Langage C# « La planche d’équilibre »



|  |  |
| --- | --- |
| Auteur | Dwenn Kaufmann |
| Date de début de projet | 24/04/2013 |
| Date de fin de projet | 28/05/2013 |
| Date de reddition du rapport | 28/05/2013 |
| Expert principal | Roberto Amico |
| Expert secondaire | Alain Jeanmaire |
| Version | 1.0 |

Table des matières

[1 Introduction 4](#_Toc357434283)

[2 Documentation de développement 4](#_Toc357434284)

[2.1 Explications détaillées du projet 4](#_Toc357434285)

[2.2 Diagramme des cas d'utilisation 5](#_Toc357434286)

[2.3 Définition des conventions applicables 5](#_Toc357434287)

[2.4 Planning de livraison global 7](#_Toc357434288)

[3 Réalisation des cas d'utilisation 8](#_Toc357434289)

[3.1 Cas d'utilisation «Login » 9](#_Toc357434290)

[3.1.1 Scénario 9](#_Toc357434291)

[3.1.2 Maquettes 9](#_Toc357434292)

[3.1.3 Analyse du scénario 9](#_Toc357434293)

[3.1.3.1 Algorithme ou Structogramme 9](#_Toc357434294)

[3.1.3.2 Explications détaillées 9](#_Toc357434295)

[3.1.4 La phase de programmation 10](#_Toc357434296)

[3.1.5 La phase de tests 10](#_Toc357434297)

[3.2 Cas d'utilisation « Paramètres » 10](#_Toc357434298)

[3.2.1 Scénario 10](#_Toc357434299)

[3.2.2 Maquettes 11](#_Toc357434300)

[3.2.3 Analyse du scénario 11](#_Toc357434301)

[3.2.3.1 Algorithme ou Structogramme 11](#_Toc357434302)

[3.2.3.2 Explications détaillées 11](#_Toc357434303)

[3.2.4 La phase de programmation 12](#_Toc357434304)

[3.2.5 La phase de tests 12](#_Toc357434305)

[3.3 Cas d'utilisation « Connecter la carte » 12](#_Toc357434306)

[3.3.1 Scénario 13](#_Toc357434307)

[3.3.2 Maquettes 13](#_Toc357434308)

[3.3.3 Analyse du scénario 13](#_Toc357434309)

[3.3.3.1 Algorithme ou Structogramme 13](#_Toc357434310)

[3.3.3.2 Explications détaillées 13](#_Toc357434311)

[3.3.4 La phase de programmation 13](#_Toc357434312)

[3.3.5 La phase de tests 14](#_Toc357434313)

[3.4 Cas d'utilisation « Ejecter la carte » 14](#_Toc357434314)

[3.4.1 Scénario 14](#_Toc357434315)

[3.4.2 Maquettes 14](#_Toc357434316)

[3.4.3 Analyse du scénario 14](#_Toc357434317)

[3.4.3.1 Algorithme ou Structogramme 14](#_Toc357434318)

[3.4.3.2 Explications détaillées 14](#_Toc357434319)

[3.4.4 La phase de programmation 15](#_Toc357434320)

[3.4.5 La phase de tests 15](#_Toc357434321)

[3.5 Cas d'utilisation « Changer d’utilisateur » 15](#_Toc357434322)

[3.5.1 Scénario 15](#_Toc357434323)

[3.5.2 Maquettes 16](#_Toc357434324)

[3.5.3 Analyse du scénario 16](#_Toc357434325)

[3.5.3.1 Algorithme ou Structogramme 16](#_Toc357434326)

[3.5.3.2 Explications détaillées 16](#_Toc357434327)

[3.5.4 La phase de programmation 16](#_Toc357434328)

[3.5.5 La phase de tests 16](#_Toc357434329)

[3.6 Cas d'utilisation « Tableau des scores » 16](#_Toc357434330)

[3.6.1 Scénario 17](#_Toc357434331)

[3.6.2 Maquettes 17](#_Toc357434332)

[3.6.3 Analyse du scénario 17](#_Toc357434333)

[3.6.3.1 Algorithme ou Structogramme 17](#_Toc357434334)

[3.6.3.2 Explications détaillées 17](#_Toc357434335)

[3.6.4 La phase de programmation 18](#_Toc357434336)

[3.6.5 La phase de tests 18](#_Toc357434337)

[4 Mode d’emplois utilisateur 18](#_Toc357434338)

[4.1 Installation de l’application 18](#_Toc357434339)

[4.2 Login 20](#_Toc357434340)

[4.3 Connecter / Déconnecter la carte 21](#_Toc357434341)

[4.4 Paramétrer l’application 21](#_Toc357434342)

[4.5 Changer d’utilisateur 22](#_Toc357434343)

[4.6 Affichage des scores 22](#_Toc357434344)

[4.6.1 Scores par utilisateur 22](#_Toc357434345)

[4.6.2 Scores par temps 23](#_Toc357434346)

[4.6.3 Scores par difficulté 23](#_Toc357434347)

[4.7 Vider les scores 23](#_Toc357434348)

[4.7.1 Vider les scores de l’utilisateur en cours uniquement 23](#_Toc357434349)

[4.7.2 Vider les scores de tous les utilisateurs 23](#_Toc357434350)

[4.8 Imprimer les scores 24](#_Toc357434351)

[4.9 Désinstallation de l’application 24](#_Toc357434352)

[5 Conclusions 26](#_Toc357434353)

[6 Annexes 26](#_Toc357434354)

[6.1 Journal de bord 26](#_Toc357434355)

[6.2 Schéma Synoptique 26](#_Toc357434356)

[6.2.1 Login, connexion de la carte et paramétrage 28](#_Toc357434357)

[6.2.2 Affichage des scores 29](#_Toc357434358)

[6.2.3 Fenêtre de jeu 30](#_Toc357434359)

[6.3 Cahier des charges 30](#_Toc357434360)

[6.4 Code source 35](#_Toc357434361)

[6.5 Références 35](#_Toc357434362)

# Introduction

Le but de ce travail est de proposer à l’utilisateur de tester son habileté à rester en équilibre sur une planche en bois rendue instable par la pose d’un demi-cylindre en son centre.

Le TIP étant un travail personnel, il doit impérativement être réalisé seul et uniquement en classe dans le cadre du CPLN.

Le langage de programmation imposé pour réaliser ce projet est le C#, langage que l’on apprend depuis 3 ans maintenant ici au CPLN.

Cette planche en bois est elle-même reliée à une carte électronique, de modèle Velleman K8055, qui sera reliée à l’ordinateur via USB.

Des capteurs, de type contact « ouverts », sont disposés de chaque côté de la planche, afin de déterminer lorsque l’utilisateur a perdu l’équilibre, et donc touche le sol avec un des deux capteurs.

Lorsque l’utilisateur décide de lancer une partie en cliquant sur le bouton « Jouer », il dispose alors d’un temps imparti où il devra faire le meilleur score, en gardant son équilibre le mieux possible.

# Documentation de développement

## Explications détaillées du projet

Le programme doit en premier afficher une fenêtre de connexion, obligeant l’utilisateur à entrer ses informations, qui sont : son Nom, son Prénom, ainsi que sa date d’anniversaire. Ces paramètres seront sauvegardés dans une base de données de type Access 2010. Si l’utilisateur a déjà, par le passé, entré ses informations, il ne sera pas enregistré dans la base de données à nouveau.

Une fois la connexion effectuée, l’utilisateur arrive sur la fenêtre de jeu. Avant de pouvoir jouer, il va d’abord devoir cliquer sur le bouton permettant d’effectuer la connexion avec la carte Velleman reliée par USB à l’ordinateur.

Si la carte est connectée à l’ordinateur, la partie va se lancer suivant le temps, la difficulté et les temps d’attente entre deux touchés, paramétrés depuis la fenêtre accessible en cliquant dans le menu sur le bouton « Paramètres ».

A tout moment, l’utilisateur peut changer d’utilisateur, ce qui le ramènera au même stade qu’à l’ouverture du programme, le forçant à s’authentifier à nouveau.

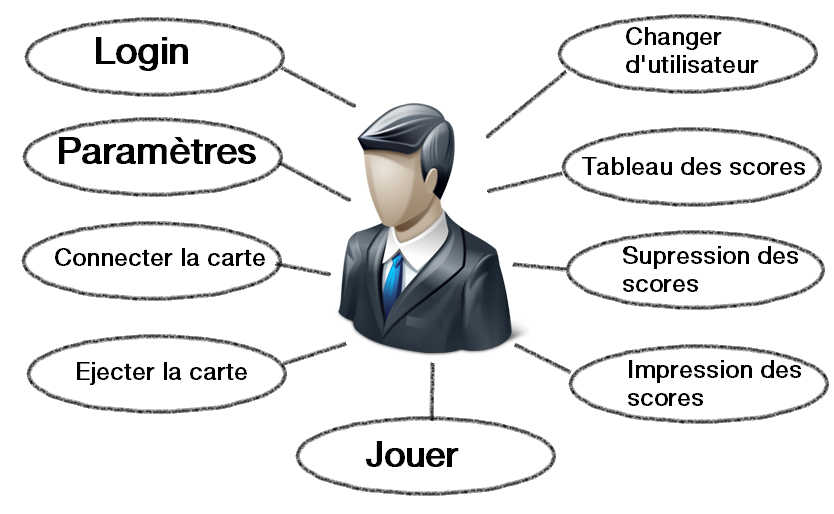
Un bouton permettant de déconnecter la carte Velleman est présent, et s’il est cliqué lorsqu’une partie est en cours, elle sera considérée comme terminée.

Le tableau des scores est accessible en cliquant sur le bouton du même nom, affichant un formulaire présentant un classement des scores triées par temps de partie, difficulté, et par nom d’utilisateur (Soit l’utilisateur en cours, soit tout le monde). L’impression des résultats est possible en cliquant sur le bouton « Imprimer » présent dans ce formulaire. Il est également possible de vider les scores de l’utilisateur en cours, ou de tous les utilisateurs.

Pendant la partie, une barre de progression indique le temps écoulé sur le temps total. Le nombre de touchés gauches et droits que l’utilisateur a fait sont affichés sous forme de texte, et des images fléchées se colorent en bleu pour signaler de quel côté le connecteur fait contact lorsque l’utilisateur perds son équilibre sur la planche.

Lorsque le temps d’une partie est écoulé, un formulaire affichant les résultats de la partie s’affiche, incluant : Le nombre de touchés totaux, le nombre de touchers gauches et droits, le temps de la partie, la difficulté. L’utilisateur est invité à cliquer sur le bouton fermant ce formulaire, et ainsi à revenir sur le formulaire de jeu.

## Diagramme des cas d'utilisation



## Définition des conventions applicables

Chaque variable typée aura un préfixe permettant de reconnaître son type plus simplement, comme appris ici au CPLN.

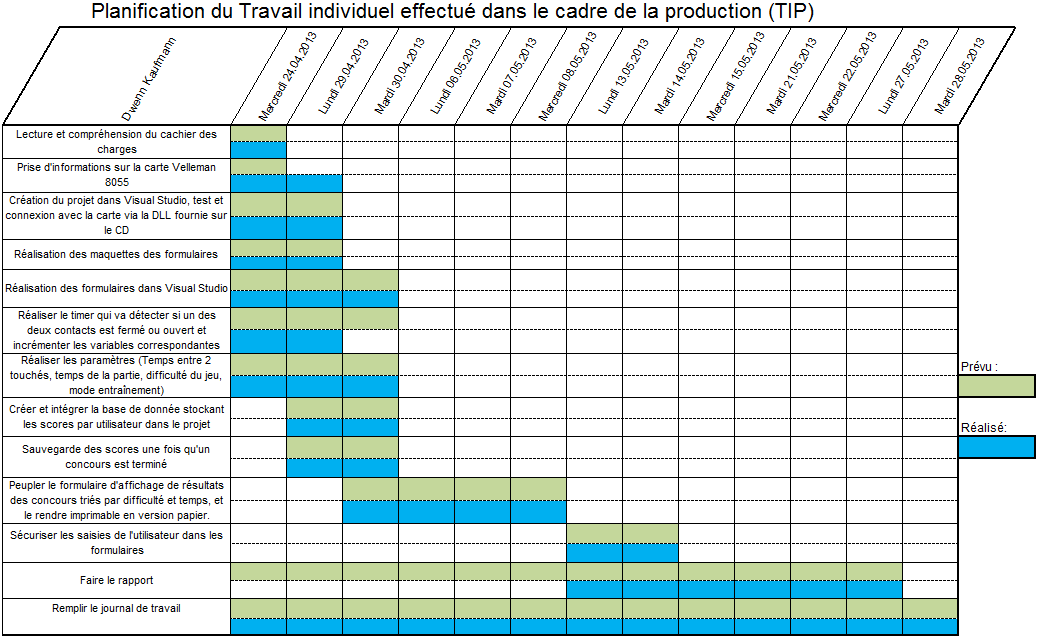
* Les variables de type Entier auront le préfixe « i »
* Les variables de type Double auront le préfixe « dbl »
* Les variables de type Texte auront le préfixe « str »
* Les variables de type Booléen auront le préfixe « b »

Pour ce qui est du nom après le préfixe, un nom le plus représentatif de l’utilité de la variable sera choisi, afin de faciliter la compréhension.

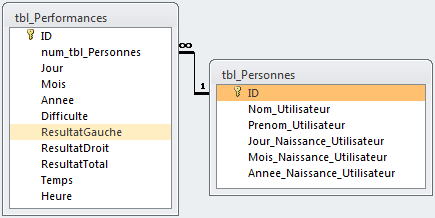
Chaque nom de variable utilise une convention appelée « camelCase », qui va consister à séparer chaque « partie » du mot par des majuscules, tout en gardant la première lettre de la variable en minuscule, pour améliorer la lisibilité et la compréhension.

