4 images 1 mot

  
  
  
  
  
  
Kadri Krasniqi / Cin4B

ETML

110 heures

Chef de projet : Gilbert Gruaz

Expert 1 : Didier Viret

Expert 2 : Frédérique Andolfatto

Table des matières

[1 Spécifications 4](#_Toc482874919)

[1.1 Titre 4](#_Toc482874920)

[1.2 Description 4](#_Toc482874921)

[1.3 Si le temps le permet 4](#_Toc482874922)

[1.4 Matériel et logiciels à disposition 4](#_Toc482874923)

[1.5 Prérequis 4](#_Toc482874924)

[1.6 Cahier des charges 4](#_Toc482874925)

[2 Planification Initiale 5](#_Toc482874926)

[3 Analyse 8](#_Toc482874927)

[3.1 Opportunités 8](#_Toc482874928)

[3.1.1 Difficultés potentielles : 8](#_Toc482874929)

[3.2 Document d’analyse et conception 8](#_Toc482874930)

[3.2.1 Maquette interface graphique 8](#_Toc482874931)

[3.2.2 Analyse base de données 11](#_Toc482874932)

[3.2.2.1 MCD 11](#_Toc482874933)

[3.2.2.2 MLD 12](#_Toc482874934)

[3.2.2.3 MPD 13](#_Toc482874935)

[3.2.3 Analyse programme 14](#_Toc482874936)

[3.3 Conception des tests 15](#_Toc482874937)

[3.4 Planification détaillée 15](#_Toc482874938)

[4 Réalisation 16](#_Toc482874939)

[4.1 Dossier de Réalisation 16](#_Toc482874940)

[4.1.1 Base de données 16](#_Toc482874941)

[4.1.2 Interface graphique 18](#_Toc482874942)

[4.1.3 Connexion c# --> sql server 19](#_Toc482874943)

[4.1.4 Exécution des requêtes SQL 19](#_Toc482874944)

[4.1.5 Récupération des images du mot en cours 20](#_Toc482874945)

[4.2 Modifications 22](#_Toc482874946)

[5 Tests 22](#_Toc482874947)

[5.1 Dossier des tests 22](#_Toc482874948)

[6 Conclusion 23](#_Toc482874949)

[6.1 Bilan des fonctionnalités demandées 23](#_Toc482874950)

[6.2 Bilan de la planification 23](#_Toc482874951)

[6.3 Bilan personnel 23](#_Toc482874952)

[7 Divers 23](#_Toc482874953)

[7.1 Journal de travail 23](#_Toc482874954)

[7.2 Bibliographie 23](#_Toc482874955)

[7.3 Webographie 23](#_Toc482874956)

[8 Annexes 23](#_Toc482874957)

# Spécifications

## Titre

Réalisation d’un quiz de type 4 images 1 mot en C#

## Description

Ce projet est réalisé dans le cadre des TPI de la session 2017 de l’ETML. Il consiste à créer une application de type « 4 images 1 mot » en c#, qui permettra aux enseignants de l’ETML de soutenir le processus d’apprentissage des élèves dans la matière qu’ils enseignent. Il s’agit d’une application simple à utiliser. À l’ouverture du programme, l’utilisateur aura le choix entre différents thèmes. Lorsqu’il aura choisi, l’application va prendre au hasard dans la base de données 1 mot en rapport avec le thème suivi de ses 4 images. Les enseignants auront la possibilité d’ajouter leurs propres images avec leurs propres mots afin d’enrichir la base de données.

## Si le temps le permet

Si le temps le permet, j’ajouterais :

* Une fonction pour compter les points
* Une fonction pour compter le temps
* Un nouveau formulaire pour pouvoir ajouter des thèmes

## Matériel et logiciels à disposition

1 ordinateur standard ETML, avec la structure habituelle

Visual studio 2015, SQL Server 2014, NotePad++

## Prérequis

Avoir suivi les modules ICH à l’etml, les projets et effectué des stages… Les modules ICH suivants sont les plus impliqués : 100, 104, 105, 303 et 306

## Cahier des charges

Le document fourni par le chef de projet fait foi. Il doit être mis en annexe 🡺 Lien sur CDC

