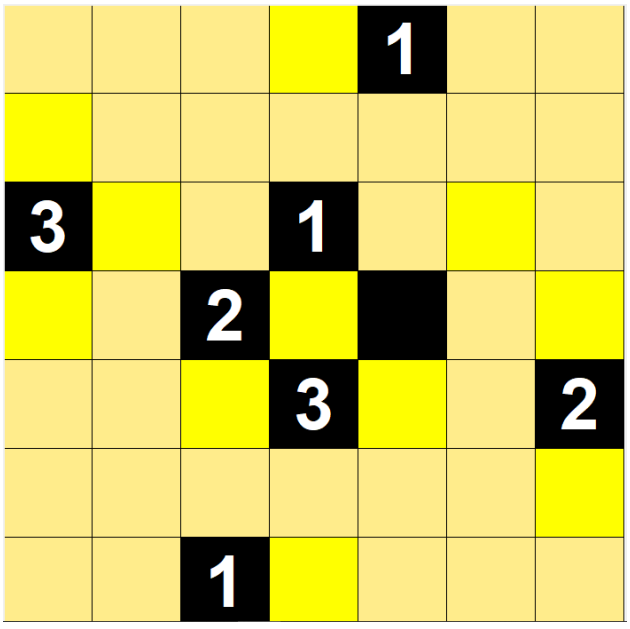
PROJET ISN

SESSION 2019

PROJET D'UN JEU :

AKARI

Dossier-projet



VIALA Alexandre

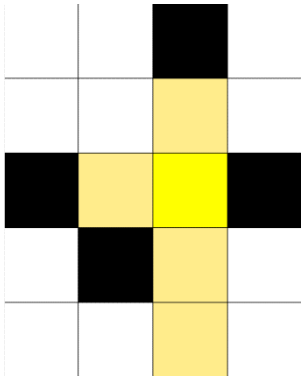
BIETH Elise

LYCÉE DU HAUT-BARR SAVERNE

SOMMAIRE

I) Introduction

La grille logique "Akari" a été créée en 2001par le site "Nikoli", un éditeur japonais spécialisé dans les jeux et les casse-têtes. Son nom vient du Japonais et signifie « lumière ».



Une grille d'Akari représente un faisceau de rues, toutes éclairées par des lanternes. Il s'agit d'un problème de la vie courante : comment éclairer un espace donné en utilisant le moins d'ampoules possible ?

II) Etude du projet

1. Choix du sujet

Nous avons choisi de programmer le jeu Akari car nous aimons les jeu de logiques et les casse-têtes. Ce jeu est peu connu et permet donc de découvrir un nouveau jeu différents du classique « SUDOKU ».

Ainsi ce jeu de logique se fait sur une grille. Il consiste à poser des lampes, qui éclairent en croix (à la manière des bombes d'un bomberman) sur une portée infinie, sur les cases de ladite grille de telle sorte que les lampes ne soient jamais éclairées par les autres lampes mais aussi en respectant les numéros des cases noires. Le but du jeu étant d'éclairer toutes les cases de la grille.

2. Cahier des charges

Nos objectifs sont de créer un jeu programmé avec Python, avec la librairie TKinter, qui respecte le cahier des charges. Ce jeu devra être facile d'utilisation et esthétique pour plaire à tout public.

Détails du cahier des charges du programme :

- générer une fenêtre avec des boutons pour un accès au 3 niveaux
- générer une grille 7x7
- permettre à l'utilisateur de cliquer sur les cases avec le clic gauche de la souris pour mettre une lampe et le clic droit pour mettre une croix
- les cases avec une croix ne devront pas être cliquables avec le clic gauche (empêcher de mettre une lampe sur une croix)
- les cases avec une lampe ne devront pas être cliquables avec le clic droit (empêcher de mettre une croix sur une lampe)
- le placement d'une lampe devra entraîner sur la ligne horizontale et la ligne verticale un éclairage autour de la lampe (passage du blanc au jaune)
- une lampe ou une croix ne devra pas pouvoir être placée sur une case noire
- le jeu devra être capable de vérifier si la solution entrée est exacte

III) Elaboration du projet

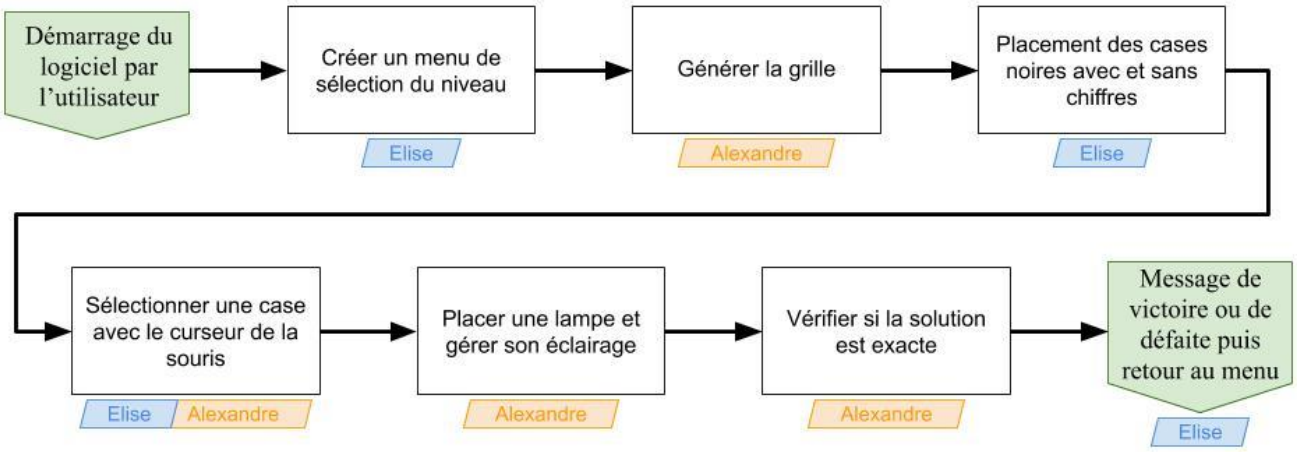
1. Les différentes phases du projet

Nous avons séparé le projet en différentes phases répondant chacune à une fonctionnalité du jeu :

- Générer la grille
- Sélectionner une case avec le curseur de la souris
- Placer des cases noires avec et sans chiffres (allant de 0 à 4)

- Placer une lampe et gérer son éclairage
- Vérifier si la solution est exacte
- Créer un menu de sélection du niveau
- Ajouter la possibilité de mettre une croix avec le clic droit

2. La répartition du travail



➔ Les fonctions que j'ai dû réaliser au cours du projet

➔ Générer la grille :

Pour générer la grille, on procède en plusieurs temps :

Tout d'abord, on commence par appeler la fonction damier à l'intérieur du programme après avoir généré un canevas. Le canevas est un espace que l'on peut créer sous tkinter et qui représente un plan, soumis à un repère où il est possible de placer uniquement des pixels en donnant à ces derniers des coordonnées relatives à la fenêtre où se situe le canevas. On donne donc certaines propriétés à ce canevas avant de l'afficher :

```
# --------- INTERACTIONS AU NIVEAU GAMEPLAY : ----------- #


    # bouton de réinitialsiation:
    bouton_reinit = Button(fen1, text='Réinitialisation', font = ("Arial", 11), command=damier, activebackground= 'LightGoldenrod1',  background='silver')

    # bouton de vérification :
    bouton_verif = Button(fen1, text='Vérification', font = ("Arial", 11), command=verification, activebackground= 'LightGoldenrod1', background='silver')

    Label(text = '',font = 200).grid()

    # création des widgets "esclaves/enfants" :
    can1 = Canvas(fen1,bg='white',height=700,width=700)

    # génération du damier à l'ouverture de la fenêtre tk
    damier()
```

Une fois que ce canevas est généré, on va donc pouvoir créé notre grille à l'aide de la fonction damier() :

```
def damier():
    global x0,y0,x1,y1, position,position_croix, m, victoire_echec

    x0,y0 = 0,0
    x1,y1 = 100,100

    if victoire_echec == 0:
        while y1 <= 700:
            #Première ligne
            while x1 <=700:
                carre(x0,y0,100,'white')
                x0 += 100
                x1 += 100
            x0=0
            y0 += 100
            x1= 100
            y1 += 100
            #print (x0,y0,x1,y1)

        print(position)

        for i in range (0,49):
            Ex = position[i]
            position_croix[i] = 'O'
            if Ex[0] != 'N':
                position[i] = 'O'

        print("réinitialisation: ", position)
        gen_grille(position)
```

Cette fonction va créer un damier en plaçant des carrés blancs. Ces derniers vont avoir une dimension de 100 de côté et vont être posés ligne par ligne : on pose d'abord tous les carrés d'une ligne (grâce à la boucle while x1 <= 700) puis on passe à la suivante (grâce à la boucle while y1 <= 700).

La fonction damier a également pour but de remettre à zéro la grille grâce à une boucle for (la boucle for i in range (0,49). Pour cela, on remplace tout ce qui n'est pas une case noire, c'est-à-dire tout ce qui a la forme 'N' ou 'Nx' et on le remplace par un 'O'. AL fonction gen_grille() permet de générer les cases noires par la suite.

➔ Placer une lampe et gérer son éclairage :

Pour placer et gérer l'éclairage des lampes, on gère d'abord le positionnement des lampes puis on gère l'éclairage des lames.