Ainsi, une méthode servant à créer des camions s’appellerait donc « creationCamion ».

## Planning de livraison global



## Modèle logique de la base de données



## D:\Desktop\TPI Dwenn Kaufmann\Documentation\Diagramme de classes.pngDiagramme de classes

# Réalisation des cas d'utilisation

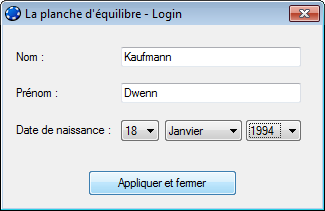
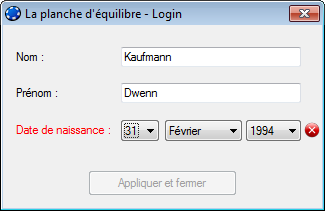
## Cas d'utilisation «Login »

La première action que va avoir à faire l’utilisateur lors de l’ouverture de l’application sera s’authentifier. Si la combinaison du nom de famille, du prénom et de la date de naissance n’a jamais été saisie, l’utilisateur sera stocké dans la base de données liée au programme.

### Scénario

1. L’utilisateur rentre son nom de famille.
2. L’utilisateur rentre son prénom.
3. L’utilisateur rentre sa date de naissance.
4. L’utilisateur appuie sur le bouton « Appliquer et fermer ».
5. Les données sont testées à la recherche d’éventuelles erreurs ou de champs vides.
6. Si la combinaison de toutes les données entrées n’est pas déjà présente dans la base de données, l’utilisateur est sauvegardé, un MessageBox l’en informe.
7. Les données de l’utilisateur sont également sauvegardées dans le fichier Params.settings pour une utilisation ultérieure.
8. Le formulaire Login.cs est caché.
9. Le formulaire Jeu.cs est affiché.

### Maquettes



### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

A l’ouverture du formulaire Login.cs, la ComboBox « cbxAnnee » est peuplée via une boucle, afin que les valeurs contenues dans la liste restent à jour, peu importe l’année d’ouverture du programme. Les années affichées commencent depuis 1940 jusqu’à l’année actuelle stockée sur l’ordinateur.

Dans les TextBox servant à renseigner le Nom ainsi que le prénom, seuls les lettres, les espaces et les tirets sont acceptés. On ne peut pas copier/coller du texte à l’intérieur.

Une fois le jour, le mois et la date de naissance renseignés, l’utilisateur pourra constater, à chaque fois qu’il modifie une de ces 3 informations, qu’elles sont contrôlées afin de déterminer si elles sont valides ou non. Si elles ne le sont pas, par exemple lors d’un 31 février, le bouton permettant d’appliquer et fermer la fenêtre se désactive, une icône d’erreur rouge surgit à côté des ComboBox, et le label « Date de naissance » devient de couleur rouge. Tout ceci disparaît une fois une date d’anniversaire valide saisie.

Si l’utilisateur laisse un champ vide, par exemple son nom, et clique sur le bouton Appliquer et fermer, une fenêtre d’erreur surgit et l’en informe.

Si toutes les données saisies sont correctes, l’utilisateur va être enregistré dans la base de données s’il n’avait jamais entré ses informations auparavant, puis les données seront stockées en tant que paramètres dans le fichier Params.settings pour être utilisées plus tard.

La fenêtre va ensuite se cacher, et le formulaire Jeu.cs va être affiché à l’écran.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « Login.cs »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Un des champs est vide. | Indique qu’aucun champ ne doit être vide. | Ok | Une fenêtre d’erreur surgit. Il faudra remplir les champs afin de continuer. |
| Des caractères spéciaux sont entrés dans un des champs. | Il est impossible d’écrire des caractères spéciaux. | Ok | Si l’on tente d’écrire un caractère spécial, hormis « -», « «’ » ou un espace, il n’est pas écrit |
|  | L’utilisateur tente de copier/coller via un raccourcis clavier ou le menu par le clic droit. | L’action n’est pas possible. | Ok | Rien ne se passe. |
|  | La date d’anniversaire n’est pas correcte. | Impossible de continuer. | Ok | Une icône d’erreur s’affiche, le bouton Appliquer se désactive, le label correspondant devient rouge. |
|  | Tous les champs sont renseignés, et la date d’anniversaire est correcte. | Cache la fenêtre, sauvegarde les valeurs et passe à la fenêtre principale. | Ok | Si l’utilisateur n’était pas déjà enregistré dans la base, il l’est maintenant. |

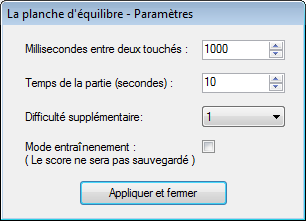
## Cas d'utilisation « Paramètres »

Depuis le formulaire Jeu.cs, un clic sur le bouton tsbParametres va afficher le formulaire Parametres.cs, permettant de modifier la configuration, tel que le temps à attendre avant de comptabiliser un deuxième toucher, le temps total de la partie, la difficulté, ainsi que de déterminer si le mode d’entrainement est activé ou non.

### Scénario

1. La méthode Dispose du timer tmrJeu est appelée pour stopper la partie en cours si elle avait lieu.
2. Le formulaire Jeu.cs est caché.
3. Le formulaire Parametres.cs est affiché.
4. Le ComboBox régissant la difficulté, la checkBox qui permet de choisir le mode entraînement, les numericUpDown permettant de saisir le nombre de millisecondes entre deux touchés ainsi que celui permettant de définir le temps de la partie sont remplis avec les valeurs stockées dans le fichier Params.settings
5. Si l’utilisateur saisit des paramètres jugés correct par l’application et clique sur le bouton btnAppliquer, ils sont sauvegardés dans le fichier Params.settings
6. Le formulaire Parametres.cs se cache
7. Le formulaire Jeu.cs s’affiche, prenant en compte les nouveaux paramètres

### Maquettes



### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Le timer tmrJeu subit la méthode « Dispose », ce qui va libérer toute ressource utilisée par celui-ci et le stopper immédiatement.

La propriété « Visible » du formulaire Jeu.cs passe à false, le formulaire Parametres.cs est déclaré et affiché.

La propriété « Value » du NumericUpDown « tbxMillisecondes » est récupérée depuis la variable iMillisecondes du fichier Params.settings.

La propriété « Value » du NumericUpDown « tbxTempsPartie » est récupérée depuis la variable iTempspartie du fichier Params.settings.

La propriété « Checked » de la CheckBox« checkEntrainement » est récupérée depuis la variable bEntrainement du fichier Params.settings.

Après un clic sur le bouton btnAppliquer, si les propriétés « Text » de tbxMillisecondes, tbxTempsPartie et cbxDifficulte ne sont pas vides, et si leur valeur sont positives et inférieures ou égales à leur propriété « Maximum », les paramètres seront sauvegardés dans le fichier Params.cs. Autrement, une erreur surgit et avertit l’utilisateur de son erreur de saisie.

La fenêtre se cache, et le formulaire Jeu.cs est affiché à nouveau.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer Params.cs** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Un ou plusieurs champs sont vides. | Indique qu’aucun champ ne doit être vide. | Ok | Il faudra remplir les champs afin de continuer. |
| Des caractères spéciaux sont entrés dans un des champs | Impossible d’écrire autre chose que des chiffres. | Ok | Les entrées claviers sont contrôlées, et le copier/coller est désactivé. |
|  | Un nombre négatif, ou inférieur/supérieur aux valeurs limites est entré dans un des champs | Il est impossible d’entrer un nombre négatif, et pour un nombre supérieur, la valeur maximal est prise | Ok |  |
|  | L’utilisateur entre des valeurs correctes dans les deux champs | Ferme la fenêtre, sauvegarde les valeurs et repasse à la fenêtre principale | Ok |  |

## Cas d'utilisation « Connecter la carte »

Connecter la carte à l’ordinateur via le câble USB puis presser le bouton « Connecter la carte » dans le formulaire principal est une étape nécessaire avant de pouvoir jouer.

Si l’utilisateur clique sur le bouton alors que la carte n’est pas connectée à l’ordinateur, un message d’erreur apparaît et l’en informe.

Une fois la carte connectée, le bouton « Connecter la carte » sera grisé et désactivé, et le bouton « Ejecter la carte » auparavant grisé et désactivé deviendra actif et d’une couleur normale, ceci afin d’éviter à l’utilisateur de connecter/éjecter la carte alors qu’elle l’est déjà.

Lors de changement de formulaire ou si le formulaire a à se réafficher, les boutons resteront dans l’état où ils étaient précédemment grâce au code inséré dans l’évènement « VisibleChanged ».

### Scénario

1. L’utilisateur clique sur le bouton « Connecter la carte ».
2. Si la carte n’est pas connectée à l’ordinateur, une fenêtre d’erreur surgit et l’action s’arrête là.
3. Si la carte est connectée et que la connexion a pu être effectuée, la propriété « Enabled » des boutons tsbConnecter, tsbEjecter et btnJouer est inversée.
4. La méthode inverserCouleurMenu est appelée, qui va inverser la couleur du bouton tsbConnecter avec le bouton tsbEjecter.
5. La variable bIsAlreadyConnected passe à true dans le fichier de paramètres, Params.settings, afin de savoir plus tard si le bouton doit être réaffiché en tant que déjà pressé ou non.

### Maquettes

C:\Users\KaufmannDw\Desktop\Capture1.PNG 🡪 C:\Users\KaufmannDw\Desktop\Capture2.PNG

### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Une variable de type int est déclarée afin de déterminer l’état futur de la carte, car la méthode de connexion fournie avec la .DLL récupérée sur le CD fourni avec la carte par le maître d’atelier renvoie un code après la tentative, indiquant si l’action est réussie ou non. (0 = carte présente, -1 = carte non connectée à l’ordinateur)

Je me sers donc de cette variable à l’aide d’un test pour vérifier si la connexion est réussie ou non. Si c’est un échec, un message apparaîtra à l’utilisateur, lui disant « Veuillez d'abord connecter la carte à l'ordinateur via le cable USB».

Si l’état de la variable est 0, la méthode inverserBoutons est appelée, servant à inverser la propriété « Enabled » entre le bouton tsbConnecter et tsbEjecter.

La couleur de ces deux boutons est ensuite inversée, permettant une meilleure visibilité de l’état de connexion de la carte pour l’utilisateur.

Dans le fichier de paramètres « Params.settings », la variable bIsAlreadyConnected passe à l’état « true » si la connexion a réussi. Ceci afin de savoir lors de réaffichages futurs de la fenêtre de jeu, l’état que doivent avoir les boutons btnConnecter et btnEjecter.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer bouton « tsbConnecter »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | La carte n’est pas connectée via USB à l’ordinateur. | Le bouton tsbConnecter ne se désactive pas pas, il reste impossible de jouer car le bouton de Jeu reste désactivé | Ok | Un message d’erreur surgit indiquant à l’utilisateur qu’il faut d’abord connecter la carte par USB avant de presser ce bouton. |
| La carte est connectée via USB à l’ordinateur. | La propriété Enabled du bouton ainsi que sa couleur s’inversent avec le bouton tsbEjecter | Ok |  |

## Cas d'utilisation « Ejecter la carte »

Le bouton tsbEjecter sert à libérer la carte, elle en sera ainsi plus utilisable avec le programme tant qu’elle ne sera pas reconnectée.

Le bouton n’est pas cliquable tant que la carte n’est pas connectée, et un clic sur le bouton met immédiatement fin à la partie en cours si elle avait lieu.

### Scénario

1. L’utilisateur clique sur le bouton « Reset ».
2. La propriété « Enabled » des boutons tsbConnecter, tsbEjecter et btnJouer est inversée.
3. La méthode inverserCouleurMenu est appelée, qui va inverser la couleur du bouton tsbConnecter avec le bouton tsbEjecter.
4. La variable bIsAlreadyConnected passe à false dans le fichier de paramètres, Params.settings, afin de savoir plus tard si le bouton doit être réaffiché en tant que déjà pressé ou non.
5. Si le timer permettant de déplacer les billes est en cours, il sera stoppé.
6. La carte est éjectée, et ne peux plus être utilisée avec le programme tant qu’elle ne sera pas reconnectée.

### Maquettes

C:\Users\KaufmannDw\Desktop\Capture2.PNG 🡪 C:\Users\KaufmannDw\Desktop\Capture1.PNG

### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

La méthode « CloseDevice » fournie dans la .DLL présente sur le CD obtenu avec la carte est appelée. Elle permet de déconnecter la carte avec le programme en cours, et libère le driver. Il sera impossible d’effectuer des actions liées à la carte par la suite, à moins de la reconnecter.

La méthode inverserBoutons est appelée, servant à inverser la propriété « Enabled » entre le bouton tsbConnecter et tsbEjecter.

La couleur de ces deux boutons est ensuite inversée, permettant une meilleure visibilité de l’état de connexion de la carte pour l’utilisateur.

Dans le fichier de paramètres « Params.settings », la variable bIsAlreadyConnected passe à l’état « false ». Ceci afin de savoir lors de réaffichages futurs de la fenêtre de jeu, l’état que doivent avoir les boutons btnConnecter et btnEjecter.