# Planification Initiale







# Analyse

## Opportunités

### Difficultés potentielles :

1. Connexion à la base de données depuis C#
2. Récupérer les images stockées dans un répertoire dont le chemin se trouve dans la base de données
3. Organisation des requêtes SQL dans le code C#
4. Distinguer les fonctionnalités en tant qu’enseignant de celle en tant qu’élève

## Document d’analyse et conception

### Maquette interface graphique



La page d’accueil ressemblera à cela. Elle propose certains thèmes que l’utilisateur pourra choisir. En bas à gauche, nous avons une icône de « paramétrage » qui va enfaite permettre aux enseignants d’ajouter un nouveau mot avec des nouvelles images.

**Interface de jeu :**



Nous arrivons à la principale fonctionnalité de l’application. Après avoir choisi un thème, l’utilisateur arrivera sur cette page. Nous avons en haut le thème choisi puis en dessous les 4 images en rapport avec le mot à trouver. Après ces images, le programme propose des lettres disposées aléatoirement. Cette liste de lettres contient les caractères pour former le mot à trouver. Une fois que l’utilisateur à trouver le mot, il peut l’écrire dans la case « Réponse ? ».

**Interface nouveau mot :**



La dernière interface graphique est la page qui permet à l’enseignant d’ajouter un nouveau mot. Il s’agit d’une interface très simple et n’importe quel utilisateur peut l’utiliser. Nous avons en premier temps une case qui va permettre à l’utilisateur d’insérer le nouveau mot. Ensuite, une liste proposera les thèmes inscrit dans la base de données et l’utilisateur devra choisir à quel thème appartient son mot. La dernière étape est de sélectionner les images. L’utilisateur est obligé d’ajouter 4 images. Une fois que l’utilisateur a bien choisi ses images dans ses propres répertoires, le programme va lui alors faire une copie dans un répertoire spécifique qui contiendra toutes les images de l’application. Lors du clique sur le bouton « Ajouter », les vérifications vont être effectuées. Tout d’abord de s’assurer que le mot ne contient que des lettres puis de vérifier que l’utilisateur à bien sélectionné 4 images accessible au programme.

### Analyse base de données

Pour mon projet TPI, il m’a été imposé de faire une base de données avec SQL server 2014. Celle-ci a été installé sur une machine virtuelle. Après une réflexion et discussion avec le chef de projet, j’ai conçu un modèle conceptuel, un modèle logique puis un modèle physique des données.

### MCD

J’ai nommé les entités comme des tables car j’étais embêté par la suite à cause du programme « JMerise ». Le programme permet de mettre les noms seulement dans le MCD, ensuite il ne nous permet plus de les modifier pour le MLD. C’est pour cela que les noms sont déjà des noms de tables.

Il s’agit d’une base de données assez simple. Nous avons 3 entités différentes qui se relient. La première est pour les images. Elle va contenir une propriété « liens » des images utilisées dans l’application. Dans cette entité nous stockeront les liens des images (lien dans le répertoire spécifique d’image). Nous voyons la relation avec l’entité des mots. Le « 0, n » signifie qu’une image peut au minimum contenir aucun mot et pas de restriction pour le maximum. En résumé, une image peut être insérer sans aucun mot dans la base de données mais ne sera pas utilisée. Elle pourra aussi décrire plusieurs mots. La deuxième entité est celle des mots. Elle contient une propriété simple de type chaine de caractère qui contiendra le mot à trouver. Cette entité a un lien avec celle que nous avons vu précédemment dont la cardinalité est « 0, n », c’est-à-dire qu’un mot peut avoir au minimum 0 image et pas de maximum. Dans notre cas le mot aura 4 images mais il s’agit d’une logique nous permettant de faire le MLD que nous verrons après. Cette entité possède un autre lien avec la suivante qui est celle pour les thèmes. Nous retrouvons une cardinalité de « 0,n ». Elle signifie qu’un mot peut n’appartenir à aucun thème ou à plusieurs thèmes. Pour finir, l’entité des thèmes qui est la dernière possède une propriété de chaine de caractère ou seront noter tous les thèmes possible. Cette dernière a un lien avec l’entité des mots et la cardinalité est de « 0, n ». Cela veut dire que un thème peut être crée sans aucun mot ou appartenir à un ou plusieurs mots.

