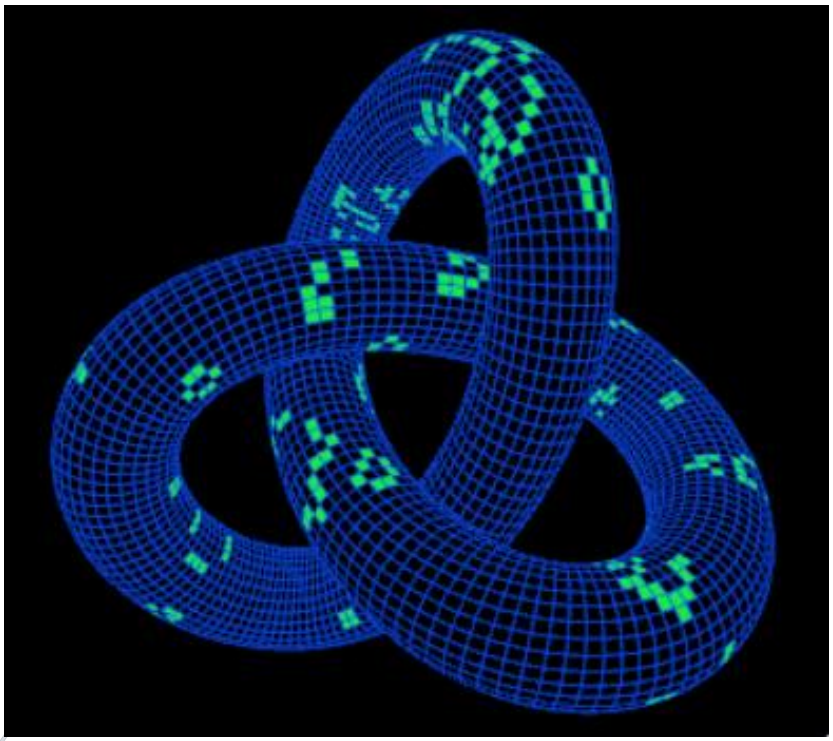
Année 2020/2021

Rapport de Projet

Le Jeu de La Vie



Damien Fournier, Théa Kieffer, Robin Difazio, Ibrahima Doumbia

Présentation du jeu

Histoire du jeu

Le jeu de la vie a été inventé par John Conway en 1970, alors qu'il est professeur de mathématiques à l'université de Cambridge, au Royaume-Unis. C'est probablement l'automate cellulaire[1] le plus connu. Le jeu de la vie est un « jeu à zéro joueur » : il ne nécessite pas d'action du joueur pendant son déroulement. Seules les conditions initiales du jeu peuvent être modifiées en début de partie.

Règles du jeu

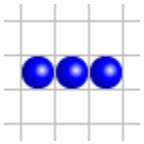
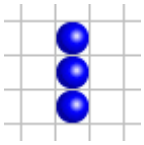
Le jeu se déroule sur une grille à deux dimensions, théoriquement infinie (mais de longueur et de largeur finies et plus ou moins grandes dans la pratique), dont les cases — qu'on appelle des « cellules », par analogie avec les cellules vivantes — peuvent prendre deux états distincts : « vivante » ou « morte ».

Une cellule possède huit voisins, qui sont les cellules adjacentes horizontalement, verticalement et diagonalement.

À chaque étape, l'évolution d'une cellule est entièrement déterminée par l'état de ses huit voisines de la façon suivante :

- une cellule morte possédant exactement trois voisines vivantes devient vivante (elle naît) ;
- une cellule vivante possédant deux ou trois voisines vivantes le reste, sinon elle meurt.

[1] Un automate cellulaire consiste en une grille régulière de « cellules » contenant chacune un « état » choisi parmi un ensemble fini et qui peut évoluer au cours du temps. L'état d'une cellule au temps *t+1* est fonction de l'état au temps *t* d'un nombre fini de cellules appelé son « voisinage ». à chaque nouvelle unité de temps, les mêmes règles sont appliquées simultanément à toutes les cellules de la grille, produisant une nouvelle « génération » de cellules dépendant entièrement de la génération précédente.

Ainsi, la configuration  donne au tour suivant la configuration  qui redonne ensuite la première.

Implémentation du jeu sous VBA

Création de la grille de jeu

Nous avons d'abord choisi de travailler avec un quadrillage de taille 50x70, avec la nomenclature suivante : une cellule vivante contient le numéro 1 tandis qu'une cellule morte contient le numéro 0. Cela permet de simplifier nos fonction VBA. Nous appliquons ensuite une mise en forme conditionnelle de sorte que les cellules vivantes apparaissent en noir tandis que celles mortes restent en blanc et nous cachons le numéro contenu dans ces cellules.

Nous obtenons cette feuille qui va nous servir de plateforme de jeu et que nous appellerons *Plateau* dans la suite :

Figure 1 : Grille du jeu

Calcul de la prochaine génération

Nous avons ensuite créé une fonction *nextG* prenant en paramètre les coordonnées d'une cellule appartenant à la feuille *Plateau* à la génération *n* et renvoyant l'état de cette cellule à la génération *n+1*.