Un test est effectué afin de savoir si une partie est en cours. Si c’est vrai, le timer tmrJeu sera stoppé, et la partie sera stoppée, affichant néanmoins les résultats à l’utilisateur.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer bouton « tsbEjecter »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Le bouton est cliqué lorsque rien ne se passe. | La carte est déconnectée | Ok |  |
| Le bouton est cliqué lorsqu’une partie est en cours. | La partie se termine, la carte est déconnectée. | Ok |  |
|  | Le bouton est cliqué lorsque la carte n’est pas connectée. | Le bouton est désactivé et non cliquable. | Ok |  |

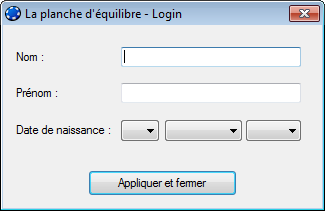
## Cas d'utilisation « Changer d’utilisateur »

Depuis le formulaire principal, il est à tout moment possible de changer d’utilisateur si une autre personne désire jouer sans devoir fermer le programme. Le bouton se trouve dans le menu du formulaire Jeu.cs.

### Scénario

1. Cliquer sur le bouton « tsbChangerUtilisateur ».
2. Le formulaire Jeu.cs est caché
3. Le formulaire Login.cs est affiché comme lors de l’ouverture du programme, forçant l’utilisateur à s’authentifier à nouveau.

### Maquettes



### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Le timer tmrJeu subit la méthode « Dispose », ce qui va libérer toute ressource utilisée par celui-ci et le stopper immédiatement.

La propriété « Visible » du formulaire Jeu.cs passe à false.

La méthode « Hide » du formulaire Jeu.cs est appelée afin de le cacher.

Le formulaire Login.cs est déclaré et affiché.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « tsbChangerUtilisateur »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | L’utilisateur clique sur le bouton lorsque rien ne se passe. | Le formulaire Login s’affiche. | Ok |  |
| L’utilisateur clique sur le bouton lorsqu’une partie est en cours. | Le formulaire Login s’affiche | Ok | La partie est stoppée nette, sans affichage des résultats. |

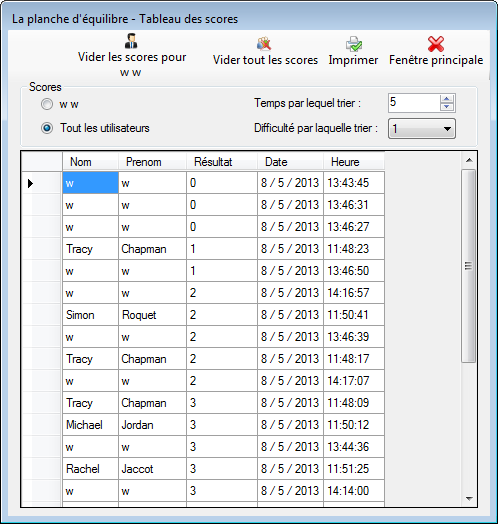
## Cas d'utilisation « Tableau des scores »

Depuis le formulaire Jeu.cs, un clic sur le bouton tsbScores va afficher le formulaire Scores.cs, permettant de consulter la liste des résultats triés par utilisateur, temps et difficulté.

### Scénario

1. Cliquer sur le bouton tsbScores depuis le formulaire Jeu.cs, le formulaire Scores.cs va s’afficher.
2. Par défaut, les scores affichés lors de l’ouverture de ce formulaire sont les mêmes paramètres que l’utilisateur a paramétrés pour sa partie.
3. L’utilisateur peut sélectionner soit uniquement ses propres résultats, soit ceux de tous les utilisateurs présents dans la base de données.
4. Les résultats peuvent être triés par temps en modifiant la valeur en seconde présente dans le TextBox « tbx\_Temps »
5. Les résultats sont également triables par difficulté, allant de Aucune à 3.

### Maquettes

### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Le timer tmrJeu subit la méthode « Dispose », ce qui va libérer toute ressource utilisée par celui-ci et le stopper immédiatement.

La propriété « Visible » du formulaire Jeu.cs passe à false.

La méthode « Hide » du formulaire Jeu.cs est appelée afin de le cacher.

Le formulaire Scores.cs est déclaré et affiché.

Dans l’évènement Load du formulaire Scores.cs, le nom et le prénom de l’utilisateur est inséré dans le bouton tsbSupressionScoresUtilisateur, le niveau de difficulté du ComboBox cbxDifficulte est rempli en fonction de la variable iDifficulte dans le fichier Params.cs, le texte présent dans le RadioButton rbtn\_Personnel est également rempli à partir d’une concaténation entre les variables strNom et strPrenom du fichier Params.cs, et la valeur du NumericUpDown tbx\_Temps est remplie en fonction de la variable iTempsPartie, toujours depuis le fichier Params.cs.

A chaque modification effectuée dans le GroupBox « gbParametres », la méthode « dataGridViewData » est appelée. Cette méthode sert à peupler le DateGridView « dataGridView » en fonction des paramètres de tris, qui sont :

* Le temps par lequel trier.
* La difficulté
* Si les scores à afficher sont seulement ceux de l’utilisateur ou ceux de tout le monde.

Une variable « Requete » de type string est déclarée, contenant les instructions SQL nécessaires afin de retourner le nom, le prénom, le résultat total, ainsi que le jour, le mois, l’année, le temps de la partie et l’heure pour chaque performance dont la difficulté et le temps correspondent aux valeurs entrées dans le NumericUpDown « tbx\_Temps » et la COmboBox « cbxDifficulte ».

Si le bouton radio permettant de n’afficher que les scores de l’utilisateur en cours est sélectionné, une clause AND vient se rajouter, permettant de ne filtrer les résultats qu’à l’utilisateur authentifié.

Les colonnes correspondantes sont ensuite créées dans le dataGridView, et les valeurs sont remplies en conséquence, une fois la requête effectuée grâce à la classe BaseDonnee.cs. Les colonnes sont ensuite redimensionnées grâce à la méthode AutoResizeColums.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « Scores.cs »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Des valeurs négatives sont entrées dans le NumericUpDown « tbx\_Temps » | Il n’est pas possible d’entrer un nombre négatif. | Ok |  |
|  | Un nombre très élevé est entré dans le NumericUpDown « tbx\_Temps ». | Une valeur égale à la propriété « Maximum » de celui-ci est remise à la place du nombre. | Ok |  |

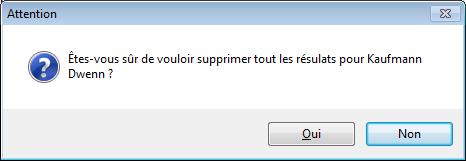
## Cas d'utilisation « Suppression des scores »

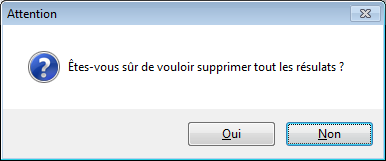
Depuis le formulaire Scores.cs, un clic sur le bouton tsbSupressionScoresUtilisateur ou tsbSupressionScoresTout va permettre de supprimer, comme leurs noms l’indique, soit de supprimer uniquement tous les résultats liés à un utilisateur, soit tous les résultats de tout les utilisateurs. Cette action est irréversible.

### Scénario

1. Cliquer soit sur le bouton « Vider les Scores pour [Nom Utilisateur] », soit sur « Vider tout les scores »
2. Un MessageBox surgit, informant l’utilisateur de l’action qu’il est sur le point d’effectuer.
3. Si l’utilisateur presse sur le bouton Oui, les données correspondantes seront effacées définitivement de la base de données et les données présentes dans le dataGridView seront actualisées.

### Maquettes



### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Si l’utilisateur répond par Oui à l’avertissement, une requête SQL sera exécutée sur la base de données.

Le contenu de la requête diffère selon le bouton pressé :

* Si le bouton « Vider les scores pour [Nom Utilisateur] » a été cliqué, alors la requête SQL utilisera l’instruction « DELETE » suivi de la clause « FROM » en utilisant l’ID de l’utilisateur récupéré depuis le fichier Params.cs afin de ne supprimer que les données liées à l’utilisateur en cours.
* Si le bouton « Vider tout les scores » a été cliqué, la requête sera plus simple : Ce sera un simple « DELETE FROM tbl\_Performances ». Vu que rien n’est spécifié après la clause FROM, toute la table sera vidée. J’ai utilisé cette méthode car apparemment l’instruction « DROP TABLE » ne fonctionnait pas sur Access.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « Scores.cs »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | Les scores sont supprimés alors qu’aucune donnée n’était présente. | La table subit quand même la requête sans erreur. | Ok |  |
|  | Un nombre très élevé est entré dans le NumericUpDown « tbx\_Temps ». | Une valeur égale à la propriété « Maximum » de celui-ci est remise à la place du nombre. | Ok |  |

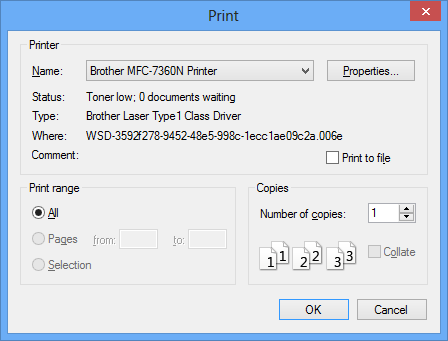
## Cas d'utilisation « Impression des scores »

Depuis le formulaire Scores.cs, un clic sur le bouton tsbImprimer va afficher à l’utilisateur une fenêtre normale d’impession permettant, s’il clique sur « OK », de lancer l’impression des résultats actuellement affichés.

### Scénario

1. Cliquer sur le bouton « Imprimer »
2. Accepter en cliquant sur « OK »
3. L’impression se lance si une imprimante a été configurée correctement dans l’ordinateur.

### Maquettes

### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Si l’utilisateur presse sur OK dans la fenêtre d’impression, le PrintDocument pdImpression subit la méthode Print, ce qui va lancer l’impression finale du dataGridView.

### La phase de programmation

Le code source est en annexe.

### La phase de tests

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « Scores.cs »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | L’utilisateur a sélectionné l’impression de ses propres scores uniquement. | Seuls ses scores seront imprimés. | Ok |  |
|  | L’utilisateur a choisi d’imprimer les scores de tous les utilisateurs. | Les scores de tous les utilisateurs seront imprimés, avec le nom et le prénom spécifié pour chaque résultat afin de différencier les personnes. | Ok |  |

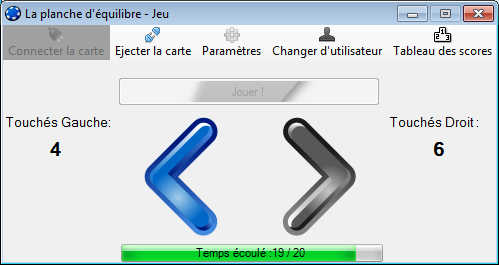
## Cas d'utilisation « Jouer »

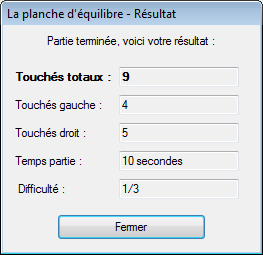
Une fois la carte connectée à l’ordinateur, il est enfin possible de jouer selon les paramètres spécifiés par l’utilisateur.

### Scénario

1. Cliquer sur le bouton jouer.
2. Les contrôles liés au jeu deviennent visibles.
3. Une barre de progression se lance, informant l’utilisateur du temps passé sur le temps total de la partie
4. A chaque perte d’équilibre de l’utilisateur, il verra une des deux flèches indiquant le côté duquel il a touché. Si l’utilisateur fait un contact dans un temps plus petit que le temps d’attente défini entre deux contacts, il verra le compteur de touchés « gauches/droits » s’incrémenter.
5. Quand le temps est écoulé, un formulaire affichant les résultats apparaît, indiquant les statistiques, puis invite l’utilisateur à le fermer.
6. Une fois le formulaire d’affichage des résultats fermé, il est possible de rejouer, en recliquant sur le bouton « Jouer ».

### Maquettes





### Analyse du scénario

#### Algorithme ou Structogramme

#### Explications détaillées

Lors d’un clic sur le bouton « btnJouer », le timer de jeu, « tmrJeu », est démarré.

La propriété du bouton btnJouer passe ensuite à false.

Une variable, permettant de mesurer le temps écoulé depuis le début de la partie, nommée « lDebut », est assignée à Evironment.TickCount.

La valeur Maximum de la ProgressBar indiquant la progression de la partie est réglée à la variable iTempsPartie du fichier Params.cs \*1000, ceci afin de fournir une animation de progression fluide.

Le panel contenant les contrôles liés à la partie en cours devient visible.

Le timer qui est maintenant démarré, tmrJeu, dont l’intervalle de tick est réglée à 10 millisecondes, va tourner en boucle jusqu’à la fin de la partie afin de détecter si un des contacts de la planche fait actuellement contact, en tenant compte du temps de « répit » laissé à l’utilisateur où deux touchés ne doivent pas être comptabilisés. A chaque Tick, le timer réactualise la valeur de la variable lNew, qui va par la suite permettre de calculer le temps écoulé à la milliseconde près.

Un test est effectué pour savoir s’il y a contact, au moyen de la DLL K8055 et de sa méthode « ReadDigitalChannel », auquel il faut passer un paramètre en input. La planche d’équilibre est reliée sur la carte Velleman sur les inputs 4 et 5, c’est pourquoi il y a deux IF testant s’il y a contact. S’il y a contact, l’image contenue dans les PictureBox indiquant quel capteur fait contact change pour une couleur bleue, et un test est effectué afin de savoir si celui-ci doit être comptabilisé, au moyen d’un principe simple : Une soustraction de lNew à lDernier. Si le résultat est supérieur à la variable iMillisecondes dans le fichier Params.settings, le nombre de touchés sera incrémenté de 1, et lDernier sera actualisé à la valeur Environnment.TickCount, ceci afin de recommencer un temps de répit de la longueur définie par l’utilisateur dans les paramètres.

