Dossier-projet ISN :
Jeu de la vie de CONWAY

Yann BOUCHET
Louise THILL
Thierry JACOBS

BOUCHET Yann

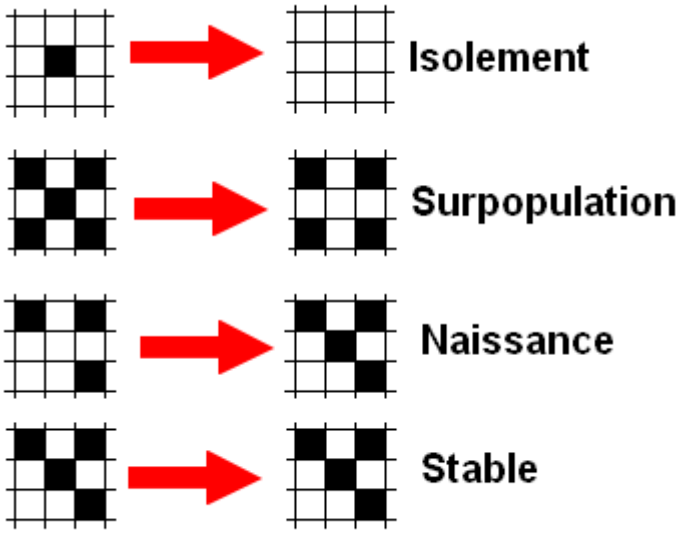
Sommaire :

BOUCHET Yann

I. Présentation du projet et origine

1) Présentation

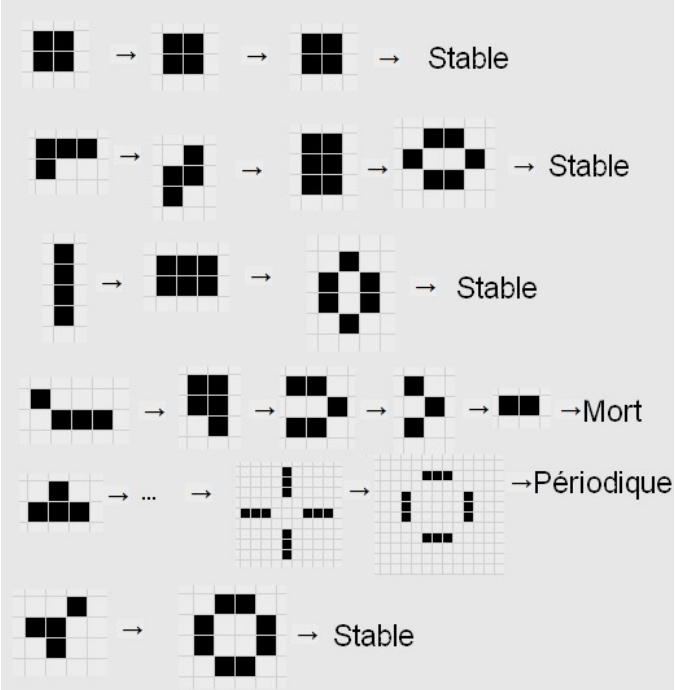
Issu d'un jeu du célèbre mathématicien britannique John HORTON CONWAY, notre projet de «Jeu de la vie» est une simulation, qui comme son nom l'indique, tenterait de «simuler la vie». Dans ce jeu un peu particulier que CONWAY qualifie lui même de «no player game» c'est-à-dire de «jeu sans joueur» dans une interview, il s'agit pour l'utilisateur de sélectionner des cases ou «cellules» sur une grille, avant de lancer une simulation au tour par tour, aux règles simples.



Règles du jeu de la vie

Une cellule ayant moins de deux voisins pendant un tour se voit ainsi mourir d'«isolement» et disparaît de la grille au tour suivant. Autre règle : si une cellule en côtoie quatre autres, elle meurt alors de «surpeuplement». Finalement, une cellule morte à trois voisins naît.
Ce sont donc grâce à ces trois règles simples que la simulation se déroule.

Pourtant, ces règles plutôt basiques amènent à un comportement vraiment très intéressant : on dit que les parties du jeu de la vie ont un caractère impossible à prévoir à l'avance. C'est ce qui rend le jeu à mon avis si unique et ludique. Pouvoir partir de presque rien, de la façon dont on le souhaite, pour finalement arriver parfois à quelque chose d'étonnant et totalement imprévu, rend l'expérience presque envoûtante.