### MLD

Nous avons maintenant le modèle logique des données. Suite aux règles de passage du MCD au MLD, nous avons maintenant des tables et non des entités et nous remplaçons les propriétés par des attributs. Les cardinalités ont été supprimées et nous avons maintenant deux nouvelles tables qui sont des tables qui ont été établies grâce aux règles de cardinalités. Vu que chaque table contenait une relation de « x, n », j’ai alors dû créé des tables pour faire le lien. Ces nouvelles tables ont pour attributs les clés primaires des autres tables.

### MPD

Nous retrouvons les 5 tables vu précédemment mais sous formes de tableaux d'intégrités. Leurs attributs sont plus détaillés dans le modèle physique des données ci-dessous. Pour chaque table nous avons son type et sa taille, la valeur par défaut, les contraintes, si l’attribut est indexé ou non, si on autorise le nul et autre.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T\_Image | | | | | | |
| Attribut | Type + Taille | Valeur par défaut | contrainte | Indexe | Nul interdit | Divers |
| id\_image | Int - 100 | - | Clé primaire | Non | Oui | NumAuto |
| imaLien | Varchar - 100 | - | - | Non | Oui | Unique |

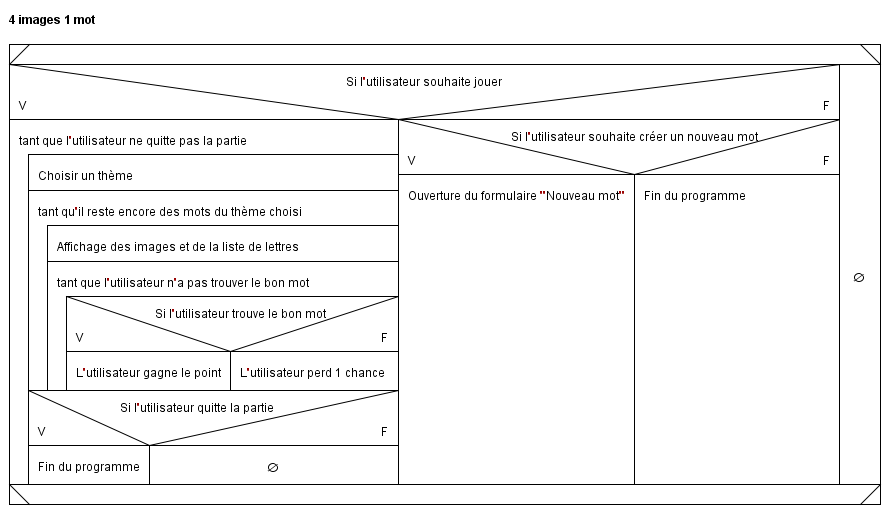
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T\_Decrire | | | | | | |
| Attribut | Type + Taille | Valeur par défaut | contrainte | Indexe | Nul interdit | Divers |
| idimag | Int - 100 | - | Clé étrangère | Non | Oui | NumAuto |
| idmot | Int - 25 | - | Clé étrangère | Non | Oui | NumAuto |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T\_Mot | | | | | | |
| Attribut | Type + Taille | Valeur par défaut | contrainte | Indexe | Nul interdit | Divers |
| id\_mot | Int - 25 | - | Clé primaire | Non | Oui | NumAuto |
| motMot | Varchar - 25 | - | - | Non | Oui | Unique |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T\_appartenir | | | | | | |
| Attribut | Type + Taille | Valeur par défaut | contrainte | Indexe | Nul interdit | Divers |
| idmot | Int - 25 | - | Clé étrangère | Non | Oui | NumAuto |
| idtheme | Int - 25 | - | Clé étrangère | Non | Oui | NumAuto |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| T\_Theme | | | | | | |
| Attribut | Type + Taille | Valeur par défaut | contrainte | Indexe | Nul interdit | Divers |
| id\_theme | Int - 25 | - | Clé primaire | Non | Oui | NumAuto |
| theTheme | Varchar - 25 | - | - | Non | Oui | Unique |

### Analyse programme

Après avoir lu le cahier des charges et avoir fait une petite analyse sur papier, j’ai créé un structogramme pour expliquer le fonctionnement du programme plus facilement. Il explique étape par étape comment le programme va réagir.