Les labels indiquant à l’utilisateur le nombre de touchés sont ensuite mis à jour, ainsi que la progression de la ProgressBar indiquant la progression de la partie en cours.

Un dernier test est effectué afin de déterminer s’il est temps de terminer la partie ou non, au moyen d’un test simple : Si lNew – lDebut /1000 est supérieur au contenu de la variable iMillisecondes dans le fichier Params.cs, la méthode StopJeu est appelée.

Lorsque StopJeu est appelée, un test est effectué sur la variable bEntraînement présente dans le fichier Params.settings afin de savoir si la partie était un entraînement ou non, ce qui entraînera le fait que les scores soient sauvegardés dans la base de données ou non.

Le timer de jeu, tmrJeu est ensuite stopé, la propriété Enabled du bouton btnJeu repasse donc à true.

Les images contenues dans les deux PictureBox sont retirées.

Le formulaire Jeu.cs est rendu invisible, puis se cache, et le formulaire affichant les résultats dans la partie est affiché, jusqu’à ce que l’utilisateur le ferme. Une fois fermé, le formulaire Jeu.cs réapparaîtra et une nouvelle partie pourra être jouée.

### La phase de programmation

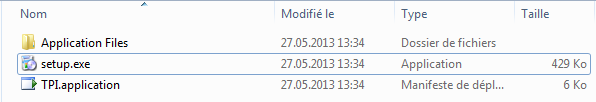
Le code source est en annexe.

### La phase de tests

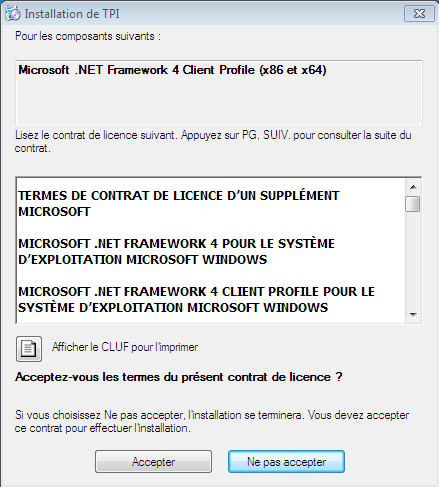
|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test à effectuer « Jeu.cs »** | | **Résultat**  **escompté** | **Résultat**  **obtenu** | **Constatation** |
| Généralités | | | | |
|  | XXXXXXXXXXXXX | XXXXXXXXXXXXX | Ok |  |
|  | XXXXXXXXXXXXX | XXXXXXXXXXXX | Ok |  |

# Mode d’emplois utilisateur

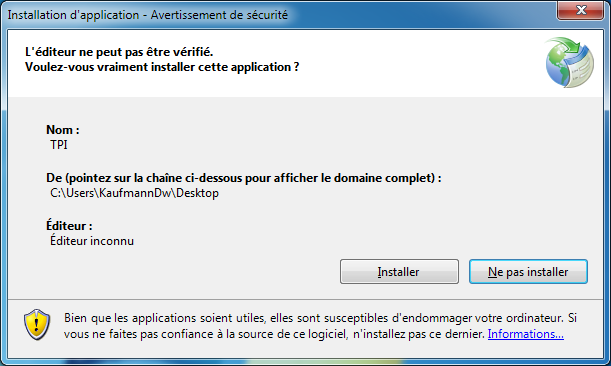
## Installation de l’application



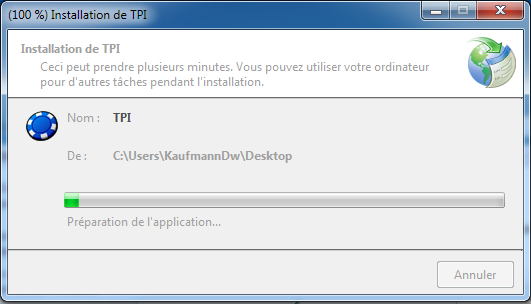
Pour installer l’application, il suffit d’ouvrir le fichier « setup.exe » fourni. Si une fenêtre vous invite à installer le fichier « Microsoft .NET Framework 4 Client Profile (x86 et x64), acceptez.



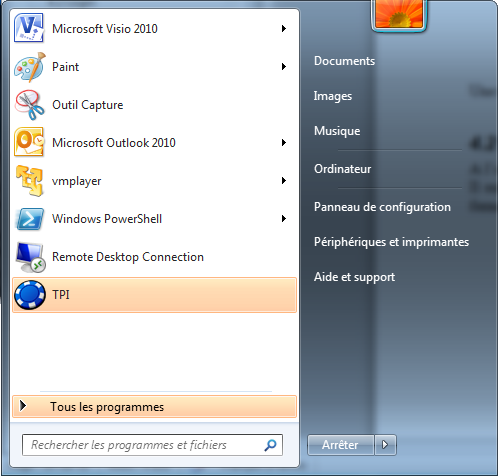
Une fenêtre va demander la permission permettant au programme de s’installer. Il faut cliquer sur le bouton « Installer ».



Un bref temps d’attente sera nécessaire avant la fin de l’installation



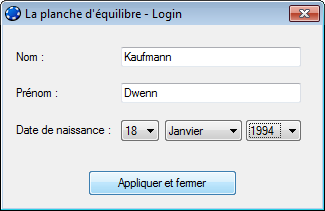
Une fois la barre de progression complète, l’installation est terminée et l’icône de l’application apparaît dans le menu démarrer.



## Login

A l’ouverture de l’application, une fenêtre invitant l’utilisateur à s’authentifier surgit.

Il suffit de rentrer son Nom, Prénom, et sa date de naissance puis de cliquer sur « Appliquer et fermer ».



## Connecter / Déconnecter la carte

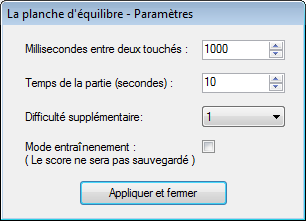
Une fois authentifié, la fenêtre de jeu apparaît. Mais avant de pouvoir jouer, il faut au préalable connecter la carte Velleman par USB à l’ordinateur, et la planche d’équilibre à celle-ci. Une fois fait, un simple clic sur le bouton « Connecter la carte » activera les boutons permettant à l’utilisateur de pouvoir jouer.

C:\Users\KaufmannDw\Desktop\Capture1.PNG 🡪 C:\Users\KaufmannDw\Desktop\Capture2.PNG

La déconnexion de la carte est possible, et stoppe la partie en cours si elle avait lieu.

## Paramétrer l’application

Dans la fenêtre de jeu, un clic sur le bouton Paramètres permet d’afficher une fenêtre dans laquelle figure toutes les options de configuration de l’application.

Millisecondes entre deux touchés : Correspond au nombre de millisecondes à attendre avant de compter à nouveau un touché lorsqu’il y a contact.

Temps de la partie : Définit le nombre, en secondes, de la partie.

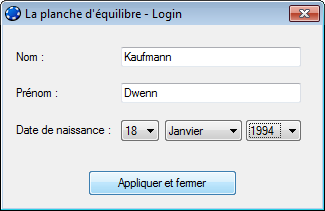
Difficulté supplémentaire : Aucune, 1, 2, ou 3. Des accessoires sont fournis avec la planche afin de corser la difficulté. L’utilisateur doit, en plus de tenir en équilibre sur la planche, porter des objets.

Mode entraînement : Si cette case est cochée, les scores ne sont pas sauvegardés à la fin d’une partie.

Un clic sur le bouton « Appliquer et fermer » applique les paramètres, ainsi lors de la prochaine partie, les modifications faites seront prises en compte.

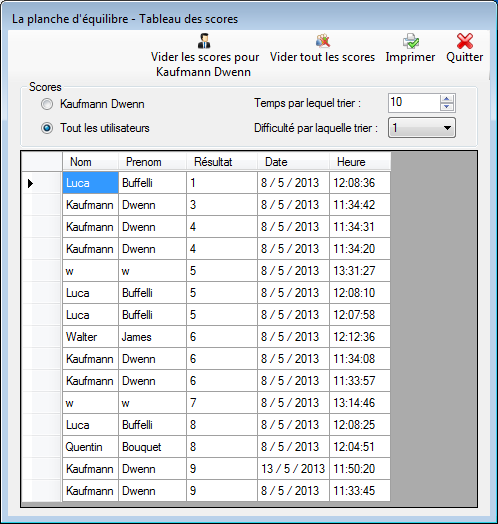
## Changer d’utilisateur

Un clic sur ce bouton renvoie simplement à la première fenêtre, obligeant l’utilisateur à s’authentifier à nouveau.

## Affichage des scores

Un clic sur le bouton « Tableau des scores » depuis le formulaire de jeu permet de faire apparaître une nouvelle fenêtre, dans laquelle figure un tableau regroupant les scores triés par temps et difficulté. Aucun bouton permettant d’actualiser n’est présent car cela se fait automatiquement à chaque modification d’un paramètre de tri.



### Scores par utilisateur

Dépendamment du choix de l’utilisateur, il est possible de trier les résultats affichés afin de ne montrer que les siens, ou ceux de tout le monde via un bouton radio.



### Scores par temps

Les scores sont triés également par le temps de la partie. L’utilisateur doit juste entrer le nombre, en secondes, du temps des parties qu’il veut consulter.



### Scores par difficulté

La difficulté est aussi un paramètre de tri dans l’affichage des scores. Allant de Aucune à 3, l’utilisateur doit sélectionner le niveau choisi afin de voir les scores correspondants.



## Vider les scores

Une fois arrivé dans la fenêtre « Tableau des scores », il est possible de supprimer ceux-ci.

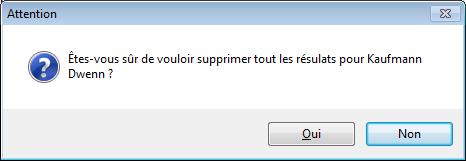
### Vider les scores de l’utilisateur en cours uniquement

Un bouton permet de faire cette action, le voici :



L’utilisateur en cours s’appelant Kaufmann Dwenn, son nom est inscrit dans le bouton afin que l’utilisateur s’aperçoive que seuls ses résultats personnels seront supprimés.

Une fenêtre d’avertissement surgit, pour éviter les erreurs, afin d’avertir l’utilisateur de l’action qu’il tente d’effectuer.



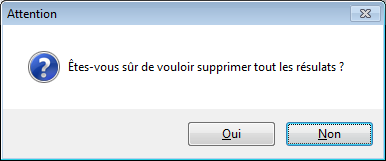
S’il clique sur « Oui », tous ses résultats personnels seront effacés définitivement et la liste de ses résultats (vides) sera affichée.

### Vider les scores de tous les utilisateurs

Un bouton permet de faire cette action, le voici :



Ce bouton permet, comme son nom l’indique, à supprimer définitivement tous les résultats de tous les utilisateurs. Cette action étant irréversible, un avertissement apparaît :



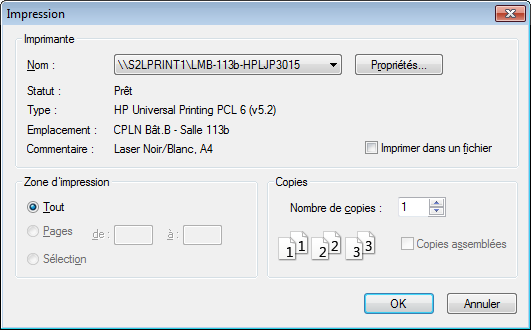
Si la réponse est oui, absolument tous les résultats seront effacés définitivement et l’affichage de la liste des résultats sera actualisé.

## Imprimer les scores

Si l’on souhaite imprimer les résultats affiché dans la fenêtre « Tableau des scores », il suffit de cliquer sur le bouton « Imprimer » :



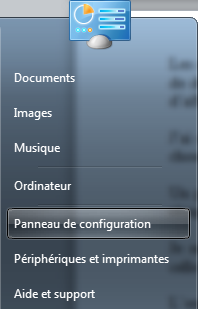
Une fenêtre « normale » d’impression apparaît alors, permettant de modifier les paramètres classiques d’impression :



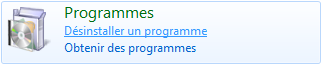
Lors d’un clic sur le bouton « Ok », l’impression des résultats se fera immédiatement.

## Désinstallation de l’application

Pour désinstaller l’application, la première étape est de cliquer sur le bouton « Panneau de configuration » dans le menu démarrer.



Ensuite, il faut cliquer sur « Désinstaller un programme ».



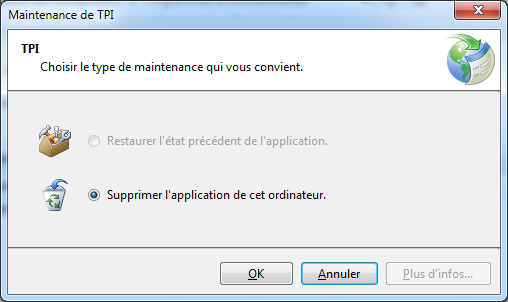
Il faut ensuite rechercher l’application, nommée « TPI »



Et cliquer sur le bouton « Désinstaller / modifier »



Finalement, il reste à sélectionner le bouton radio « Supprimer l’application de cet ordinateur. », puis de presser sur « OK ».



La désinstallation est un succès.

# Conclusions

Lors du développement de mon projet, la prise d’information et la compréhension de l’utilisation de la carte Velleman n’as pas été un souci car des exemples étaient présents sur le web, ainsi que sur le CD fourni par le maître d’atelier.

Les objectifs du cahier des charges ont tous étés atteints. Le programme est en effet capable de déterminer l’habileté de quelqu’un à rester en équilibre sur une planche rendue instable, et d’afficher un classement des scores ainsi que de l’imprimer.

