réalisation, le fait que je prenne du retard sur les deux fonctions principales du programme m’a empêché d’ajouter certaine fonction, qui aurait rendu mon application meilleure.

Si le projet était à refaire, je prendrais un peu plus de temps sur mon analyse. Ceci m’aurais permis d’éviter quelques petits soucis comme la fonction de multiplication ou les nombres à virgule. Sinon je pense que je ne changerais pas énormément ma façon de travailler car j’estime avoir un rythme assez soutenu.

Pour les points positifs du projet, je noterai un chef de projet et des experts toujours prêt à me donner un coup de main ou à répondre à mes questions.

Ce projet m’a aussi appris à mieux gérer mon temps. Avant je passais beaucoup de temps sur la recherche de solutions pour que tout fonctionne. Maintenant je document pourquoi ça ne marche pas, et j’y reviens quand j’ai du temps ou une idée. Il m’a aussi appris à mieux gérer mon code, et à mieux optimiser ma façon de coder.

Si je devais donner suite à ce projet, je reviendrais sur les améliorations possible données au point 4.2.4. Et j’essaierait de continuer à optimiser mon code pour le rendre plus pérformant.

Je tiens à remercier les personnes suivantes :

* Daniel Jorge et M. Melly : Pour leur idée du multi-threading
* Sacha Grenier : pour avoir corrigé un bout de mon rapport
* Le chef de projet et les experts : pour le retour régulier et leurs idées d’améliorations

Stéphane Pittier

ETML

07.06.2017

# Divers

## Journal de travail

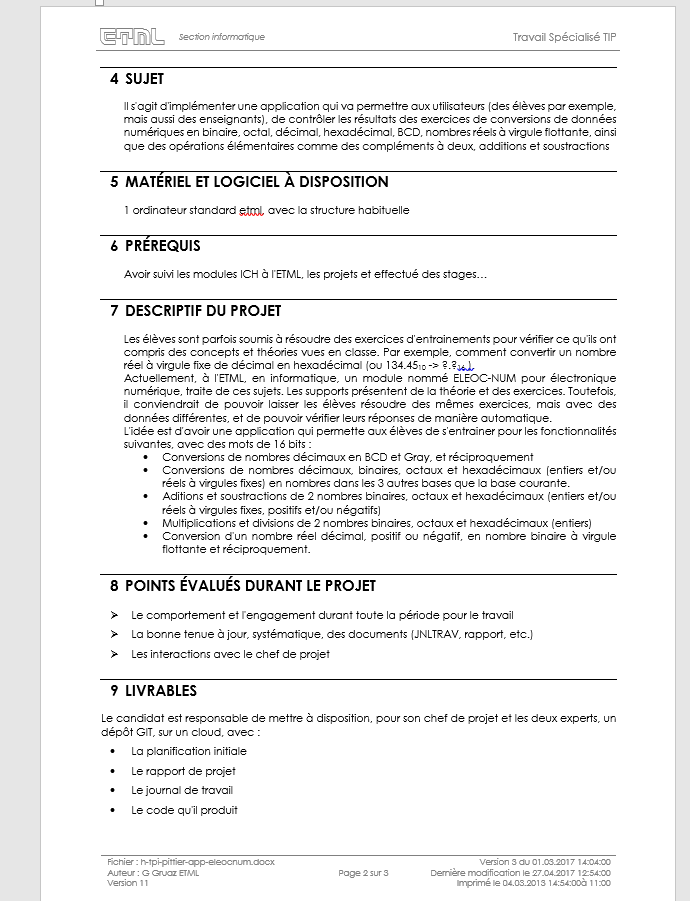
[Le journal de travail se trouve dans les annexes au chapitre 8.5](#_Journal_de_travail)

## Webographie

* Analyse concurrentielle
  + [www.aly-abbara.com/utilitaires/convertisseur/convertisseur\_chiffres.html](http://www.aly-abbara.com/utilitaires/convertisseur/convertisseur_chiffres.html)
  + [www.sebastienguillon.com/test/javascript/convertisseur.html](http://www.sebastienguillon.com/test/javascript/convertisseur.html)
* Convertisseur avec nombre à virgule
  + <http://coderstoolbox.net/number/>
* Documentations c#
  + <https://docs.microsoft.com/en-us/dotnet/csharp/>
  + <https://msdn.microsoft.com/fr-fr/library>
* C# Variable en lecture seule
  + <https://openclassrooms.com/forum/sujet/c-erreur-indexeur-ne-peut-pas-etre-assigne-41392>
* C# méthode parseInt
  + <https://msdn.microsoft.com/en-us/library/1kc6b02f(v=vs.100).aspx>
* C# RegEx
  + <https://msdn.microsoft.com/fr-fr/library/hs600312(v=vs.110).aspx>
* Lien du Git
  + <https://github.com/StephanePittier/TPI>

# Annexes

## Cahier des Charges



## Manuel Utilisateur

## Normes de codage ETML

## Code source

### FormConverter

### FormCalculate

## Journal de travail

## Planification détaillée