Nous apercevons qu’il y a deux principales fonctionnalités dans le programme. La première est de choisir un thème puis de découvrir le mot qui se cache. La seconde est réservée aux enseignants et sera bloqué par un code d’accès. Elle consiste à pouvoir ajouter un mot avec ses 4 images dans la base de données.

Ce paragraphe décrit le fonctionnement de manière détaillée.

Autant que possible de manière graphique, imagée, tableaux, etc.

Tous les cas particuliers devraient y être spécifiés…

S’il y a des fonctionnalités à développer :

Découpage en étapes, en modules, en fonctionnalités, etc.

Formulaires, interfaces graphiques, pages web, etc.

Schémas de navigation, schémas événementiels, structogramme, pseudocode, etc.

Si le projet inclut une base de données :

Dictionnaire des données

Modèle conceptuel des données, modèles logique des données.

Si le projet inclut implémentation système/réseau :

Schéma réseau, plan d’adressage, etc.

Descriptif installation, guide de mise en œuvre, etc.

## Conception des tests

Les tests de l’application se feront pendant le projet et après chaque nouvelle fonctionnalité ajoutée. Il s’agira de petits tests pour voir si le programme fonctionne bien. Vers la fin du projet, j’effectuerais des tests un peu plus spécifiques qui auront pour but de tester le programme et de chercher s’il y a des erreurs à corriger.

Les tests généraux du programme à la fin du projet seront ceux-ci :

|  |  |
| --- | --- |
| Jeu |  |
| Case « Réponse » | Mettre des mauvaises réponses  Mettre des chiffres, caractères spéciaux  Ne rien mettre |

|  |  |
| --- | --- |
| Nouveau mot |  |
| Case « Mot » | Mettre des chiffres, caractères spéciaux  Ne rien mettre  Mettre un long mot de plus de 20 caractères  Mettre un seul caractère |
|  |  |
| Ajouter images | Ajouter un document « word, pdf… » |
|  | Ne pas mettre d’images  Mettre seulement 3 images |

Ce paragraphe permet de spécifier la stratégie de test qui sera menée au point 5.1

Qui, quand, avec quelles données, dans quel ordre, etc.

Avec quels matériels, quels services, etc.

## Planification détaillée

A ce stade, après l’analyse complète du projet, un planning détaillé et complet (avec tâches, sous-tâches, dépendances, durée, …) peut être finalisé.

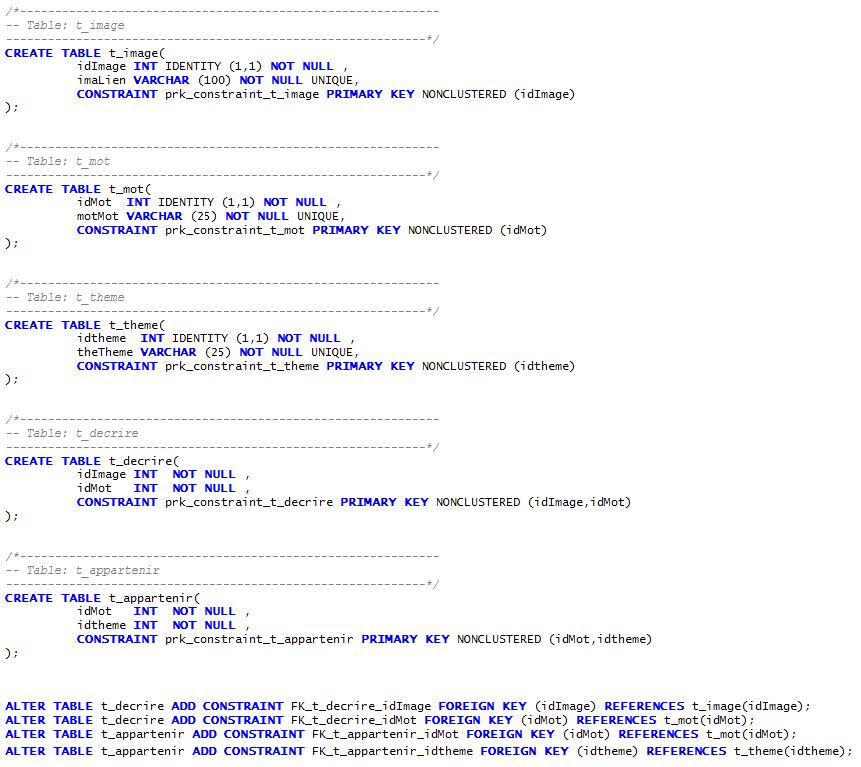
Le planning détaillé doit s’inscrire dans le planning initial. Il faut que l’on puisse situer cette planification détaillée par rapport à la planification initiale.

