

Un jeu du Sudoku

SAE 2.01- Développement d'une
application

- Groupe ■■■■ -

■■■■■ ■■■■■, ■■■■■ ■■■■■,
■■■■■■ ■■■■■■■, ■■■■■ ■■■■■

9 juin 2024

Table des matières

1	Déroulement de l'application	2
1.1	Page d'accueil	2
1.2	Partie de jeu	2
1.3	Partie gagné/perdu	3
1.4	Paramètres	3
2	Description des fonctionnalités implémentées	4
2.1	Données de joueur persistantes	4
2.2	Système de paramètres en fonction du nombre de cases	4
2.3	Ajout d'un bouton de pause	5
2.4	Synchronisation du joueur entre la page d'accueil et la page de statistiques	5
2.5	Système de génération de Sudoku aléatoire faisable	6
3	Modification des attentes	7
3.1	Images et sons	7
3.2	Redimensionnement des interfaces	7
4	Structure des données et organisation des fichiers	7
4.1	Structure des données dans le module	7
4.2	Données des joueurs	7
4.3	Données de la grille de Sudoku	7
4.4	Gestion des fichiers	8
4.5	Chargement des données	8
4.6	Sauvegarde des données	8
4.7	Gestion des paramètres de configuration	8
4.8	Schéma	9
5	Code Source	9
5.1	Main.vb	9
5.2	Accueil.vb	9
5.3	Jeu.vb	12
5.4	SettingsForm.vb	15
5.5	StatisticsForm.vb	18
5.6	Settings.vb	22
5.7	PlayerDataModule.vb	23
5.8	SudokuGrid.vb	26

1 Déroulement de l'application

1.1 Page d'accueil

Sur la page d'accueil, l'utilisateur saisit son pseudo ou le sélectionne pour jouer. L'utilisateur clique ensuite sur "Jouer" et la partie se lance.

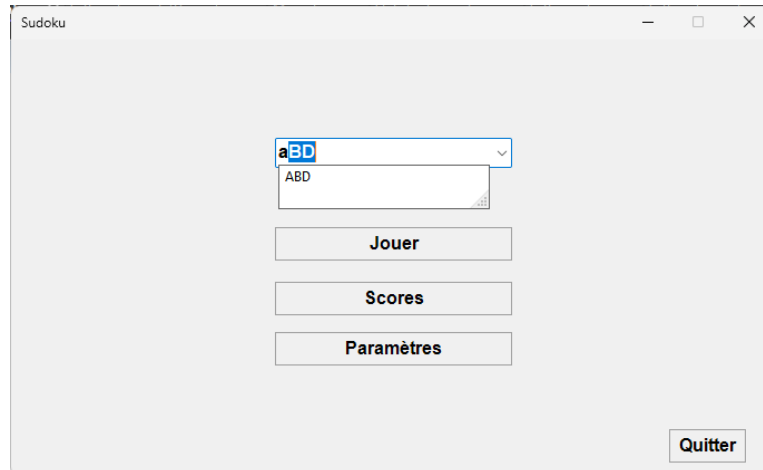


FIGURE 1 – Formulaire Accueil.vb

1.2 Partie de jeu

Le joueur a 7 minutes pour trouver la solution à ce Sudoku. Il est possible que le joueur trouve cela trop simple ou beaucoup trop difficile.

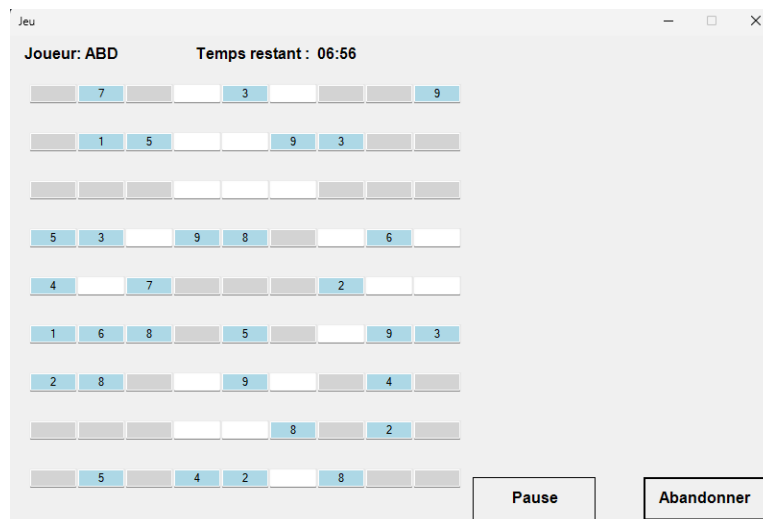
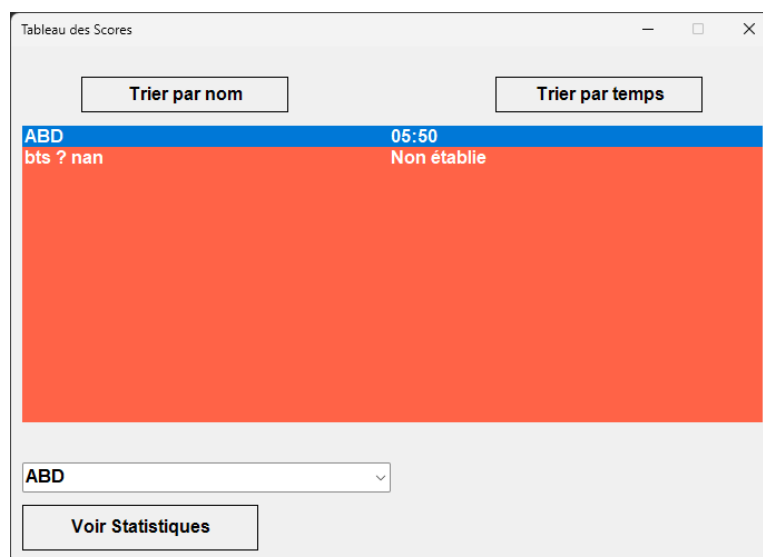


FIGURE 2 – Formulaire Jeu.vb

1.3 Partie gagné/perdu

Félicitations, il vient de gagner en seulement 5 minutes et 50 secondes ! Il se compare désormais aux autres joueurs enregistrés sur l'ordinateur. Cependant, s'il perd ou abandonne, il est redirigé vers la page d'accueil.