Structures basiques connues du jeu de la vie (Image modifiée)

La partie a alors deux manières de se dérouler. La première est une simulation infinie. En effet, certaines configurations de cellules particulières ont des propriétés leur permettant une survie infinie. Ces configurations dites «stables» ou «périodiques» conservent leur statut si elles ne sont pas en contact d'une autre structure pouvant ainsi détruire leur configuration.

La seconde possibilité est une partie au nombre de tours finis. Dans ce cas, après un nombre X de tours, toutes les cellules auront été détruites. Certaines configurations sont d'ailleurs vouées à une mort rapide si elles ne rentrent pas en contact avec un autre groupe de cellules.

Finalement, si deux groupes de cellules rentrent en contact, il y a alors une possibilité de fusion, créant ainsi une structure plus grande qui évoluera, elle aussi, d'une manière différente. Il est également à noter qu'il existe beaucoup d'autres structures par exemple au caractère périodique et que ceci est juste un résumé des formes les plus basiques trouvées dans le jeu. Il est existe de même des structures impossible à réaliser avec les règles du jeu de la vie. Par exemple, le «Jardin d'Éden» est une de ces structures. Il est uniquement possible de la réaliser manuellement.

2) Origine du projet

Lorsque les professeurs nous ont parlé du projet de fin d'année, nous ne savions pas vraiment quoi réaliser. Nous étions d'accord pour tenter quelque chose d'original et pouvant être ludique à utiliser. Passionné de problèmes logiques, j'ai orienté mes recherches sur une chaîne anglaise de vidéos sur les mathématiques, sur YouTube, nommée **Numberphile**. Je suis alors tombé par hasard sur l'**interview de CONWAY** qui m'a beaucoup intrigué. Pour en savoir plus, j'ai recherché d'autres vidéos sur le sujet sur le site. Le vidéaste français **ScienceEtonnante** et sa **vidéo** sur le jeu de la vie a alors confirmé mes attentes. J'en ai ensuite parlé à mes coéquipiers en expliquant le principe du jeu. Le retour positif a finalement confirmé le choix du projet de fin d'année.

BOUCHET Yann

II. Brainstorming : besoins de l'utilisateur

Après la sélection du jeu, nous avons réalisé un «Brainstorming» afin de comprendre les besoins de l'utilisateur lors de l'utilisation du programme. Chacun proposa alors des idées de fonctionnalités à apporter pour préparer le cahier des charges. Ensuite, nous avons réalisé un dessin de la façon dont nous imaginions l'interface et ses boutons de façon à avoir un modèle à suivre lors de sa création. Le modèle retenu fut simple : une grande grille prenant les trois quarts de l'écran dans le coin en haut à gauche, des boutons et des informations sur la simulation (nombre de tours, de cellules vivantes …) sur le côté droite, des outils (démarrage, sauvegarde ...) en bas. Le tout devait rester sobre visuellement et facile à comprendre.

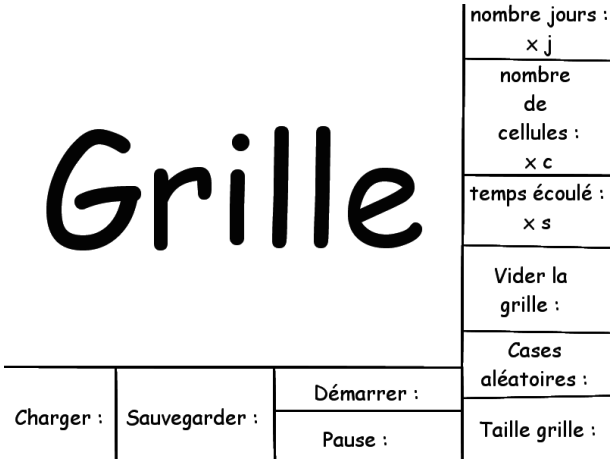


Schéma réalisé pendant le brainstorming

Pour finir, il a fallu penser à une structure globale du programme et de la façon dont les actions se dérouleraient. Ainsi chaque bouton devait être lié à une ou plusieurs fonctions particulières. Certains boutons devaient eux aussi réinitialiser certaines variables ou pouvoir être activés sous certaines conditions (exemple : on ne doit pas pouvoir vider la grille pendant que la partie se déroule, cela pourrait créer des problèmes potentiellement difficiles à gérer).