# Réalisation

## Dossier de Réalisation

### Base de données

Après avoir créé le MCD, MLD avec le programme JMerise, celui-ci nous permet de générer le script pour pouvoir créer la base de données très facilement. Il nous propose de choisir la plateforme utilisée et produit le code en fonction du langage. Pour ma part, j’ai demandé à cette application de me générer le script pour un SQL Server et voilà le résultat :



Il s’agit d’un script très simple et assez clair. Tout de même j’ai dû faire quelques petites recherches sur internet avant d’exécuter ce code pour bien comprendre le fonctionnement et la création de la base de données.

En effet, je ne comprenais pas la propriété « IDENTITY (1,1) » qui se trouve après chaque clé primaire. Il s’agit enfaite tout simplement de définir que l’attribut s’incrémente tout seul. L’équivalent d’un « AUTO\_INCREMENT » pour MYSQL. Par contre elle a une fonction en plus qui permet de définir à quel chiffre commence l’incrémentation et de combien va elle augmenter.

Exemple :

CREAT TABLE t\_exemple(

idExemple INT IDENTITY (5,10) NOT NULL

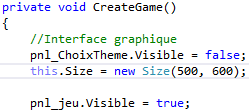
) ;

Dans ce cas, le premier exemple qui sera créé commencera par l’identifiant numéro 5 et le deuxième sera le numéro 15. Car l’incrémentation se fait de 10 en 10.

### Interface graphique

Pour l’interface graphique, j’ai décidé de faire seulement deux formulaires séparés. Le premier est celui du menu d’accueil et du jeu, le deuxième est celui de l’ajout d’un nouveau mot. L’interface de l’accueil a été faite directement grâce à l’espace de travail « Design » de Visual studio. Cet espace de travail permet de crée visuellement l’application, d’ajouter nos boutons ou d’autres contrôles en faisant un « drag and drop » depuis la « toolbox ». C’est un outil très utile qui nous permet de faire des interfaces simples assez rapidement. Les boutons et autres contrôles de la page d’accueil ont tous étés mis dans un « panel ». Il s’agit d’un conteneur qui nous permettra par la suite de l’afficher ou le faire disparaître. L’interface de jeu a par contre elle était faite par code et aussi dans un panel. J’ai trouvé mieux de faire comme ça car on garde tous sur une fenêtre et nous n’avons pas besoin d’avoir plusieurs fenêtres inutiles. Pour mieux expliquer, lorsque l’application se lance nous arrivons directement sur l’accueil et donc au choix du thème. Après avoir choisi le thème, je rends le panel invisible et fait apparaître le deuxième qui lui a été créé par code. Il s’agit donc de la même fenêtre c’est juste le contenu qui va changer.

Pour la partie qui permet d’ajouter un nouveau mot j’ai préféré faire un nouveau formulaire car je trouve que c’est une autre fonctionnalité qui doit être indépendante et dont tout le monde n’aura pas l’accès. Cette interface a donc aussi été faite avec l’espace de travail « Design ». Lorsque l’enseignant qui aura les accès clique sur le bouton pour ajouter un nouveau mot. Le nouveau formulaire s’ouvrira et il pourra alors ajouter son mot.