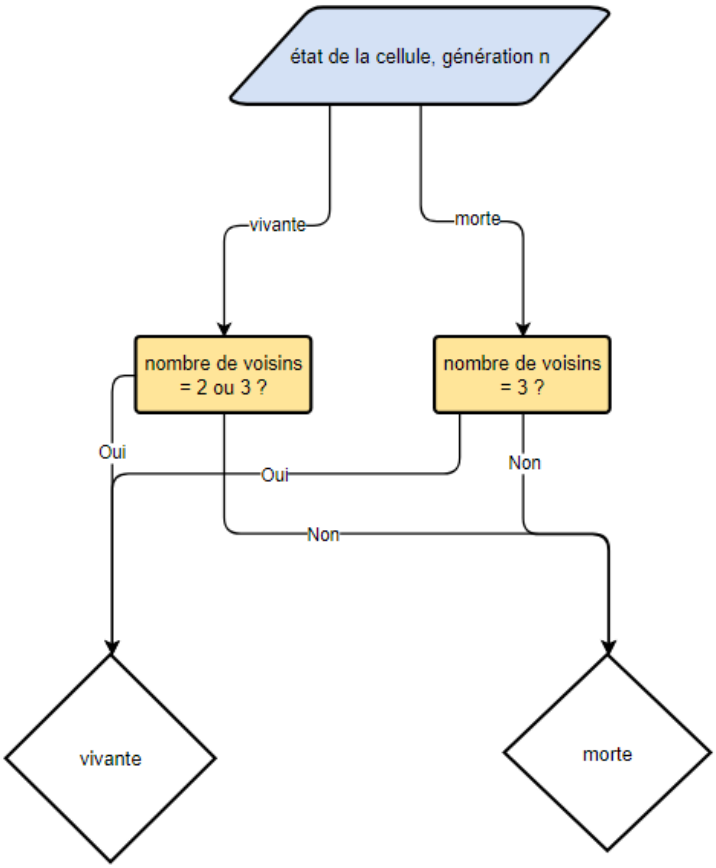


Figure 2 : algorithme de nextG

Nous ne pouvons pas remplacer directement l'état d'une cellule de génération *n* dans *Plateau* sans avoir au préalable calculé tous les états pour la génération *n+1*[2]. Nous avons donc créé une nouvelle feuille *nextGen,* qui contient l'état de toutes les cellules de *Plateau* à la génération d'après. Cette feuille n'est d'aucune utilité pour le joueur et est donc cachée.

[2] Le calcul du nombre de voisins pour une cellule (i, j) dépend de l'état des cellules adjacentes. Ainsi, en remplaçant directement l'état d'une cellule de génération *n* par son état de génération *n+1*, les calculs peuvent être faux pour l'état de la cellule (i+1, j) en *n+1*.

Nous avons ensuite créé une fonction *test* qui, dans sa première version, nous a permis d'observer l'évolution de notre grille sur *Plateau* à l'aide d'une boucle qui appelle *nextG* sur toutes les cellules de la grille puis qui modifie les valeurs ainsi obtenues dans *nextGen*.

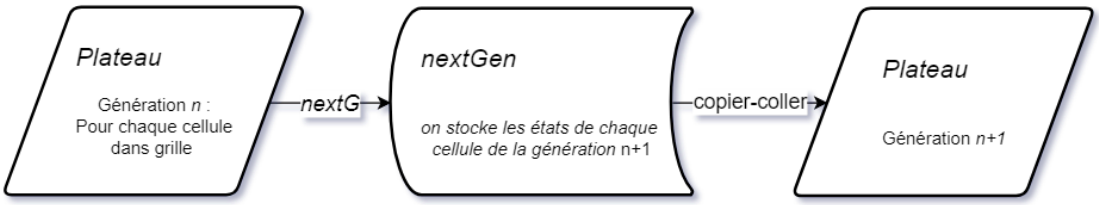


Figure 3 : algorithme de la fonction test

Implémentation du programme

Nous avons ensuite créé une procédure *jouer* qui appelle notre fonction *test* dans une boucle. Dans un premier temps, nous ne nous sommes pas préoccupés des possibles conditions d'arrêt[3] de cette boucle hormis un arrêt manuel du joueur à l'aide d'un bouton.

Pour le bouton, nous avons choisi une commande ActiveX car plus flexible d'utilisation qu'un contrôle de formulaire classique. Il s'agit du même bouton, appelé *CommandButton1*. Au démarrage, quand la simulation n'est pas en cours, le bouton contient le texte « jouer ». Dès que l'on clique dessus, il appelle la procédure *jouer* et change de texte (« Arrêter ») et de couleur de fond. En cliquant de nouveau dessus, on stoppe la boucle appelant *test* dans *jouer* ; la simulation s'arrête et le bouton reprend son apparence initiale.

Figure 4 : les deux formes de CommandButton1

[3] Différentes conditions d'arrêt comme un nombre maximal d'itérations ou l'aboutissement à une forme stable seront développés dans III. Interface de jeu.

```
47  Private Sub CommandButton1_Click()
48
49      'C'est le bouton pour lancer/arrêter le jeu.
50      'C'est un activeX, c'est pour ça qu'il se trouve sur Feuil1,
51      'il ne peut pas être dans module1
52      If CommandButton1.Caption = "Arrêter" Then
53      'le jeu est en cours, on veut l'arrêter
54      CommandButton1.Caption = "Jouer"
55      CommandButton1.BackColor = &HFF00FF
56      arret = True
57      'permet d'arrêter le jeu quand on clique
58      'sur le commandButton "jouer" de caption "Arrêter"
59
60      ElseIf CommandButton1.Caption = "Jouer" Then  'on veut démarrer le jeu
61      CommandButton1.Caption = "Arrêter"
62      CommandButton1.BackColor = &HFF797C
63      Call jouer
64      End If
65  End Sub
```

Figure 5 : le code derrière CommandButton1

L'interface de jeu

Présentation et structure

Le principe du jeu et l'implémentation du code sont assez simple. Nous avons donc concentré nos efforts sur l'expérience de jeu pour l'utilisateur. Malgré un design simple et épuré, le jeu permet d'effectuer de nombreuses actions. La grille de jeu occupe une place centrale tandis qu'un menu est disponible à droite et permet d'interagir avec la simulation. Une fenêtre en bas à droite indique le statut de la simulation et certains paramètres choisis par l'utilisateur.

Figure 6 : l'interface du jeu

Hormis le bouton « Jouer/Arrêter » (*CommandButton1*), tous les autres boutons sont des contrôles de formulaire. Ils se trouvent tous dans *Module1* et permettent d'accéder à d'autres userforms (*Aleatoire, Sauvegarder, UserForm 1 = Librairie, Options*), d'effacer la grille (le statut de chaque cellule est mort), de la remplir de cellules vivantes ou de lire les instructions du jeu.

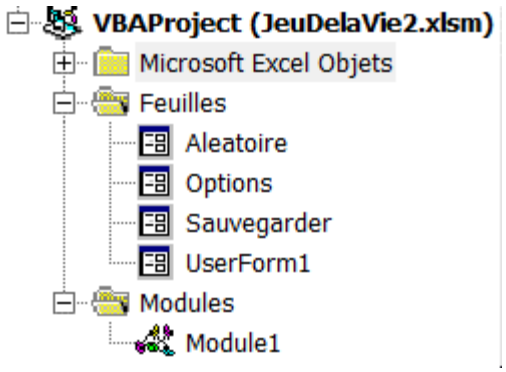


Figure 7 : structure de l'interface

Sauvegarder un modèle

Le bouton *sauvegarder* permet d'enregistrer le modèle présent sur la grille de jeu.