J’ai eu du plaisir durant mon travail car j’apprécie la programmation en langage C#, c'est la chose qui m'a toujours intéressée le plus ici au CPLN.

Un point important a été de comprendre comment utiliser la .DLL fournie sur le CD afin de récupérer les informations sur les contacts de la carte électronique.

Je suis maintenant plus à l’aise pour ce qui est de l’utilisation d’une DLL annexe, de plus celle fournie n’était pas prévue à l’origine pour le C#, mais le C++.

L’organisation quant aux tâches à effectuer, et l’attribution d’un temps par tâche est important, c’est pourquoi j’ai fait attention à y réfléchir correctement dès le début.

# Annexes

## Journal de bord

Le journal de bord est en annexe

## Schéma Synoptique

### Login, connexion de la carte et paramétrage



### Affichage des scores



### Fenêtre de jeu



## Cahier des charges

Page vide car remplacée par les feuilles vertes du cahier des charges après impression.

Page vide car remplacée par les feuilles vertes du cahier des charges après impression.

Page vide car remplacée par les feuilles vertes du cahier des charges après impression.

Page vide car remplacée par les feuilles vertes du cahier des charges après impression.

## Code source

### Login.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

//Permets l'utilisation des utilitaires de base de données

using System.Data.Common;

using System.Data.OleDb;

//Permets l'utilisation du fichier de config

using System.Configuration;

namespace TPI

{

public partial class Login : Form

{

public Login()

{

InitializeComponent();

}

bool bDateValide = false;

private void btnAppliquer\_Click(object sender, EventArgs e)

{

controleDate(sender, e);

if (tbxNom.Text == "" || tbxPrenom.Text == "" || cbxJour.Text == "" || cbxMois.Text == "" || cbxAnnee.Text == "")

{

MessageBox.Show("Veuillez remplir tous les champs", "Champs vides", MessageBoxButtons.OK, MessageBoxIcon.Error);

}

else

{

string strDate = cbxJour.Text + "/" + (cbxMois.SelectedIndex + 1).ToString() + "/" + cbxAnnee.Text;

if (bDateValide == true) //Si la date est valide

{

int NbUser = 0;

DbDataReader oRdr;

//Cette requête va retourner le nombre de personne qui ont ce nom/prenom dans la base de données

oRdr = BaseDonnee.Request("SELECT COUNT(Nom\_Utilisateur) FROM tbl\_Personnes WHERE Nom\_Utilisateur='" + (tbxNom.Text).Replace("'", "''") + "' AND Prenom\_Utilisateur='" + (tbxPrenom.Text).Replace("'", "''") + "' AND Jour\_Naissance\_Utilisateur='" + (cbxJour.Text).Replace("'", "''") + "' AND Mois\_Naissance\_Utilisateur='" + (cbxMois.Text).Replace("'", "''") + "' AND Annee\_Naissance\_Utilisateur='" + (cbxAnnee.Text).Replace("'", "''") + "';");

while (oRdr.Read())

{

NbUser = (int)oRdr[0];

}

//Si il n'y a personne sous ce nom, l'utilisateur va être ajouté dans la base

if (NbUser == 0)

{

//Cette requête va ajouter l'utilisateur en fonction des textbox et combobox

BaseDonnee.Request("INSERT INTO tbl\_Personnes(Nom\_Utilisateur,Prenom\_Utilisateur,Jour\_Naissance\_Utilisateur,Mois\_Naissance\_Utilisateur,Annee\_Naissance\_Utilisateur) VALUES ('" + tbxNom.Text.Replace("'", "''") + "','" + tbxPrenom.Text.Replace("'", "''") + "','" + cbxJour.Text.Replace("'", "''") + "','" + cbxMois.Text.Replace("'", "''") + "','" + cbxAnnee.Text.Replace("'", "''") + "');");

MessageBox.Show("Vous avez été correctement enregistré(e), amusez-vous bien!", "Inscription validée", MessageBoxButtons.OK, MessageBoxIcon.Information);

}

ActiveForm.Hide(); //Cache le formulaire de Login

Jeu Jeu = new Jeu();

Jeu.Show(); //Affiche le formulaire de Jeu

Params.Default.strNom = tbxNom.Text;

Params.Default.strPrenom = tbxPrenom.Text;

Params.Default.strJourNaissance = cbxJour.Text;

Params.Default.strMoisNaissance = cbxMois.Text;

Params.Default.strAnneeNaissance = cbxAnnee.Text;

}

}

DbDataReader iD;

iD = BaseDonnee.Request("SELECT tbl\_Personnes.ID FROM tbl\_Personnes WHERE Nom\_Utilisateur='" + (tbxNom.Text).Replace("'", "''") + "' AND Prenom\_Utilisateur='" + (tbxPrenom.Text).Replace("'", "''") + "' AND Jour\_Naissance\_Utilisateur='" + (cbxJour.Text).Replace("'", "''") + "' AND Mois\_Naissance\_Utilisateur='" + (cbxMois.Text).Replace("'", "''") + "' AND Annee\_Naissance\_Utilisateur='" + (cbxAnnee.Text).Replace("'", "''") + "';");

while (iD.Read())

{

Params.Default.iID = Convert.ToInt32(iD["ID"]);

Params.Default.Save();

}

}

private void controleDate(object sender, EventArgs e)

{

lblDatenaissance.ForeColor = Color.Black; //Remet la couleur par défaut (noir) au label

DateTime dt;

string strDate = cbxJour.Text + "/" + (cbxMois.SelectedIndex + 1).ToString() + "/" + cbxAnnee.Text;

if (DateTime.TryParse(strDate, out dt)) //Si la date est valide

{

if (dt.CompareTo(DateTime.Now) < 1) //Si la date n'est pas dans le futur

{

pbxErreur.ImageLocation = @"";

bDateValide = true;

btnAppliquer.Enabled = true;

}

else //Si la date est dans le futurs

{

pbxErreur.ImageLocation = @"error.png";

bDateValide = false;

lblDatenaissance.ForeColor = Color.Red;

btnAppliquer.Enabled = false;

}

}

else //Si la date n'est pas valide.

{

if (cbxJour.Text != "" && cbxMois.Text != "" && cbxAnnee.Text != "")

{

pbxErreur.ImageLocation = @"error.png";

bDateValide = false;

lblDatenaissance.ForeColor = Color.Red;

btnAppliquer.Enabled = false;

}

}

}

private void Login\_Load(object sender, EventArgs e)

{

//Peuple la liste déroulante des années, afin que lorsque l'on passe à une nouvelle année, la liste se mette à jour d'elle même.

cbxAnnee.Items.Clear();

int[] iAnnee;

int iAnneeDepart = 1940;

iAnnee = new int[(Convert.ToInt32(DateTime.Now.Year + 1) - iAnneeDepart)];

for (int iCpt = 0; iCpt < (Convert.ToInt32(DateTime.Now.Year + 1) - iAnneeDepart); iCpt++)

{

iAnnee[iCpt] = (iAnneeDepart + iCpt);

cbxAnnee.Items.Add(iAnnee[iCpt]);

}

}

private void tbxKeyPress(object sender, KeyPressEventArgs e) //N'accepte que les lettres, les espaces, le caractère "-", le caractère "'".

{

e.Handled = !char.IsControl(e.KeyChar) && !char.IsLetter(e.KeyChar) && !char.IsSeparator(e.KeyChar) && e.KeyChar != Convert.ToChar("-") && e.KeyChar != Convert.ToChar("'");

}

}

}

### Parametres.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace TPI

{

public partial class Parametres : Form

{

public Parametres()

{

InitializeComponent();

}

private void Parametres\_Load(object sender, EventArgs e)

{

//Peuple les Textbox, la Combobox et la checkbox depuis le fichier Params.settings

tbxMillisecondes.Text = Params.Default.iMillisecondes.ToString();

tbxTempsPartie.Text = Params.Default.iTempsPartie.ToString();

cbxDifficulte.SelectedIndex = Params.Default.iDifficulte;

checkEntrainement.Checked = Params.Default.bEntrainement;

//Permet d'appuyer sur le bouton appliquer lorsque l'utilisateur appuie sur la touche "Enter"

this.AcceptButton = btnAppliquer;

}

private void btnAppliquer\_Click(object sender, EventArgs e)

{

//Déclaration et reset du message contenant les erreurs

string strMessageErreur = "";

//Test des valeurs entrées par l'utilisateur

if (tbxMillisecondes.Text != "" && tbxTempsPartie.Text != "" && cbxDifficulte.Text != "") //Si les trois champs ne sont pas vides

{

int iTemp;

bool bValide = false;

//Teste la valeur entrée pour les millisecondes

bValide = int.TryParse(tbxMillisecondes.Value.ToString(), out iTemp);

if (iTemp < 0)

{

strMessageErreur += "Le nombre de millisecondes doit être positif\n";

}

if (iTemp > tbxMillisecondes.Maximum)

{

strMessageErreur += "Le nombre de millisecondes doit être inférieur à " + tbxMillisecondes.Maximum + "\n";

}

//Teste la valeur entrée pour le temps de la partie

bValide = int.TryParse(tbxTempsPartie.Value.ToString(), out iTemp);

if (iTemp < 0)

{

strMessageErreur += "Le temps par concours doit être positif\n";

}

if (iTemp > tbxTempsPartie.Maximum)

{

strMessageErreur += "Le temps par concours doit être inférieur à " + tbxTempsPartie.ToString() + "\n";

}

}

else //Si un des champs est vide

{

strMessageErreur += "Tout les champs doivent être remplis";

}

//Affichage de l'erreur s'il y en a une

if (strMessageErreur != "")

{

MessageBox.Show(strMessageErreur, "Erreur de saisie");

}

else //Si tout les champs sont remplis correctement

{

this.Hide();

Form Jeu = new Jeu();

Jeu.Show();

//Teste si le mode entraînement est activé ou non

if (checkEntrainement.Checked)

{

Params.Default.bEntrainement = true;

}

else

{

Params.Default.bEntrainement = false;

}

Params.Default.iMillisecondes = Convert.ToInt32(tbxMillisecondes.Text);

Params.Default.iTempsPartie = Convert.ToInt32(tbxTempsPartie.Text);

Params.Default.iDifficulte = (cbxDifficulte.SelectedIndex);

Params.Default.Save(); //Sauvegarde les modifications.

}

}

private void Parametres\_FormClosing(object sender, FormClosingEventArgs e)

{

Environment.Exit(0); //Fermture propre au cas où l'utilisateur arriverait à fermer le formulaire, bien que la controlbox ne soit plus présente.

}

private void tbxMillisecondes\_KeyPress(object sender, KeyPressEventArgs e)

{//Vérifie l'entrée clavier faite par l'utilisateur afin d'en retirer les caractères non désirés dans un numericUpDown

e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);

}

private void tbxTempsPartie\_KeyPress(object sender, KeyPressEventArgs e)

{//Vérifie l'entrée clavier faite par l'utilisateur afin d'en retirer les caractères non désirés dans un numericUpDown

e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);

}

}

}

### Resultat.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

namespace TPI

{

public partial class Resultat : Form

{

int iGauche;

int iDroite;

public Resultat(int iTouchesGauche, int iTouchesDroit)

{

InitializeComponent();

iGauche = iTouchesGauche;

iDroite = iTouchesDroit;

}

private void Resultat\_Load(object sender, EventArgs e)

{

tbxDifficulte.Text = Params.Default.iDifficulte.ToString()+"/3";

tbxTempsPartie.Text = Params.Default.iTempsPartie.ToString()+" secondes";

tbxTouchesTotaux.Text = (iGauche + iDroite).ToString();

tbxTouchesGauche.Text = iGauche.ToString();

tbxTouchesDroit.Text = iDroite.ToString();

}

private void btnFermer\_Click(object sender, EventArgs e)

{

this.Visible = false;

this.Hide();

Form Jeu = new Jeu();

Jeu.Show();

}

}

}

### Scores.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

//Permets l'utilisation des utilitaires de base de données

using System.Data.Common;

using System.Data.OleDb;

//Permets l'utilisation du fichier de config

using System.Configuration;

//Permets l'utilisation des fonctions d'impression

using System.Drawing.Printing;

using System.Threading; //Va permette de mettre en pause le temps que la supression des données se fasse.

namespace TPI

{

public partial class Scores : Form

{

public Scores()

{

InitializeComponent();

this.pdImpression.PrintPage += new PrintPageEventHandler(this.pdImpression\_PrintPage);

}

//Va permettre l'impression

PrintDocument pdImpression = new PrintDocument();

DataGridViewPrinter dgvPrinter; //Objet permettant l'impression

string strMessageErreur = "";

private void dataGridViewData(object sender, EventArgs e)

{

if (tbx\_Temps.Text != "-")

{

string Temps = tbx\_Temps.Value.ToString();

//Construction de la requête selon les options choisies par l'utilisateur

string Requete = "SELECT tbl\_Personnes.Nom\_Utilisateur, tbl\_Personnes.Prenom\_Utilisateur, tbl\_Performances.ResultatTotal, tbl\_Performances.Jour, tbl\_Performances.Mois, tbl\_Performances.Annee, tbl\_Performances.Temps, tbl\_Performances.Heure" + " FROM tbl\_Personnes" + " INNER JOIN tbl\_Performances ON tbl\_Personnes.ID = tbl\_Performances.num\_tbl\_Personnes";

Requete += " WHERE tbl\_Performances.Temps=" + tbx\_Temps.Value;

Requete += " AND tbl\_Performances.Difficulte=" + cbxDifficulte.SelectedIndex + "";

//Test si l'utilisateur veux tout le monde ou uniquement ses résultats

if (rbtn\_Personnel.Checked)