Dans ce petit bout de code on peut voir le début de la fonction « CreateGame » qui va créer l’interface de jeu. La première opération qui se fait c’est celle que nous avons vu plus haut. La fonction va cacher le premier panel qui est « pnl\_ChoixTheme ».

Ensuite, le programme agrandi le formulaire pour pouvoir afficher l’interface de jeu qui prend plus de place. Nous pouvons voir que l’interface de jeu sera de 500 pixels de largeur et 600 de hauteur.

Après avoir fait tout ça, le programme va rendre le deuxième panel visible pour que le contenu du formulaire soit le jeu.

### Connexion C# --> SQL server

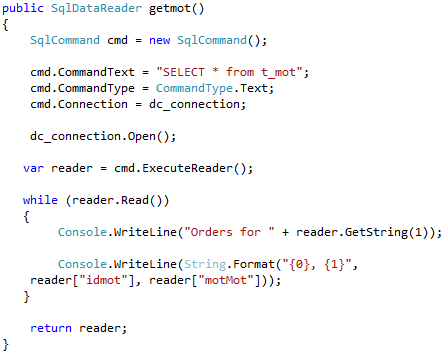
J’ai décidé de faire la connexion entre l’application C# et le serveur SQL avec l’espace de noms « System.Data ». Cet espace de noms permet de travailler avec différentes classes qui vont nous permettre de gérer efficacement les données provenant de notre serveur SQL. La classe qui va justement nous permettre de faire la connexion entre ces derniers est la classe « sqlconnection ». Pour cela, il faut d’abord initialiser une nouvelle connexion puis lui passer en paramètre la « connectionString ». Celle-ci sert à indiquer au programme le chemin et la syntaxe pour aller se connecter à la base de données. Exemple :

SqlConnection sqlConnection1 = new SqlConnection("Data Source=(localdb)\\MSSQLLocalDB;Initial Catalog=4images1mot;Connect Timeout=30;ApplicationIntent=ReadWrite");

Le nom de la nouvelle connexion est « sqlConnection1 » et la connectionString est en rouge. Elle donne la source de la base de données qui est en local. Le nom de la base de donnée qui est « 4images1mot » et d’autres informations.

### Exécution des requêtes SQL

J’ai créé une classe spécifique pour les requêtes SQL qui s’appelle « dbConnexion ». J’ai trouvé que c’était mieux de séparer le code en C# qui va s’occuper de l’application elle-même et le code qui va interagir avec la base de données. Le but était de simplifier le code (le rendre plus compréhensible). Cela me permettra plus tard d’appeler mes requêtes SQL depuis mon code en utilisant seulement les noms des fonctions et sans avoir besoins de mettre du SQL. Exemple :



Nous pouvons voir qu’il s’agit d’une fonction « getmot ». Elle permet de récupérer tous de la table « t\_mot ». C’est une fonction test pour voir si la connexion à la base de données à bien fonctionner et voir si l’application arrive à récupérer des données. Nous voyons la classe « SqlCommand » qui va nous permettre de créer une nouvelle commande SQL. Après avoir créé cette variable il faut lui passer les paramètres. Le premier est de lui donner la requête SQL. Ceci se fait directement entre guillemet. Dans ce cas de test, il s’agit d’une requête simple qui est « SELECT \* FROM t\_mot ». La commande s’exécute avec la fonction « ExecuteReader() » et je récupère le tout dans une variable que j’appelle « reader ». Après avoir récupérer ces données dans une variable, il est encore impossible de les lire. C’est pour cela qu’il faut utiliser la fonction « read » qui va lire les données et les traduire.

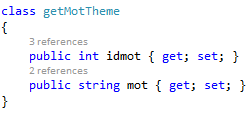
L’appelle de cette fonction que nous avons vue est toute simple. Depuis la classe où se trouve le code de l’application il suffit d’initialiser la classe que nous voyons de voir et d’appeler la fonction de connexion à la base de données puis la requête SQL. Exemple :



L’initialisation de la base de données s’appelle « fm\_connexion » et nous appelons justement la fonction « getmot ».