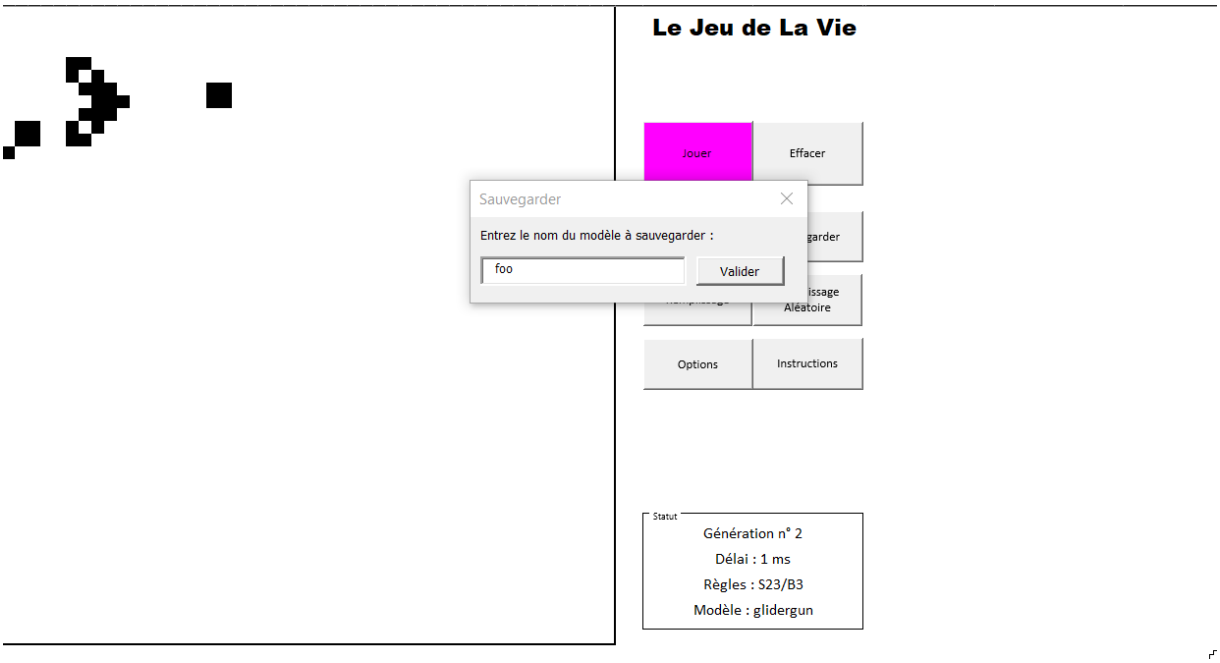


Figure 8 : sauvegarder un modèle

Derrière cette naïve interface se cache une plage nommée appelée *Sauvegardes* contenant tous les noms de sauvegarde de l'utilisateur. Cette plage nommée est une liste dynamique[4] régie par la formule suivante :

Figure 9 : une plage dynamique : Sauvegardes

En cliquant sur *Valider,* le nom rentré par l'utilisateur est ajouté à la fin de la liste dynamique *Sauvegardes,* une nouvelle feuille du même nom[5] est créée contenant les valeurs sur la grille de *Plateau*. Cette feuille est cachée de l'utilisateur[6].

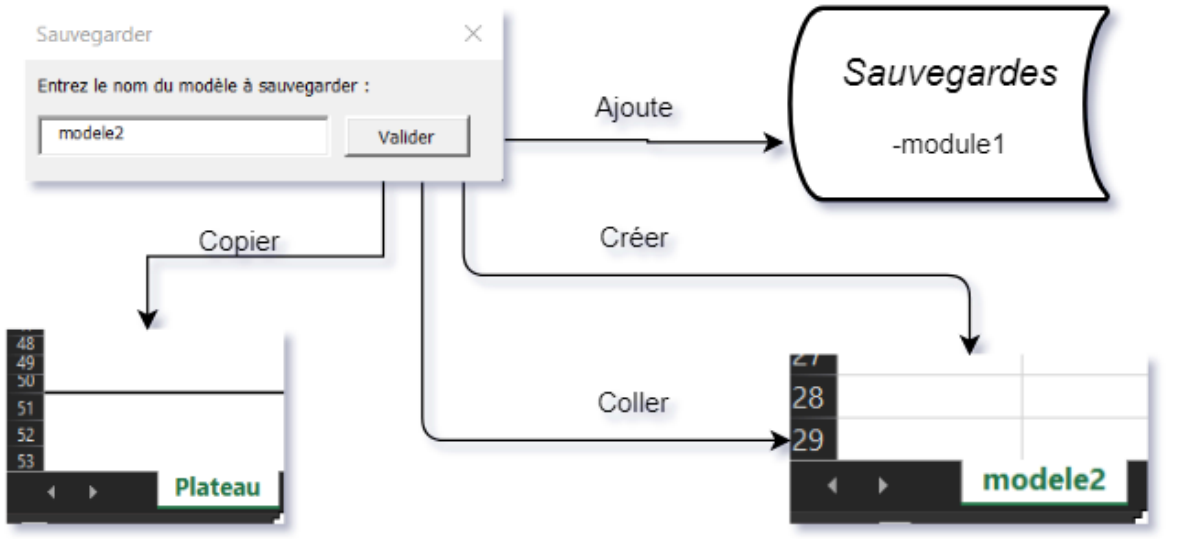


Figure 10 : algorithme de sauvegarde de modèle

Charger un modèle

Le bouton *Charger* permet d'utiliser un modèle préexistant ou un modèle sauvegardé par l'utilisateur. Il ouvre l'userform *UserForm1,* que l'on appellera par la suite *Librairie* par souci

[4] Cette liste dynamique nous sera particulièrement utile lors du chargement de modèles sauvegardés.