C'est donc avec toutes ces idées notées au propre sur cahier des charges et la validation des professeurs que le projet débuta.

III. Partage des tâches au sein de l'équipe

Le cahier des charges étant prêt, le partage des tâches devait logiquement s'effectuer. Nous avons tenté de rendre le travail équitable. Cependant, nous verrons après dans la partie suivante qu'il a fallu un remaniement du partage. Thierry devait s'occuper de l'interface et de ses variables ainsi que de la boucle principale. Louise devait créer la grille ainsi que de s'occuper des cases aléatoires. Je devais finalement m'occuper de la sauvegarde et du chargement de grilles prédéfinies afin de montrer certaines structures intéressantes à l'utilisateur.

Nous avons tous travaillé de notre côté sur des fichiers séparés afin de faciliter les tests et de trouver plus facilement les potentielles erreurs. Les versions de nos travaux respectifs suivaient une règle : changement de version si une grosse modification est apportée. Afin de fusionner nos programmes, il nous a fallu attendre l'interface quasiment complète de Thierry car sans elle nous ne pouvions utiliser les boutons et les variables nécessaires à nos fonctions. Il a donc été nécessaire de focaliser son attention sur les parties les plus essentielles d'abord pour ne pas ralentir le projet. Ensuite, une fois la fonction intégrée au bon endroit, il suffisait de modifier une ou plusieurs variables clés pour que le programme fonctionne de nouveau et qu'il puisse aussi nous fournir des statistiques à afficher par exemple.

BOUCHET Yann

IV. Travail personnel : Surestimation du temps temps de travail nécessaire et modification du plan des tâches

1) Surestimation du temps de travail nécessaire et modification du plan des tâches

Lors de l'étape de programmation, j'ai rapidement réalisé que les fonctions outils de sauvegarde et de charge allaient vite être terminées. Un changement dans la manière dont nous pensions la grille (nous sommes passé d'une forme carrée à une forme rectangulaire pour avoir un terrain de jeu plus grand) m'a facilité les conditions pour qu'un fichier puisse être chargé. Il a donc fallu rééquilibrer le travail avec Thierry et on a donc décidé que je m'occuperai de la boucle principale du jeu afin d'alléger sa charge de travail et lui permettre de focaliser son attention sur le chronomètre pour que celui-ci corresponde à nos attentes : un chronomètre précis, s'actualisant seul. Ce choix s'est finalement révélé judicieux et nous a permis d'être plus efficaces.

2) Travail personnel

Mon travail alors réalisé, que je vais présenter et expliquer, va se séparer en quatre sous parties, chacune représentant un morceau différent de code. Le tout sera rangé dans l'ordre chronologique de création.

a) Système de charge

```
def load():
    """Ouvre un fichier .gol/.txt et recupere la grille correspondant"""
    global listeGrille
    print("Etat Pause :", etatPause)
    if etatPause and differenceTemps==0:
        filename = loader.askopenfilename(defaultextension='.gol',
                                          filetypes=(('Text files', '*.txt'), ('GOL files', '*.gol'), ('All files', '*.*')))
        # askopenfilename(".gol"=extension par defaut, filetypes=types de fichiers autorisés)
        # Le tout dans une variable : Si aucun choix effectué (ferme la fenetre) -> variable = None, Sinon variable = True
        if not filename:
            return box.showinfo("Erreur", "Aucun fichier sélectionné")# Termine la fonction, c'est a dire qu'on ne lit pas plus loin

        with open(filename, 'r') as file: # Ouvre le fichier en mode lecture texte sous la variable file
            line = None
            listeGrilleTemp = list()

            while line != "": # Tant que fichier pas termine
                line = file.readline().strip()
                listeGrilleTemp.append(list(line))
            del listeGrilleTemp[-1] # C'est une liste vide

            file.close() # Ferme le fichier

            if not correctArea(listeGrilleTemp, listeGrille): # Appelle la fonction qui verfie la qualite de la grille, retourne un booleen
                box.showinfo("Erreur", "La grille ne correspond pas aux attentus, elle ne sera pas chargée")
                return
            else:
                grille.delete("all") # Supprime l'ancienne grille
                listeGrille = list(listeGrilleTemp)
                nouvelleGrille("charge", listeGrilleTemp)
                compteur(listeGrille)
```