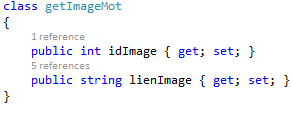
### Récupération des images du mot en cours

La récupération des images avec le bon mot était, je pense, la plus grande étape de l’application. Tout d’abord j’ai commencé à écrire plusieurs requêtes pour récupérer les données dans des listes. Je ne pouvais pas récupérer plusieurs informations comme par exemple « l’ID avec le lien de l’image » mais je devais récupérer l’id dans une variable puis le lien dans une autre. Ce qui me faisait deux fonctions et deux requêtes différentes. J’ai donc crée deux classe qui contiennent chacune une liste d’objet. La première liste d’objet est « getMotTheme ». Celle-ci va permettre de récupérer l’id du mot et le mot lui-même. Tous les mots du thème choisi seront dans cette liste d’objet.

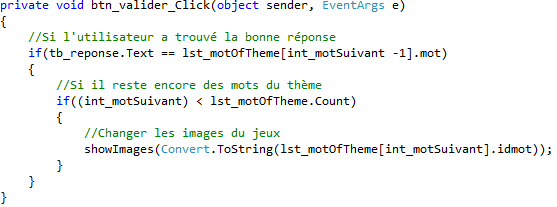


Nous pouvons voir qu’il s’agit d’une classe très simple. L’objet contient les propriétés suivantes : idmot qui est de type « int » et mot de type « string ».

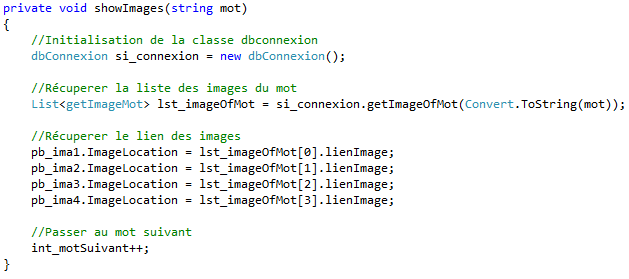
Après avoir récupérer tous les mots du thème qui nous intéresse, nous allons nous occuper de récupérer les images pour chaque mot. C’est ici que nous allons faire appel à la deuxième classe qui est « getImageMot ».

Encore une fois, nous apercevons que la classe est simple. Nous avons en premier la propriété idImage qui est de type « int » et le lien de l’image qui est de type « string ».

Après avoir choisi le thème, l’application va donc chercher tous les mots qui sont dans le thème puis aller chercher les images du premier mot et les afficher. Une fois que le jeu est en place, le programme attend que l’utilisateur clique sur le bouton « valider » pour donner sa réponse. Le programme va vérifier si le mot et correct ou pas et s’il est juste, il va appeler la fonction pour récupérer les images en lui donnant comme paramètre le deuxième mot.



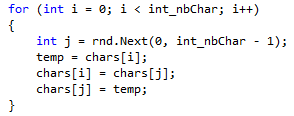
Ci-dessus nous avons la fonction de validation. La première étape est de vérifier si le mot est égal au mot en cours. Le programme regarde ensuite s’il reste encore des mots du thème choisi, si oui il appel la fonction qui gère les images « showImages » que nous pouvons voir ci-dessous.



La fonction « showImages » va nous permettre de récupérer les images puis de les afficher. Premièrement elle initialise la classe qui contient les requêtes « dbConnexion » en « si\_connexion ». Ensuite, elle va appeler la fonction qui se charge d’exécuter la requête pour retourner les images du mot. En paramètre, nous donnons le mot en cours. Après avoir récupérer ces images, nous allons les afficher dans les pictureBox dans l’interface graphique du programme.

### Affichage des lettres aléatoire

L’affichage des lettres aléatoire, du mot, permet d’aider l’utilisateur à trouver le mot. Cette fonction a été plus simple que prévue et j’ai passé moins de temps que prévu dans la planification pour cette partie. Le mot a été découpé et toutes les lettres du mot ont été placé dans la table (Chars). Ce qui va nous permettre de compter le nombre de lettres puis de les mélanger. Voici la fonction :



Il s’agit en effet d’une boucle for qui va s’exécuter au nombres de lettres que contient le tableau (chars). Le programme va ensuite choisir un numéro aléatoire qui évidemment sera un chiffre entre 0 et le nombre de lettres. Après avoir choisi ce chiffre, la fonction copie la lettre se trouvant dans l’index du chiffre aléatoire pour la mettre dans la première case du tableau, mais avant ça, elle copie la lettre qui sera remplacé pour ne pas la perdre et la met à la place de celle qui la remplace.