[5] Excel ne permet pas de créer de feuille possédant le même nom qu'une feuille déjà existante. Ainsi, le module *Sauvegarde* possède un contrôle de nom renvoyant une erreur si le nom n'est pas valide et la liste dynamique *Sauvegardes* ne possède pas de doublons.

[6] Par manque de temps, nous n'avons pas pu créer de bouton permettant de modifier les modèles sauvegardés par l'utilisateur. Les feuilles sont seulement cachées. Il est fortement déconseillé de les modifier manuellement.

de clarté. *Librairie* est composé de deux pages. La première page contient les modèles préenregistrés et la seconde, les modèles enregistrés par l'utilisateur.

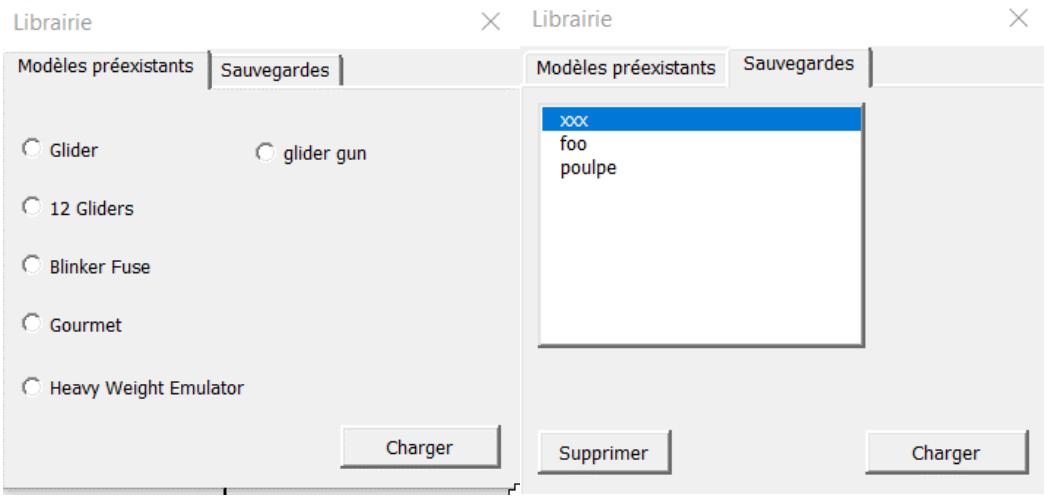


Figure 11 : l'userform Librairie

Les modèles préexistants

Il s'agit de modèles que nous avons ajouter manuellement au jeu. Ce sont des modèles plutôt connus qui permettent de donner à l'utilisateur une idée des possibilités du jeu. Ces modèles se trouvent dans des feuilles cachées de l'utilisateur. En cliquant sur *Charger*, la procédure privée *Charger_Click()* récupère le nom du modèle choisi avec les OptionButtons, le stocke dans une variable globale appelée *modele* et appelle la fonction *chargerTable* située dans le code VBA de la feuille *Plateau*. *chargerTable* prend comme argument une chaîne de caractère correspondant au nom de la feuille sur laquelle le modèle se trouve, copie ses valeurs et les collent sur la grille de *Plateau*.

```
Private Sub blinkerfuseOpt_Click()

    'OptionButton de nom blinkerfuse'
    If blinkerfuseOpt.Value = True Then UserForm1.modele = "blinkerfuse"

End Sub

Private Sub Charger_Click()

    Feuil1.chargerTable (UserForm1.modele)
    Worksheets("Plateau").Cells(47, 72).Value = "Modèle : " & modele
    'Affiche le nom du modèle dans le statut du jeu'
    UserForm1.Hide
    Unload UserForm1

End Sub
```

Figure 12 : récupérer une valeur parmi un ensemble de OptionButton et la procèdure Charger_Click()

```
Sub chargerTable(modele)

        'modele est récupérée dans l'userform Librairie.
        'chargerTable permet de charger le modele renseigné déjà existant.
        Worksheets(modele).Range("A1", "BR50").Copy
        Worksheets("Plateau").Range("A1", "BR50").PasteSpecial Paste:=xlPasteValues
        Worksheets("Plateau").Range("A1", "BR50").PasteSpecial Paste:=xlPasteValues
        'deux fois car ne fonctionne pas toujours en simple
        Worksheets("Plateau").Range("A1").Select
        'pour ne pas avoir tout le tableau selectionné juste après la copie

End Sub
```

Figure 13 : la procèdure chargerTable

Les modèles sauvegardés

Il s'agit des modèles sauvegardés par l'utilisateurs via la commande *Sauvegarder*. Cette page contient une listbox *ListBox1* de source[7] la liste dynamique *Sauvegardes* introduite en III. ii. Et deux CommandButtons *supprimerSauvegarde* et *chargerSauvegarde*.

Par une méthode analogue à celle présentée en a), *chargerSauvegarde* récupère le nom du modèle choisi et appelle *chargerTable* dans la feuille *Plateau*.

Le bouton *supprimerSauvegarde* permet de supprimer un modèle sauvegardé. En cliquant dessus, il supprime entièrement la ligne[8] contenant le nom du modèle dans la liste dynamique *Sauvegardes*. En revanche, il est impossible de supprimer tous les modèles sauvegardés.

[7] ListBox1.RowSource = *Sauvegardes*

[8] Cela permet de faire remonter dans la liste tous les éléments situés à la suite. Nous n'avons donc pas d'élément sans texte.

En effet, la *ListBox1* est reliée à la liste *Sauvegardes*, or celle-ci ne peux pas ne pas contenir aucun élément, autrement la formule de la figure 8 contiendrait un problème de référence. Pour pallier ce problème, nous avons créé une procédure permettant de verrouiller le bouton *supprimerSauvegarde* si le nombre d'éléments de la liste *Sauvegardes* est égal à 1.

```
Private Sub enableSupprimer()
'on ne peut pas supprimer une sauvegarde si c'est la seule restante

    If Worksheets("Plateau").Range("Sauvegardes").Rows.Count = 1 Then
    'Combien de lignes dans Sauvegardes
        UserForm1.supprimerSauvegarde.Enabled = False
    Else
        UserForm1.supprimerSauvegarde.Enabled = True
    End If

End Sub
```

Figure 14 : La procèdure enableSupprimer

De même, si l'on souhaite supprimer le premier élément de *listBox1*, on ne peut pas simplement supprimer l'élément de la liste correspondant car sa référence absolue est utilisée dans la formule définissant *Sauvegardes*. Pour remédier à ce problème, nous remplaçons chaque éléments (sauf le dernier) de *Sauvegarde* par l'élément juste en dessous de lui, puis nous supprimons la dernière ligne de *Sauvegardes* (qui contient donc un doublon)[9].