Code correspondant au système de charge

Le système de charge est donc la première fonction que j'ai réalisé. En effet, celle-ci ne demandait pas l'utilisation de l'interface, ni de grille mais uniquement d'un fichier texte que je pouvais créer moi même. Avant toute chose, il a fallu décider de la «forme» de la case, c'est à dire de la manière dont la case serait traitée. Peu avant le projet, nous avons découvert l'étude du binaire, des octets, des images pour un ordinateur. Un nombre binaire n'a de valeur que 0 et 1 pour Faux et Vrai. Les cellules du jeu de la vie ont également ce système de valeur : elles sont vivantes ou mortes. Par conséquent j'ai décidé de nommer les cellules mortes 0 et les cellules vivantes 1. Nous avons d'ailleurs fait de cela une constante pour tout le programme, tout particulièrement pour Louise qui devait ensuite préparer la conversion de la grille chargée en jeu. Ainsi, tous les fichiers de chargements sont uniquement remplis de 0 ou de 1.

La fonction s'intitule ainsi «load()» (traduit par charger). La fonction commence par un string avec triple guillemets. C'est une «docstring». Le but d'une docstring est de résumer partiellement l'objectif de la fonction et de donner son résultat. Ici la fonction ouvre un fichier de format spécial et récupère la grille. Elle est ensuite chargée en jeu.
Nous vérifions l'état des variables liés à la pause. En effet, nous ne voulons pas charger de grille lors d'une partie en cours, cela peut poser des problèmes. On fait ensuite appel à une fonctionnalité de Tkinter permettant d'ouvrir un explorateur de fichier afin de faire un choix. Cette fonction nous permet alors de rendre le choix de fichier plus facile. Elle prend en paramètre l'extension de base, c'est-à-dire le type de fichier qu'elle recherche d'abord puis tous les types autorisés. Si un choix est effectué, elle retourne Vrai, sinon elle retourne None, ce qui équivaut à «rien». Il suffit donc juste de vérifier ensuite cela. Si la variable n'est pas vraie, elle est donc obligatoirement None, on retourne alors un message d'erreur avec une autre fonction de Tkinter : la «messagebox». Cette fonction prend en paramètre le titre à afficher ainsi que le message d'erreur à donner à l'utilisateur.

Maintenant, nous savons que l'utilisateur semble avoir utilisé un fichier valide, on récupère donc son contenu. La ligne «with open(filename, 'r') as file:» permet justement de préparer la récupération. Celle-ci ouvre alors le fichier précédemment récupéré en mode «read» (lire en français). Ce mode lit le texte écrit dans le fichier. A noter qu'il existe d'autres modes comme 'rb' (traduit par «lire les bytes») qui récupère les bytes du fichier, ce qui peut servir dans d'autres cas. Ensuite, le fichier est stocké dans une variable file.
On définit ensuite des variables temporaires de stockage. On obtient alors, dans une boucle tant que, chaque ligne du fichier que l'on lit avec la fonction readline(). Cette ligne est ensuite transformée pour préparer la vérification. On ferme ensuite le fichier, pour éviter tout problème.

```
def correctArea(cells, listeGrille):
    """
    Recupere la liste des cellules
    et verifie si la grille correspond aux attendus
    Pas de chiffres autres que 0 ou 1 ainsi que la taille correspondante
    """
    if not cells or len(cells)!=len(listeGrille):
        print("Fichier vide ou grille de mauvaise taille")
        return False
    for i in range(len(cells)):
        if len(cells[i])!=len(listeGrille[i]):
            print("Grille de mauvaise taille")
            return False
        for j in range(len(cells[i])):
            if not cells[i][j].isdigit() or int(cells[i][j])>1:
                print("Nombre invalide")
                return False
    return True
```

Code de la fonction CorrectArea()

Occupons nous à présent de la fonction correctArea(). Cette fonction vérifie pour nous la qualité de la grille et retourne un booléen Vrai ou Faux si la grille correspond à nos attentes (bonne taille, pas de texte …). Elle prend en arguments la liste récupérée du fichier ainsi que la grille actuelle, afin de comparer les deux plus facilement. On vérifie d'abord si la liste cells, c'est-à-dire notre liste fraîchement récupérée est vide ou si la taille ne correspond pas en largeur. On affiche alors un message en console et on retourne Faux, sinon on continue. On continue ensuite par vérifier la taille de chaque ligne puis si chaque élément de liste est bien un nombre plus petit que 2. Dans le cas contraire, on affiche une erreur dans la console et on retourne à nouveau Faux. Si la grille parvient à passer cette série de tests on retourne alors Vrai : nous pouvons utiliser cette grille.