Cette partie permet de reproduire ou reprendre le projet par un tiers.

Pour chaque étape, il faut décrire sa mise en œuvre. Typiquement :

Versions des outils logiciels utilisés (OS, applications, pilotes, librairies, etc.)

Configurations spéciales des outils (Equipements, PC, machines, outillage, etc.)

Code source commenté des éléments logiciels développés.

Modèle physique d’une base de données.

Arborescences des documents produits.

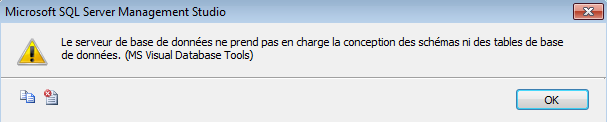
Schémas, plans d’adressages, plan de nommage, etc.

Il faut décrire le parcours de réalisation et justifier les choix.

## Modifications

### Changement Microsoft SQL Server Management Studio

Après avoir installé mon serveur SQL 2014, j’ai créé une nouvelle base de données grâce au script de JMerise. La base de données a très bien fonctionné et j’arrivais bien à récupérer des données depuis l’application C#. Le problème était que je n’arrivais pas à ouvrir mes tables en mode création et à utiliser quelques autres fonctionnalités depuis le SQL Server Management Studio. Voici le message d’erreur :



Suite à quelques recherches, j’ai vu qu’il s’agissait d’un problème de version. J’ai donc essayer d’installer la dernière version du management studio qui est la 2016 et après avoir installé celle-ci, j’ai enfin réussi à manipuler mes tables et base de données comme je le voulais avec toutes les fonctionnalités que propose Microsoft.

Historique des modifications demandées (ou nécessaires) aux spécifications détaillées.

Date, raison, description, etc.

# Tests

## Dossier des tests

On dresse le bilan des tests effectués (qui, quand, avec quelles données…) sous forme de procédure. Lorsque cela est possible, fournir un tableau des tests effectués avec les résultats attendus et obtenus, ainsi que les actions à entreprendre en conséquence (et une estimation de leur durée).

Si des tests prévus dans la stratégie n'ont pas pu être effectués :

raison, décisions, etc.

Liste des bugs répertoriés avec la date de découverte et leur état:

Corrigé, date de correction, corrigé par, etc.

# Conclusion

## Bilan des fonctionnalités demandées

Il s’agit de reprendre point par point les fonctionnalités décrites dans les spécifications de départ et de définir si elles sont atteintes ou pas, et pourquoi.

Si ce n’est pas le cas, estimer en « % » ou en « temps supplémentaire » le travail qu’il reste à accomplir pour terminer le tout.

## Bilan de la planification

Distinguer et expliquer les tâches qui ont généré des retards ou de l'avance dans la gestion du projet. Indiquer les différences entre les planifications initiales et détaillées avec le journal de travail.

## Bilan personnel

Si c’était à refaire:

Qu’est-ce qu’il faudrait garder ? Les plus et les moins ?

Qu’est-ce qu’il faudrait gérer, réaliser ou traiter différemment ?

Qu’est que ce projet m’a appris ?

Suite à donner, améliorations souhaitables, …

Remerciements, signature, etc.

# Divers

## Journal de travail

Date, activité (description qui permet de reproduire le cheminement du projet), durée, liens et références sur des documents externes. Lorsqu’une activité de recherches a été entreprise, il convient d’énumérer ce qui a été trouvé, avec les références.

## Bibliographie

Références des livres, revues et publications utilisés durant le projet.

## Webographie

Références des sites Internet consultés durant le projet.

# Annexes

Cahier des charges

Listing du code source (partiel ou, plus rarement complet)

Guide(s) d’utilisation et/ou guide de l’administrateur

Etat ou « dump » de la configuration des équipements (routeur, switch, robot, etc.).

Extraits de catalogue, documentation de fabricant, etc.