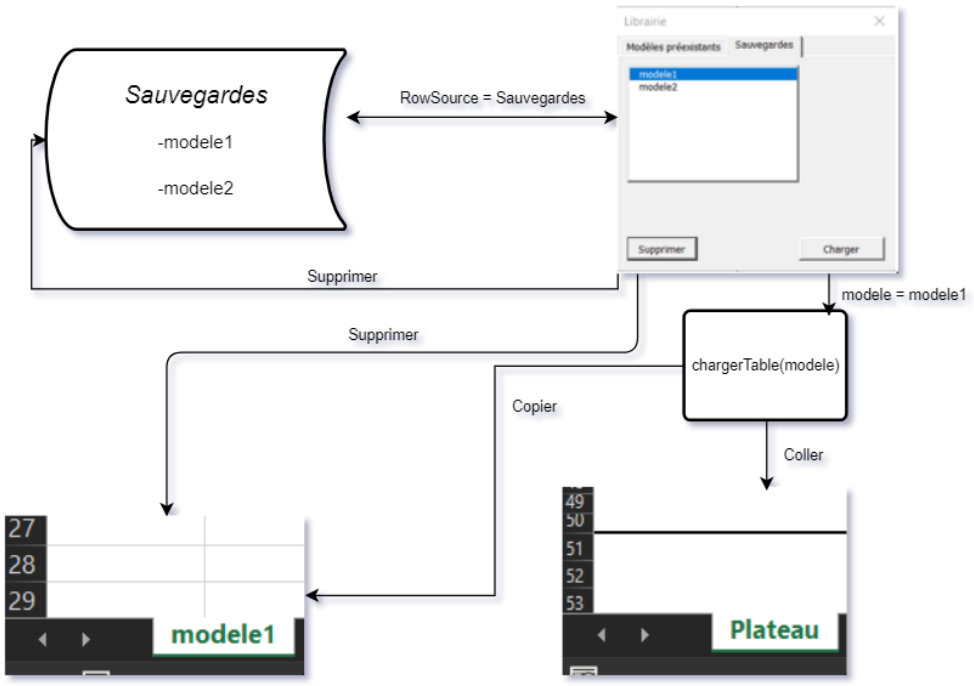


Figure 15 : algorithme de suppression et de chargement d'un modèle sauvegardé

[9] Il s'agit en fait d'un cycle de matrice $\begin{pmatrix} 1 & 2 & \cdots & n-1 & n \\ 2 & 3 & \cdots & n & 1 \end{pmatrix}$ auquel on supprime la dernière colonne.

Options

Le bouton *Options*[10] permet de changer les règles de naissance et de survie, de donner une limite au nombre de générations que l'on souhaite observer et de régler le délai entre chaque génération.

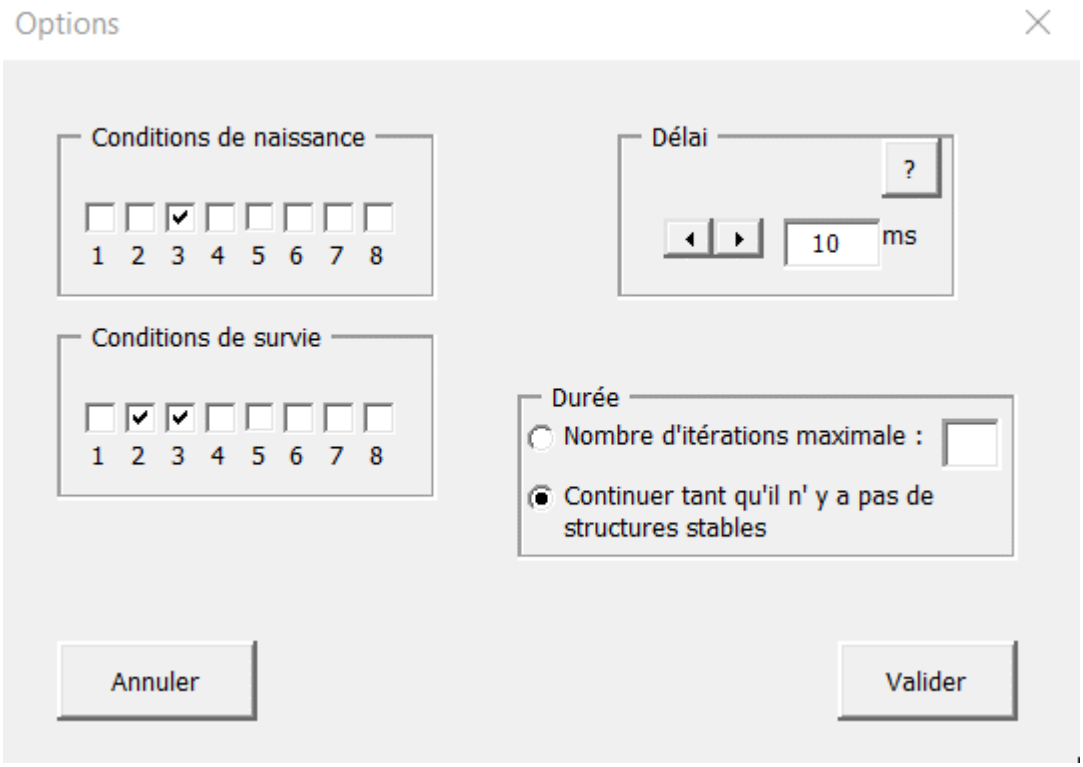


Figure 16 : l'UserForm Options

Changer les règles

A l'ouverture du jeu, les valeurs par défaut sont celles décrites dans les instructions du jeu, c'est-à-dire, deux ou trois cellules voisines vivantes pour rester en vie et trois cellules voisines vivantes pour naître (S23/B3 : Successor = 2 or 3 / Born = 3). On peut changer ces valeurs en cochant les conditions de naissance et de survie que l'on souhaite. Cela se répercute naturellement dans le calcul de la prochaine génération avec la fonction *nextG.* Afin de récupérer les valeurs de l'UserForm *Options*, nous utilisons deux variables globales *regleSurvie* et *regleNaissance* qui sont des chaînes de caractère. Ces variables prennent une succession de chiffres correspondant aux règles choisies. Par exemple en S23/B3, regleSurvie