Pour gérer le positionnement des lampes, on utilise la fonction poser_lampe() :

```
def poser_lampe(event):                                                 # Cette focntion sert à positionner des lampes
    global posLampeX, posLampeY, positionX, positionY, X, Y, place_liste, position_croix, victoire_echec

    if victoire_echec == 0:
        posLampeX = (event.x // 100)*100                                # On fait une division entière puis on multiplie par 100 pour se positionner dans le coin supérieur gauche de la case
        posLampeY = (event.y // 100)*100

        if posLampeX >= 700:                                            # On vérifie si le joueur a bien cliqué sur une case et on rectifie si ce n'est pas le cas
            posLampeX = ((event.x-50) // 100)*100

        if posLampeY >= 700:
            posLampeY = ((event.y-50) // 100)*100

        matrice(posLampeX,posLampeY)                                    # On regarde ce qu'il y a dans la case
        int(place_liste)                                                # C'est le résulatat de ce l'opération effectué avec matrice()
        Ex = position[place_liste]
        longueur = len(Ex)

        if Ex[0] == 'N' or position_croix[place_liste] == 'X':          # on n'effectue aucune opération si l'utiisateur appuie sur une case noire ou sur une croix
            rien = 0
        elif Ex[0] == 'L' :                                             # Si la case cliquée possède une lampe, on enlève cette lampe

            if Ex[1] == '0':
                carre(posLampeX,posLampeY,100,'white')
                position[place_liste] = 'O'
            else:
                if longueur == 3:
                    Z = Ex[1]+ Ex[2]                                    # On va garder le même niveau de luminosité à la case par cette opération
                    position[place_liste] = 'E' + Z
                elif longueur == 2:
                    Z = Ex[1]
                    position[place_liste] = 'E' + Z
                carre(posLampeX,posLampeY,100,'LightGoldenrod1')
            obscurite()

        elif position[place_liste] == 'O' :                             # Si la case cliquée ne possède rien alors on pose une lampe
            carre(posLampeX,posLampeY,100,'yellow')
            position[place_liste] = 'L0'
            eclairage()

        elif Ex[0] == 'E':                                              # Si la case cliquée est éclairée alors on va gérer faire en sorte que la lampe posée ait un niveau de lumniosité identique à la case éclairée
            carre(posLampeX,posLampeY,100,'yellow')
            if longueur == 2:
                position[place_liste] = 'L' + Ex[1]
            elif longueur == 3:
                position[place_liste] = 'L' + Ex[1] + Ex[2]
            eclairage()
```

Cette fonction permet d'utiliser l'information envoyée par l'utilisateur quand il clique sur le bouton droit ou gauche de la souris quelque part sur le canevas pour trouver d'abord sa position puis pour poser une lampe (ou une croix dans le cas de la fonction poser _croix()). On ne pose une lampe que si on est sur uene case blanche ou dorée. On ne pose donc pas de lampe si on a une case noire ou une croix. S'il y a une case avec une lampe alors on enlève cette dernière.

Pour gérer l'éclairage on va actualiser l'éclairage à chaque fois qu'on pose une lampe grâce à la fonction eclairage() et à chaque fois qu'on retire une lampe à l'aide de la fonction obscurite() :

```
def eclairage():                                                    # Cette fonction va permettre de gérer l'éclairage d'une lampe lorsqu'on en pose une
    global place_liste, lumiere, lumiere_d, lumiere_g, posLampeX, posLampeY, Ex, position_croix

    lumiere = place_liste                                           # On va d'abord indiquer au programme où se situe notre lampe
    """ On va ensuite gérer l'éclairage dans cahque point cardinaux, en indiquant la première case qui doit être éclairée en bas, à droite, à gauche et en haut """
    lumiere_d = lumiere + 1
    lumiere_g = lumiere -1
    lumiere_h = lumiere - 7
    lumiere_b = lumiere + 7

    debug_lumiere_b = 0                                             # Cette variable est là pour pallier un bug qui apparaît uniquement dans la boucle d'éclairage vers le bas

    while (lumiere_d) % 7 != 0:                                     # On commence par éclairer tout ce qui est à droite
        print('lumiere_d')
        Ex = position[lumiere_d]                                    # cette variable va stocker le nombre d'éclairage sur une même case
        if Ex[0] != 'N':                                            # On ne traite que les cases qui ne sont pas noires

            if Ex[0] == 'L':                                        # Si on a une case lampe alors dans ce cas on passe à la case suivante
                E(Ex,'L', 1)
                position[lumiere_d] = Ex
                lumiere_d += 1
            elif Ex[0] == 'E':                                      # Grâce à cette variable, on va vérifier si le premier terme dans la case est un E, ce qui signifierait que la case est éclairée
                E(Ex,'E', 1)
                position[lumiere_d] = Ex
                lumiere_d += 1
            else:
                position[lumiere_d] = 'E1'                          # On actualise la liste qui stocke toutes les informations
                grille(lumiere_d)                                   # On transforme la place dans notre liste en des coordonnées dans la grille à l'aide de la fonction grille()
                carre(xh,yh,100,'LightGoldenrod1')                  # On crée un carré pour chaque case éclairée
                if position_croix[lumiere_d] == 'X':                # On regarde s'il y a une croix et on la replace dans ce cas. On fait la même opération pour chaque case éclairée
                    croix(xh, yh)
                lumiere_d += 1                                      # On incrémente pour passer à la case suivante

            print(position)

        else:                                                       # Mais on ne remplace pas les cases noires, ni les cases d'après
            break                                                   # Le break permet de couper la boucle while pour qu'aucune autre case ne soit éclairée par après

    while lumiere_b <= 48:                                          # On éclaire enfin le bas
        print('lumiere_b')
        Ex = position[lumiere_b]
        if Ex[0] != 'N':                                            # On ne traite que les cases qui ne sont pas noires
            if Ex[0] == 'L':
                E(Ex,'L', 1)
                position[lumiere_b] = Ex
                lumiere_b += 7                                      # Dans ce cas, on passe simplement à la case suivante
            elif Ex[0] == 'E':
                E(Ex,'E', 1)
                position[lumiere_b] = Ex
                lumiere_b += 7
            else:
                position[lumiere_b] = 'E1'                          # On actualise la liste qui stocke toutes les informations
                grille(lumiere_b)                                   # On transforme la place dasn notre liste en des coordonnées dans la grille à l'aide de la fonction grille()
                carre(xh,yh,100,'LightGoldenrod1')                  # On crée un carré pour chaque case éclairée
                if position_croix[lumiere_b] == 'X':
                    croix(xh, yh)
                lumiere_b += 7                                      # On incrémente de 7 pour passer à la case suivante

                print(position)

        else:                                                       # Mais on ne remplace pas les cases noires, ni les cases d'après
            break                                                   # Le break permet de couper la boucle while pour qu'aucune autre case ne soit éclairée par après

        debug_lumiere_b += 1
        if debug_lumiere_b == 10:                                   # La boucle se répète 6 fois au maximum en fonctionnement normal donc si le nombre d'itérations est supérieur à 6
            break                                                   # Le programme s'arrête et on peut retenter de cliquer
```

Ici on peut voir la fonction eclairage() partiellement(Normalement on éclaire également à gauche et en haut. Cependant les boucles sont très similaires dans les deux cas cités, il faut juste modifier la condition (% 7 != 0 reste identique pour la gauche et pour le haut la condition est > = 0) et il faut inverser le signe des indentations)

. La fonction obscurite() a un fonctionnement très similaire, cependant pour cette dernière, quand on utilise la fonction E(En, lettre, a), l'argument 'a' est égale à (-1). En effet, cet argument permet d'indiquer de combien ou va indenter le 'x' du string 'Ex'. Quand on éclaire on indente de 1 et quand on enlève de l'éclairage on indente de (-1).

➔ **Vérifier si la solution est exacte :**

Pour vérifier si la solution est exacte, nous avions d'abord pensé à résoudre chaque grille créée puis à créer une liste solution que nous aurions comparée à la liste fournie par le jouer après résolution. Cependant, nous avons préféré opté pour une solution beaucoup plus fiable en effectuant une vérification sur chaque case où l'on regarde plusieurs critères :

-Si le nombre de lampe autour d'une case noire respecte le nombre inscrit sur ladite case noire.

-Si aucune lampe n'est éclairée par une autre

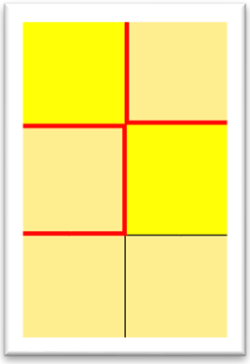
Avec ces critères en tête, on a pu réalisé une boucle for qui va regarder case par case si les conditions sont respectées :

```
def verification():                                                     # Cette fonction permet de vérifier si la réponse donnée par l'utilisateur est correcte
    global nbre_chance, compteur, fen1, victoire_echec

    if victoire_echec == 0:                                             # On vérifie avec ce test si l'écran de victoire ou défaite est actif. On empêche la réinitialisation si jamais c'est le cas
        erreurs = 0

        for i in range (0,49):                                          # On vérifie chaque case à l'aide d'une boucle for
            Ex = position[i]

            if Ex[0] == 'N':                                            # On regarde si la case est une case noire
                l = len(Ex)

                if l == 2:                                              # Il y a des caes noires 'N' et des cases 'Nx', x étant un nombre entre 1 et 4. si le nom de la case fait moins de deux caractères alors on peut mettre ce que l'on désire
                    nbre_noir = int(Ex[1])                              # On prend le nombre de la case noire qui va nous indiquer combien il faut de lampes autour
                    nbre_lampe = 0                                      # On réinitialise le nombre de lampe à chaque itération de la boucle for
                    verif_h, verif_b, verif_d, verif_g = i - 7, i + 7, i + 1, i - 1

                    if verif_h < 0:                                     #Cette opération est ici pour éviter qu'on aille "en dehors" de la grille, on envoie donc 'R' pour indiquer qu'il n'y a rien au-dessus
                        Ex_h = 'R'
                    else:
                        Ex_h = position[verif_h]

                    if verif_b >= 49:                                   # Même opération mais pour le bas
                        Ex_b = 'R'
                    else:
                        print(i)
                        Ex_b = position[verif_b]

                    if verif_d > 49 or (verif_d // 7) != (i // 7):      # Cette opération permet de vérifier si la case considérée est bien sur la même ligne
                        Ex_d = 'R'
                    else:
                        Ex_d = position[verif_d]

                    if verif_g < 0 or (verif_g // 7) != (i // 7):       # Même fonctionnement que pour la vérification à gauche
                        Ex_g = 'R'
                    else:
                        Ex_g = position[verif_g]

                    if Ex_h[0] == 'L':                                  # On vérifie si chaque case autour correspond à une lampe ou non et on incrémante le nombre de lampes à chaque fois qu'on trouve une lampe
                        nbre_lampe += 1
                        print("en haut : ", nbre_lampe)

                    if Ex_b[0] == 'L':
                        nbre_lampe += 1
                        print("en bas : ", nbre_lampe)

                    if Ex_d[0] == 'L':
                        nbre_lampe += 1
                        print("à droite : ", nbre_lampe)

                    if Ex_g[0] == 'L':
                        nbre_lampe += 1
                        print("à gauche : ", nbre_lampe)

                    print("opération logique nbre lampe != nbre_noir: ",nbre_lampe != nbre_noir)

                    if nbre_lampe != nbre_noir:                         # Une fois  qu'on a le nombre de lampe, on compare le nombre sur la case noire et le nombre de lampe : si le nombre est différent alors on compte une erreur
                        erreurs += 1
                        print("erreur cases noires :", erreurs, "numéro case : ", i)

            if Ex[0] == 'L' and Ex[1] != '0':                           # On considère les cases lampe et si on voit que le nombre associé est différent de '0' alors la lampe est éclairée par une autre. On compte donc une erreur
                erreurs += 1
                print("erreur lampe :", erreurs)

            if Ex == 'O':                                               # S'il y a une case blanche, c'est le même tarif : on compte une erreur.
                erreurs += 1
```

On vérifie pour chaque case en premier lieu si la case est une case noire avec un chiffe, on regarde dans ce cas si le nombre de lampe autour est le bon. On regarde ensuite si la case est une case lampe, dans ce cas, si elle est éclairée par une autre, on va également compter une erreur. Une fois que les erreurs ont été décomptées, on regarde si le nombre d'erreur est bien nul et on informe l'utilisateur sur l'état de sa partie.

