



Rendu de projet

4 Memory

Module « Interface Homme Machine »

Antoine BANHA
Groupe 102

DUT 1ère Année



Table des matières

Table des matières.....	2
Introduction.....	3
Les formulaires.....	4
Le menu principal.....	4
Le paramètres.....	5
Le classement.....	8
Le jeu.....	10
Note technique sur le cartes.....	12
Enregistrement des joueurs.....	13
LoadPlayers.....	13
SavePlayers.....	14
Ordonnancement des formulaires.....	15
Conclusion.....	16



Introduction

Ce projet a consisté à reproduire le célèbre jeu de *Ravensburger* : le « Memory » en Visual Basic.NET, un langage que je n'avais jamais vraiment touché auparavant (je l'ai tout de même utilisé à la période précédente pour cette même matière, du VBA sur Access).

J'ai pu remettre au goût du jour ce fameux jeu grâce à l'exploitation de certains aspects du jeu de plateau moderne sur ordinateur comme l'implémentation de thèmes utilisateur et notamment un mode de triche à l'activation très 1986.

Commençons avec la première interface utilisateur...

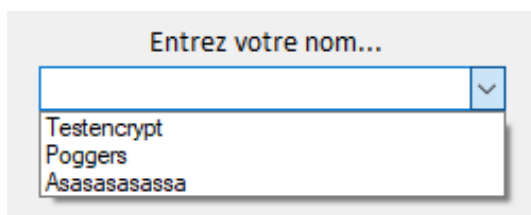




Les formulaires

Le menu principal

Le menu principal du 4 Memory consiste de 5 boutons et un champ « ComboBox » permettant de rechercher les derniers joueurs et d'ainsi jouer à plusieurs reprises avec le même nom et même statistiques de jeu.




(pas d'idées pour les pseudos 🤖)



Ce menu vous présente 4 boutons permettant de naviguer dans l'interface du 4 Memory :

- **Jouer** permet de lancer une partie avec le nom entré par l'utilisateur dans la ComboBox.
 - Le nom du joueur doit contenir au moins 3 caractères.
 - Le nom est en suite capitalisé en « ProperCase ».
 - Le nom du joueur ne doit pas contenir les caractères de séparations qui sont utilisés pour sauvegarder les données du joueur.
 - Par défaut : « | | ».
- **Paramètres** permet d'accéder aux paramètres du jeu.
- **Scores** permet d'accéder au classement local sur le jeu.
- **Quitter** permet de quitter l'application avec une confirmation.

- Une confirmation est également demandée quand la fermeture de la fenêtre est engendrée par un « ALT+F4 » ou le bouton de fermeture 

Le paramètres

Le formulaire de paramétrage du 4 Memory contient plusieurs options modifiant l'aspect du jeu :

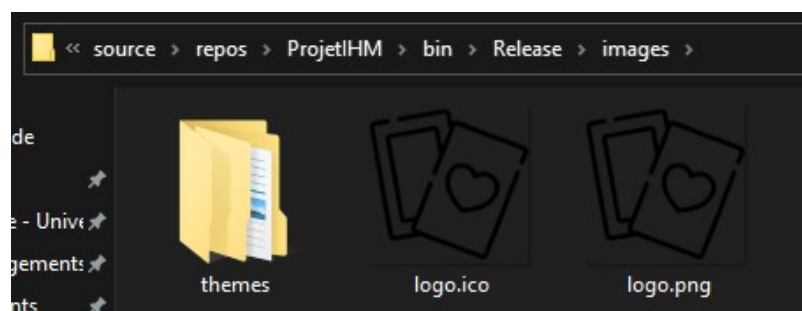


Les paramètres sont enregistrés dans le **registre** pour ainsi pouvoir sauvegarder de façon globale les paramètres du jeu. Les clés sont enregistrés sous la ruche `HKEY_CURRENT_USER/SOFTWARE` pour ainsi permettre chaque utilisateur de l'ordinateur d'avoir leur propre paramètres de jeu.

Il est possible alors de modifier :

- Le **répertoire** de sauvegarde des joueurs.
 - Le bouton « Parcourir » permet d'ouvrir une fenêtre de sélection de répertoire.
 - Une fois un répertoire sélectionné, un nouveau dossier appelé « 4Memory_Data » est créé contenant un fichier « players.ihm » qui décrira ligne par ligne, chaque joueur enregistrés.