[10] En réalité, en cliquant sur le bouton *Options*, on ouvre l'UserForm *Options*

= 23 et regleNaissance = 3. Cela nous permet ensuite d'afficher les règles à l'utilisateur dans la fenêtre « statut » du jeu.

```
Private Sub validerOptions_Click()
    regleNaissance = ""
    'On veut savoir quelles règles sont actives sur le plateau sans avoir à cliquer sur options
    If CheckBox1.Value = True Then regleNaissance = regleNaissance & "1"
    If CheckBox2.Value = True Then regleNaissance = regleNaissance & "2"
    If CheckBox3.Value = True Then regleNaissance = regleNaissance & "3"
    If CheckBox4.Value = True Then regleNaissance = regleNaissance & "4"
    If CheckBox5.Value = True Then regleNaissance = regleNaissance & "5"
    If CheckBox6.Value = True Then regleNaissance = regleNaissance & "6"
    If CheckBox7.Value = True Then regleNaissance = regleNaissance & "7"
    If CheckBox8.Value = True Then regleNaissance = regleNaissance & "8"

    regleSurvie = ""
    If CheckBox10.Value = True Then regleSurvie = regleSurvie & "1"
    If CheckBox11.Value = True Then regleSurvie = regleSurvie & "2"
    If CheckBox12.Value = True Then regleSurvie = regleSurvie & "3"
    If CheckBox13.Value = True Then regleSurvie = regleSurvie & "4"
    If CheckBox14.Value = True Then regleSurvie = regleSurvie & "5"
    If CheckBox15.Value = True Then regleSurvie = regleSurvie & "6"
    If CheckBox16.Value = True Then regleSurvie = regleSurvie & "7"
    If CheckBox17.Value = True Then regleSurvie = regleSurvie & "8"

    Worksheets("Plateau").Cells(45, 72).Value = "Règles : " & "S" & regleSurvie & "/B" & regleNaissance
    Worksheets("Plateau").Cells(43, 72).Value = "Délai : " & SpinButton1.Value & " ms"
    Options.Hide 'On ne peut pas unload car on utilise directement les valeurs des CheckBox dans nextG

End Sub
```

Figure 17 : validation des options

On affiche également le délai choisi par l'utilisateur (de 10 ms par défaut). L'utilisateur peut varier le délai en rentrant manuellement une valeur numérique dans le textBox ou en utilisant la toupie. Il faut donc relier ces deux outils.

```
Private Sub SpinButton1_Change()
    delaiTextBox = SpinButton1.Value    'on lie la toupie à la textbox
End Sub

Private Sub delaiTextBox_Change()
    SpinButton1.Value = Val(delaiTextBox)    'on lit les valeurs rentrées manuellement dans la textbox à la toupie
End Sub
```

Figure 18 : liaison SpinButton / TextBox

On peut aussi cocher le bouton d'option « Nombre d'itérations maximale » et renseigner le nombre voulu. Cela modifie la fonction *jouer*, qui, au lieu de boucler tant que la simulation n'est pas stable, va boucler jusqu'à cette limite.

Implémenter les nouvelles règles

Les règles de naissance et de survie viennent simplement modifier le calcul de la prochaine génération dans *nextG*. On récupère les valeurs de chaque règle directement dans le UserForm *Options*. Cela nous permet de ne pas avoir à créer seize nouvelles variables (chacune correspondant à S1, S2 , ⋯, B1, ⋯, B8) et simplifie grandement le code de *nextG*.

La contrepartie est que l'on ne peut plus Unload ce UserForm une fois ouvert une première fois.

```
Public Sub nextG(ligne, colonne) 'Procèdure qui renvoie l'état d'une cellule à la prochaine génération, en fonctions des règles choisies
    Dim compteur As Integer
    Dim i As Integer
    Dim j As Integer
    compteur = 0
    For i = ligne - 1 To ligne + 1 'boucle qui relève les valeurs des cases adjacentes en comptant elle-même
        For j = colonne - 1 To colonne + 1
            If i = 0 Then i = 50  'on aurait pu utiliser offset mais cela aurait posé problème pour les coins du tableau'
            If i = 51 Then i = 1
            If j = 0 Then j = 70
            If j = 71 Then j = 1
            compteur = compteur + Worksheets("Plateau").Cells(i, j)
        Next
    Next
    compteur = compteur - Worksheets("Plateau").Cells(ligne, colonne).Value  'si la case est vivante on soustrait 1 au compteur

    If Worksheets("Plateau").Cells(ligne, colonne).Value = 0 Then  'conditions de naissance selon les règles choisies dans Options
            If Options.CheckBox1.Value = True And compteur = 1 _
                Or Options.CheckBox2.Value = True And compteur = 2 _
                Or Options.CheckBox3.Value = True And compteur = 3 _
                Or Options.CheckBox4.Value = True And compteur = 4 _
                Or Options.CheckBox5.Value = True And compteur = 5 _
                Or Options.CheckBox6.Value = True And compteur = 6 _
                Or Options.CheckBox7.Value = True And compteur = 7 _
                Or Options.CheckBox8.Value = True And compteur = 8 Then
                Worksheets("nextGen").Cells(ligne, colonne).Value = 1

            Else
                Worksheets("nextGen").Cells(ligne, colonne).Value = 0

            End If
        Else   'conditions de survie i.e. la case est vivante et vaut 1
            If Options.CheckBox10.Value = True And compteur = 1 _
                Or Options.CheckBox11.Value = True And compteur = 2 _
                Or Options.CheckBox12.Value = True And compteur = 3 _
                Or Options.CheckBox13.Value = True And compteur = 4 _
                Or Options.CheckBox14.Value = True And compteur = 5 _
                Or Options.CheckBox15.Value = True And compteur = 6 _
                Or Options.CheckBox16.Value = True And compteur = 7 _
                Or Options.CheckBox17.Value = True And compteur = 8 Then
                Worksheets("nextGen").Cells(ligne, colonne).Value = 1

            Else
                Worksheets("nextGen").Cells(ligne, colonne).Value = 0

            End If
        End If

End Sub
```