Selon le booléen reçu, on affiche alors le message d'erreur ou on continue. Dans le cas d'une réussite, on supprime alors l'ancienne grille, plus précisément tous les objets du Canvas «grille». La ligne suivante permet de copier la grille vérifiée dans la variable utilisée par le programme. On crée alors une nouvelle grille avec la fonction nouvelleGrille() qui prend en paramètre le mode et la grille. Attention, cette fonction ayant des paramètres prédéfinis, il n'est pas obligé d'en rajouter, c'est le cas par exemple pour créer une grille vide. Finalement, on met à jour le compteur de cases vivantes.

On a donc récupéré notre grille sauvegardée, elle est affichée et utilisable sur le programme. Cela est donc très utile pour certaines structures particulières. On peut les conserver afin de les réutiliser, les modifier. Cependant, à ce moment là du projet, on ne pouvait que les écrire nous même. Il semblait alors logique de créer un système pouvant le faire pour nous. Cela apporterait une économie de temps et d'efforts !

b) Système de sauvegarde

```
def save():
    """Sauvegarde la grille"""
    global listeGrille
    print("Etat Pause :", etatPause)
    if etatPause and differenceTemps==0:
        directory = loader.asksaveasfile(initialdir=".", title="My GOL", defaultextension=".gol",
                                         filetypes=(('Text files', '*.txt'), ('GOL files', '*.gol'), ('All files', '*.*')))
        # Le tout dans une variable : Si aucun choix effectué (ferme la fenetre) -> variable = None, Sinon variable = True
        if directory:
            texte = listeATexte(listeGrille)
            directory.write(texte) # Ecrit dans le fichier
            directory.close() # Ferme le fichier
```

Code correspondant au système de sauvegarde

Nous avons donc vu que le système de charge engendrait alors l'utilité d'un système de sauvegarde. La fonction save() (traduit par sauvegarde) a donc pour objectif de prendre une grille créée par l'utilisateur et de la transformer en fichier texte utilisable plus tard.

On commence comme avant par récupérer la grille actuelle et vérifier si l'état de pause correspond aux besoins : on ne doit pas sauvegarder une grille si la partie a démarré. C'est ensuite l'arrivée d'une nouvelle fonction de Tkinter nommée asksaveasfile() ouvrant elle aussi un explorateur de fichiers. Celle-ci prend de nombreux arguments :

- le lieu ouvert par défaut. Ici, il correspond à '.'. C'est la façon que Python utilise pour parler du dossier où le programme a été lancé.
- Le titre de la fenêtre
- L'extension utilisée par défaut
- Les types autorisés pour sauvegarder

Comme la dernière fois, cette fonction retourne Vrai ou None selon l'action réalisée. On fait donc une vérification. Cette fois, on vérifie si cela est Vrai, pour ensuite sauvegarder.
On fait alors appel à la fonction listeATexte().

```
def listeATexte(listeGrille):
    texte = ""
    for i in range(len(listeGrille)):
        for j in range(len(listeGrille[i])):
            if listeGrille[i][j] == "0":
                texte += "0"
            else:
                texte += "1"
        texte += "\n"
    return texte
```

Code de la fonction listeATexte()

Cette fonction prend en argument la grille actuelle. Rappel, cette liste est constituée de listes contenant uniquement des chaînes de caractères '0' ou '1'. On crée alors une variable pour stocker le texte à sauvegarder dans le fichier. Pour chaque objet de chaque liste on regarde si la cellule est vivante ou non et on ajoute le bon nombre à la suite de la variable de stockage. On ajoute finalement des sauts de lignes en fin de chaque sous-liste correspondant à une ligne avant de retourner la grille convertie en texte.

Puisque la grille est convertie, il suffit maintenant juste de la sauvegarder. Précédemment, la fonction qui nous à permis de choisir, crée en réalité un objet de la même manière que l'on ouvre un fichier. On demande alors simplement de sauvegarder avec la fonction write() de Python et le préfixe «directory» qui correspond à la variable qui contient le choix. On ferme alors le fichier pour éviter les problèmes.