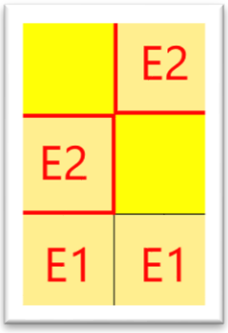
3. Les problèmes et les solutions envisagées

Nous voulions au début créer des grilles avec des cases noires et des chiffres aléatoires, cependant nous ne pourrions pas savoir si la grilles a une solution possible. C'est pour cela que nous avons créé nous même les listes afin que toutes les grilles soient réalisables. Nous en avons ainsi créé plusieurs par niveau pour avoir une plus vaste possibilité de jeu.



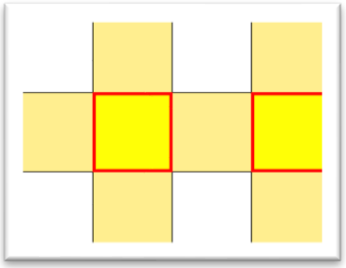
L'autre problème auquel nous nous sommes heurtés fut l'affichage dynamique de l'éclairage. Nous ne voyions pas comment faire en sorte, lorsqu'une case était éclairée par deux lampes pour que celle-ci reste éclairée même lorsque l'on enlève une des deux (voir image ci-contre). Nous avons donc trouvé une solution qui a permis de faciliter grandement notre travail. Il faut avant tout savoir que nous utilisons une liste pour actualiser l'état des cases au niveau du programme. Dans cette liste, nous stockons les informations des cases sous formes de courts sigles :

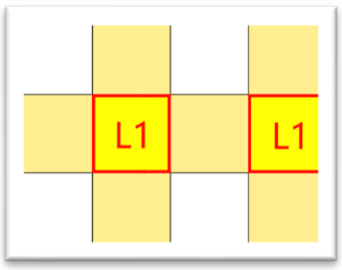
- 'O' correspond à une case vide et donc blanche
- 'Nx' correspond à une case noire avec un nombre x compris entre 0 et 4 ou pas de chiffre (donc pas de x).
- 'Ex ' correspond à une case éclairée avec un nombre x derrière (le nombre va de 1 à, potentiellement, 12.

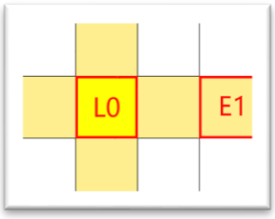


L'astuce repose justement sur ce nombre x associé aux cases éclairées. Comme on peut le voir sur l'image ci-contre, on associe d'abord aux cases non-éclairées (c'est-à-dire égales à 'O') la valeur 'E1'. Si par la suite une case éclairée vient à être éclairée par une autre alors on incrémente le nombre associé à la case éclairée (à l'aide d'une fonction E() qui est détaillée dans le programme). Une case 'E1' devient ainsi une case 'E2'. Cette incrémentation permet de plus facilement mettre à jour les cases éclairées quand on va retirer une lampe. En effet, en la retirant, on va « retirer de la lumière » dans les quatre directions possibles et on va donc se contenter de décrémenter chaque case 'Ex'. Une case 'E2' va alors simplement devenir une case 'E1' et sera donc encore éclairée mais si la case était 'E1' alors la case va passer à l'état 'O' et devenir blanche.

Nous avons ensuite utilisé cette astuce dans le cas des lampes pour également résoudre un problème si jamais une case lampe est éclairée par une autre lampe (voir illustration ci-contre). Dans ce cas-là, de la même façon qu'avant, nous associons une valeur à la lampe qui de base est 'L0'. A chaque fois que la lampe sera éclairée par une autre, on l'incrémentera de 1 (comme on peut le voir sur la deuxième illustration). Quand on retirera la lampe, pour savoir si on doit changer la case lampe en case éclairée, on va regarder le numéro associé à la lampe et si ce dernier est supérieur à 0, on se contente de remplacer la case par une case éclairée et si la valeur de la lampe est égale à 0 alors on la change en case blanche.

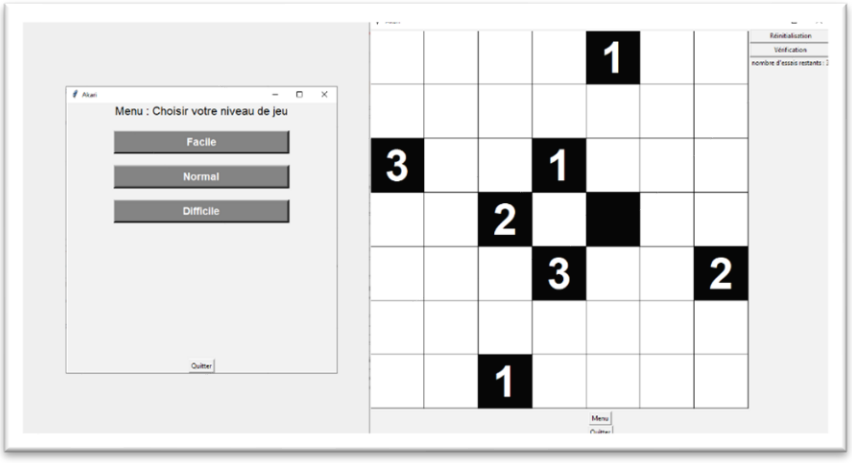






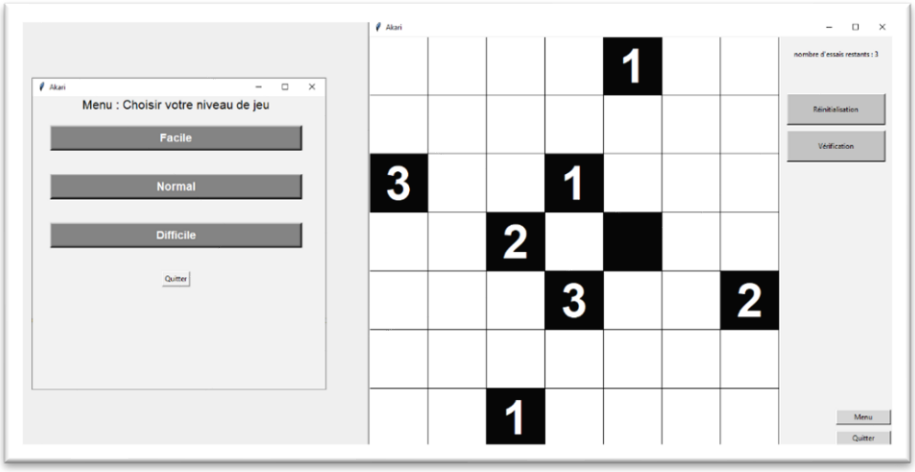
Résultat après l'enlèvement de la lampe

Enfin, nous voulions mettre en page notre menu de manière élégante en plaçant les options (boutons, étiquettes, etc.) sur un des côtés quand nous jouons au jeu. Nous avons d'abord opté pour la méthode pack() de tkinter mais celle-ci de ne nous donnait pas un résultat satisfaisant (pack() permet uniquement de fixer des objets, dans un seul des points cardinaux, c'est-à-dire qu'on fixe l'objet soit à droite, soit à gauche, soit en haut, ou en bas sans possibilité de nuancer). Nous nous sommes donc penchés sur la méthode grid() qui permet bien plus de possibilités (voir ci-dessous les modifications apportées par le passage de pack() à grid()).