- Un dossier a été utilisé au cas où plusieurs fichiers serait nécessaire plus tard dans un potentiel développement de l'application.
- Les joueurs sont enregistrés de façon à ce qu'un utilisateur lambda ne puisse pas les modifier sans avoir une clé privée écrite en dur dans le programme (pas très sécurisé car il y a la possibilité de décompiler le programme avec un outil comme *dotPeek* et obtenir ainsi la clé privée)
- Le répertoire est enregistré dans le registre avec la clé « SaveDirectory ».
- La **durée d'une partie** de 4 Memory.
 - Une limite de 59 minutes et 59 secondes est imposée pour éviter de partir dans des extrêmes.
 - Le texte est remplacé si cette restriction n'est pas respectée.
 - La durée est enregistrée dans le registre avec la clé « GameTime » et est formatée avec un séparateur pour ainsi être en mesure de *Split* au moment du chargement des paramètres.
- L'autorisation de **pause** pendant une partie.
 - 2 états possible car c'est une *CheckBox*, *True* ou *False*.
 - Si la pause est à *True*, le bouton Pause de l'interface de jeu sera activé, sinon celui-ci ne sera pas activé en jeu.
 - L'autorisation de pause est enregistrée dans le registre avec la clé « Pause ».
- Le **thème** du jeu.
 - Un thème est un dossier d'images présentes dans le dossier où se situe l'exécutable du programme sous le repertoire « images/themes ».



- Il est très facile de créer son propre thème : il suffit de créer un dossier dans le dossier « themes » et d'y insérer les **6 images** nécessaires au bon fonctionnement du jeu.
 - Les images doit avoir exactement les noms :
 - *cartes[0-4].png* pour les cartes.
 - *cartedefault.png* pour la face cachée des cartes.
 - Si une image est manquante, le thème ne sera pas affiché dans le *Dropdown* de sélection de thème.
- La prévisualisation du thème est rafraîchie lors du changement de thème :



- Le nom du thème sélectionné est enregistré dans la registre avec la clé « Theme ».

Pendant le développement du jeu, j'ai eu besoin de voir la génération des cartes pour ainsi vérifier que les sets se génèrent correctement. Pour cela, j'ai implémenté un bouton secret permettant alors d'afficher/cacher les cartes restantes de la grille et d'ainsi **tricher**.

Ce mode de triche est complètement **disponible** mais son activation n'est pas à la portée de tout le monde !

Il faut utiliser le **Code Konami** sur le formulaire des paramètres pour ainsi afficher le merveilleux bouton sur l'interface de jeu.



Selon Wikipédia :

Ce code peut être utilisé dans de nombreux [jeux vidéo](#) de Konami, activant généralement une option secrète. Au cours du jeu, le joueur doit mettre la partie en pause, puis presser sur sa [manette de jeu](#) la séquence de boutons suivante :



Attention

Aucunes statistiques n'est sauvegardée quand le mode triche est activé, même si le bouton n'est pas utilisé pendant la partie. On vous voit les tricheurs.

Le classement

Le programme va garder en mémoire pendant les parties plusieurs paramètres :

- Le nom du joueur saisi par l'utilisateur avant de démarrer la partie.
 - Si le nom du joueur n'existe pas dans la mémoire, celui-ci est initialisé.
- Les carrés max trouvés par le joueur, c'est-à-dire, le plus grand nombre de carrés trouvés par le joueur pendant une partie.
- Le temps pris par le joueur pour avoir trouvé les carrés max.
- Les parties jouées par le joueur.
- Le temps cumulé en jeu.

Rechercher un joueur

Affichage décroissant

	Nom	Carrés max	Temps pris	Parties jouées	Temps cumulé
	Testencrypt	5	00:52	1	00:00:54
	Aaaaaa	0	00:00	0	00:00:11
▶	Ggg	0	00:00	0	00:00:00

Classement disponible en cliquant le bouton « Scores » du menu principal.

Il est possible de consulter sur ce formulaire les données des joueurs sur le tableau mais également sur la ComboBox qui viendra alors auto-compléter avec le nom des joueurs.

Les données du tableau ci-dessus sont stockées dans une *DataTable*.