Nom	Temps
ABD	05:50

bts ? nan Non établie

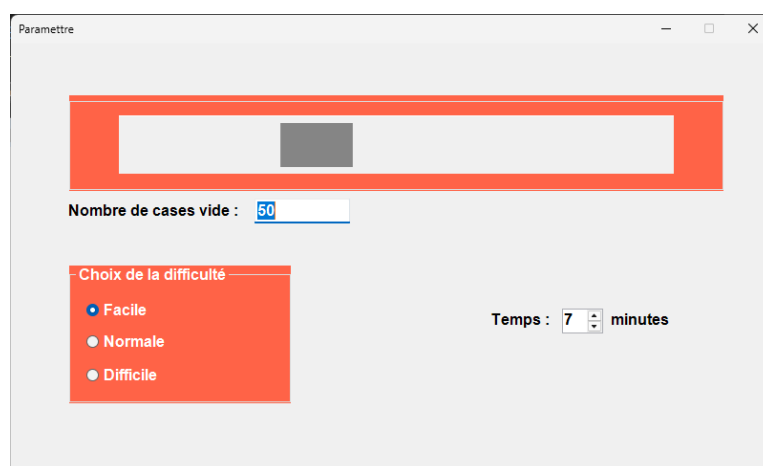
ABD

Voir Statistiques

FIGURE 3 – Formulaire StatisticsForm.vb

1.4 Paramètres

Notre joueur a trouvé la partie trop dure ou trop facile ? Il lui suffit alors d'aller dans les paramètres depuis la page d'accueil.



Paramètre

Nombre de cases vide : 50

Choix de la difficulté

- ☒ Facile
- ☐ Normale
- ☐ Difficile

Temps : 7 minutes

FIGURE 4 – Formulaire SettingsForm.vb

2 Description des fonctionnalités implémentées

2.1 Données de joueur persistantes

Cette fonctionnalité permet de sauvegarder et de restaurer les données des joueurs, telles que les scores et les statistiques. Les données sont stockées de manière persistante pour garantir qu'elles sont conservées entre les sessions de jeu.

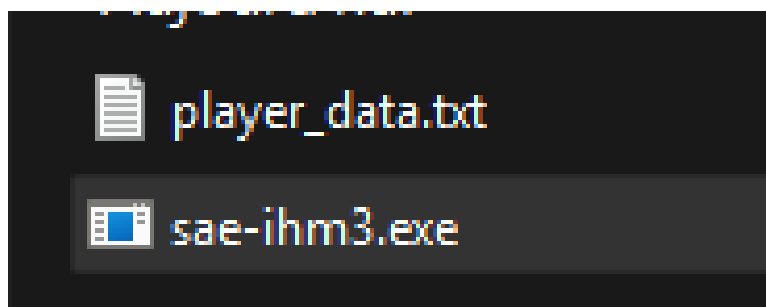


FIGURE 5 – Fichier player_data.txt

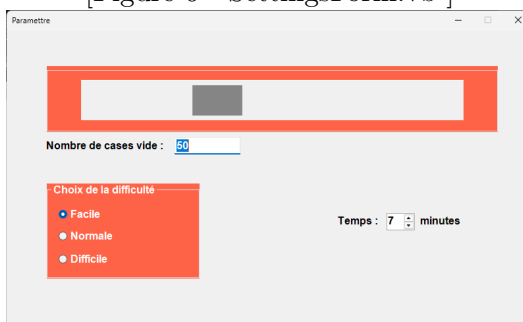
Les données des joueurs sont stockées dans un fichier texte nommé "player_data.txt".

2.2 Système de paramètres en fonction du nombre de cases

Un système de paramètres a été implémenté pour permettre aux utilisateurs de personnaliser le nombre de cases dans le jeu de Sudoku.

- Sélection du nombre de cases à remplir par le joueur, grâce à des radio-boutons, une scrollbar et une textbox synchronisés.
- Les utilisateurs peuvent désormais modifier le temps imparti pour résoudre un Sudoku. Cette fonctionnalité permet de définir des limites de temps différentes en fonction des préférences et des niveaux de compétence des joueurs.

[Figure 6 - SettingsForm.vb]



2.3 Ajout d'un bouton de pause

Un bouton de pause a été ajouté pour permettre aux joueurs de mettre le jeu en pause. Lorsque le jeu est en pause, le temps est arrêté, la grille est masquée et les joueurs peuvent reprendre la partie à leur convenance.

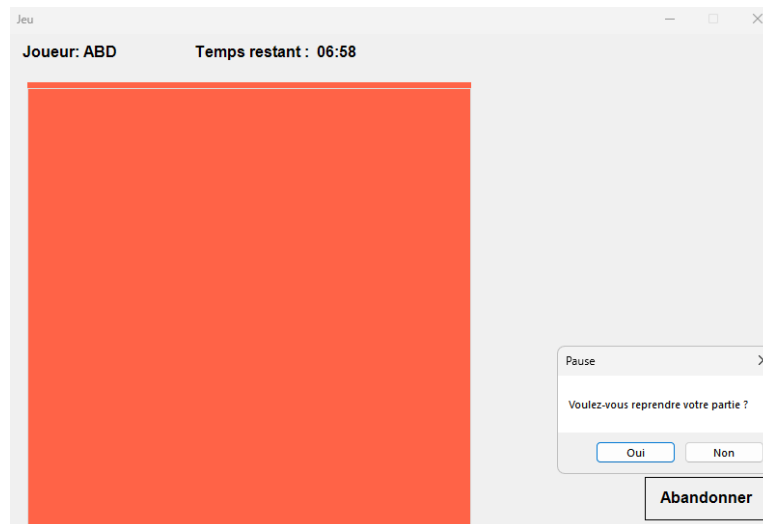


FIGURE 6 – Formulaire Jeu.vb (en pause)

2.4 Synchronisation du joueur entre la page d'accueil et la page de statistiques

Cette fonctionnalité assure que la sélection du joueur est synchronisée entre la page d'accueil et la page de statistiques. Ainsi, la sélection du joueur est également synchronisée avec la page de jeu, comme nous avons pu le voir précédemment. Cela permet une transition fluide des informations affichées.