Menus avec pack()

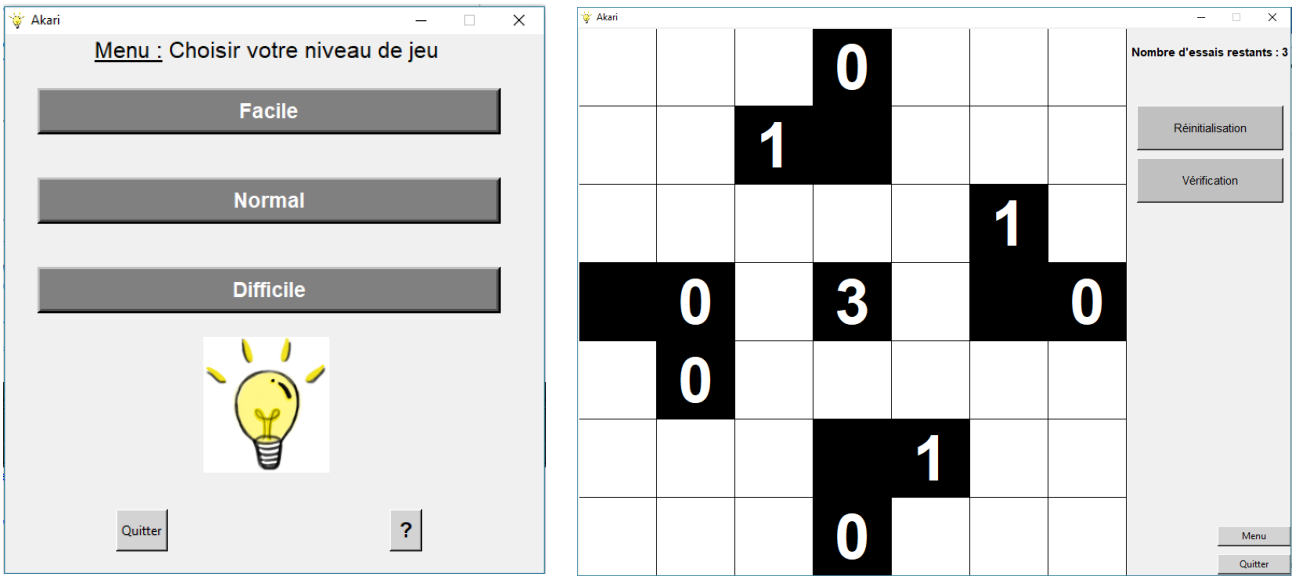
Menus avec grid()



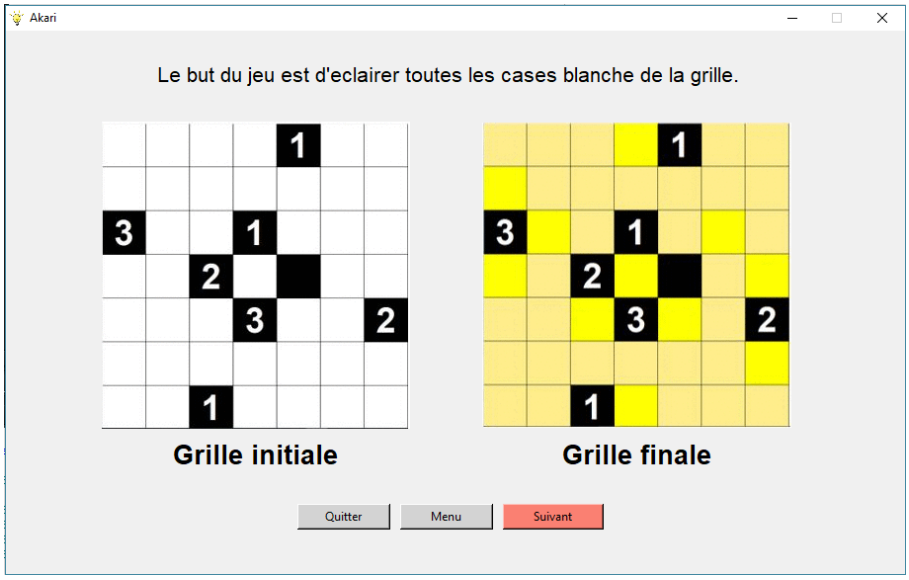
IV) Résultats et perspectives

1. Etat final du projet (bilan)

Nous avons réussi à créer le jeu utilisant toutes les fonctionnalités définis dans le cahier des charges. Nous avons ainsi abouti à une version fonctionnelle, ne comportant pas de défauts majeurs, comprenant trois niveaux de difficulté, et parfaitement jouable.



Capture d'écran du menu Akari et d'une des grilles du jeu



Capture d'écran de la première page du tutoriel d'Akari

2. Amélioration et extensions possibles

Nous pourrions tenter de générer des grilles totalement aléatoires.

Nous pourrions aussi penser à ajouter un chronomètre pour tenter de résoudre les grilles le plus rapidement possible.

Nous pourrions enfin, rendre la fenêtre réajustable en utilisant les spécificités de la méthode grid(), de la méthode geometry() et des méthodes rowconfigure() et columnconfigure() toutes trois implémentées dans la bibliothèque tkinter.

V) Conclusion

1. Impressions personnelles

VI) Annexes

1. Bibliographie et « webographie »

Pour les différentes fonctions de la bibliothèque TKinter :
http://tkinter.fdex.eu/index.html

Pour l'interface graphique :

https://www.youtube.com/watch?v=N4M4W7JPOL4

Pour la mise en page avec la balise grid :