L'affichage des données se fait par une *DataGridView* car celle-ci est facile à utiliser, je peux alors créer des colonnes facilement en appelant par exemple :

```
playersTable.Columns.Add(New DataColumn("Carrés max", GetType(Integer)))
```

Pour l'insertion des données, j'utilise alors les *Rows* de ma *DataTable* : j'itère dans mon tableau de joueur et j'ajoute une ligne à la *DataTable*.

En suite, pour afficher et ordonner le tableau, je viens assigner la *DataSource* à la *DefaultView* de la *DataTable* qui est ordonnée au préalable grâce à une syntaxe ressemblant de près ou de loin à l'instruction « ORDER BY » du SQL :

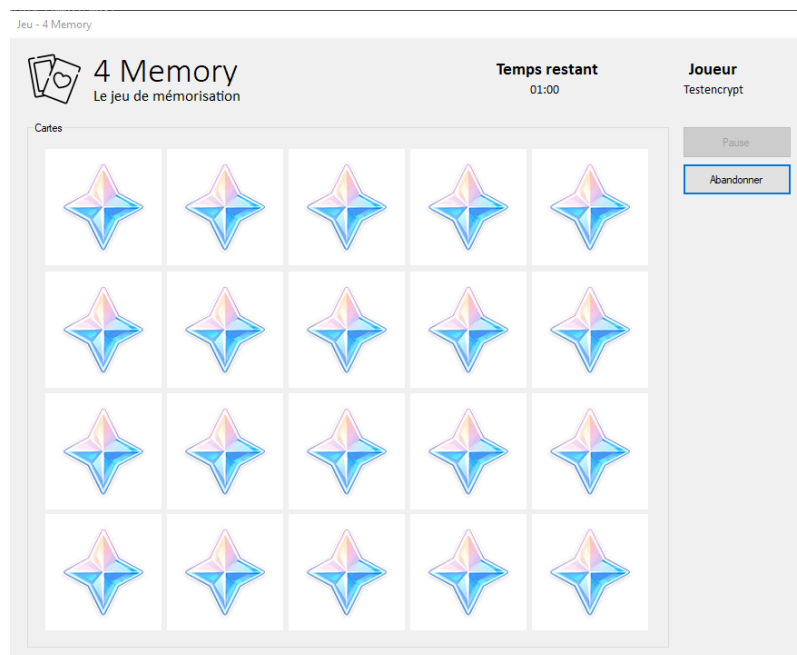
```
"Carrés max DESC, Temps pris ASC"
```

Pour inverser l'ordre d'affichage avec le bouton au dessus du tableau, il me suffit alors de modifier cette expression avec « ASC » ou « DESC ».

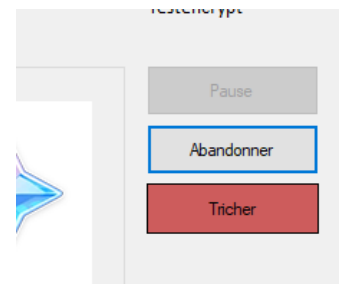
Les *DataTable* pour moi étaient le meilleur moyen de formater les données joueurs et j'ai eu une très bonne expérience avec.

Le jeu

Après avoir rempli les formalités du formulaire d'accueil pour pouvoir jouer, il est temps de passer sur le formulaire de jeu.



Nous pouvons également apercevoir le temps restant, le nom du joueur ainsi qu'un bouton « **Pause** » et « **Abandonner** » et si le Mode Triche est activé, un bouton « **Triche** ».