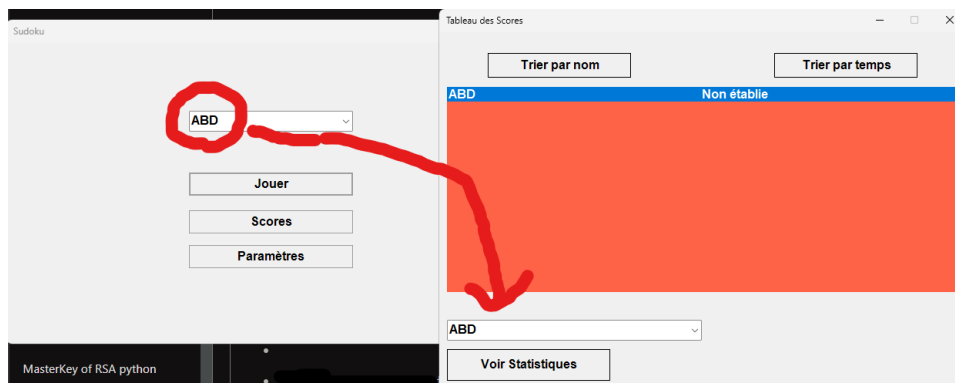


FIGURE 7 – sync Accueil -> StatisticForm.vb and Jeu.vb

2.5 Système de génération de Sudoku aléatoire faisable

Un système de génération de Sudoku aléatoire a été mis en place pour créer des grilles de Sudoku qui sont toujours résolvables. L'algorithme garantit que chaque grille générée a une solution unique et respecte les règles du Sudoku.

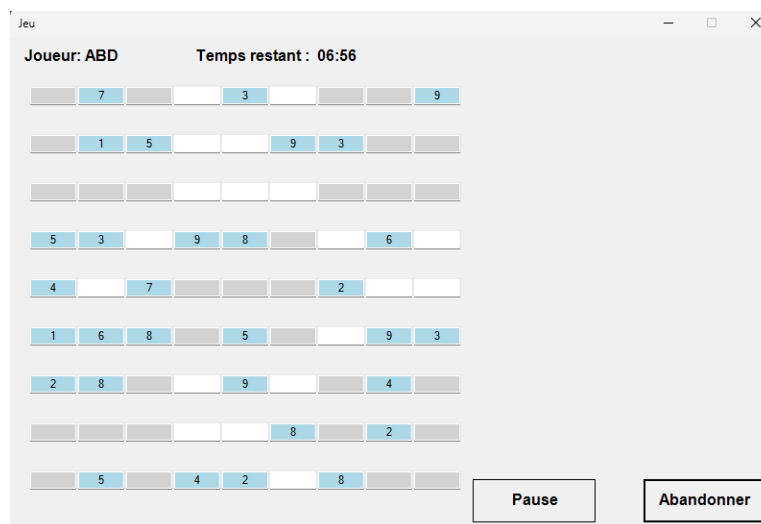


FIGURE 8 – Formulaire Jeu.vb

3 Modification des attentes

3.1 Images et sons

Nous avons envie de rajouter de la musique et des images, mais cela aurait nécessité une gestion de fichiers à ne pas perdre. En effet, si l'utilisateur perd son document de statistiques, le jeu continue de fonctionner, alors que s'il perd une image, cela peut grandement affecter la lisibilité de l'interface. De plus, le document de statistiques se régénère automatiquement, alors que les images ne peuvent pas être récupérées facilement. Pour éviter ces problèmes qui sont plus des caprices que des réelles améliorations, nous avons décidé de supprimer le système d'images et de sons.

3.2 Redimensionnement des interfaces

De même, dans un souci de lisibilité et de rendre le développement plus rapide, nous avons fixé une taille à chaque formulaire, alors que nous souhaitions à l'origine faire des formulaires redimensionnables.

4 Structure des données et organisation des fichiers

4.1 Structure des données dans le module

Le module gère les données des joueurs et de la grille de Sudoku. Voici une vue d'ensemble de la structure des données :

4.2 Données des joueurs

Chaque joueur est représenté par la structure `Player` qui contient les informations suivantes :

- Nom du joueur (`Name`)
- Meilleur temps réalisé (`BestTime`)
- Nombre de parties jouées (`GamesPlayed`)
- Temps de jeu total (`TotalPlayTime`)

Ces données sont stockées dans une liste (`playerList`) qui est manipulée par les différentes fonctions du module.

4.3 Données de la grille de Sudoku

La grille de Sudoku est représentée par un tableau bidimensionnel (`grid`). La résolution de la grille est également stockée dans un tableau bidimensionnel (`solveGrid`) pour la comparaison lors de la vérification de la victoire du joueur. La fonctionnalité de génération de la grille, de validation des entrées

du joueur et de vérification de la solution est également implémentée dans ce module.

4.4 Gestion des fichiers

Les données des joueurs sont stockées dans un fichier texte (`player_data.txt`). Voici comment cela est réalisé :

4.5 Chargement des données

Lorsque l'application démarre, les données des joueurs sont chargées à partir du fichier texte dans la fonction `LoadPlayerData()`. Ces données sont ensuite stockées dans la liste `playerList`.

4.6 Sauvegarde des données

Lorsque l'application se ferme ou qu'une modification est effectuée sur les données des joueurs, les données sont sauvegardées dans le fichier texte à l'aide de la fonction `SavePlayerData()`.

4.7 Gestion des paramètres de configuration

Les paramètres de configuration, tels que la difficulté du jeu et le temps total de jeu, sont stockés dans un autre module (`Settings.vb`). Voici comment cela est géré :

- Les paramètres sont définis comme des variables globales dans le module `Settings.vb`.
- Les valeurs par défaut sont définies pour la difficulté et le temps total de jeu.
- Ces paramètres peuvent être récupérés et modifiés à tout moment à l'aide des fonctions définies dans le module `Settings.vb`.
- Les paramètres sont chargés au démarrage de l'application et peuvent être sauvegardés lorsqu'ils sont modifiés.