https://stackoverrun.com/fr/q/9448134

2. Le code source commenté du programme réalisé

```
# --------- IMPORTATION FONCTIONS DES LIBRAIRIES : ----------- #

from tkinter import *
from random import randrange



# --------- CARACTERISTIQUES DE LA FENETRE PRINCIPALE : ----------- #


global fen1
fen1 = Tk()         #Il s'agit de la fenêtre maître qui pose les règles

# Nom de la fenêtre :
fen1.title("Akari")

#Taille de base de la fenêtre:
fen1.geometry('500x500')
fen1.minsize(500, 500)
fen1.resizable(width = False, height = False)

# Logo de la fenêtre
fen1.iconbitmap("lampe.ico")



# --------- PARTIE GAMEPLAY : ----------- #


def akari_gameplay(matrice_grille):
    global position, nbre_chance, compteur, position_croix, victoire_echec
    """ On définit deux grilles : une pour les croix, complétement vide ( avec
'O') et une qui dépend de la grille choisie dans le menu (maztrice_grille) """
```

```
    """ Cette variable peremt de savoir si le joueur a gagné, elle empêche le
joueur d'appuyer sur n'importe quel bouton lorsque l'écran de victoire ou d'échec
est actif """
    victoire_echec = 0

    position_croix =
["0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","
0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0","0"
,"0","0","0","0","0","0","0","0"]
    position =  matrice_grille
    nbre_chance = "Nombre d'essais restants : 3"

    def carre(xA,yA,cote,couleur):
# Cette fonction a pour rôle de créer un carré dans le canvas. Cette fonction est
utilisée avec différentes couleurs ( doré, jaune, blanc et noir)
        xB = xA + cote
        yB = yA + cote
        can1.create_rectangle(xA, yA, xB, yB, fill=couleur)

    def croix(xC, yC):
# Cette fonction a pour rôle de dessiner une croix dans le canvas comportant la
grille du jeu
        xA = xC + 20
        yA = yC + 20
        xB = xA + 60
        yB = yA + 60
        print("xA,yA,xB,yB : ",xA, yA, xB, yB)
        can1.create_line(xA, yA, xB, yB, width = 15, fill = 'red')
        can1.create_line(xB, yA, xA, yB, width = 15, fill = 'red')

    def matrice(x,y):
# Cette fonction a pour rôle de changer les coordonnées d'une case en place dans
la liste qui nous permet d'interpréter la grille

        global place_liste
# On divise les deux coordonnées par 100, pour avoir des chiffres entre 0 et 7. A
chaque ligne, on augmente de 7, c'est pour cela qu'on multiplie y
        place_liste = (y//100)*7+(x//100)
# par 7 et on additionne le résultat avec le nombre sur x.


    def grille(h):
# Cette fonction a le rôle inverse de la fonction matrice(), elle permet de
changer une place dans la liste par des coordonnées dans le canevas
        global xh, yh
        yh = (h // 7)*100
        xh = (h % 7)*100

    def E(En, lettre, a):
# Cette fonction sert à indenter un nombre dans un string. Elle est utilisée
notamment pour les cases 'Ex' et 'Lx'
        global Ex
        if len(En) == 2:
            x = En[1]
            x = str( int(En[1]) + a)
# La case étant déjà éclairée, on va la mettre à jour pour lui signaler qu'elle
```

```
sera éclairée une fois de plus
        elif len(En) == 3:
            x = En[1] + En[2]
            x = str( int(En[1] + En[2]) + a)
        Ex = lettre + x
# On ajoute +1 à l'éclairage (cela sert par après pour enlever des lumières)

    def damier():
# Cette fonction sert à générer la grille ou remettre à zéro la grille
        global x0,y0,x1,y1, position,position_croix, m, victoire_echec

        x0,y0 = 0,0
        x1,y1 = 100,100

        if victoire_echec == 0:
# On vérifie avec ce test si l'écran de victoire ou défaite est actif. On empêche
la réinitialisation si jamais c'est le cas
            while y1 <= 700:
# Cette boucle While permet de traiter chaque ligne l'une après l'autre
                #Première ligne
                while x1 <=700:
#Cette boucle While permet de traiter chaque case d'une ligne l'une après l'autre
                    carre(x0,y0,100,'white')
# On place des carrés blancs pour créer les cases
                    x0 += 100
                    x1 += 100
                x0=0
                y0 += 100
                x1= 100
                y1 += 100
                #print (x0,y0,x1,y1)

            print(position)

            for i in range (0,49):
# Cette boucle for a pour objectif de réinitialiser la grille en plaçant, dans la
liste qui représente la grille, des 'O' pour signifier que cette
                Ex = position[i]
# case est blanche. On traite pour cela chaque case ( de 0 à 48)
                position_croix[i] = 'O'
                if Ex[0] != 'N':
                    position[i] = 'O'

            print("réinitialisation: ", position)
            gen_grille(position)
# On génère ensuite la grille à l'aide la liste remise à jour

    def gen_grille(matrice_grille):
# Cette fonction a pour objectif de placer les cases noires avec et sans chiffres
sur la grille
        global xh, yh
        for i in range (0,49):
# On traite également chaque case une après l'autre comme pour la fonction
damier()
            case_traitee = matrice_grille[i]
            if case_traitee[0] != 'N':
                cent = 100
```

```
            elif case_traitee[0] == 'N':
                grille(i)
                carre(xh,yh,100,'black')
                long = len(case_traitee)
                if long == 2:
                    xh += 50
                    yh += 50
                    num_case = case_traitee[1]
                    can1.create_text(xh, yh, text = num_case,font=('Arial', 60,
'bold'), fill ='white')

    def verification():
# Cette fonction permet de vérifier si la réponse donnée par l'utilisateur est
correcte
        global nbre_chance, compteur, fen1, victoire_echec

        if victoire_echec == 0:
# On vérifie avec ce test si l'écran de victoire ou défaite est actif. On empêche
la réinitialisation si jamais c'est le cas
            erreurs = 0

            for i in range (0,49):
# On vérifie chaque case à l'aide d'une boucle for
                Ex = position[i]

                if Ex[0] == 'N':
# On regarde si la case est une case noire
                    l = len(Ex)

                    if l == 2:
# Il y a des cases noires 'N' et des cases 'Nx', x étant un nombre entre 1 et 4.
si le nom de la case fait moins de deux caractères alors on peut mettre ce que
l'on désire
                        nbre_noir = int(Ex[1])
# On prend le nombre de la case noire qui va nous indiquer combien il faut de
lampes autour
                        nbre_lampe = 0
# On réinitialise le nombre de lampe à chaque itération de la boucle for
                        verif_h, verif_b, verif_d, verif_g = i - 7, i + 7, i + 1,
i - 1

                        if verif_h < 0:
#Cette opération est ici pour éviter qu'on aille "en dehors" de la grille, on
envoie donc 'R' pour indiquer qu'il n'y a rien au-dessus
                            Ex_h = 'R'
                        else:
                            Ex_h = position[verif_h]

                        if verif_b >= 49:
# Même opération mais pour le bas
                            Ex_b = 'R'
                        else:
                            print(i)
                            Ex_b = position[verif_b]

                        if verif_d > 49 or (verif_d // 7) != (i // 7):
# Cette opération permet de vérifier si la case considérée est bien sur la même
```

```
ligne
                            Ex_d = 'R'
                        else:
                            Ex_d = position[verif_d]

                        if verif_g < 0 or (verif_g // 7) != (i // 7):
# Même fonctionnement que pour la vérification à gauche
                            Ex_g = 'R'
                        else:
                            Ex_g = position[verif_g]

                        if Ex_h[0] == 'L':
# On vérifie si chaque case autour correspond à une lampe ou non et on incrémente
le nombre de lampes à chaque fois qu'on trouve une lampe
                            nbre_lampe += 1
                            print("en haut : ", nbre_lampe)

                        if Ex_b[0] == 'L':
                            nbre_lampe += 1
                            print("en bas : ", nbre_lampe)

                        if Ex_d[0] == 'L':
                            nbre_lampe += 1
                            print("à droite : ", nbre_lampe)

                        if Ex_g[0] == 'L':
                            nbre_lampe += 1
                            print("à gauche : ", nbre_lampe)

                        print("opération logique nbre lampe != nbre_noir:
",nbre_lampe != nbre_noir)

                        if nbre_lampe != nbre_noir:
# Une fois qu'on a le nombre de lampe, on compare le nombre sur la case noire et
le nombre de lampe : si le nombre est différent alors on compte une erreur
                            erreurs += 1
                            print("erreur cases noires :", erreurs, "numéro case :
", i)

                if Ex[0] == 'L' and Ex[1] != '0':
# On considère les cases lampe et si on voit que le nombre associé est différent
de '0' alors la lampe est éclairée par une autre. On compte donc une erreur
                    erreurs += 1
                    print("erreur lampe :", erreurs)

                if Ex == 'O':
# S'il y a une case blanche, c'est le même tarif : on compte une erreur.
                    erreurs += 1
                    print("erreur vide :", erreurs)

            print(erreurs)
# On affiche le nombre d'erreurs


            if erreurs != 0:
# Si on a fait une faute
                l_nbre_chance = len(nbre_chance)
```

```
# C'est le texte qui nous est affiché pour nous donner notre nombre de fautes. On
regarde la longueur totale
                essai_prec = int(nbre_chance[l_nbre_chance-1])
# On récupère le numéro à la fin du nombre d'essais et on le change en chiffre
                essai = essai_prec - 1
# On enlève un à ce nombre
                essai_str, essai_prec_str = str(essai), str(essai_prec)
# On remet le nombre et le nombre précédent en string
                nbre_chance = nbre_chance.replace(essai_prec_str, essai_str)
# On remplace le nombre précédent dans la phrase par le nouveau nombre
                print(nbre_chance)

                if essai_prec > 0:
# on vérifie si avant de cliquer le nombre était supérieur à 0
                    children = fen1.winfo_children()
# Si c'est le cas on le supprime et on le remplace par le nouveau nombre au niveau
de l'affichage
                    l_children = len(children)
                    children[l_children - 1].destroy()
                    compteur =  Label(fen1, text = nbre_chance, font=("Arial", 11,
'bold'))
                    compteur.grid(row = 0, column = 1,pady = 20, sticky = 'new')

                if essai == 0:
# s'il nous reste aucun essai, un message d'échec arrive et on revient au menu
                    victoire_echec = 1
                    can1.create_rectangle(50,200,650,400, fill = 'white', width =
5)
                    can1.create_text(360, 250, text="Vous avez PERDU !!!",
font=('Arial', 40, "bold"), fill = 'red')
                    can1.create_text(360, 350, text="Retentez votre chance ...",
font=('Arial', 30, "bold"), fill = 'red')
                    can1.after(3000, change_frame, 'retour_menu')


            if erreurs == 0:
# Si on a fait aucune faute alors un message de victoire arrive et on revient au
menu après
                victoire_echec = 1