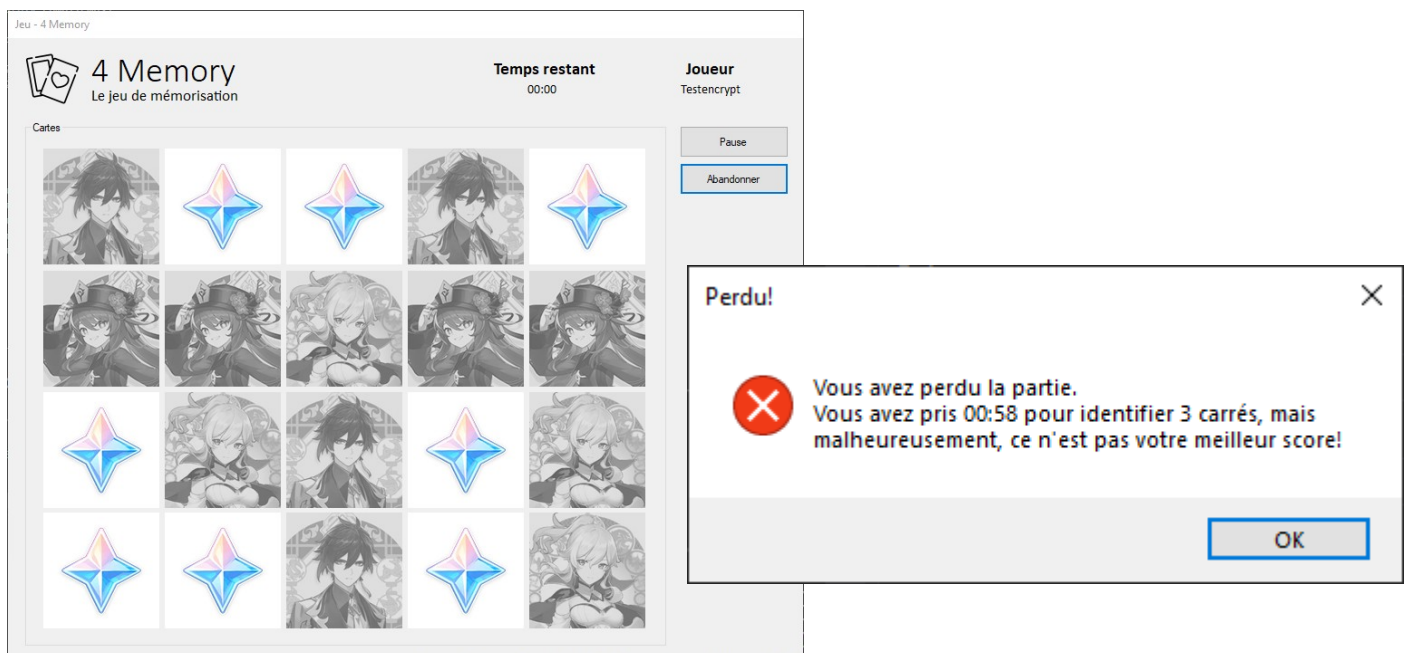
Une grille de taille 5 par 4 est alors générée avec le thème choisi pour l'utilisateur dans les paramètres, ici, le thème « genshin » a été sélectionné.

La génération entraîne une randomisation d'une liste de « Card » qui à ce moment là ne fait aucune référence aux labels présent sur l'IHM. Après que la randomisation ait eu lieu, on vient itérer dans les labels et on vient alors les référencer dans le tableau de « Card », c'est également à ce moment là qu'on vient leur donner le « AddHandler » permettant de déclencher un Sub quand il y a un évènement de click.

Une fois qu'un set de carte est trouvé, les cartes sont alors grisées et ne peuvent plus être dévoilées ou cachées par le mode triche.



L'ensemble du jeu fonctionne et la détection de fin de partie mentionne également certains détails sur la partie, comme le temps pris pour avoir trouvé les sets de cartes ou un gentil message si le mode triche était activé.



Note technique sur le cartes

Les cartes sont stockées dans une classe « Card » ayant comme propriétés, un identifiant, le label, l'image par défaut, le nom du set de l'image (par exemple, carte1, carte2 etc.) et un booléen pour l'affichage de la carte sur le jeu.

Quand le joueur vient cliquer une carte, celle-ci se retourne et dévoile alors l'image stockée dans l'attribut « image ». Si la prochaine carte cliquée fait partie du même imageSet que la précédente, alors on continue, sinon, les images affichées restent affichées pendant 1 seconde. Pour réaliser cette attente sans affecter le temps restant et les événements de clic, j'ai utilisé une fonction utilisant un DoWhile et une variable d'ignorance du clic qui est mise à « True » quand cette période de 1 seconde se déroule.

```
Public Sub Wait(ByVal seconds As Single)
    Static start As Single
    start = Microsoft.VisualBasic.Timer()
    Do While Microsoft.VisualBasic.Timer() < start + seconds
        Application.DoEvents()
    Loop
End Sub
```

Fonction utilisée pour l'attente.



Enregistrement des joueurs

Le programme garde en mémoire plusieurs statistiques déjà énumérée dans la partie parlant du classement des joueurs, nous allons ici nous focaliser sur l'enregistrement des joueurs dans le fichier « players.ihm ».