Nous avons donc fini tout le système utilitaire. Maintenant, il ne nous reste que le boucle principale. Ce travail n'est pas des moindres, il est primordial au fonctionnement global du logiciel. Sans elle, le jeu n'a pas d'intérêt et ne peut fonctionner !

c) Boucle principale

```
def jeu():
    global listeGrille, debut, jours
    listeGrille = vie(listeGrille)
    grille.delete("all")
    nouvelleGrille("charge", listeGrille)
    jours+=1
    joursSVar.set(jours)
    compteur(listeGrille)
    debut = root.after(50, jeu)
```

Code de la boucle principale

Entrons dans le cœur du programme : la boucle principale. C'est ici que la vie des cellules est décidée. Contrairement à avant où j'expliquais les lignes dans le sens d'exécution, je vais d'abord débuter ici par la dernière ligne qui est très importante avant de retourner dans l'ordre logique.

Cette ligne permet la récursivité de cette fonction, c'est-à-dire qu' à chaque fois que cette ligne est exécutée, elle appellera la fonction jeu() qui gère les cellules. Cette fonction after() prend deux arguments et un préfixe. Le préfixe correspond à la variable qui contient l'objet Tkinter, dans notre cas, elle s'appelle root. Le premier argument est un délai en ms avant l'appel de la fonction nommée dans le second argument. Grâce à cela, les tours peuvent s'enchaîner sans aucune action de l'utilisateur et de façon contrôlée et à intervalle de temps régulier. Le programme ne change donc pas instantanément les grilles de façon à ce que l'on puisse observer la simulation. De plus, cette fonction est stockée dans une variable. Cela sert plus loin dans le programme, dans la fonction pause(). Dans celle-ci, se situe une ligne avec le fonction after_cancel() qui prend en argument une variable comme «debut» dans notre cas correspondant au after(). Elle annule le précédent appel de la fonction after(), ce qui permet donc de faire une pause au programme.

Retournons à l'explication du début de la fonction. On «global» tout d'abord certaines variables utiles à la fonction, ce qui nous permet de les modifier ensuite. La fonction vie() prend un argument : la grille actuelle. On stocke d'ailleurs celle-ci dans la variable de la grille actuelle car cette fonction retourne le tour suivant du jeu.

```
def vie(listeGrilleTexte):
    nouveauTour = list()
    for i in range(nbCase):
        nouveauTour.append(["0" for i in range(nbCase-38)])

    for i in range(len(listeGrilleTexte)):
        for j in range(len(listeGrilleTexte[i])):
            nbVoisins = voisins(i, j, listeGrilleTexte)
            if listeGrilleTexte[i][j] == "1" and nbVoisins < 2:
                nouveauTour[i][j] = "0"
            elif listeGrilleTexte[i][j] == "1" and nbVoisins > 3:
                nouveauTour[i][j] = "0"
            elif listeGrilleTexte[i][j] == "0" and nbVoisins == 3:
                nouveauTour[i][j] = "1"
            else:
                nouveauTour[i][j] = listeGrilleTexte[i][j]
    return nouveauTour
```

Code la fonction vie()

On instancie d'abord une liste avec l'instruction list(). Celle-ci fait appel à la classe d'objet liste. Cela équivaut simplement à une liste vide. Pour chaque case en longueur, on ajoute une liste de '0' de taille nbCase-38. On a donc créé une matrice remplie de '0' de même taille que notre grille actuelle. Ensuite pour chaque cellule de la grille actuelle, on compte son nombre de voisins. Selon le résultat, on modifie alors dans notre nouvelle matrice l'état de chaque cellule. Finalement on retourne la matrice du nouveau tour totalement modifiée : c'est notre nouvelle grille.

```
def voisins(i, j, listeGrilleTexte):
    total = 0
    tailleLigne = len(listeGrilleTexte)
    tailleColonne = len(listeGrilleTexte[0])
    for k in range(i-1, i+2):
        if k < 0:
            ligne = tailleLigne-1
        elif k > tailleLigne-1:
            ligne = 0
        else:
            ligne = k
        for l in range(j-1, j+2):
            if l < 0:
                colonne = tailleColonne-1
            elif l > tailleColonne-1:
                colonne = 0
            else:
                colonne = l
            # On a alors un couple de coordonnées (x;y)
            if not (ligne == i and colonne == j):
                if listeGrilleTexte[ligne][colonne] == "1":
                    if not (k != ligne or l != colonne):
                        total += 1
    return total
```

Code de la fonction voisins()