                can1.create_rectangle(50,200,650,400, fill = 'white', width = 5)
                can1.create_text(360, 250, text="Vous avez GAGNE !!!",
font=('Arial', 40,"bold"), fill = 'green')
                can1.create_text(360, 350, text="Passez au niveau supérieur ...",
font=('Arial', 30, "bold"), fill = 'green')
                can1.after(3000, change_frame, 'retour_menu')



    def poser_croix(event):
# Cette fonction permet de gérer le positionnement d'une croix
        global place_liste, position_croix, victoire_echec
        if victoire_echec == 0:
# On vérifie avec ce test si l'écran de victoire ou défaite est actif. On empêche
la réinitialisation si jamais c'est le cas
            posCroixX = (event.x // 100)*100
            posCroixY = (event.y // 100)*100
```

```
            if posCroixX >= 700:
# On vérifie si le joueur a bien cliqué sur une case et on rectifie si ce n'est
pas le cas
                posCroixX = ((event.x-50) // 100)*100

            if posCroixY >= 700:
                posCroixY = ((event.y-50) // 100)*100

            matrice(posCroixX, posCroixY)
# On change l'endroit cliqué en position dans la liste avec la fonction matrice()
            E_croix = position_croix[place_liste]
            E_grille = position[place_liste]

            if E_grille[0] == 'N' or E_grille[0] == 'L':
# On empêche le joueur de placer une croix sur une case lampe ou une case noire
                sucre = 0
            elif E_croix == 'X':
# On regarde s'il y a déjà une croix sur la case cliquée. On va enlever cette
croix

                if E_grille[0] == 'E':
# On regarde d'abord si la croix est sur une case éclairée ou une case blanche.
                    carre(posCroixX,posCroixY,100,'LightGoldenrod1')
# Si la case est éclairée, on remplace la croix par un carré doré
                else :
                    carre(posCroixX,posCroixY,100,'white')
# Si la case est blanche, on remplace la croix par un carré blanc
                position_croix[place_liste] = 'O'
# On remplace ensuite dans la liste donnant la position des croix le 'X' par un
'O' pour dire qu'il n'y a plus de croix sur cette case
                print(position_croix)

            elif E_croix == 'O':
# On regarde enfin si la case cliquée ne possède pas de croix et n'est pas un case
noire ou une case jaune.
                croix(posCroixX, posCroixY)
# Si c'est le cas, on pose une croix à l'aide de la fonction croix()
                position_croix[place_liste] = 'X'
# On actualise la liste donnant la position des croix pour dire qu'il y a
maintenant une croix dans la case cliquée
                print(position_croix)



    def poser_lampe(event):
# Cette focntion sert à positionner des lampes
        global posLampeX, posLampeY, positionX, positionY, X, Y, place_liste,
position_croix, victoire_echec

        if victoire_echec == 0:
            posLampeX = (event.x // 100)*100
# On fait une division entière puis on multiplie par 100 pour se positionner dans
le coin supérieur gauche de la case
            posLampeY = (event.y // 100)*100

            if posLampeX >= 700:
# On vérifie si le joueur a bien cliqué sur une case et on rectifie si ce n'est
```

```
pas le cas
                posLampeX = ((event.x-50) // 100)*100

            if posLampeY >= 700:
                posLampeY = ((event.y-50) // 100)*100

            matrice(posLampeX,posLampeY)
# On regarde ce qu'il y a dans la case
            int(place_liste)
# C'est le résultat de ce l'opération effectué avec matrice()
            Ex = position[place_liste]
            longueur = len(Ex)

            if Ex[0] == 'N' or position_croix[place_liste] == 'X':
# on n'effectue aucune opération si l'utilisateur appuie sur une case noire ou sur
une croix
                rien = 0
            elif Ex[0] == 'L' :
# Si la case cliquée possède une lampe, on enlève cette lampe

                if Ex[1] == '0':
                    carre(posLampeX,posLampeY,100,'white')
                    position[place_liste] = 'O'
                else:
                    if longueur == 3:
                        Z = Ex[1]+ Ex[2]
# On va garder le même niveau de luminosité à la case par cette opération
                        position[place_liste] = 'E' + Z
                    elif longueur == 2:
                        Z = Ex[1]
                        position[place_liste] = 'E' + Z
                    carre(posLampeX,posLampeY,100,'LightGoldenrod1')
                obscurite()

            elif position[place_liste] == 'O' :
# Si la case cliquée ne possède rien alors on pose une lampe
                carre(posLampeX,posLampeY,100,'yellow')
                position[place_liste] = 'L0'
                eclairage()

            elif Ex[0] == 'E':
# Si la case cliquée est éclairée alors on va gérer faire en sorte que la lampe
posée ait un niveau de luminosité identique à la case éclairée
                carre(posLampeX,posLampeY,100,'yellow')
                if longueur == 2:
                    position[place_liste] = 'L' + Ex[1]
                elif longueur == 3:
                    position[place_liste] = 'L' + Ex[1] + Ex[2]
                eclairage()

            print(position)



    def eclairage():
# Cette fonction va permettre de gérer l'éclairage d'une lampe lorsqu'on en pose
une
```

```
        global place_liste, lumiere, lumiere_d, lumiere_g, posLampeX, posLampeY,
Ex, position_croix

        lumiere = place_liste
# On va d'abord indiquer au programme où se situe notre lampe
        """ On va ensuite gérer l'éclairage dans cahque point cardinaux, en
indiquant la première case qui doit être éclairée en bas, à droite, à gauche et en
haut """
        lumiere_d = lumiere + 1
        lumiere_g = lumiere -1
        lumiere_h = lumiere - 7
        lumiere_b = lumiere + 7

        debug_lumiere_b = 0
# Cette variable est là pour pallier un bug qui apparaît uniquement dans la boucle
d'éclairage vers le bas

        while (lumiere_d) % 7 != 0:
# On commence par éclairer tout ce qui est à droite
            print('lumiere_d')
            Ex = position[lumiere_d]
# cette variable va stocker le nombre d'éclairage sur une même case
            if Ex[0] != 'N':
# On ne traite que les cases qui ne sont pas noires

                if Ex[0] == 'L':
# Si on a une case lampe alors dans ce cas on passe à la case suivante
                    E(Ex,'L', 1)
                    position[lumiere_d] = Ex
                    lumiere_d += 1
                elif Ex[0] == 'E':
# Grâce à cette variable, on va vérifier si le premier terme dans la case est un
E, ce qui signifierait que la case est éclairée
                    E(Ex,'E', 1)
                    position[lumiere_d] = Ex
                    lumiere_d += 1
                else:
                    position[lumiere_d] = 'E1'
# On actualise la liste qui stocke toutes les informations
                    grille(lumiere_d)
# On transforme la place dans notre liste en des coordonnées dans la grille à
l'aide de la fonction grille()
                    carre(xh,yh,100,'LightGoldenrod1')
# On crée un carré pour chaque case éclairée
                    if position_croix[lumiere_d] == 'X':
# On regarde s'il y a une croix et on la replace dans ce cas. On fait la même
opération pour chaque case éclairée
                        croix(xh, yh)
                    lumiere_d += 1
# On incrémente pour passer à la case suivante

                print(position)

            else:
# Mais on ne remplace pas les cases noires, ni les cases d'après
                break
# le break permet de couper la boucle while pour qu'aucune autre case ne soit
```

```
éclairée par après
            print(position)


        while (lumiere_g + 1) % 7 != 0:
# On éclaire ensuite tout ce qui est à gauche
            print('lumiere_g')
            Ex = position[lumiere_g]
            if Ex[0] != 'N':
# On ne traite que les cases qui ne sont pas noires
                if Ex[0] == 'L':
# Si on a une case lampe alors dans ce cas on passe à la case précédente
                    E(Ex, 'L', 1)
                    position[lumiere_g] = Ex
                    lumiere_g -= 1

                elif Ex[0] == 'E':
                    E(Ex,'E', 1)
                    position[lumiere_g] = Ex
                    lumiere_g -= 1

                else:
                    position[lumiere_g] = 'E1'
# On actualise la liste qui stocke toutes les informations
                    grille(lumiere_g)
# On transforme la place dans notre liste en des coordonnées dans la grille à
l'aide de la fonction grille()
                    carre(xh,yh,100,'LightGoldenrod1')
# On crée un carré pour chaque case éclairée
                    if position_croix[lumiere_g] == 'X':
                        croix(xh, yh)
                    lumiere_g -= 1
# On incrémente pour passer à la case précédente


                print(position)

            else:
# Mais on ne remplace pas les cases noires, ni les cases d'après
                break
# le break permet de couper la boucle while pour qu'aucune autre case ne soit
éclairée par après
            print(position)


        while lumiere_h >= 0:
# On éclaire ensuite la colonne en commençant par le haut
            print('lumiere_h')
            Ex = position[lumiere_h]
            if Ex[0] != 'N':
# On ne traite que les cases qui ne sont pas noires
                if Ex[0] == 'L':
                    E(Ex, 'L', 1)
                    position[lumiere_h] = Ex
                    lumiere_h -= 7
# Dans ce cas, on passe simplement à la case suivante
                elif Ex[0] == 'E':
```

```
                    E(Ex,'E', 1)
                    position[lumiere_h] = Ex
                    lumiere_h -= 7
                else:
                    position[lumiere_h] = 'E1'
# On actualise la liste qui stocke toutes les informations
                    grille(lumiere_h)
# On transforme la place dans notre liste en des coordonnées dans la grille à
l'aide de la fonction grille()
                    carre(xh,yh,100,'LightGoldenrod1')
# On crée un carré pour chaque case éclairée
                    if position_croix[lumiere_h] == 'X':
                        croix(xh, yh)
                    lumiere_h -= 7
# On incrémente de -7 pour passer à la case suivante

                print(position)

            else:
# Mais on ne remplace pas les cases noires, ni les cases d'après
                break
# le break permet de couper la boucle while pour qu'aucune autre case ne soit
éclairée par après


        while lumiere_b <= 48:
# On éclaire enfin le bas
            print('lumiere_b')
            Ex = position[lumiere_b]
            if Ex[0] != 'N':
# On ne traite que les cases qui ne sont pas noires
                if Ex[0] == 'L':
                    E(Ex,'L', 1)
                    position[lumiere_b] = Ex
                    lumiere_b += 7
# Dans ce cas, on passe simplement à la case suivante
                elif Ex[0] == 'E':
                    E(Ex,'E', 1)
                    position[lumiere_b] = Ex
                    lumiere_b += 7
                else:
                    position[lumiere_b] = 'E1'
# On actualise la liste qui stocke toutes les informations
                    grille(lumiere_b)
# On transforme la place dasn notre liste en des coordonnées dans la grille à
l'aide de la fonction grille()
                    carre(xh,yh,100,'LightGoldenrod1')
# On crée un carré pour chaque case éclairée
                    if position_croix[lumiere_b] == 'X':