4.8 Schéma

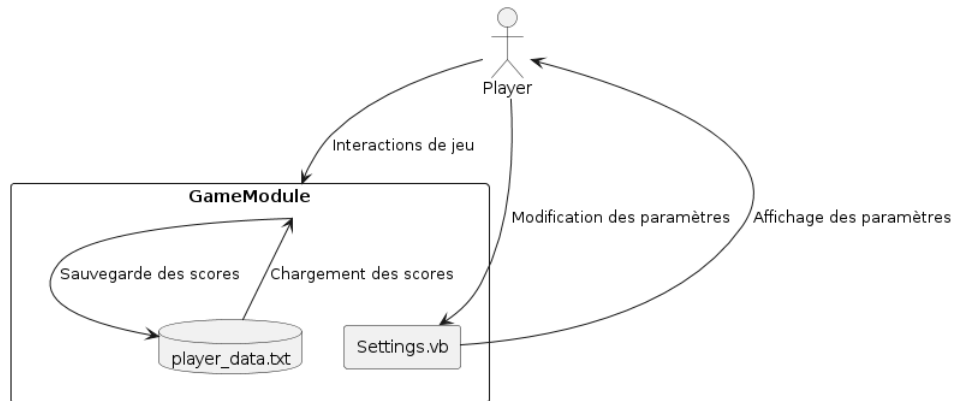


FIGURE 9 – Schéma de la gestion des données

5 Code Source

5.1 Main.vb

```
Module Main
    Sub main()
        LoadPlayerData()
        Application.EnableVisualStyles()
        For Each player As Player In getPlayerList()
            Accueil.cmbPlayerNames.Items.Add(player.Name)
        Next
        Application.Run(Accueil)
        SavePlayerData()
        Application.Exit()
    End Sub
End Module
```

5.2 Accueil.vb

```
Public Class Accueil

    Private Sub Accueil_Load(sender As Object, e As EventArgs) Handles MyBase.Load
        Me.FormBorderStyle = FormBorderStyle.FixedDialog
        Me.MaximizeBox = False
    End Sub
End Class
```

```

    If getCurrentPlayerId() > -1 Then
        cmbPlayerNames.Text = getCurrentPlayer().Name
    End If

    cmbPlayerNames.AutoCompleteMode =
        AutoCompleteMode.SuggestAppend
    cmbPlayerNames.AutoCompleteSource =
        AutoCompleteSource.ListItems
End Sub

Private Sub cmbPlayerNames_SelectedIndexChanged(
    sender As Object, e As EventArgs) Handles
    cmbPlayerNames.SelectedIndexChanged
    selectPlayer(cmbPlayerNames.SelectedItem.ToString
    )
End Sub

Private Sub btnNewGame_Click(sender As Object, e As
    EventArgs) Handles btnNewGame.Click
    Dim playerName As String = cmbPlayerNames.Text.
        Trim()
    If playerName <> "" Then
        If Not getPlayerNames().Contains(playerName)
            Then
                addNewPlayer(playerName)
                cmbPlayerNames.Items.Add(playerName)
            Else
                selectPlayer(playerName)
            End If
        End If

        Jeu.Show()

        Me.Hide()
    Else
        MsgBox("Veuillez s lectionner ou saisir un
            nom de joueur.", MsgBoxStyle.Critical, "
            Erreur")
    End If
End Sub

Private Sub btnQuit_Click(sender As Object, e As
    EventArgs) Handles btnQuit.Click

    If MsgBox("Voulez-vous vraiment quitter l'
        application ?", MsgBoxStyle.YesNo, "Quitter ?
        ") = MsgBoxResult.Yes Then
        Me.Close()
    End If
End Sub

```

```

        End If
    End Sub

    Private Sub btnScores_Click(sender As Object, e As
        EventArgs) Handles btnScores.Click

        Me.Hide()
        StatisticsForm.cmbPlayerNames.Text =
            cmbPlayerNames.Text
        StatisticsForm.Show()

    End Sub

    Private Sub btnSettings_Click(sender As Object, e As
        EventArgs) Handles btnSettings.Click

        SettingsForm.Show()

        Me.Hide()
    End Sub
End Class

```

5.3 Jeu.vb

```
Public Class Jeu
    Dim gridSize As Integer = 9
    Dim gridTextBox(gridSize - 1, gridSize - 1) As
        TextBox
    Dim timerSpan As Integer

    Dim timerString As String = "Temps restant : "
    Dim Timer1 As New Timer With {
        .Interval = 1000
    }

    Dim hideBox As GroupBox = New GroupBox With {
        .Width = 460,
        .Height = 460,
        .BackColor = Color.Tomato,
        .Location = New Point(20, 50),
        .Visible = False
    }
    Dim win As Boolean = False

    Private Sub Jeu_Load(sender As Object, e As EventArgs
    ) Handles MyBase.Load
        timerSpan = getTotalTime()

        Me.FormBorderStyle = FormBorderStyle.FixedDialog
        Me.MaximizeBox = False

        sudoku(gridSize, gridSize)

        For row As Integer = 0 To gridSize - 1
            For column As Integer = 0 To gridSize - 1
                Dim cases As New TextBox With {
                    .Width = 50,
                    .Height = 50,
                    .Location = New Point(20 + (row * 50)
                    , 50 + (column * 50)),
                    .TextAlign = HorizontalAlignment.
                        Center,
                    .Tag = New Point(row, column)
                }

                If (row \ 3 + column \ 3) Mod 2 = 0 Then
                    cases.BackColor = Color.LightGray
                Else
                    cases.BackColor = Color.White
                End If
            Next column
        Next row
    End Sub
End Class
```

```

        End If

        If EstCaseInitiale(row, column) Then
            cases.Text = ObtenirValeurInitiale(
                row, column).ToString()
            cases.ReadOnly = True
            cases.BackColor = Color.LightBlue
        End If

        AddHandler cases.TextChanged, AddressOf
            TextBox_TextChanged

        gridTextBox(row, column) = cases
        Me.Controls.Add(cases)
    Next
Next

lblJoueur.Text = "Joueur: " & getCurrentPlayer().
    Name
UpdateTimerString()
AddHandler Timer1.Tick, AddressOf Timer1_Tick

addAGameCurrPlayer()

Me.Controls.Add(hideBox)
hideBox.BringToFront()
Timer1.Start()
End Sub

Private Sub UpdateTimerString()
    lblTempsRestant.Text = $"{{timerString}} {{timerSpan
        \ 60:00}}:{{timerSpan Mod 60:00}}"
End Sub

Private Function EstCaseInitiale(row As Integer,
    column As Integer) As Boolean
    Return ObtenirValeurInitiale(row, column) > 0
End Function

Private Function ObtenirValeurInitiale(row As Integer
    , column As Integer) As Integer
    Return getGrid()(row, column)
End Function

Private Sub btnAbandonner_Click_1(sender As Object, e
    As EventArgs) Handles btnAbandonner.Click
    If MsgBox("Voulez-vous vraiment quitter l'
        application ?", MsgBoxStyle.YesNo, "
        Abandonner ?") = MsgBoxResult.Yes Then