Pour gérer au mieux les joueurs, j'ai décidé de créer un module « PlayersModule » qui contiendra alors la classe « Player » contenant les statistiques de celui-ci mais également une méthode « LoadPlayer » qui viendra lire la ligne cryptographiée du fichier texte pour ainsi la décoder, la diviser par son séparateur et changer les variables de cette instance Player. J'ai également mis en œuvre le processus inverse dans une autre méthode pour ainsi récupérer le joueur sous format cryptographié. J'ai utilisé le module disponible sur la documentation de la cryptographie .NET pour ainsi cryptographier les données facilement.

Le module ne possède alors qu'à lui seul le tableau des joueurs mais également 2 fonctions utilisée par le jeu : « LoadPlayers » et « SavePlayers ».

LoadPlayers

La fonction LoadPlayers permet de lire les joueurs dans le fichier « players.ihm » du répertoire sélectionné dans les paramètres du jeu. Si le répertoire sélectionné est vide, nous allons alors créer le fichier vide. Dans le cas où le fichier contient des joueurs, nous lisons le fichier avec **StreamReader** qui lira alors ligne par ligne le fichier : c'est à ce moment là qu'on instancie la classe Player et que j'utilise la méthode LoadPlayer de cette instance et en suite, on ajoute ce joueur dans le tableau. Une fois la lecture du fichier terminée, nous fermons le fichier avec la méthode Close du StreamReader.

Cette fonction est exécutée à chaque lancement de l'application et en cas de rafraîchissement des données actuelles.

La fonction SavePlayers permet de sauvegarder les joueurs dans le fichier « players.ihm ». Dans le cas où le fichier n'existe pas, nous le créons. En suite, nous faisons une itération dans le tableau des joueurs pour récupérer leur String à partir de la méthode ToString de l'instance. Nous récupérons alors la version cryptographiée du joueur qui est alors enregistrée sur la ligne actuellement sélectionnée par le **StreamWriter**. Après l'écriture, nous fermons le fichier avec sa méthode Close et nous en profitons pour recharger les joueurs et actualiser les scores du formulaires de classement.

Cette solution aurait pu être remplacée par une sauvegarde dans un fichier binaire. Encore une fois, la cryptographie utilisée pour la sauvegarde des joueurs n'est vraiment pas sûre sachant qu'elle utilise une clé privée qui est écrite en dur dans le programme. J'ai à ma disposition plusieurs solutions mais qui s'avèrent au même niveau que graver la clé privée en dur dans le programme : par exemple, nous pourrions générer la clé privée et la stocker dans le registre Windows mais celle-ci serait alors retrouvée encore plus facilement que de décompiler le programme sur dotPeek.

 players.ihm - Bloc-notes

Fichier Edition Format Affichage Aide

```
JT3sX3xws0qzJMM1/Rn12Bjn+TS9WVEn1zb3y3JDKGX53KOVbcjt3GS/gicoSQJFTlu9oMduMwY=
J5wVnhNcX074DA22dEXSxTNCKmoK5u+E2IIhE61SbPLqoe3xuRHp3A==
vpq1xu1TPZwa71RfmON1qySoisGdFa975KLiOEK9WzhJO9fqLVWcLaFGBeybR8KsXKcNmFSvv2k=
D+zpTzrfaVCcD9G0EIw8v9FgWdQxcSHEj95iE9Zt3R3pjpIUgrOIBCZRKqBjV8BP
```

Exemple de sauvegarde des joueurs.

