Université de Technologie de Compiègne

Devoir 3

Pascal Quach, Korantin Toczé

UV: *Maîtrise des systèmes informatiques (SR01)*
Date de rendu: *30 Décembre 2019*
Semestre: *A19*

Jeu de la vie en Python

```
1 # On construit un simple tableau de taille n x n
def __init__(self, n):
3         self.tableau = [[bool(0) for j in range(n)] for i
            in range(n)]
```

Listing 1: Classe `Tableau` - Constructeur

On initialise les booléens d'un tableau de taille $n * n$ à **faux**.

```
# Méthode d'initialisation du tableau.
## Remise à zéro du tableau
## Cases en vie aléatoirement avec un certain pourcentage
def initialiser_tableau(self, n, pourcentage_vie):
        nombre_de_blocs = n*n
        nombre_de_blocs_vivants = 0
        pourcentage_vie_actuel = 0 # Variable pour comparer

        # On met le tableau à zéro.
        for i in range(0, n):
            for j in range(0, n):
                self.tableau[i][j] = False

        # Boucle pour les cellules en vie
        while (float(pourcentage_vie_actuel) <
           pourcentage_vie and pourcentage_vie != float(0)):

            # Coordonnées aléatoires
            x_random = random.randint(0, n - 1)
            y_random = random.randint(0, n - 1)

            # Vérification que la case n'est pas déjà en vie
            ## Si c'est le cas, on choisit une autre case
               en parcourant le tableau
            while(self.tableau[x_random][y_random] == True):
                if (x_random == n-1):
                    x_random=0
                    if (y_random==n-1):
                        y_random=0
                    else:
                        y_random=y_random+1
                else:
                    x_random=x_random+1

            self.tableau[x_random][y_random] = True #
               Cellule en vie

            # Calcul du pourcentage
            nombre_de_blocs_vivants += 1
            pourcentage_vie_actuel =
               float(nombre_de_blocs_vivants/nombre_de_blocs)
```

Listing 2: Classe `Tableau` - Méthode `initialiser_tableau`

La fonction initialiser tableau est appelé lorsqu'on appuie sur le bouton "Initialiser". Il faut donc réinitialiser les valeurs du tableau à faux. Ensuite, on cherche à initialiser la grille entière de telle sorte à ne pas dépasser un certain pourcentage de vie. A cette fin, on se sert de la fonction `randint` pour générer aléatoirement des coordonnées. On incrémente le pourcentage de vie actuel à chaque fois qu'on rajoute une cellule vivante.

S'il se trouve que les coordonnées générées ont déjà été initialisées, on parcourt la grille de telle sorte à prendre la cellule à droite. Si cela n'est pas possible, on passe à la ligne suivante et ainsi de suite.

```
# Retourne le booléen d'une case (i,j)
    def chercher_case(self, i,j, n):

        # Mode d'adressage prenant en comptant les bords
        ## Le coefficient est présent pour ajuster en cas
           de débordement trop conséquent
        return self.tableau[(i+n*10)%n-1][(j+n*10)%n-1]
```

Listing 3: Classe `Tableau` - Méthode `chercher_case`

Cette méthode est utilisée pour se renseigner sur la valeur de la case (i, j). On utilise un mode d'adressage prenant en compte les cases au bord de la grille. On utilise un coefficient pour éviter les débordements. On retire 1 pour ajuster à l'index du tableau.

```
# Calcul du nombre de voisins d'une case
    def nombre_de_voisins(self, i, j, n):
        nbr_de_voisins = 0
        nbr_de_voisins += 1 if self.chercher_case(i-1, j-1,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i-1, j,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i-1, j+1,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i, j-1,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i, j+1,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i+1, j-1,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i+1, j,
           n) else 0
        nbr_de_voisins += 1 if self.chercher_case(i+1, j+1,
           n) else 0
        return nbr_de_voisins
```

Listing 4: Classe `Tableau` - Méthode `nombre_de_voisins`

On cherche les booléens des huit cases autour pour compter le nombre de voisins.

```
# Mise à jour du tableau
    ## On crée un deuxième Tableau qui est une copie du
       premier
    ## On met à jour le premier Tableau à partir du deuxième
    def mise_a_jour_grille(self, n):

        # Copie du tableau
        copie_tableau = Tableau(n)
        for i in range(n):
            for j in range(n):
                copie_tableau.tableau[i][j] =
                   self.tableau[i][j]

        # Analyse grille
        for i in range(n):
            for j in range(n):
                if (copie_tableau.nombre_de_voisins(i, j,
                   n) < 2 or
                   copie_tableau.nombre_de_voisins(i, j, n)
                   > 3):
                    self.tableau[i][j] = False
                elif (copie_tableau.nombre_de_voisins(i, j,
                   n) == 3):
                    self.tableau[i][j] = True
```