                        croix(xh, yh)
                    lumiere_b += 7
# On incrémente de 7 pour passer à la case suivante

                    print(position)

            else:
# Mais on ne remplace pas les cases noires, ni les cases d'après
```

```
                break
# le break permet de couper la boucle while pour qu'aucune autre case ne soit
éclairée par après

            debug_lumiere_b += 1
            if debug_lumiere_b == 10:
# La boucle se répète 6 fois au maximum en fonctionnement normal donc si le nombre
d'itérations est supérieur à 6 ( ici 10 pour avoir une marge de sécurité) alors
                break
# le programme s'arrête et on peut retenter de cliquer


    def obscurite():
# Cette fonction effectue le résultat inverse de la fonction eclairage(), il
permet en outre d'enlever tout l'éclairage quand on retire une lampe

        global place_liste, blanc, blanc_d, blanc_g, blanc_h, blanc_b, posLampeX,
posLampeY, Ex, position_croix

        blanc = place_liste
# Le fonctionnement est quasiment le même que pour la fonction eclairage.
        blanc_d = blanc + 1
        blanc_g = blanc -1
        blanc_h = blanc - 7
        blanc_b = blanc + 7
        debug_lumiere_b = 0

        while (blanc_d) % 7 != 0:
# On enlève d'abord l'éclairage d'abord à droite
            print('blanc_d')
            Ex = position[blanc_d]
# cette variable va stocker le nombre d'éclairage sur une même case

            if Ex[0] != 'N' :
# On ne traite que les cases qui ne sont pas noires

                if Ex[0] == 'L':
                    E(Ex,'L',(-1))
                    position[blanc_d] = Ex
# Ex est le résultat donné par la fonction E(), dans ce cas, il enlève 1 au string
'Lx'
                    blanc_d += 1
                elif Ex[0] == 'E':
# Grâce à cette variable, on va vérifier si le premier terme dans la case est un
E, ce qui signifierait que la case est éclairée
                    E(Ex,'E',(-1))
                    if Ex == 'E0':
# Une case non éclairée est une case 'O', on va donc placer une case non éclairée
à la place de la case éclairée
                        position[blanc_d] = 'O'
                        grille(blanc_d)
                        carre(xh,yh,100,'white')
                        if position_croix[blanc_d] == 'X':
# On regarde s'il y a une croix et on la replace dans ce cas. On fait la même
opération pour chaque case à laquelle on retire un éclairage
                            croix(xh, yh)
                        blanc_d += 1
```

```
                    else:
                        position[blanc_d] = Ex
                        blanc_d += 1

                print(position)

            else:
                break

            print(position)

        while (blanc_g + 1) % 7 != 0:
# On enlève l'éclairage à gauche ensuite
            print('blanc_g')
            Ex = position[blanc_g]

            if Ex[0] != 'N' :
# On ne traite que les cases qui ne sont pas noires

                if Ex[0] == 'L':
                    E(Ex, 'L',(-1))
                    position[blanc_g] = Ex
                    blanc_g -= 1

                elif Ex[0] == 'E':
                    E(Ex,'E',(-1))
                    if Ex == 'E0':
                        position[blanc_g] = 'O'
                        grille(blanc_g)
                        carre(xh,yh,100,'white')
                        if position_croix[blanc_g] == 'X':
                            croix(xh, yh)
                        blanc_g -= 1
                    else:
                        position[blanc_g] = Ex
                        blanc_g -= 1


                print(position)

            else:
                    break

            print(position)


        while blanc_h >= 0:
# On enlève l'éclairage ensuite dans la colonne en commençant par le haut
            print('blanc_h')
            Ex = position[blanc_h]

            if Ex[0] != 'N' :
# On ne traite que les cases qui ne sont pas noires

                if Ex[0] == 'L':
                    E(Ex, 'L',(-1))
                    position[blanc_h] = Ex
```

```
                    blanc_h -= 7
                elif Ex[0] == 'E':
                    E(Ex,'E',(-1))
                    if Ex == 'E0':
                        position[blanc_h] = 'O'
                        grille(blanc_h)
                        carre(xh,yh,100,'white')
                        if position_croix[blanc_h] == 'X':
                            croix(xh, yh)
                        blanc_h -= 7
                    else:
                        position[blanc_h] = Ex
                        blanc_h -= 7
# On incrémente de -7 pour passer à la case suivante

            else:
                break


            print(position)

        while blanc_b <= 48:
# On enleve l'eclairage en bas enfin par le bas
            print('blanc_b')
            Ex = position[blanc_b]

            if Ex[0] != 'N' :
# On ne traite que les cases qui ne sont pas noires

                if Ex[0] == 'L':
                    E(Ex,'L',(-1))
                    position[blanc_b] = Ex
                    blanc_b += 7
                elif Ex[0] == 'E':
                    E(Ex,'E',(-1))
                    if Ex == 'E0':
                        position[blanc_b] = 'O'
                        grille(blanc_b)
                        carre(xh,yh,100,'white')
                        if position_croix[blanc_b] == 'X':
                            croix(xh, yh)
                        blanc_b += 7
                    else:
                        position[blanc_b] = Ex
                        blanc_b += 7

                print(position)

            else:
                break

            debug_lumiere_b += 1
            if debug_lumiere_b == 10:
# La boucle se répète 6 fois au maximum en fonctionnement normal donc si le nombre
d'itérations est supérieur à 6 ( ici 10 pour avoir une marge de sécurité) alors
                break
# le programme s'arrête et on peut retenter de cliquer
```

```
# --------- INTERACTIONS AU NIVEAU GAMEPLAY : ----------- #


    # bouton de réinitialsiation:
    bouton_reinit = Button(fen1, text='Réinitialisation', font = ("Arial", 11),
command=damier, activebackground= 'LightGoldenrod1',  background='silver')

    # bouton de vérification :
    bouton_verif = Button(fen1, text='Vérification', font = ("Arial", 11),
command=verification, activebackground= 'LightGoldenrod1', background='silver')

    Label(text = '',font = 200).grid()

    # création des widgets "esclaves/enfants" :
    can1 = Canvas(fen1,bg='white',height=700,width=700)

    # génération du damier à l'ouverture de la fenêtre tk
    damier()

    #compteur d'erreurs :
    compteur =  Label(fen1, text = nbre_chance, font=("Arial", 11, 'bold'))


    """On donne la position/caractéristiques de chaque widget sur la fen1 à cet
endroit :"""

    can1.grid(row = 0, rowspan = 21 )
    bouton_reinit.grid(row = 1, column = 1,pady = 10,padx = 10,ipadx = 40, sticky
= 'snew')
    bouton_verif.grid(row = 2, column = 1,padx = 10, ipadx = 40, sticky = 'snew' )
    compteur.grid(row = 0, column = 1,pady = 20, sticky = 'new')

    """ Ensuite on crée des interactions avec les boutons de la souris : clic
gauche et clic droit :"""

    # on génère une fonction qui permet de détecter le clic droit de la souris sur
la fenêtre et de lancer la fonction "poser_croix"
    can1.bind("<Button-3>", poser_croix)

    # on génère une fonction qui permet de détecter le clic gauche de la souris
sur la fenêtre et de lancer la fonction "poser_lampe"
    can1.bind("<Button-1>", poser_lampe)
    chaine = Label(can1)



# --------- PARTIE MENU : ----------- #
```

```
def fenetre_jeu(tuto):

    global bouton_quitter, bouton_menu, frame_option, bouton_suivant

    frame_option = Frame(fen1)

    bouton_quitter = Button(frame_option,text='Quitter', command=fen1.destroy,
background = 'lightgrey')        # on définit les caractéristiques du bouton
QUITTER, permettant de quitter le jeu
    bouton_menu = Button(frame_option, text='Menu', command=lambda:
change_frame('retour_menu'), background = 'lightgrey')      # on définit les
caractéristiques du bouton MENU, permettant de retourner au menu du jeu

    if tuto == 1:

        bouton_quitter.grid(row = 0, column = 0, ipadx = 23, pady = 5, sticky =
'e')        # on définit les positions du bouton QUITTER
        bouton_menu.grid(row = 0, column = 1, ipadx = 26, padx = 10, pady = 5,
sticky = 'e')        # on définit les positions du bouton MENU


    else:

        frame_option.grid(row = 3,rowspan = 18,column = 1, sticky = 'se')

        bouton_quitter.grid(row = 1, column = 0, ipadx = 23, pady = 5, sticky =
'e')        # on définit les positions du bouton QUITTER
        bouton_menu.grid(row = 0, column = 0, ipadx = 26, pady = 5, sticky = 'e')
# on définit les positions du bouton MENU




def tutoriel(master):

    global bouton_quitter, bouton_menu, bouton_suivant

    """ Crée un tutoriel pour le jeu """
    fenetre_jeu(1)

    # définition des boutons

    bouton_suivant = Button(frame_option, text='Suivant', command=lambda:
change_frame('tuto1'), background = 'salmon')



    # création des images
    can = Canvas(fen1, width = 700, height = 350)

    can.create_image(157, 157, image = image_vide)
    texte1 = can.create_text(157,340, text = 'Grille initiale', font=("Arial", 20,
'bold'))

    can.create_image(543, 157, image = image_rempli)
```

```
    texte2 = can.create_text(543,340, text = 'Grille finale', font=("Arial", 20,
'bold'))

    explication1 = Label(fen1, text = "Le but du jeu est d'eclairer toutes les
cases blanche de la grille.", font=("Arial", 16))


    can.grid(row = 1, column = 1,columnspan = 1, padx = 50, sticky = 'nw')
    frame_option.grid(row = 2, column = 1, columnspan = 2, padx = 250,pady = 30,
sticky = 'snew')
    explication1.grid(row = 0, column = 0, columnspan = 2, padx = 150,pady = 30,
sticky ='new')
    bouton_suivant.grid(row = 0, column = 2, ipadx = 26, pady = 5, sticky = 'e')

def tuto1(master):

    global bouton_quitter, bouton_menu, bouton_suivant

    """ Crée un tutoriel pour le jeu """
    fenetre_jeu(1)

    # définition des boutons

    bouton_suivant = Button(frame_option, text='Suivant', command=lambda:
change_frame('tuto2'), background = 'salmon')


    # création des images
    can = Canvas(fen1,  width = 900, height = 350)

    can.create_image(150, 150, image = image_eclairage)
    can.create_text(150, 260, text = 'Une lampe éclaire horizontalement',
font=("Arial", 12, 'italic'))
    can.create_text(150, 280, text = 'et verticalement uniquement', font=("Arial",
12, 'italic'))

    can.create_image(450, 150, image = image_eclairage_noir)
    can.create_text(450, 300, text = "L'éclairage de la lampe", font=("Arial", 12,
'italic'))
    can.create_text(450, 320, text = "est bloqué lorsqu'il rencontre",
font=("Arial", 12, 'italic'))
    can.create_text(450, 340, text = "une case noire", font=("Arial", 12,
'italic'))

    can.create_image(750, 150, image = image_2eclairage)
    can.create_text(750, 265, text = 'Mais une lampe ne peut', font=("Arial", 12,
'italic'))
    can.create_text(750, 285, text = 'être éclairée par une autre', font=("Arial",
12, 'italic'))

    explication2 = Frame (fen1)
    explication2_1 = Label(explication2,text = 'Eclairage:', font =("Arial", 16,
'underline') )
    explication2_2 = Label(explication2, text = 'Pour poser/enlever une lampe,
appuyer sur le clic gauche de la souris', font =("Arial", 14, "italic"))