```

```

        Me.Close()
    End If
End Sub

Private Sub Jeu_FormClosing(sender As Object, e As
    FormClosingEventArgs) Handles MyBase.FormClosing
    Timer1.Stop()
    SavePlayerData()

    If win Then
        StatisticsForm.Show()
    Else
        Accueil.Show()
    End If
End Sub

Private Sub Timer1_Tick(sender As Object, e As
    EventArgs)
    timerSpan -= 1
    addTimeCurrPlayer(1)
    UpdateTimerString()
    If timerSpan = 0 Then
        Timer1.Stop()
        MsgBox("Dommage, vous y tiez presque mais
            vous n'avez pas eu le temps de finir !",
            MsgBoxStyle.Critical, "Vous avez perdu !"
        )
        Me.Close()
    End If
End Sub

Private Sub TextBox_TextChanged(sender As Object, e
    As EventArgs)
    Dim currCase As TextBox = CType(sender, TextBox)
    Dim coordinates As Point = CType(currCase.Tag,
        Point)
    Dim row As Integer = coordinates.X
    Dim column As Integer = coordinates.Y

    If Not isValid(row, column, currCase.Text) Then
        currCase.BackColor = Color.Red
    Else
        If (row \ 3 + column \ 3) Mod 2 = 0 Then
            currCase.BackColor = Color.LightGray
        Else
            currCase.BackColor = Color.White
        End If
    End If
End Sub

```

```

        If isSolved() Then
            Timer1.Stop()
            Dim time As Integer = getTotalTime() -
                timerSpan
            updatePlayerBestTime(time)
            MsgBox($"Vous avez r ussi en {time \
                60:00} minute(s) {time Mod 60:00}
                seconde(s)", MsgBoxStyle.Information,
                "F licitation !")
            win = True
            Me.Close()
        End If
    End If

End Sub

Private Sub btnPause_Click(sender As Object, e As
EventArgs) Handles btnPause.Click
    Timer1.Stop()
    hideBox.Visible = True
    btnPause.Visible = False

    If MsgBox("Voulez-vous reprendre votre partie ?",
        MsgBoxStyle.YesNo, "Pause") = MsgBoxResult.
        Yes Then
        Timer1.Start()
        hideBox.Visible = False
        btnPause.Visible = True
    Else
        Me.Close()
    End If
End Sub
End Sub
End Class

```

5.4 SettingsForm.vb

```

Public Class SettingsForm
    Dim inMovement As Boolean = False
    Dim loadFinished As Boolean = False
    Private Sub SettingsForm_Load(sender As Object, e As
EventArgs) Handles MyBase.Load
        ' D sactiver le redimensionnement de la fen tre
        et l'agrandissement
        Me.FormBorderStyle = FormBorderStyle.FixedDialog
    End Sub
End Class

```



```

Me.MaximizeBox = False

HScrollBarDiff.Minimum = getMinDiff()
HScrollBarDiff.Maximum = getMaxDiff()
HScrollBarDiff.Value = getDiff()
TextBoxCurrentValue.Text = getDiff().ToString()

HScrollBarDiff.SmallChange = 1
HScrollBarDiff.LargeChange = 5
NumericUpDown1.Value = getTotalTime() \ 60
loadFinished = True
End Sub

Private Sub updateDifficulty(value As Integer)
    setDiff(value)

    HScrollBarDiff.Value = value
    TextBoxCurrentValue.Text = value.ToString()
    inMovement = True
    If value > getMaxDiff() - HScrollBarDiff.
        LargeChange * 2 Then
        radioHard.Checked = True
    ElseIf value >= getNormalDiff() Then
        radioNormal.Checked = True
    Else
        radioEasy.Checked = True
    End If
    inMovement = False
End Sub

Private Sub StatisticsForm_FormClosing(sender As
    Object, e As FormClosingEventArgs) Handles MyBase
    .FormClosing
    ' Fermer le formulaire et revenir au formulaire d
    'accueil
    Dim mainForm As New Accueil()
    mainForm.Show()
End Sub

Private Sub HScrollBarDiff_Scroll(sender As Object, e
    As ScrollEventArgs) Handles HScrollBarDiff.
    Scroll
    updateDifficulty(HScrollBarDiff.Value)
End Sub

Private Sub LabelCurrentValue_TextChanged(sender As
    Object, e As EventArgs) Handles
    TextBoxCurrentValue.TextChanged

```

```

Dim value As Integer
If Integer.TryParse(TextBoxCurrentValue.Text,
    value) And value <= getMaxDiff() And value >=
    getMinDiff() Then
    updateDifficulty(value)
    TextBoxCurrentValue.BackColor = Color.White
Else
    TextBoxCurrentValue.BackColor = Color.Red
End If
End Sub

Private Sub radioEasy_CheckedChanged(sender As Object
, e As EventArgs) Handles radioEasy.
CheckedChanged
    If Not inMovement Then
        updateDifficulty(getMinDiff())
    End If
End Sub

Private Sub radioNormal_CheckedChanged(sender As
Object, e As EventArgs) Handles radioNormal.
CheckedChanged
    If Not inMovement Then
        updateDifficulty(getNormalDiff())
    End If
End Sub

Private Sub radioHard_CheckedChanged(sender As Object
, e As EventArgs) Handles radioHard.
CheckedChanged
    If Not inMovement Then
        updateDifficulty(getMaxDiff())
    End If
End Sub

Private Sub NumericUpDown1_ValueChanged(sender As
Object, e As EventArgs) Handles NumericUpDown1.
ValueChanged
    If loadFinished Then
        If NumericUpDown1.Value > getMaxTimeMinute()
Then
            NumericUpDown1.Value = getMaxTimeMinute()
        ElseIf NumericUpDown1.Value <
getMinTimeMinute() Then
            NumericUpDown1.Value = getMinTimeMinute()
        Else
            setTime(NumericUpDown1.Value)
        End If
    End If
End Sub