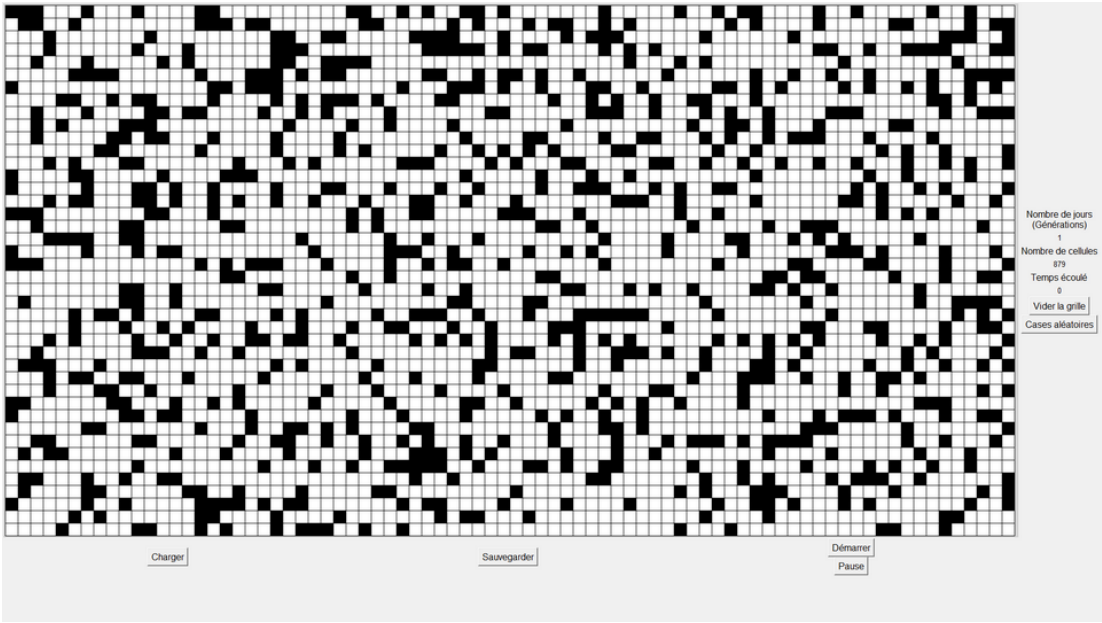
Revenons donc sur le comptage des voisins. La fonction voisins() prend en arguments la valeur variable des boucles de la fonction vie ainsi que la grille actuelle. Ainsi, on peut s'occuper de chaque cellule de la grille actuelle. On instancie d'abord certaines variables utiles dans la boucle : le total, la taille d'une ligne, la taille d'une colonne. Je ne vais pas m'attarder sur les détails de cette double boucle. Cela ne servirait à rien et serait redondant. En résumé, ces doubles boucles nous donnent des couples de coordonnées autour de la cellule étudiée de coordonnée (i;j). Ensuite, on fait une série de vérifications. On cherche tout d'abord à savoir si on ne vérifie pas la case elle-même en tant que voisin. On évite alors un faux positif. En second lieu, on regarde si la cellule étudiée vaut '1'. Finalement, si c'est le cas on vérifie alors si le couple de coordonnées (k;l) possède au moins une coordonnée dans le couple (ligne;colonne) : c'est un voisin ! On augmente donc le nombre total de voisin pour la case sélectionnée. On retourne finalement le total qui sera utilisé comme expliqué précédemment.

Prenons tout de même un exemple pour éclaircir le tout. Supposons que tailleLigne et tailleColonne valent respectivement 80 et 42. Le couple (i;j) vaut (44;32). Suivons un tour de boucle et vérifions si nous trouvons un voisin. Nous avons donc une boucle de variable k allant de 43 à 46. La variable k étant plus grande que 0 et plus petite que tailleLigne-1, on obtient ligne = k avec k = 43. Dans la boucle suivante allant de 31 à 34 de variable l, on fait également le même constat : colonne = l avec l = 31. Ainsi, la première condition est vérifiée. La variable ligne n'est pas égale à i et la variable colonne n'est pas égale à j. Si on suppose également que la cellule de la grille aux coordonnées (44;32) est une cellule vivante, on passe finalement à la dernière condition. Cette ligne peut se lire autrement : «Si (k == ligne ou l == colonne)». Nous observons que les deux conditions sont vérifiées, on augmente alors le total de voisins et on débute un nouveau tour de boucle. Notons qu'une des deux conditions aurait suffit puisque le mot clé utilisé est un «or» et non un «and».