```

```
    can.grid(row = 1, column = 0,columnspan = 2, sticky = 'nw')
    frame_option.grid(row = 2, column = 0, columnspan = 2, padx = 300, pady = 40,
sticky = 'snew')
    bouton_suivant.grid(row = 0, column = 2, ipadx = 26, pady = 5, sticky = 'e')

    explication2.grid(row = 0, column = 0, columnspan = 2, padx = 150,pady = 10,
sticky ='new')
    explication2_1.grid(row = 0, column = 0, sticky = 'nsew' )
    explication2_2.grid(row = 1, column = 0, sticky = 'nsew')

def tuto2(master):

    global bouton_quitter, bouton_menu, bouton_suivant

    """ Crée un tutoriel pour le jeu """
    fenetre_jeu(1)


    # création des images
    can = Canvas(fen1,  width = 900, height = 300)

    can.create_image(150, 150, image = image_croix)
    can.create_text(150, 240, text = 'On ne peut poser aucune lampe',
font=("Arial", 12, 'italic'))
    can.create_text(150, 260, text = "autour d'une case 0,", font=("Arial", 12,
'italic'))
    can.create_text(150, 280, text = "on peut donc mettre 4 croix", font=("Arial",
12, 'italic'))

    can.create_image(450, 150, image = image_1)
    can.create_text(450, 240, text = "On ne peut mettre qu'une lampe",
font=("Arial", 12, 'italic'))
    can.create_text(450, 260, text = "autour d'une case 1,", font=("Arial", 12,
'italic'))
    can.create_text(450, 280, text = "on peut donc ensuite mettre 3 croix",
font=("Arial", 12, 'italic'))

    can.create_image(750, 150, image = image_4)
    can.create_text(750, 240, text = "On peut mettre 4 lampe,", font=("Arial", 12,
'italic'))
    can.create_text(750, 260, text = "soit le maximum, autour d'une",
font=("Arial", 12, 'italic'))
    can.create_text(750, 280, text = "case 4, donc aucune croix, etc.",
font=("Arial", 12, 'italic'))

    explication3 = Frame (fen1)
    explication3_1 = Label(explication3,text = 'Fonctionnalités des cases noires
:', font =("Arial", 16, 'underline') )
    explication3_2 = Label(explication3, text = 'Pour poser/enlever une croix,
appuyer sur le clic droit de la souris', font =("Arial", 14, "italic"))
    explication3_3 = Label(explication3, text = "Il faut poser autant de lampes
autour d'une case noire que le numéro indiqué par ladite case", font =("Arial",
14, "italic"))


    can.grid(row = 1, column = 0,columnspan = 2, sticky = 'nw')
    frame_option.grid(row = 2, column = 0, columnspan = 2, padx = 350, pady = 40,
```

```
sticky = 'snew')

    explication3.grid(row = 0, column = 0, columnspan = 2, padx = 50, pady = 10,
sticky ='new')
    explication3_1.grid(row = 0, column = 0,pady = 10, sticky = 'nsew' )
    explication3_2.grid(row = 1, column = 0, sticky = 'nsew')
    explication3_3.grid(row = 2, column = 0, sticky = 'snew')




def lvl1(master):

    """ Crée une frame pour le niveau Facile """
    fenetre_jeu(0)

    x = randrange(1,6)
# on génère un nombre aléatoire pour choisir une des 5 grilles pour le niveau 1
    niveau1 = {1:
["O","O","O","O","N0","O","O","O","O","O","N3","O","O","O","O","O","N","O","O","O","O","O","N0
","O","O","O","O","O","N","O","O","O","O","O","N1","O","O","O","O","O","N2","O","O","O","O","O","N
2","O","O","O","O","O","N0","O","O","O","O"],
    2:
["O","N","O","O","N3","O","O","O","N","O","O","O","N","N","N2","O","O","O","O","O","O"
,"O","O","O","O","O","N","O","O","O","O","O","O","O","O","O","O","O","N1","N2","N1","O","O","O","O
","N1","O","O","O","N","O","O","O","N","O"],
    3:
["O","O","O","O","O","N1","O","O","O","O","O","O","O","O","O","O","O","N3","O","O","O","N1","O","O
","O","O","O","O","N2","O","O","N","O","O","O","O","O","O","N3","O","O","O","N2","O","O","O","O","O","
O","O","O","O","O","O","N1","O","O","O","O"],
    4:
["O","O","O","O","N2","O","N1","O","O","N","O","O","O","O","O","O","O","O","O","O","O","O","O","O"
,"O","O","N","O","O","O","O","O","O","N","O","O","O","O","O","O","O","O","O","O","O","O","O","O","O","
O","O","N2","O","O","N3","O","O","N","O","O","O"],
    5:
["O","O","O","O","N0","O","O","O","O","O","O","O","N1","N","O","O","O","O","O","O","O","O","O","N1
","O","O","N","N0","O","O","N3","O","O","N","N0","O","O","N0","O","O","O","O","O","O","O","O","O","O","N",
"N1","O","O","O","O","O","O","N0","O","O","O","O"]}
    lvl1 = niveau1[x]

    akari_gameplay(lvl1)



def lvl2(master):
    """ Crée une frame pour le niveau Normal """
    fenetre_jeu(0)

    x = randrange(1,6)
# on génère un nombre aléatoire pour choisir une des 5 grilles pour le niveau 2
    niveau2 = {1:
["O","O","O","O","N0","O","O","O","O","O","N0","O","O","O","O","N3","O","O","O","O","O","O","O","O
","O","O","N1","O","O","O","O","O","O","O","N","O","O","O","O","O","O","O","O","O","O","N3","O","O","O
","O","N","O","O","O","O","O","N1","O","O","O"],
    2:
["O","O","N3","O","O","N1","O","O","O","O","O","O","N","O","O","O","O","N","O","O","O","O","O","O"
```

```
,"N","O","N1","O","N1","O","N","O","N","O","O","O","O","O","N","O","O","O","N1","O
","O","O","O","O","N1","O","N2","O","O"],
    3:
["O","O","O","O","N0","O","O","O","N2","N","O","O","N","O","N","O","O","O","O","N0
","O","O","O","O","O","O","O","O","O","O","N1","O","O","O","O","N","O","N2","O","O","N
1","N0","O","O","O","N2","O","O","O","O"],
    4:
["O","N1","O","O","O","N","O","N2","O","O","O","O","O","N","O","O","N","O","N","O"
,"O","O","O","O","O","O","O","O","O","O","O","O","N","O","N2","O","O","N2","O","O","O","O"
,"O","N","O","N2","O","O","O","N1","O"],
    5:
["O","N","O","N0","O","O","O","O","O","N0","O","O","O","O","N0","N","O","O","O","O","O","O
","O","N0","O","O","O","N","O","O","N","O","O","O","O","O","O","O","O","N1","N1","O","O","
O","N0","O","O","O","O","O","N1","O","N","O"]}
    lvl2 = niveau2[x]

    akari_gameplay(lvl2)



def lvl3(master):
    """ Crée une frame pour le niveau Difficile """
    fenetre_jeu(0)

    x = randrange(1,6)
# on génère un nombre aléatoire pour choisir une des 5 grilles pour le niveau 3
    niveau3 = {1:
["O","O","O","N","O","O","O","O","O","O","N3","O","O","O","O","O","O","O","O","O","O","N0"
,"O","N2","O","O","N1","O","O","N1","O","N1","O","O","O","O","O","O","O","O","O","O","O","
N","O","O","O","O","O","N1","O","O","O"],
    2:
["O","N0","O","N","O","O","O","O","O","O","O","N0","O","O","N2","O","O","O","O","O","O","O
","O","N","N1","O","O","O","N1","N0","O","O","O","O","O","O","O","O","N2","O","O","N",
"O","O","O","O","O","O","O","N","O","N1","O"],
    3:
["O","O","O","O","N","O","O","O","N4","O","O","O","N1","O","N","O","N","O","N1","O
","O","O","O","O","O","O","O","O","O","O","O","O","N1","O","N","O","N2","O","N1","O","O","
O","N2","O","O","O","N","O","O","O","O"],
    4:
["N1","O","O","O","O","O","O","N","O","O","N1","O","O","O","O","O","O","O","N0","O","N0","
N1","O","O","O","O","O","O","O","O","O","O","O","N","N0","O","N1","O","O","O","O","O","O","O",
"N1","O","O","N1","O","O","O","O","O","O","N"],
    5:
["O","O","N1","O","O","N1","O","N2","O","O","O","O","O","O","O","O","O","N","O","N2","
O","N","O","O","O","O","O","O","O","O","N","O","N0","O","N","O","O","O","O","O","O","O","O
","O","N2","O","N2","O","O","N","O","O"]}
    lvl3 = niveau3[x]


    akari_gameplay(lvl3)



def menu(master):
# C'est la fonction qui permet de créer notre menu principal, tout va graviter
autour de lui
```

```
    """ On va maintenant créer une case pour chaque objet affiché :"""

    instructions = Frame(fen1)
    instructions1 = Label(instructions, text='Menu :', font=("Arial",
16,'underline'))
    instructions2 = Label(instructions, text='Choisir votre niveau de jeu',
font=("Arial", 16))

    bouton_facile = Button(fen1, text='Facile', font =("Arial", 14,'bold'),
command=lambda: change_frame('niveau1'), activebackground= 'LightGoldenrod1',
background='grey', foreground = 'white', borderwidth = '4')
    bouton_moyen = Button(fen1, text='Normal', font =("Arial", 14,'bold'),
command=lambda: change_frame('niveau2'), activebackground= 'LightGoldenrod1',
background='grey', foreground = 'white', borderwidth = '4')
    bouton_difficile = Button(fen1, text='Difficile',font =("Arial", 14,'bold'),
command=lambda: change_frame('niveau3'), activebackground= 'LightGoldenrod1',
background='grey', foreground = 'white', borderwidth = '4')

    # création de l'image du menu
    can = Canvas(fen1,  width = 100, height = 128)
    can.create_image(191.5, 64, image = image_menu)

    bouton_tuto = Button(fen1, text = '?', font =("Arial", 14,'bold'),
command=lambda: change_frame('tuto'), background = 'lightgrey')

    bouton_quitter = Button(fen1, text='Quitter', command=fen1.destroy, background
= 'lightgrey')


    # emplacement des boutons
    instructions.grid(row = 0, column = 0,padx = 80, columnspan = 2, sticky =
'nsew')
    instructions1.grid(row= 0, column = 0)
    instructions2.grid(row= 0, column = 1)


    bouton_facile.grid(row = 2, column =0, columnspan = 2, pady = 20, padx = 30,
ipadx = 170, sticky = 'nsew')
    bouton_moyen.grid(row = 3,column = 0, columnspan = 2, pady = 20, padx = 30,
ipadx = 170, sticky = 'nsew')
    bouton_difficile.grid(row = 4, column = 0, columnspan = 2, pady = 20, padx =
30, ipadx = 170, sticky = 'sew')

    can.grid(row = 5, column = 0, columnspan = 2, padx = 50, sticky = 'nsew')

    bouton_tuto.grid(row = 6, column = 1, pady = 30, sticky = 'ns')

    bouton_quitter.grid(row = 6, column = 0, pady = 30, sticky = 'ns')

    return fen1


def change_frame(frame_name):
# Cette fonction permet d'échanger de fenêtre.
    global B
    children = fen1.winfo_children()
```

```
# Cette méthode permet de récupérer tous les boutons, désignés comme widget
enfants, sous la forme d'une liste
    print(children)
    niveau = boite_a_fenetre[frame_name]
# Ici niveau va corresponde à une fonction définie par le dico {boite_a_fenetre}
    nbre_total = len(children)
    nbre_bouton = 0

    while nbre_bouton + 1 <= nbre_total:
# On supprime les widgets tour à tour
        frame_info = children[nbre_bouton]
        frame_info.destroy()
        nbre_bouton += 1

    if frame_name == 'retour_menu':
        fen1.geometry('500x500')
        fen1.minsize(500, 500)
        new_frame = niveau(fen1)
        new_frame.grid()

    elif frame_name == 'tuto':
        fen1.geometry('910x550')
        new_frame =niveau(fen1)

    elif frame_name == 'tuto1':
        fen1.geometry('910x550')
        new_frame =niveau(fen1)

    elif frame_name == 'tuto2':
        fen1.geometry('910x550')
        new_frame =niveau(fen1)


    else:
        fen1.geometry('910x700')
        fen1.minsize(910, 700)
        new_frame = niveau(fen1)
# On crée une nouvelle fenêtre dans laquelle on va mettre notre niveau


# image menu
image_menu = PhotoImage(file = 'lampe.gif')

# images tuto
image_croix = PhotoImage(file = 'Croix.gif')
image_eclairage = PhotoImage(file = 'Eclairage.gif')
image_2eclairage = PhotoImage(file = 'Double_eclairage.gif')
image_eclairage_noir = PhotoImage(file = 'Eclairage_noir.gif')
image_1 = PhotoImage(file = '1.gif')
image_4 = PhotoImage(file = '4.gif')
image_vide = PhotoImage(file = 'vide.gif')
image_rempli = PhotoImage(file = 'rempli.gif')

frame_info = menu(fen1)


frame_info.grid()
```

```
# la méthode grid() permet de mieux agencer tout ce que nous avons sur notre menu.


boite_a_fenetre = {'niveau1': lvl1, 'niveau2': lvl2, 'niveau3': lvl3, 'tuto':
tutoriel, 'tuto1' : tuto1, 'tuto2' : tuto2, 'retour_menu': menu} # On utilise un
dictionnaire pour stocker différentes fonctions utilisées par la suite par la
fonction change_frame

fen1.mainloop() # démarrage du réceptionnaire d'événements
```