```

```

        End If
    End If

End Sub

Private Sub NumericUpDown1_Text(sender As Object, e
    As EventArgs) Handles NumericUpDown1.TextChanged
    If NumericUpDown1.Value > getMaxTimeMinute() Then
        NumericUpDown1.Value = getMaxTimeMinute()
    ElseIf NumericUpDown1.Value < getMinTimeMinute()
    Then
        NumericUpDown1.Value = getMinTimeMinute()
    End If
End Sub
End Class

```

5.5 StatisticsForm.vb

```

Public Class StatisticsForm

    Private nameShorted As Boolean = True

    Private Sub StatisticsForm_Load(sender As Object, e
        As EventArgs) Handles MyBase.Load
        Me.FormBorderStyle = FormBorderStyle.FixedDialog
        Me.MaximizeBox = False

        FillListBoxesByName()
        If cmbPlayerNames.Items.Count < Accueil.
            cmbPlayerNames.Items.Count Then
            For Each player As Player In getPlayerList()
                cmbPlayerNames.Items.Add(player.Name)
            Next
        End If

        cmbPlayerNames.SelectedItem = getCurrentPlayer().
            Name
        cmbPlayerNames.AutoCompleteMode =
            AutoCompleteMode.SuggestAppend
        cmbPlayerNames.AutoCompleteSource =
            AutoCompleteSource.ListItems
    End Sub

    Private Sub FillListBoxesByName()
        Dim sortedPlayersByName = getPlayerList().OrderBy
            (Function(player) player.Name)
        PlayerListBox.Items.Clear()
    End Sub

```

```

TimeListBox.Items.Clear()

For Each player As Player In sortedPlayersByName
    PlayerListBox.Items.Add(player.Name)
    If player.BestTime = getNoBestTime() Then
        TimeListBox.Items.Add("Non    tableie ")
    Else
        TimeListBox.Items.Add($"{player.BestTime
                                \ 60:00}:{player.BestTime Mod 60:00}"
                                )
    End If
Next
End Sub

Private Sub FillListBoxesByTime()
    Dim sortedPlayersByTime = getPlayerList().OrderBy
        (Function(player) player.BestTime)
    PlayerListBox.Items.Clear()
    TimeListBox.Items.Clear()

    For Each player As Player In sortedPlayersByTime
        PlayerListBox.Items.Add(player.Name)
        If player.BestTime = getNoBestTime() Then
            TimeListBox.Items.Add("Non    tableie ")
        Else
            TimeListBox.Items.Add($"{player.BestTime
                                    \ 60:00}:{player.BestTime Mod 60:00}"
                                    )
        End If
    Next
End Sub

Private Sub btnShowStats_Click(sender As Object, e As
EventArgs) Handles btnShowStats.Click
    If cmbPlayerNames.SelectedItem IsNot Nothing Then
        Dim playerName As String = cmbPlayerNames.
            SelectedItem.ToString()

        Dim selectedPlayer As Player = getPlayerList
            ().Find(Function(player) player.Name =
                playerName)

        Dim bestTime As String
        If selectedPlayer.BestTime = getNoBestTime()
            Then
            bestTime = "Non    tableie "
        Else

```

```

        bestTime = $"{selectedPlayer.BestTime \
        60:00}:{selectedPlayer.BestTime Mod
        60:00}"
    End If

    Dim hours As Integer = selectedPlayer.
        TotalPlayTime \ 3600
    Dim minu As Integer = selectedPlayer.
        TotalPlayTime \ 60 - hours * 60
    Dim sec As Integer = selectedPlayer.
        TotalPlayTime Mod 60

    Dim message As String = $"Statistiques de {
        selectedPlayer.Name} :" & Environment.
        NewLine &
        $"Meilleur temps : {
            bestTime}" &
            Environment.
            NewLine &
            $"Nombre de parties
            jou es : {
                selectedPlayer.
                GamesPlayed}" &
                Environment.
                NewLine &
                $"Temps de jeu total
                : {hours:00}:{
                    minu:00}:{sec:00}
                "

    MsgBox(message, MsgBoxStyle.Information, "
        Statistiques du joueur")
Else
    MsgBox("Veuillez s lectionner un joueur.",
        MsgBoxStyle.Exclamation, "Erreur")
End If
End Sub

Private Sub btnSortByName_Click(sender As Object, e
    As EventArgs) Handles btnSortByName.Click
    If Not nameShorted Then
        FillListBoxesByName()
        nameShorted = True
    End If
End Sub

```

```

Private Sub btnSortByTime_Click(sender As Object, e
    As EventArgs) Handles btnSortByTime.Click
    If nameShorted Then
        FillListBoxesByTime()
        nameShorted = False
    End If
End Sub

Private Sub PlayerListBox_SelectedIndexChanged(sender
    As Object, e As EventArgs) Handles PlayerListBox
    .SelectedIndexChanged
    If PlayerListBox.SelectedIndex <> -1 Then
        Dim playerName As String = PlayerListBox.
            SelectedItem.ToString()
        TimeListBox.SelectedIndex = PlayerListBox.
            SelectedIndex
        cmbPlayerNames.SelectedItem = playerName
    End If
End Sub

Private Sub TimeListBox_SelectedIndexChanged(sender
    As Object, e As EventArgs) Handles TimeListBox.
    SelectedIndexChanged
    If TimeListBox.SelectedIndex <> -1 Then
        PlayerListBox.SelectedIndex = TimeListBox.
            SelectedIndex
        PlayerListBox.Select()
    End If
End Sub

Private Sub StatisticsForm_FormClosing(sender As
    Object, e As FormClosingEventArgs) Handles MyBase
    .FormClosing
    Accueil.Show()
End Sub

Private Sub cmbPlayerNames_SelectedIndexChanged(
    sender As Object, e As EventArgs) Handles
    cmbPlayerNames.SelectedIndexChanged
    Dim playerName As String = cmbPlayerNames.
        SelectedItem.ToString()

    PlayerListBox.SelectedItem = playerName
    TimeListBox.SelectedIndex = PlayerListBox.
        SelectedIndex
End Sub