{

Requete += " AND (tbl\_Personnes.ID = " + Params.Default.iID.ToString() + " )";

}

Requete += " ORDER BY tbl\_Performances.ResultatTotal ASC";

DbDataReader oRdr = BaseDonnee.Request(Requete);

DataTable dt = new DataTable();

//Création de la datatable pour des résultats d'un utilisateur

if (rbtn\_Personnel.Checked)

{

dt.Columns.Add("Résultat", Type.GetType("System.Int32"));

dt.Columns.Add("Date", Type.GetType("System.String"));

dt.Columns.Add("Heure", Type.GetType("System.String"));

//Peuple la datatable

try //Le Try-Catch Evite les erreurs si l'utilisateur rentre trop vite des frappes au clavier dans le formulaire Scores ( A Cause de l'evenement TextChanged du numericUpDown)

{

while (oRdr.Read())

{

DataRow r = dt.NewRow();

r["Résultat"] = oRdr["ResultatTotal"].ToString();

r["Date"] = oRdr["Jour"] + " / " + oRdr["Mois"] + " / " + oRdr["Annee"];

r["Heure"] = oRdr["Heure"].ToString();

dt.Rows.Add(r);

}

}

catch

{

}

}

else

{

//création de datatable pour plusieurs utilisateurs

dt.Columns.Add("Nom", Type.GetType("System.String"));

dt.Columns.Add("Prenom", Type.GetType("System.String"));

dt.Columns.Add("Résultat", Type.GetType("System.Int32"));

dt.Columns.Add("Date", Type.GetType("System.String"));

dt.Columns.Add("Heure", Type.GetType("System.String"));

//Peuple la datatable

try //Le Try-Catch Evite les erreurs si l'utilisateur rentre trop vite des frappes au clavier dans le formulaire Scores ( A Cause de l'evenement TextChanged du numericUpDown)

{

while (oRdr.Read())

{

DataRow r = dt.NewRow();

r["Nom"] = oRdr["Nom\_Utilisateur"].ToString();

r["Prenom"] = oRdr["Prenom\_Utilisateur"].ToString();

r["Résultat"] = oRdr["ResultatTotal"].ToString();

r["Date"] = oRdr["Jour"] + " / " + oRdr["Mois"] + " / " + oRdr["Annee"];

r["Heure"] = oRdr["Heure"].ToString();

dt.Rows.Add(r);

}

}

catch

{

}

}

//Affiche la DataTable dans le DataGridView

dataGridView.DataSource = dt;

if (rbtn\_Personnel.Checked)

{

//Donne l'ordre dans lequel les colonnes doivent être afficher

dataGridView.Columns["Résultat"].DisplayIndex = 0;

dataGridView.Columns["Date"].DisplayIndex = 1;

dataGridView.Columns["Heure"].DisplayIndex = 2;

}

else

{

dataGridView.Columns["Nom"].DisplayIndex = 0;

dataGridView.Columns["Prenom"].DisplayIndex = 1;

dataGridView.Columns["Résultat"].DisplayIndex = 2;

dataGridView.Columns["Date"].DisplayIndex = 3;

dataGridView.Columns["Heure"].DisplayIndex = 4;

}

dataGridView.AutoResizeColumns(DataGridViewAutoSizeColumnsMode.AllCells); //Permet de faire que la largueur de chaque colonne s'ajuste automatiquement.

}

else

{

strMessageErreur = "Le caractère \"-\" est interdit";

}

//Affichage de l'erreur s'il y en a une

if (strMessageErreur != "")

{

MessageBox.Show(strMessageErreur, "Erreur de saisie");

}

}

private void Scores\_Load(object sender, EventArgs e)

{

tsbSupressionScoresUtilisateur.Text = "Vider les scores pour \n" + Params.Default.strNom + " " + Params.Default.strPrenom;

cbxDifficulte.SelectedIndex = Params.Default.iDifficulte; //Affiche la valeur de la difficulté en cours par défaut, sinon il apparaît vide.

rbtn\_Personnel.Text = Params.Default.strNom + " " + Params.Default.strPrenom;

tbx\_Temps.Value = Params.Default.iTempsPartie;

}

private void tsbImprimer\_Click(object sender, EventArgs e)

{

if(print() == true)

{

pdImpression.Print(); //Va finaliser l'impression

}

}

private bool print()

{

//Ouverture d'une boite de dialogue d'impression

PrintDialog MyPrintDialog = new PrintDialog();

if (MyPrintDialog.ShowDialog() != DialogResult.OK)

return false;

//Affectation des options selectionnées dans la boite de dialogue d'impression

pdImpression.DocumentName = "Résultats";

pdImpression.PrinterSettings = MyPrintDialog.PrinterSettings;

pdImpression.DefaultPageSettings = MyPrintDialog.PrinterSettings.DefaultPageSettings;

//Création du contenu du titre de l'impression

string strTitre = "";

if (rbtn\_Personnel.Checked)

strTitre = "Résultats pour :\n" + Params.Default.strNom + " " + Params.Default.strPrenom + " \n " + Params.Default.strJourNaissance + " " + Params.Default.strMoisNaissance + " " + Params.Default.strAnneeNaissance +"\nDifficulté supplémentaire : " + Params.Default.iDifficulte.ToString() + "/3 \nTemps: " + Params.Default.iTempsPartie+" secondes";

else

strTitre = "Classement\n\n\nDifficulté supplémentaire : " + Params.Default.iDifficulte.ToString() + "/3 \nTemps: " + Params.Default.iTempsPartie+" secondes";

//Création de l'objet à imprimer

dgvPrinter = new DataGridViewPrinter(dataGridView,

pdImpression, true, true, strTitre, new Font("Microsoft sans serif", 20,

FontStyle.Bold, GraphicsUnit.Point), Color.Black, true);

return true;

}

// L'évenement d'impression de l'objet MyPrintDocument

private void pdImpression\_PrintPage(object sender, PrintPageEventArgs e)

{

bool more = dgvPrinter.DrawDataGridView(e.Graphics); //Détecte s'il reste d'autres pages

if (more == true)

e.HasMorePages = true;

}

private void tsbQuitter\_Click(object sender, EventArgs e)

{

this.Hide();

Form Jeu = new Jeu();

Jeu.Show();

}

private void Scores\_FormClosing(object sender, FormClosingEventArgs e)

{

Environment.Exit(0); //Fermture propre au cas où l'utilisateur arriverait à fermer le formulaire, bien que la controlbox ne soit plus présente.

}

private void tsbSupressionScoresTout\_Click(object sender, EventArgs e)

{

DialogResult drConfirmation = MessageBox.Show("Êtes-vous sûr de vouloir supprimer tout les résulats ?",

"Attention",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question,

MessageBoxDefaultButton.Button2); //Le bouton non aura le focus par défaut.

if (drConfirmation == DialogResult.Yes)

{

string Requete = "DELETE FROM tbl\_Performances"; //Vu qu'il n'y a pas de clause WHERE, toute la table sera vidée. (Truncate ne fonctionne apparemment pas dans une base Access).

DbDataReader oRdr = BaseDonnee.Request(Requete);

Thread.Sleep(3000); //Pour qu'au performClick, toute action se soit déjà terminée.

rbtn\_Tous.PerformClick(); //Clique sur le bouton pour afficher les scores personnels, vu qu'on viens de supprimer tout les résulats de l'utilisateur en cours. ( Afin de rafraîchir)

}

}

private void tsbSupressionScoresUtilisateur\_Click(object sender, EventArgs e)

{

DialogResult drConfirmation = MessageBox.Show("Êtes-vous sûr de vouloir supprimer tout les résulats pour "+Params.Default.strNom+" "+Params.Default.strPrenom+" ?",

"Attention",

MessageBoxButtons.YesNo,

MessageBoxIcon.Question,

MessageBoxDefaultButton.Button2); //Le bouton non aura le focus par défaut.

if (drConfirmation == DialogResult.Yes) //Supression des résultats liés à l'utilisateur en cours

{

string Requete = "DELETE tbl\_Performances.\* FROM tbl\_Performances INNER JOIN tbl\_Personnes ON tbl\_Personnes.ID = tbl\_Performances.num\_tbl\_Personnes"; //Vu qu'il n'y a pas de clause WHERE, toute la tabel sera vidée. (Truncate ne fonctionne apparemment pas dans une base Access).

Requete += " WHERE " + "(tbl\_Personnes.ID = " + Params.Default.iID.ToString() + " );";

DbDataReader oRdr = BaseDonnee.Request(Requete);

Thread.Sleep(3000);

rbtn\_Personnel.PerformClick(); //Clique sur le bouton pour afficher les scores personnels, vu qu'on viens de supprimer tout les résulats de l'utilisateur en cours. ( Afin de rafraîchir)

}

}

private void update(object sender, KeyPressEventArgs e)

{ //Vérifie l'entrée clavier faite par l'utilisateur afin d'en retirer les caractères non désirés dans un numericUpDown

e.Handled = !char.IsDigit(e.KeyChar) && !char.IsControl(e.KeyChar);

Thread.Sleep(100);

dataGridViewData(sender, e);

}

}

}

### Jeu.cs

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Windows.Forms;

//Permets l'utilisation des utilitaires de base de données

using System.Data.Common;

using System.Data.OleDb;

//Permets l'utilisation du fichier de config

using System.Configuration;

namespace TPI

{

public partial class Jeu : Form

{

public Jeu()

{

InitializeComponent();

}

//Déclaration des variables permettant de compter le nombre de contacts lors d'un concours.

int iGauche = 0;

int iDroite = 0;

//Déclaration des variable permettant de dénombrer deux touchers distincts.

long lDebut; //Va permetre de compter le temps total de chaque concour afin de savoir quand l'arrêter.

long lDernier = Environment.TickCount;

long lNew;

DateTime dateDebut = DateTime.Now;

private void Jeu\_FormClosing(object sender, FormClosingEventArgs e)

{

//Permet de fermer correctement l'application sans causer d'erreurs

//dûes au fait qu'il y ait une connexion en cours avec la Carte.

K8055.CloseDevice();

Params.Default.bIsAlreadyConnected = false; //Repasse à false pour la prochaine ouverture du programme

Params.Default.Save();

Environment.Exit(0);

}

private void btnConnexion\_Click(object sender, EventArgs e)

{

int iEtat = 0; //Permet de détecter si la Carte sera connectée.

iEtat = K8055.OpenDevice(0); //Si iEtat = 0, carte connectée. Si iEtat = -1, carte non présente.

if (iEtat != 0)

{

MessageBox.Show("Veuillez d'abord connecter la carte à l'ordinateur via le cable USB.");

}

else //Si la connection a réussi

{

inverserBoutons(); //Débloque les boutons.

}

}

private void resetCompteur()

{

iGauche = 0;

iDroite = 0;

lblNombreDroit.Text = "0";

lblNombreGauche.Text = "0";

}

private void tmrJeu\_Tick(object sender, EventArgs e)

{

lNew = Environment.TickCount;

if (K8055.ReadDigitalChannel(5) == true)

{

if (lNew - lDernier >= Params.Default.iMillisecondes)

{

iDroite++;

lDernier = Environment.TickCount;

}

pbxRight.ImageLocation = @"right.png";

}

else if (K8055.ReadDigitalChannel(4) == true)

{

if (lNew - lDernier >= Params.Default.iMillisecondes)

{

iGauche++;

lDernier = Environment.TickCount;

}

pbxLeft.ImageLocation = @"left.png";

}

else

{

pbxRight.ImageLocation = @"rightx.png";

pbxLeft.ImageLocation = @"leftx.png";

}

lblNombreDroit.Text = iDroite.ToString();

lblNombreGauche.Text = iGauche.ToString();

if((Convert.ToInt32(lNew-lDebut) < progressBar1.Maximum))

{

progressBar1.Value = Convert.ToInt32(lNew - lDebut); //Valeur souhaitée

progressBar1.Value = Convert.ToInt32(lNew - lDebut - 1); //Valeur souhaitée -1

progressBar1.Value = Convert.ToInt32(lNew - lDebut); //Valeur finale souhaitée. Ceci pour éviter l'animation sur les progressbar, car lors d'un temps cours, exemple 5 secondes, le temps que l'animation de la probgressbar finisse, la partie était deja temrinée, n'affichant jamais la fin. Ceci corrige le problème.

}

progressBar1.Refresh();

using (Graphics gr = progressBar1.CreateGraphics())

{

gr.DrawString("Temps écoulé : " + (lNew - lDebut) / 1000 + " / " + Params.Default.iTempsPartie.ToString()+" seconde(s)",

SystemFonts.DefaultFont,

Brushes.Black,

new PointF(progressBar1.Width / 2 - (gr.MeasureString("Temps écoulé : " + (lNew - lDebut) / 1000 + " / " + Params.Default.iTempsPartie.ToString() + " seconde(s)",

SystemFonts.DefaultFont).Width / 2.0F),

progressBar1.Height / 2 - (gr.MeasureString("Temps écoulé : " + (lNew - lDebut) / 1000 + " / " + Params.Default.iTempsPartie.ToString() + " seconde(s)",

SystemFonts.DefaultFont).Height / 2.0F)));

}

//Teste si le concour est terminé (Nouveau temps - temps du debut)/1000 car c'est en millisecondes

if ((lNew - lDebut) / 1000 >= Params.Default.iTempsPartie)

{

StopJeu();

}

}

private void inverserBoutons()

{

//Inverse la propriété enabled de tout les boutons.

tsbConnecter.Enabled = !tsbConnecter.Enabled;

tsbEjecter.Enabled = !tsbEjecter.Enabled;

btnJouer.Enabled = !btnJouer.Enabled;

}

private void btnJouer\_Click(object sender, EventArgs e)

{

//Timer de jeu.

tmrJeu.Start();

btnJouer.Enabled = false;

lDebut = Environment.TickCount; //Va permettre de savoir quand terminer le concours.

progressBar1.Maximum = Params.Default.iTempsPartie \*1000; //x1000 pour que ce soit plus fluide lorsqu'il n'y a que 10 secondes par exemple, ce sera moins par "à-coup".

pnlJeu.Visible = true;

}

private void StopJeu()

{

//Teste si la performance doit être sauvegardée

if (Params.Default.bEntrainement == false) //Sauvegarde le concours seulement si l'utilisateur n'as pas coché l'option "Entraînement".