Figure 19 : la procédure nextG

Par défaut, à l'ouverture du classeur, on met les règles S23/B3 et pas de limite d'itération.

```
Private Sub Auto_Open()

    Options.CheckBox3.Value = True  'par défault, à l'ouverture du jeu, on met les règles S23/B3 et pas de limite d'itération
    Options.CheckBox11.Value = True
    Options.CheckBox12.Value = True
    Options.OptionButton2.Value = True

End Sub
```

Figure 20 : paramètres par défaut à l'ouverture du classeur

Afin d'utiliser le délai, nous utilisons la fonction Sleep(). Nous avons aussi besoin des modules suivants.

```
#If VBA7 Then
    Declare PtrSafe Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As LongPtr)
#Else
    Declare Sub Sleep Lib "kernel32" (ByVal dwMilliseconds As Long)
#End If
```

Figure 21 : module délai

Les conditions d'arrêt

La simulation peut s'arrêter sous plusieurs conditions, que l'on a entré un nombre d'itérations maximale ou pas. D'abord, l'utilisateur peut décider d'arrêter la simulation en cliquant sur « Arrêter » (*CommandButton1*). Dans ce cas, le booléen *arret* prend la valeur True.

Une autre condition d'arrêt est l'obtention d'un état stable de la simulation ; la disposition des cellules ne bougera plus. C'est par exemple le cas pour cette configuration :

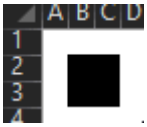


Figure 22 : état stable

Nous ajoutons donc à la fonction *test* (fonction qui appelle la procédure *nextG* sur toutes les cellules puis qui copie/colle le résultat obtenu dans *Plateau*), un test permettant de détecter s'il y a eu un changement entre l'état actuel et l'état calculé suivant. On introduit un booléen *different* qui renvoie True tant qu'il y a au moins une cellule qui diffère dans son état à la génération *n+1* par rapport à la génération *n* et False sinon. Dès que *different* est False, on assigne à *arret* la valeur True.

```
168   Sub test()
169
170       Dim ligne As Integer
171       Dim col As Integer
172       For ligne = 1 To 49
173           For col = 1 To 69
174               Call Feuil1.nextG(ligne, col)
175               If Worksheets("nextGen").Cells(ligne, col) <> Worksheets("Plateau").Cells(ligne, col) Then different = True
176           Next
177       Next
178       If different = False Then arret = True 'On initialise different = False à chaque call de test()
179
180       Worksheets("nextGen").Range("A1", "BR50").Copy
181       Worksheets("Plateau").Range("A1", "BR50").PasteSpecial Paste:=xlPasteValues
182       Worksheets("Plateau").Range("A1").Select
183
184   End Sub
```

Figure 23 : la procédure test() et sa condition d'arrêt

Tant que la simulation n'est pas stable, à chaque appel de *test()* par *jouer()*, *different* aura pris la valeur True. Or *test()* peut uniquement changer la valeur de *different* de False à True. Ainsi, à chaque appel de *test()* dans *jouer()*, on initialise avant *different* à False.

Finalement, la dernière condition d'arrêt existe quand on atteint la limite d'itération imposée par l'utilisateur. Cette potentielle limite est directement utilisé par *jouer()* depuis l'UserForm *Options.*