```

```

Private Sub cmbPlayerNames_TextChanged(sender As
Object, e As KeyPressEventArgs) Handles
cmbPlayerNames.KeyPress
    If Not e.KeyChar = ChrW(Keys.Return) Then
        Return
    End If

    Dim playerName As String = cmbPlayerNames.Text.
Trim()

    If getPlayerList().Any(Function(player) player.
Name = playerName) Then
        PlayerListBox.SelectedItem = playerName
        PlayerListBox.Select()
    End If
End Sub
End Class

```

5.6 Settings.vb

```

Module Settings
    Dim totalTime As Integer = 7 * 60
    Dim difficulty As Integer = 50

    Dim minDiff As Integer = 40
    Dim maxDiff As Integer = 75
    Dim normalDiff As Integer = 55

    Dim maxTimeMinute As Integer = 59
    Dim minTimeMinute As Integer = 1

    Function getTotalTime() As Integer
        Return totalTime
    End Function

    Function getDiff() As Integer
        Return difficulty
    End Function

    Function getMinDiff() As Integer
        Return minDiff
    End Function

    Function getNormalDiff() As Integer
        Return normalDiff
    End Function

```

```

End Function

Function getMaxDiff() As Integer
    Return maxDiff
End Function

Sub setDiff(diff As Integer)
    difficulty = diff
End Sub

Sub setTime(m As Integer)
    totalTime = m * 60
End Sub

Function getMaxTimeMinute() As Integer
    Return maxTimeMinute
End Function

Function getMinTimeMinute() As Integer
    Return minTimeMinute
End Function

Function getNoBestTime() As Integer
    Return (maxTimeMinute + 1) * 60
End Function

End Module

```

5.7 PlayerDataModule.vb

```

Imports System.IO

Module PlayerDataModule

    Structure Player
        Dim Name As String
        Dim BestTime As Integer
        Dim GamesPlayed As Integer
        Dim TotalPlayTime As Integer
    End Structure

    Dim playerList As New List(Of Player)
    Dim idCurrPlayer As Integer = -1

    Function getPlayerNames() As List(Of String)

```



```

    Dim res As New List(Of String)
    For Each player As Player In playerList
        res.Add(player.Name)
    Next
    Return res
End Function

Sub addNewPlayer(playerName As String)
    Dim np As Player

    np.Name = playerName
    np.BestTime = getNoBestTime()
    np.GamesPlayed = 0
    np.TotalPlayTime = 0

    playerList.Add(np)
    idCurrPlayer = playerList.Count - 1
    SavePlayerData()
End Sub

Sub selectPlayer(playerName As String)
    For i As Integer = playerList.Count - 1 To 0 Step -1
        If (playerName = playerList(i).Name) Then
            idCurrPlayer = i
            i = 0
        End If
    Next
End Sub

Sub addTimeCurrPlayer(seconds As Integer)
    Dim np As Player
    np = playerList(idCurrPlayer)

    np.TotalPlayTime = playerList(idCurrPlayer).
        TotalPlayTime + seconds

    playerList(idCurrPlayer) = np

    'playerList(idCurrPlayer).TotalPlayTime +=
    '    TimeSpan.FromSeconds(seconds)
End Sub

Public Sub addAGameCurrPlayer()

    Dim np As Player
    np = playerList(idCurrPlayer)
    np.GamesPlayed = playerList(idCurrPlayer).
        GamesPlayed + 1

```

```

        playerList(idCurrPlayer) = np

        'playerList(idCurrPlayer).GamesPlayed += 1
    End Sub

    ' Chargement des donn es des joueurs depuis un
    fichier
    Sub LoadPlayerData()
        playerList.Clear()
        playerList = LoadPlayer()
    End Sub

    Private Function LoadPlayer() As List(Of Player)
        Dim pl As New List(Of Player)
        If File.Exists("player_data.txt") Then
            Using sr As New StreamReader("player_data.txt")
                While Not sr.EndOfStream
                    Dim line As String = sr.ReadLine()
                    Dim parts() As String = line.Split(",
                        "c)
                    Dim player As New Player With {
                        .Name = parts(0),
                        .BestTime = Integer.Parse(parts
                            (1)),
                        .GamesPlayed = Integer.Parse(
                            parts(2)),
                        .TotalPlayTime = Integer.Parse(
                            parts(3))
                    }

                    pl.Add(player)
                End While
            End Using
        End If

        Return pl
    End Function

    ' Enregistrement des donn es des joueurs dans un
    fichier
    Sub SavePlayerData()
        Using sw As New StreamWriter("player_data.txt")
            For Each player As Player In playerList
                sw.WriteLine($"{player.Name},{player.
                    BestTime},{player.GamesPlayed},{
                    player.TotalPlayTime}")
            Next
        End Using
    End Sub

```

```

        End Using
    End Sub

    Sub updatePlayerBestTime(timespan As Integer)

        If timespan < getCurrentPlayer().BestTime Then
            Dim np As Player
            np = playerList(idCurrPlayer)
            np.BestTime = timespan

            playerList(idCurrPlayer) = np
        End If
    End Sub

    Function getCurrentPlayer() As Player
        If idCurrPlayer > -1 Then
            Return playerList(idCurrPlayer)
        Else
            Return Nothing
        End If
    End Function

    Function getCurrentPlayerId() As Integer
        Return idCurrPlayer
    End Function

    Function getPlayerList() As List(Of Player)
        Return playerList
    End Function

End Module

```

5.8 SudokuGrid.vb

```

Imports System.Runtime.InteropServices.WindowsRuntime

Module SudokuGrid
    Dim grid(,) As Integer
    Dim solveGrid(,) As Integer
    Dim sx As Integer
    Dim sy As Integer
    Dim nbZero As Integer
    Dim rand As Random
    Dim A As Integer(,) = {{3, 6, 7, 9, 4, 1, 2, 8, 5},