{

//Ajoute la performance

BaseDonnee.Request("INSERT INTO tbl\_Performances (num\_tbl\_Personnes,Jour,Mois,Annee,Difficulte,ResultatTotal,ResultatGauche,ResultatDroit,Temps,Heure) VALUES ('" + Params.Default.iID.ToString() + "','" + DateTime.Now.Day.ToString() + "','" + DateTime.Now.Month.ToString() + "','" + DateTime.Now.Year.ToString() + "','" + Params.Default.iDifficulte + "','" + (iGauche+iDroite) + "','" + iGauche + "','" + iDroite + "','" + Params.Default.iTempsPartie + "','" + DateTime.Now.ToString("HH:mm:ss") + "');");

}

tmrJeu.Stop();

btnJouer.Enabled = true;

pbxLeft.ImageLocation = @"";

pbxRight.ImageLocation = @"";

pnlJeu.Visible = false;

//Va afficher les résultats

this.Visible = false;

this.Hide();

Form Resultat = new Resultat(iGauche, iDroite);

resetCompteur(); //Se fait après avoir passé les paramètres au formulaire Resultat que l'on va afficher, pour ne pas perdre les valeurs.

Resultat.Show();

}

public void inverserCouleurMenu(ToolStripButton bouton1, ToolStripButton bouton2)

{

//Cette méthode va inverser la couleur des deux boutons passés en paramètres.

bouton1.BackColor = bouton2.BackColor;

bouton2.BackColor = Control.DefaultBackColor; //Remet la couleur de base du controle.

}

private void Jeu\_VisibleChanged(object sender, EventArgs e)

{

if (Params.Default.bIsAlreadyConnected == true)

{

tsbConnecter.PerformClick(); //Permet de recliquer sur le bouton lorsque le formulaire se recharge alors que la carte est déjà connectée, exemple après être revenu du formulaire Paramètres.

}

}

private void tsbConnecter\_Click(object sender, EventArgs e)

{

tsbConnecter.Enabled = false;

int iEtat = 0; //Permet de détecter si la Carte sera connectée.

iEtat = K8055.OpenDevice(0); //Si iEtat = 0, carte connectée. Si iEtat = -1, carte non présente.

tsbConnecter.Enabled = true;

if (iEtat != 0)

{

MessageBox.Show("Veuillez d'abord connecter la carte à l'ordinateur via le cable USB.");

}

else //Si la connection a réussi

{

inverserBoutons(); //Débloque les boutons

inverserCouleurMenu(tsbConnecter, tsbEjecter); //Inverse les couleurs du menu.

Params.Default.bIsAlreadyConnected = true;

Params.Default.Save();

}

}

private void tsbEjecter\_Click\_1(object sender, EventArgs e)

{

K8055.CloseDevice();

inverserBoutons();

inverserCouleurMenu(tsbEjecter, tsbConnecter);

Params.Default.bIsAlreadyConnected = false;

Params.Default.Save();

if(tmrJeu.Enabled)

{//La partie n'est stoppée que si elle est en cours.

StopJeu();

}

btnJouer.Enabled = false;

}

private void tsbParametres\_Click\_1(object sender, EventArgs e)

{

tmrJeu.Dispose(); //Stop net le timer de jeu, sans le laisser finir quoi que ce soit

this.Visible = false;

this.Hide();

Form Parametres = new Parametres();

Parametres.Show();

}

private void tsbScores\_Click\_1(object sender, EventArgs e)

{

tmrJeu.Dispose(); //Stop net le timer de jeu, sans le laisser finir quoi que ce soit

this.Visible = false;

this.Hide();

Form Scores = new Scores();

Scores.Show();

}

private void tsbChangerUtilisateur\_Click(object sender, EventArgs e)

{

tmrJeu.Dispose(); //Stop net le timer de jeu, sans le laisser finir quoi que ce soit

this.Visible = false;

this.Hide();

Form Login = new Login();

Login.Show();

}

}

}

### BaseDonnee.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

//Permets l'utilisation des utilitaires de base de données

using System.Data.Common;

using System.Data.OleDb;

//Permets l'utilisation du fichier de config

using System.Configuration;

using System.Threading;

namespace TPI

{

public class BaseDonnee

{

public static DbDataReader Request(string Request)

{

DbProviderFactory dbpf = DbProviderFactories.GetFactory("System.Data.Oledb");

DbConnection Connection = dbpf.CreateConnection();

Connection.ConnectionString = "Provider=Microsoft.ACE.OLEDB.12.0;Data source=TPI.mdb; Persist Security Info=False;";

DbCommand oCmd = Connection.CreateCommand();

oCmd.CommandText = Request;

DbDataReader oRdr = null;

try { Connection.Open(); } //Le Try-Catch Evite les erreurs si l'utilisateur rentre trop vite des frappes au clavier dans le formulaire Scores. ( A Cause de l'evenement TextChanged du numericUpDown)

catch { }

//Vérifie si la requête doit donner un retour

if (Request.IndexOf("SELECT") > -1)

{

try { oRdr = oCmd.ExecuteReader(); } //Le Try-Catch Evite les erreurs si l'utilisateur rentre trop vite des frappes au clavier dans le formulaire Scores. ( A Cause de l'evenement TextChanged du numericUpDown)

catch { }

}

else

{

oCmd.ExecuteNonQuery();

}

return oRdr;

}

}

}

### K8055D.cs

using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Runtime.InteropServices; //Permet d'accéder à la DLL

namespace TPI

{

public class K8055

{

[DllImport("K8055D.dll")]

public static extern int OpenDevice(int CardAddress);

[DllImport("K8055D.dll")]

public static extern void CloseDevice();

[DllImport("K8055D.dll")]

public static extern int ReadAnalogChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern void ReadAllAnalog(int Data1, int Data2);

[DllImport("K8055D.dll")]

public static extern void OutputAnalogChannel(int Channel, int Data);

[DllImport("K8055D.dll")]

public static extern void OutputAllAnalog(int Data1, int Data2);

[DllImport("K8055D.dll")]

public static extern void ClearAnalogChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern void SetAllAnalog();

[DllImport("K8055D.dll")]

public static extern void ClearAllAnalog();

[DllImport("K8055D.dll")]

public static extern void SetAnalogChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern void WriteAllDigital(int Data);

[DllImport("K8055D.dll")]

public static extern void ClearDigitalChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern void ClearAllDigital();

[DllImport("K8055D.dll")]

public static extern void SetDigitalChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern void SetAllDigital();

[DllImport("K8055D.dll")]

public static extern bool ReadDigitalChannel(int Channel);

[DllImport("K8055D.dll")]

public static extern int ReadAllDigital();

[DllImport("K8055D.dll")]

public static extern int ReadCounter(int CounterNr);

[DllImport("K8055D.dll")]

public static extern void ResetCounter(int CounterNr);

[DllImport("K8055D.dll")]

public static extern int SetCounterDebounceTime(int CounterNr, int DebounceTime);

}

}

### DataGridViewPrinter.cs

using System;

using System.Text;

using System.Collections;

using System.Collections.Generic;

using System.Drawing;

using System.Drawing.Printing;

using System.Data;

using System.Windows.Forms;

class DataGridViewPrinter

{

// The DataGridView Control which will be printed

private DataGridView TheDataGridView;

// The PrintDocument to be used for printing

private PrintDocument ThePrintDocument;

// Determine if the report will be

// printed in the Top-Center of the page

private bool IsCenterOnPage;

// Determine if the page contain title text

private bool IsWithTitle;

// The title text to be printed

// in each page (if IsWithTitle is set to true)

private string TheTitleText;

// The font to be used with the title

// text (if IsWithTitle is set to true)

private Font TheTitleFont;

// The color to be used with the title

// text (if IsWithTitle is set to true)

private Color TheTitleColor;

// Determine if paging is used

private bool IsWithPaging;

// A static parameter that keep track

// on which Row (in the DataGridView control)

// that should be printed

static int CurrentRow;

static int PageNumber;

private int PageWidth;

private int PageHeight;

private int LeftMargin;

private int TopMargin;

private int RightMargin;

private int BottomMargin;

// A parameter that keep track

// on the y coordinate of the page,

// so the next object to be printed

// will start from this y coordinate

private float CurrentY;

private float RowHeaderHeight;

private List<float> RowsHeight;

private List<float> ColumnsWidth;

private float TheDataGridViewWidth;

// Maintain a generic list to hold start/stop

// points for the column printing

// This will be used for wrapping

// in situations where the DataGridView will not

// fit on a single page

private List<int[]> mColumnPoints;

private List<float> mColumnPointsWidth;

private int mColumnPoint;

// The class constructor

public DataGridViewPrinter(DataGridView aDataGridView,

PrintDocument aPrintDocument,

bool CenterOnPage, bool WithTitle,

string aTitleText, Font aTitleFont,

Color aTitleColor, bool WithPaging)

{

TheDataGridView = aDataGridView;

ThePrintDocument = aPrintDocument;

IsCenterOnPage = CenterOnPage;

IsWithTitle = WithTitle;

TheTitleText = aTitleText;

TheTitleFont = aTitleFont;

TheTitleColor = aTitleColor;

IsWithPaging = WithPaging;

PageNumber = 0;

RowsHeight = new List<float>();

ColumnsWidth = new List<float>();

mColumnPoints = new List<int[]>();

mColumnPointsWidth = new List<float>();

// Claculating the PageWidth and the PageHeight

if (!ThePrintDocument.DefaultPageSettings.Landscape)

{

PageWidth =

ThePrintDocument.DefaultPageSettings.PaperSize.Width;

PageHeight =

ThePrintDocument.DefaultPageSettings.PaperSize.Height;

}

else

{

PageHeight =

ThePrintDocument.DefaultPageSettings.PaperSize.Width;

PageWidth =

ThePrintDocument.DefaultPageSettings.PaperSize.Height;

}

// Claculating the page margins

LeftMargin = ThePrintDocument.DefaultPageSettings.Margins.Left;

TopMargin = ThePrintDocument.DefaultPageSettings.Margins.Top;

RightMargin = ThePrintDocument.DefaultPageSettings.Margins.Right;

BottomMargin = ThePrintDocument.DefaultPageSettings.Margins.Bottom;

// First, the current row to be printed

// is the first row in the DataGridView control

CurrentRow = 0;

}

// The function that calculate

// the height of each row (including the header row),

// the width of each column (according

// to the longest text in all its cells including

// the header cell), and the whole DataGridView width

private void Calculate(Graphics g)

{

if (PageNumber == 0)

// Just calculate once

{

SizeF tmpSize = new SizeF();

Font tmpFont;

float tmpWidth;

TheDataGridViewWidth = 0;

for (int i = 0; i < TheDataGridView.Columns.Count; i++)

{

tmpFont = TheDataGridView.ColumnHeadersDefaultCellStyle.Font;

if (tmpFont == null)

// If there is no special HeaderFont style,

// then use the default DataGridView font style

tmpFont = TheDataGridView.DefaultCellStyle.Font;

tmpSize = g.MeasureString(

TheDataGridView.Columns[i].HeaderText,

tmpFont);

tmpWidth = tmpSize.Width;

RowHeaderHeight = tmpSize.Height;

for (int j = 0; j < TheDataGridView.Rows.Count; j++)

{

tmpFont = TheDataGridView.Rows[j].DefaultCellStyle.Font;

if (tmpFont == null)

// If the there is no special font style of the

// CurrentRow, then use the default one associated

// with the DataGridView control

tmpFont = TheDataGridView.DefaultCellStyle.Font;

tmpSize = g.MeasureString("Anything", tmpFont);

RowsHeight.Add(tmpSize.Height);

tmpSize =

g.MeasureString(

TheDataGridView.Rows[j].Cells[i].