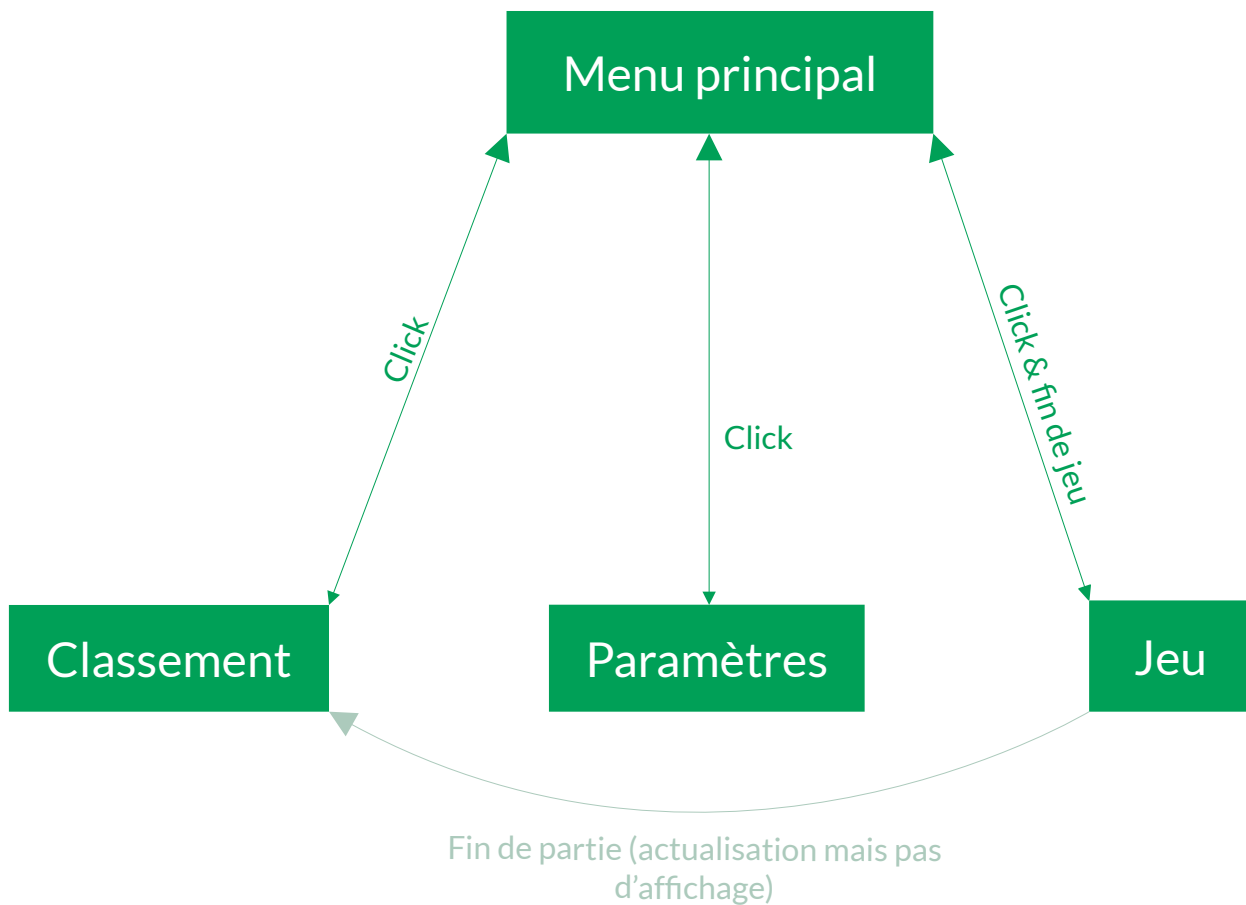
Une ligne décryptée ressemblerai alors plutôt à :

Antoine| |5| |52| |1| |54

Les temps sont stockés en secondes et sont convertis à la volée.



Ordonnancement des formulaires



Note technique

Le classement est rafraîchi par les fins de parties, les changements de répertoire et le démarrage du jeu par le module PlayerModule.



Conclusion

Encore une fois, le programme n'est pas très sécurisé contre les tricheurs qui s'y connaissent un peu en décompilation. Beaucoup de solutions sont possibles pour contrer au maximum ce soucis mais considérant que ce projet est une sorte d'introduction au VB.NET, je ne suis pas allé plus loin.

La sauvegarde des joueurs n'est pas si conventionnelle que ça, j'aurais pu passer par une sauvegarde dans une base de donnée ou encore, un fichier .XML, malgré tout, je pense que ma solution est assez sécurisée contre les petits malfrats.

L'interface utilisateur serait également à améliorer avec une étude UX un peu plus poussée, mais encore, c'est peut-être un peu trop avancé pour le moment. On pourrait alors parler de « persona » et ainsi imaginer qui utiliserait notre jeu.

J'aurai pu également ajouter une sorte de tutoriel avec une MessageBox Yes/No pour les nouveaux joueurs enregistrés dans le tableau des joueurs.

Ce premier projet en VB.NET a été super intéressant et a pu réellement mettre en application ce que j'ai appris pendant les cours, les TP mais également mes connaissances personnelles car j'avais utilisé pour le premier projet Access, du VBA.

Merci d'avoir lu mon rendu !

Antoine BANHA

Groupe 102

DUT 1ère Année