```

```

        {1, 5, 2, 6, 8, 3, 4, 9, 7},
        {4, 9, 8, 7, 5, 2, 1, 6, 3},
        {7, 4, 6, 1, 9, 5, 8, 3, 2},
        {8, 1, 9, 2, 3, 7, 6, 5, 4},
        {2, 3, 5, 8, 6, 4, 7, 1, 9},
        {9, 2, 1, 5, 7, 8, 3, 4, 6},
        {5, 8, 4, 3, 2, 6, 9, 7, 1},
        {6, 7, 3, 4, 1, 9, 5, 2, 8}
    }

    Dim B As Integer(,) = {{4, 2, 7, 9, 5, 1, 6, 8, 3},
        {6, 3, 9, 2, 8, 4, 7, 5, 1},
        {8, 5, 1, 7, 6, 3, 9, 2, 4},
        {5, 1, 4, 8, 9, 6, 3, 7, 2},
        {9, 7, 6, 4, 3, 2, 8, 1, 5},
        {2, 8, 3, 1, 7, 5, 4, 9, 6},
        {3, 9, 2, 6, 1, 8, 5, 4, 7},
        {7, 4, 5, 3, 2, 9, 1, 6, 8},
        {1, 6, 8, 5, 4, 7, 2, 3, 9}
    }

    Dim C As Integer(,) = {{9, 8, 5, 1, 3, 2, 4, 7, 6},
        {2, 7, 3, 9, 4, 6, 5, 1, 8},
        {4, 1, 6, 5, 7, 8, 9, 2, 3},
        {1, 6, 7, 4, 8, 9, 2, 3, 5},
        {8, 5, 2, 6, 1, 3, 7, 9, 4},
        {3, 9, 4, 2, 5, 7, 8, 6, 1},
        {7, 4, 1, 3, 9, 5, 6, 8, 2},
        {5, 2, 9, 8, 6, 1, 3, 4, 7},
        {6, 3, 8, 7, 2, 4, 1, 5, 9}
    }

    ' Constructeur pour initialiser la grille avec un
    ' grid complet et valide
    Sub sudoku(sizeX As Integer, sizeY As Integer)
        sx = sizeX - 1
        sy = sizeY - 1
        ReDim grid(sx, sy)
        rand = New Random(DateTime.Now.GetHashCode())
        ' Générer une grille complète et valide
        GenerateGrid()
        solveGrid = grid.Clone()
        ReplaceWithZeros(getDiff())
        nbZero = getDiff()
    End Sub

    Sub GenerateGrid()
        Dim grids As New List(Of Integer(,))
        grids.Add(A)
    End Sub

```

```

        grids.Add(B)
        grids.Add(C)

        RandomSudoku(grids)

    End Sub

    Private Sub RandomSudoku(gridOptions As List(Of
        Integer(,)))
        Dim chosenGrid As Integer(,) = gridOptions(rnd.
            Next(0, gridOptions.Count))

        Dim rotations As Integer = rnd.Next(0, 4)
        For i As Integer = 0 To rotations - 1
            chosenGrid = RotateGrid(chosenGrid)
        Next

        grid = chosenGrid
    End Sub

    Private Function RotateGrid(grid As Integer(,)) As
        Integer(,)
        Dim n As Integer = grid.GetLength(0)
        Dim rotatedGrid(n - 1, n - 1) As Integer

        For i As Integer = 0 To n - 1
            For j As Integer = 0 To n - 1
                rotatedGrid(i, j) = grid(n - j - 1, i)
            Next
        Next

        Return rotatedGrid
    End Function

    Private Sub ReplaceWithZeros(count As Integer)
        While count > 0
            Dim posX As Integer = rnd.Next(0, sx + 1)
            Dim posY As Integer = rnd.Next(0, sy + 1)
            If Not grid(posX, posY) = 0 Then
                grid(posX, posY) = 0
                count -= 1
            End If
        End While
    End Sub

    Function getGrid() As Integer(,)
        Return grid
    End Function

```

```

Function isValid(x As Integer, y As Integer, text As
String) As Boolean
    Dim validBefore As Boolean = Not grid(x, y) = 0
    Dim validAfter As Boolean = trimValid(x, y, text)

    ' Mise à jour du compteur si la validité a
    changé
    If validBefore AndAlso Not validAfter Then
        nbZero += 1 ' Incrémenter le compteur si la
        valeur devient invalide
        grid(x, y) = 0
    ElseIf Not validBefore AndAlso validAfter Then
        nbZero -= 1 ' Décrémenter le compteur si la
        valeur devient valide
    End If

    ' Retourner true si la nouvelle valeur est valide
    Return validAfter

End Function

Private Function trimValid(x As Integer, y As Integer
, text As String) As Boolean
    text = text.Trim()

    If text = "" Then
        grid(x, y) = 0
        Return True
    End If

    Dim value As Integer

    If Not Integer.TryParse(text, value) Then
        Return False
    End If

    ' Vérifier si la valeur est dans la plage de 1
    9
    If value < 1 OrElse value > 9 Then
        Return False
    End If

    Return testValid(x, y, value)
End Function

Private Function testValid(x As Integer, y As Integer
, value As Integer) As Boolean

```

```

' V rifier si la valeur n'est pas d j
pr sente dans la m me colonne
For j As Integer = 0 To sx
    If grid(x, j) = value Then
        Return False
    End If
Next

' V rifier si la valeur n'est pas d j
pr sente dans la m me ligne
For i As Integer = 0 To sy
    If grid(i, y) = value Then
        Return False
    End If
Next

' V rifier si la valeur n'est pas d j
pr sente dans la m me r gion 3x3
Dim startX As Integer = (x \ 3) * 3
Dim startY As Integer = (y \ 3) * 3

For i As Integer = startX To startX + 2
    For j As Integer = startY To startY + 2
        If grid(i, j) = value Then
            Return False
        End If
    Next
Next

' Si la valeur passe toutes les v rifications ,
elle est valide
grid(x, y) = value
Return True
End Function

Function isSolved() As Boolean
    If nbZero > 0 Then
        Return False
    End If

    For i As Integer = 0 To sx
        For j As Integer = 0 To sy
            ' Si les lments correspondants ne
            sont pas gaux , les grilles ne sont
            pas identiques
            If solveGrid(i, j) <> grid(i, j) Then
                Return False
            End If
        Next
    Next

```

```
        End If
    Next
Next

Return True
End Function
End Module
```