EditedFormattedValue.ToString(),

tmpFont);

if (tmpSize.Width > tmpWidth)

tmpWidth = tmpSize.Width;

}

if (TheDataGridView.Columns[i].Visible)

TheDataGridViewWidth += tmpWidth;

ColumnsWidth.Add(tmpWidth);

}

// Define the start/stop column points

// based on the page width and

// the DataGridView Width

// We will use this to determine

// the columns which are drawn on each page

// and how wrapping will be handled

// By default, the wrapping will occurr

// such that the maximum number of

// columns for a page will be determine

int k;

int mStartPoint = 0;

for (k = 0; k < TheDataGridView.Columns.Count; k++)

if (TheDataGridView.Columns[k].Visible)

{

mStartPoint = k;

break;

}

int mEndPoint = TheDataGridView.Columns.Count;

for (k = TheDataGridView.Columns.Count - 1; k >= 0; k--)

if (TheDataGridView.Columns[k].Visible)

{

mEndPoint = k + 1;

break;

}

float mTempWidth = TheDataGridViewWidth;

float mTempPrintArea = (float)PageWidth - (float)LeftMargin -

(float)RightMargin;

// We only care about handling

// where the total datagridview width is bigger

// then the print area

if (TheDataGridViewWidth > mTempPrintArea)

{

mTempWidth = 0.0F;

for (k = 0; k < TheDataGridView.Columns.Count; k++)

{

if (TheDataGridView.Columns[k].Visible)

{

mTempWidth += ColumnsWidth[k];

// If the width is bigger

// than the page area, then define a new

// column print range

if (mTempWidth > mTempPrintArea)

{

mTempWidth -= ColumnsWidth[k];

mColumnPoints.Add(new int[] { mStartPoint, mEndPoint });

mColumnPointsWidth.Add(mTempWidth);

mStartPoint = k;

mTempWidth = ColumnsWidth[k];

}

}

// Our end point is actually

// one index above the current index

mEndPoint = k + 1;

}

}

// Add the last set of columns

mColumnPoints.Add(new int[] { mStartPoint, mEndPoint });

mColumnPointsWidth.Add(mTempWidth);

mColumnPoint = 0;

}

}

// The funtion that print the title, page number, and the header row

private void DrawHeader(Graphics g)

{

CurrentY = (float)TopMargin;

// Printing the page number (if isWithPaging is set to true)

if (IsWithPaging)

{

PageNumber++;

string PageString = "Page " + PageNumber.ToString();

StringFormat PageStringFormat = new StringFormat();

PageStringFormat.Trimming = StringTrimming.Word;

PageStringFormat.FormatFlags = StringFormatFlags.NoWrap |

StringFormatFlags.LineLimit | StringFormatFlags.NoClip;

PageStringFormat.Alignment = StringAlignment.Far;

Font PageStringFont = new Font("Tahoma", 8, FontStyle.Regular,

GraphicsUnit.Point);

RectangleF PageStringRectangle =

new RectangleF((float)LeftMargin, CurrentY,

(float)PageWidth - (float)RightMargin - (float)LeftMargin,

g.MeasureString(PageString, PageStringFont).Height);

g.DrawString(PageString, PageStringFont,

new SolidBrush(Color.Black),

PageStringRectangle, PageStringFormat);

CurrentY += g.MeasureString(PageString,

PageStringFont).Height;

}

// Printing the title (if IsWithTitle is set to true)

if (IsWithTitle)

{

StringFormat TitleFormat = new StringFormat();

TitleFormat.Trimming = StringTrimming.Word;

TitleFormat.FormatFlags = StringFormatFlags.NoWrap |

StringFormatFlags.LineLimit | StringFormatFlags.NoClip;

if (IsCenterOnPage)

TitleFormat.Alignment = StringAlignment.Center;

else

TitleFormat.Alignment = StringAlignment.Near;

RectangleF TitleRectangle =

new RectangleF((float)LeftMargin, CurrentY,

(float)PageWidth - (float)RightMargin - (float)LeftMargin,

g.MeasureString(TheTitleText, TheTitleFont).Height);

g.DrawString(TheTitleText, TheTitleFont,

new SolidBrush(TheTitleColor),

TitleRectangle, TitleFormat);

CurrentY += g.MeasureString(TheTitleText, TheTitleFont).Height;

}

// Calculating the starting x coordinate

// that the printing process will start from

float CurrentX = (float)LeftMargin;

if (IsCenterOnPage)

CurrentX += (((float)PageWidth - (float)RightMargin -

(float)LeftMargin) - mColumnPointsWidth[mColumnPoint]) / 2.0F;

// Setting the HeaderFore style

Color HeaderForeColor =

TheDataGridView.ColumnHeadersDefaultCellStyle.ForeColor;

if (HeaderForeColor.IsEmpty)

// If there is no special HeaderFore style,

// then use the default DataGridView style

HeaderForeColor = TheDataGridView.DefaultCellStyle.ForeColor;

SolidBrush HeaderForeBrush = new SolidBrush(HeaderForeColor);

// Setting the HeaderBack style

Color HeaderBackColor =

TheDataGridView.ColumnHeadersDefaultCellStyle.BackColor;

if (HeaderBackColor.IsEmpty)

// If there is no special HeaderBack style,

// then use the default DataGridView style

HeaderBackColor = TheDataGridView.DefaultCellStyle.BackColor;

SolidBrush HeaderBackBrush = new SolidBrush(HeaderBackColor);

// Setting the LinePen that will

// be used to draw lines and rectangles

// (derived from the GridColor property

// of the DataGridView control)

Pen TheLinePen = new Pen(TheDataGridView.GridColor, 1);

// Setting the HeaderFont style

Font HeaderFont = TheDataGridView.ColumnHeadersDefaultCellStyle.Font;

if (HeaderFont == null)

// If there is no special HeaderFont style,

// then use the default DataGridView font style

HeaderFont = TheDataGridView.DefaultCellStyle.Font;

// Calculating and drawing the HeaderBounds

RectangleF HeaderBounds = new RectangleF(CurrentX, CurrentY,

mColumnPointsWidth[mColumnPoint], RowHeaderHeight);

g.FillRectangle(HeaderBackBrush, HeaderBounds);

// Setting the format that will be

// used to print each cell of the header row

StringFormat CellFormat = new StringFormat();

CellFormat.Trimming = StringTrimming.Word;

CellFormat.FormatFlags = StringFormatFlags.NoWrap |

StringFormatFlags.LineLimit | StringFormatFlags.NoClip;

// Printing each visible cell of the header row

RectangleF CellBounds;

float ColumnWidth;

for (int i = (int)mColumnPoints[mColumnPoint].GetValue(0);

i < (int)mColumnPoints[mColumnPoint].GetValue(1); i++)

{

// If the column is not visible then ignore this iteration

if (!TheDataGridView.Columns[i].Visible) continue;

ColumnWidth = ColumnsWidth[i];

// Check the CurrentCell alignment

// and apply it to the CellFormat

if (TheDataGridView.ColumnHeadersDefaultCellStyle.

Alignment.ToString().Contains("Right"))

CellFormat.Alignment = StringAlignment.Far;

else if (TheDataGridView.ColumnHeadersDefaultCellStyle.

Alignment.ToString().Contains("Center"))

CellFormat.Alignment = StringAlignment.Center;

else

CellFormat.Alignment = StringAlignment.Near;

CellBounds = new RectangleF(CurrentX, CurrentY,

ColumnWidth, RowHeaderHeight);

// Printing the cell text

g.DrawString(TheDataGridView.Columns[i].HeaderText,

HeaderFont, HeaderForeBrush,

CellBounds, CellFormat);

// Drawing the cell bounds

// Draw the cell border only if the HeaderBorderStyle is not None

if (TheDataGridView.RowHeadersBorderStyle !=

DataGridViewHeaderBorderStyle.None)

g.DrawRectangle(TheLinePen, CurrentX, CurrentY, ColumnWidth,

RowHeaderHeight);

CurrentX += ColumnWidth;

}

CurrentY += RowHeaderHeight;

}

// The function that print a bunch of rows that fit in one page

// When it returns true, meaning that

// there are more rows still not printed,

// so another PagePrint action is required

// When it returns false, meaning that all rows are printed

// (the CureentRow parameter reaches

// the last row of the DataGridView control)

// and no further PagePrint action is required

private bool DrawRows(Graphics g)

{

// Setting the LinePen that will be used to draw lines and rectangles

// (derived from the GridColor property of the DataGridView control)

Pen TheLinePen = new Pen(TheDataGridView.GridColor, 1);

// The style paramters that will be used to print each cell

Font RowFont;

Color RowForeColor;

Color RowBackColor;

SolidBrush RowForeBrush;

SolidBrush RowBackBrush;

SolidBrush RowAlternatingBackBrush;

// Setting the format that will be used to print each cell

StringFormat CellFormat = new StringFormat();

CellFormat.Trimming = StringTrimming.Word;

CellFormat.FormatFlags = StringFormatFlags.NoWrap |

StringFormatFlags.LineLimit;

// Printing each visible cell

RectangleF RowBounds;

float CurrentX;

float ColumnWidth;

while (CurrentRow < TheDataGridView.Rows.Count)

{

// Print the cells of the CurrentRow only if that row is visible

if (TheDataGridView.Rows[CurrentRow].Visible)

{

// Setting the row font style

RowFont = TheDataGridView.Rows[CurrentRow].DefaultCellStyle.Font;

// If the there is no special font style of the CurrentRow,

// then use the default one associated with the DataGridView control

if (RowFont == null)

RowFont = TheDataGridView.DefaultCellStyle.Font;

// Setting the RowFore style

RowForeColor =

TheDataGridView.Rows[CurrentRow].DefaultCellStyle.ForeColor;

// If the there is no special RowFore style of the CurrentRow,

// then use the default one associated with the DataGridView control

if (RowForeColor.IsEmpty)

RowForeColor = TheDataGridView.DefaultCellStyle.ForeColor;

RowForeBrush = new SolidBrush(RowForeColor);

// Setting the RowBack (for even rows) and the RowAlternatingBack

// (for odd rows) styles

RowBackColor =

TheDataGridView.Rows[CurrentRow].DefaultCellStyle.BackColor;

// If the there is no special RowBack style of the CurrentRow,

// then use the default one associated with the DataGridView control

if (RowBackColor.IsEmpty)

{

RowBackBrush = new SolidBrush(

TheDataGridView.DefaultCellStyle.BackColor);

RowAlternatingBackBrush = new

SolidBrush(

TheDataGridView.AlternatingRowsDefaultCellStyle.BackColor);

}

// If the there is a special RowBack style of the CurrentRow,

// then use it for both the RowBack and the RowAlternatingBack styles

else

{

RowBackBrush = new SolidBrush(RowBackColor);

RowAlternatingBackBrush = new SolidBrush(RowBackColor);

}

// Calculating the starting x coordinate

// that the printing process will

// start from

CurrentX = (float)LeftMargin;

if (IsCenterOnPage)

CurrentX += (((float)PageWidth - (float)RightMargin -

(float)LeftMargin) -

mColumnPointsWidth[mColumnPoint]) / 2.0F;

// Calculating the entire CurrentRow bounds

RowBounds = new RectangleF(CurrentX, CurrentY,

mColumnPointsWidth[mColumnPoint], RowsHeight[CurrentRow]);

// Filling the back of the CurrentRow

if (CurrentRow % 2 == 0)

g.FillRectangle(RowBackBrush, RowBounds);

else

g.FillRectangle(RowAlternatingBackBrush, RowBounds);

// Printing each visible cell of the CurrentRow

for (int CurrentCell = (int)mColumnPoints[mColumnPoint].GetValue(0);

CurrentCell < (int)mColumnPoints[mColumnPoint].GetValue(1);

CurrentCell++)

{

// If the cell is belong to invisible

// column, then ignore this iteration

if (!TheDataGridView.Columns[CurrentCell].Visible) continue;

// Check the CurrentCell alignment

// and apply it to the CellFormat

if (TheDataGridView.Columns[CurrentCell].DefaultCellStyle.

Alignment.ToString().Contains("Right"))

CellFormat.Alignment = StringAlignment.Far;

else if (TheDataGridView.Columns[CurrentCell].DefaultCellStyle.

Alignment.ToString().Contains("Center"))

CellFormat.Alignment = StringAlignment.Center;

else

CellFormat.Alignment = StringAlignment.Near;

ColumnWidth = ColumnsWidth[CurrentCell];

RectangleF CellBounds = new RectangleF(CurrentX, CurrentY,

ColumnWidth, RowsHeight[CurrentRow]);

// Printing the cell text

g.DrawString(

TheDataGridView.Rows[CurrentRow].Cells[CurrentCell].

EditedFormattedValue.ToString(), RowFont, RowForeBrush,

CellBounds, CellFormat);

// Drawing the cell bounds

// Draw the cell border only

// if the CellBorderStyle is not None

if (TheDataGridView.CellBorderStyle !=

DataGridViewCellBorderStyle.None)

g.DrawRectangle(TheLinePen, CurrentX, CurrentY,

ColumnWidth, RowsHeight[CurrentRow]);

CurrentX += ColumnWidth;

}

CurrentY += RowsHeight[CurrentRow];

// Checking if the CurrentY is exceeds the page boundries

// If so then exit the function and returning true meaning another

// PagePrint action is required

if ((int)CurrentY > (PageHeight - TopMargin - BottomMargin))

{

CurrentRow++;

return true;

}

}

CurrentRow++;

}

CurrentRow = 0;

// Continue to print the next group of columns

mColumnPoint++;

if (mColumnPoint == mColumnPoints.Count)

// Which means all columns are printed

{

mColumnPoint = 0;

return false;

}

else

return true;

}

// The method that calls all other functions

public bool DrawDataGridView(Graphics g)

{

try

{

Calculate(g);

DrawHeader(g);

bool bContinue = DrawRows(g);

return bContinue;

}

catch (Exception ex)

{

MessageBox.Show("Operation failed: " + ex.Message.ToString(),

Application.ProductName + " - Error", MessageBoxButtons.OK,

MessageBoxIcon.Error);

return false;

}

}

}

## Références

Classe C# regroupant les fonctions liées à la carte 8055 :

<http://www.protung.ro/2009/08/velleman-k8055-c-test-application/>

Empêcher le copier/coller dans les contrôles  (Matthew Gertz):

<http://social.msdn.microsoft.com/Forums/en-US/vbide/thread/1bf46245-4696-44f0-ab36-5fb51cb546cd/>

Icônes des menus :

<http://findicons.com/>

Intercepter les touches dans un contrôle et les filtrer : (BFree)

<http://stackoverflow.com/questions/463299/how-do-i-make-a-textbox-that-only-accepts-numbers>

Mettre du texte dans une progressBar :

<http://www.codeproject.com/Articles/31406/Add-the-Percent-or-Any-Text-into-a-Standard-Progre>

Vérifier la validité d’une date :

<http://msdn.microsoft.com/fr-fr/library/vstudio/ch92fbc1.aspx>

Dwenn Kaufmann, le mardi 28 mai 2013.