Maintenant que nous avons fini le processus lié à la fonction vie(), terminons l'explication de la fonction jeu(). Nous avons donc récupéré la liste correspondant au nouveau tour et celle-ci est déjà stockée dans la variable correspondante. On supprime donc tous les objets du Canvas «grille» afin de créer la nouvelle avec la fonction nouvelleGrille() expliquée précédemment. Pour finir, on modifie les variables affichées en incrémentant le nombre de jours puis, avec la fonction compteur() qui prend en argument la grille, le nombre de cellules vivantes.

Notre jeu peut alors pleinement fonctionner. Voyons le résultat final.

V. Le résultat final : Le jeu de la vie, prend vie ?



Capture d'écran du programme réalisé (Taille : 50%)

Voila le résultat final ! Comme prévu pendant notre brainstorming, nous avons réalisé l'interface désirée. Elle est sobre et compréhensible, chose qui nous semble primordial pour une application. Les variables fonctionnent bien en temps réel et s'actualisent de façon fluide. La simulation fonctionne parfaitement. On retrouve notamment les différentes structures connues. L'aléatoire permet de faire des grilles vraiment intéressantes et variées.

Capture d'écran des différentes sauvegardes de découverte

Le système de charge permet par exemple d'utiliser certaines grilles personnalisées de structures plus complexes connues que nous avons refaites. L'utilisateur peut correctement sauvegarder sa grille favorite et la partager s'il le souhaite.On a donc réussi à recréer ce jeu de la vie de CONWAY dans Python grâce à Tkinter.

VI. Conclusion

Notre jeu de la vie correspond globalement à l'idée que je me faisais du programme. Cependant, il me manque quelques fonctionnalités que j'aurais jugées utiles mais le temps étant compté, nous avons dû faire ce qui nous semblait le plus important en premier. J'aurais d'abord souhaité ajouter la possibilité de changer la taille de grille comme il était prévu au départ. Cela aurait engendré une refonte du système de charge ainsi que de la création de la grille. L'utilisateur aurait pu lui même modifier la taille de la grille grâce à un slider présent sur le côté de l'interface selon ses envies. Ensuite, j'aurais souhaité pouvoir rendre modifiable l'interface, c'est-à-dire pouvoir laisser l'utilisateur choisir les couleurs des cases, de l'interface. Finalement, j'aurais souhaité ajouter la possibilité d'appliquer des règles variantes au jeu de la vie comme le HighLife. Ainsi, on aurait pu créer un menu de choix de règles supplémentaires avec l'explication des règles pour chaque mode. Après l'épreuve, je compte d'ailleurs tenter d'ajouter ces fonctionnalités dans la mesure du possible et finalement mettre le programme sur mon GitHub, un site de partage de programmes informatiques.

Malgré tout, le programme reste ludique et intéressant comme je le souhaitais. Le projet est donc une réussite pour moi. Le travail en équipe a été amusant et a respecté l'échéance finale. Ce projet confirme finalement mon choix d'orientation dans le développement informatique et plus précisément la sécurité informatique.

BOUCHET Yann

Bibliographie :

Page de garde :

- https://upload.wikimedia.org/wikipedia/commons/thumb/3/30/John_H_Conway_2005.jpg/1200px-John_H_Conway_2005.jpg
- https://mathematrec.files.wordpress.com/2016/09/screen-shot-2016-09-04-at-10-10-52-am.png

Autres :

1. Inspirations :
   a) ScienceEtonnante :
      - https://www.youtube.com/watch?v=S-W0NX97DB0
      - https://www.youtube.com/channel/UCaNlbnghtwlsGF-KzAFThqA

   b) Numberphile :
      - https://www.youtube.com/watch?v=R9Plq-D1gEk
      - https://www.youtube.com/channel/UCoxcjq-8xIDTYp3uz647V5A

2. Sources et images tierces :
   - https://fr.wikipedia.org/wiki/John_Horton_Conway
   - http://www.math.cornell.edu/~lipa/mec/4life2.png
   - https://fr.wikipedia.org/wiki/HighLife_(automate_cellulaire)
   - https://fr.wikipedia.org/wiki/Jardin_d%27%C3%89den_(automate_cellulaire)
   - https://github.com/
3. Images personnelles :
   - Règles (Thierry)
   - Schéma brainstorming