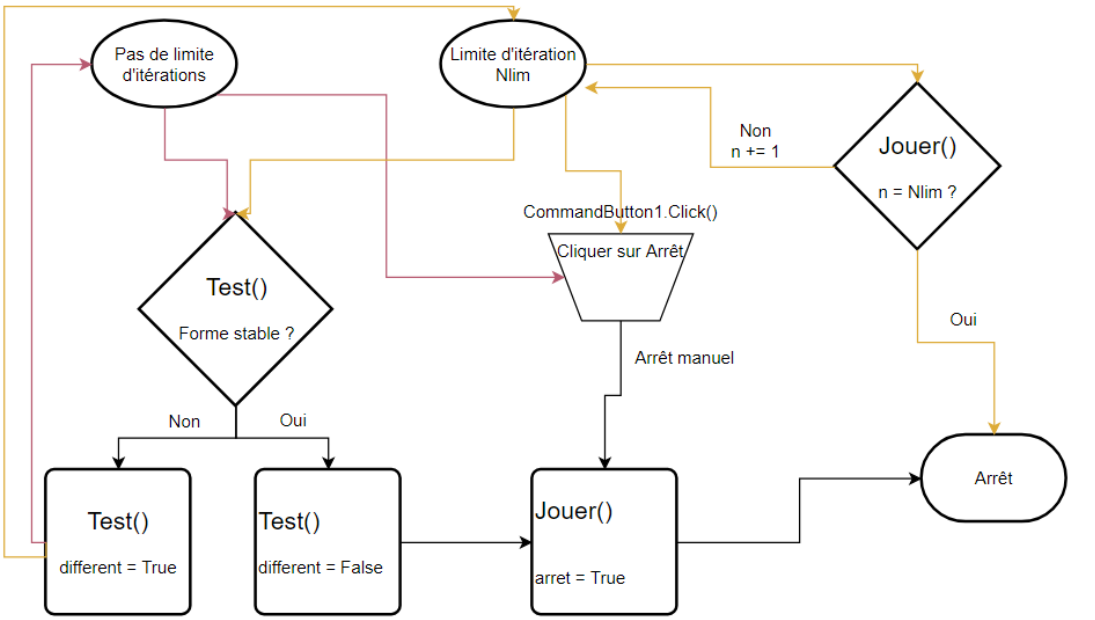


Figure 24 : différentes conditions d'arrêt

```
Sub jouer()

    arret = False
    generationN = 0
    If Options.OptionButton2.Value = True Then 'cas sans limite d'itérations
        While Not arret
            Worksheets("Plateau").Cells(41, 72).Value = "Génération n° " & generationN   'génération n dans le statut de la simulation
            different = False
            Application.ScreenUpdating = False  'smooth transitions between states, permet notamment d'interagir pendant la simulation
            Application.EnableEvents = False
            Call test
            generationN = generationN + 1
            Application.EnableEvents = True
            Application.ScreenUpdating = True
            DoEvents
            Sleep (Options.delaiTextBox)  'delai que l'utilisateur a rentré

        Wend
        If Not different Then MsgBox ("Forme stable")
        Feuil1.CommandButton1.Caption = "Jouer"  'Le programme est arrêté -> on ne propose pas l'option arrêt mais l'option jouer
        Feuil1.CommandButton1.BackColor = &HFF00FF
    End If

    If Options.iterationsOpt.Value = True Then 'cas avec limite d'itération
        Dim i As Integer
        For i = 0 To Options.iterationTextBox  'limite imposée par l'utilisateur
            If Not arret Then
            Worksheets("Plateau").Cells(41, 72).Value = "Génération n° " & i & " / " & Options.iterationTextBox
            Application.ScreenUpdating = False  'smooth transitions between states
            Application.EnableEvents = False
            Call test
            Application.EnableEvents = True
            Application.ScreenUpdating = True
            DoEvents
            Sleep (Options.delaiTextBox)
            End If
            If Not different Then  'le programme s'arrête
            MsgBox ("Forme stable")
            Feuil1.CommandButton1.Caption = "Jouer"  'Le programme est arrêté -> on ne propose pas l'option arrêt mais l'option jouer
            Feuil1.CommandButton1.BackColor = &HFF00FF
            Exit For
            End If
        Next i
        Feuil1.CommandButton1.Caption = "Jouer"  'Le programme est arrêté -> on ne propose pas l'option arrêt mais l'option jouer
        Feuil1.CommandButton1.BackColor = &HFF00FF
    End If

End Sub
```

Figure 25 : la fonction jouer()

Remplissages

Le bouton *Remplir* permet de remplir la grille de cellules vivantes. Le bouton *Remplissage Aléatoire* ouvre l'UserForm *Aleatoire*.

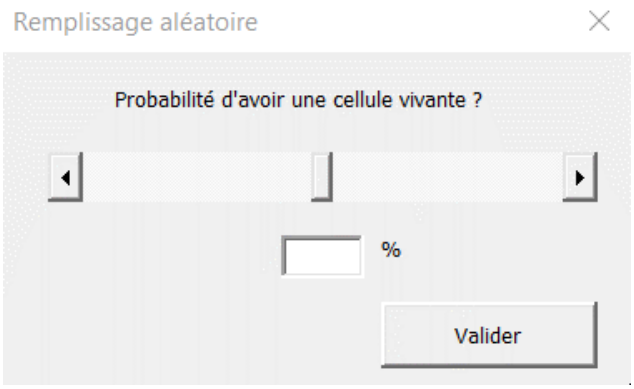


Figure 26 : l'UserForm Aleatoire

Celui-ci dispose d'un curseur variant de 0 à 100 ainsi qu'un textBox où l'on peut rentrer manuellement la valeur souhaitée. Le curseur est lié à la textBox par la procédure suivante.

```
263   Private Sub curseurRemplissageAleatoire_Change()
264
265       probaTextBox = curseurRemplissageAleatoire.Value  'met le pourcentage choisi sur le curseur dans le textBox
266
267   End Sub
```

Figure 27 : liaison curseur / textBox

En cliquant sur « Valider », la procédure suivante se déclenche.

```
253   Private Sub ValiderRemplissageAleatoire_Click()
254
255       Call plateauAleatoire
256       aleatoire.Hide
257       'On ne peut pas unload car on utilise curseurRemplissageAleatoire.Value
258       Worksheets("Plateau").Cells(46, 72) = "Modèle : Aléatoire (" & probaTextBox & " %" & " de cellules vivantes)"
259
260   End Sub
```

Figure 28 : procédure Valider Remplissage Aléatoire

On utilise ici la procédure plateauAleatoire qui assigne pour chaque cellule dans la grille l'état vivant ou mort suivant la probabilité renseignée dans *Aleatoire*.

```
271   Sub plateauAleatoire()
272
273       Dim i, j As Integer
274       For i = 1 To 50
275           For j = 1 To 70
276               Worksheets("Plateau").Cells(i, j).Value = coinFlip
277           Next
278       Next
279
280   End Sub
```

Figure 29 : la procédure plateauAleatoire

Cette procédure fait appel à la fonction coinFlip qui n'est autre qu'un schéma de Bernoulli de probabilité le pourcentage rentré dans *Aleatoire*.

```
283   Public Function coinFlip() As Integer  'génère un lancé avec la proba donnée dans le curseur
284
285       If Rnd() < (aleatoire.curseurRemplissageAleatoire.Value / 100) Then
286           coinFlip = 1
287       Else
288           coinFlip = 0
289       End If
290
291   End Function
```

Figure 30 : la fonction coinFlip

Conclusion

Le jeu de la vie est un projet informatique assez classique car plutôt simple à coder. Nous pensons avoir fait un bon travail au niveau de l'implémentation de l'algorithme. De nombreuses modifications peuvent cependant être apportées.

D'un point de vue performance, il est possible d'améliorer la rapidité du code et du temps de calcul entre chaque génération, notamment en utilisant un objet tableau contenant les calculs de *nextG* (il remplacerait la feuille *nextGen*) et en simplifiant les conditions d'arrêt avec l'utilisation de GoTo. Concernant l'interface utilisateur, nous n'avons pas réussi à établir une méthode privée permettant de changer le statut de plusieurs cellules à la fois par un cliquer-glisser. Il faut donc se contenter de changer l'état de chaque cellule un à un, ce qui prend un temps fou. Nous aurions également voulu proposer une option afin de modifier les modèles que l'utilisateur a sauvegardé. Nous ne l'avons pas fait par manque de temps.

Finalement, nous sommes satisfaits du rendu de notre projet. Avec un peu de recul, c'est un projet idéal pour débuter en VBA. Il nous a permis d'utiliser un large éventail de fonctionnalités tout en augmentant progressivement en complexité et il est certain que nous sommes tous ressortis enrichis de cette expérience.