Listing 5: Classe `Tableau` - Méthode `mise_a_jour_grille`

La mise à jour de la grille s'effectue en déclarant un deuxième tableau et en copiant les valeurs du premier dans celui-ci. Il suffit ensuite d'effectuer l'analyse des cases.

```
# Constructeur de l'application
    def __init__(self, couleur_cellule, couleur_fond,
       taille_fenetre, taille_grille, statut,
       pourcentage_vie, vitesse_animation):

        # Parametres
        self.couleur_cellule = couleur_cellule # Couleur
           d'une cellule en vie
        self.couleur_fond = couleur_fond # Couleur d'une
           cellule morte
        self.taille_fenetre = taille_fenetre # Taille de la
           fenetre qui contiendra le jeu de la vie en pixels
        self.taille_grille = taille_grille # Taille de la
           GRILLE du jeu de la vie en nombre de blocs
        self.taille_bloc =
           self.taille_fenetre/self.taille_grille # Calcul
           de la taille d'un bloc
        self.vitesse_animation = vitesse_animation #
           Vitesse de l'animation
        self.pas = int(1000/vitesse_animation) # Le pas est
           par défaut à 1s, vitesse de 1 à 100. Vitesse de
           0.1 s à 1.0s
        self.statut = False # Le booléen du jeu. Marche ou
           Arrêt
        self.pourcentage_vie = pourcentage_vie # Le
           pourcentage de vie demandé pour l'initialisation
           du jeu
        self.grille = Tableau(self.taille_grille) #
           Initialisation d'un objet grille

        # Interface Graphique Utilisateur
        ## Configuration Tkinter
        self.root = tk.Tk()
        self.root.title("SR01 - Devoir 3 - Jeu de la vie")
        self.root.resizable(False, False) # Non extensible

        ## Frame Menu des boutons sur le côté
        self.menu_cote = tk.Frame(self.root, width =
           self.taille_fenetre*0.3, height =
           self.taille_fenetre, bg = "#f0f0f0", relief =
           "flat", borderwidth = 2)
        self.menu_cote.pack(expand = False, fill = "both",
           side = "right", anchor = "nw")
```

Listing 6: Classe `Jeu_de_la_vie` - Constructeur - Partie 1

On déclare tous les paramètres du jeu de la vie, notamment la taille de la fenêtre, la taille de la grille, le statut par défaut, le pourcentage de vie initial, etc. Une application **Tkinter** est lancée, et on définit un premier *Frame* qui correspond au menu sur le côté.

```
        ## Frame Fenetre principale
        ### Frame
        self.principale = tk.Frame(self.root, width =
            self.taille_fenetre, height =
            self.taille_fenetre, bg = "white")
        self.principale.pack(expand = False, fill = "both",
            side = "left")
        ### Canevas de dessin
        self.canevas = tk.Canvas(self.principale, width =
            taille_fenetre, height = taille_fenetre, bg =
            "white")
        self.canevas.pack(fill = 'both', expand = 1)

        ## Boutons de l'application
        self.bouton_lancer = tk.Button(self.menu_cote, text
            = "Lancer", relief = "solid", fg = "blue", width
            = 20, command =self.lancer_jeu)
        self.bouton_lancer.pack(side = "top")

        self.bouton_arreter = tk.Button(self.menu_cote,
            text = "Arreter", relief = "solid", fg = "blue",
            width = 20, command =self.arreter_jeu)
        self.bouton_arreter.pack(side = "top")

        self.bouton_init = tk.Button(self.menu_cote, text =
            "Initialiser", relief = "solid", fg = "blue",
            width = 20, command =self.initialiser_jeu)
        self.bouton_init.pack(side = "top")

        self.bouton_quitter = tk.Button(self.menu_cote,
            text = "Quitter", relief = "solid", fg= "blue",
            width = 20, command = self.root.destroy)
        self.bouton_quitter.pack(side = "bottom")
```

Listing 7: Classe `Jeu_de_la_vie` - Constructeur - Partie 2

Le deuxième *Frame* correspond à celle du *canvas* sur lequel on va dessiner. Les quatre boutons de l'application sont déclarées et on donne les bonnes méthodes à lancer comme argument pour *command*.

```
        ## Sliders de l'application
        self.bouton_vitesse = tk.Scale(self.menu_cote,
           orient = "horizontal", from_ = 1, to = 100,
           length = 180, variable = vitesse_animation, bg =
           "#f0f0f0", relief = "flat", highlightthickness =
           0, label = "Vitesse", fg = "blue", command =
           self.slider_vitesse_animation)
        self.bouton_vitesse.set(self.vitesse_animation) #
           Mise du slider sur la valeur passé en paramètre
        self.bouton_vitesse.pack(side = "bottom")

        self.bouton_pourcentage_vie =
           tk.Scale(self.menu_cote, orient = "horizontal",
           from_= 0, to = 1, length = 180,  variable =
           pourcentage_vie, resolution = 0.001, bg =
           "#f0f0f0", relief = "flat", highlightthickness =
           0, label = "% de Vie", fg = "blue", bd = "0",
           command = self.slider_pourcentage_vie)
        self.bouton_pourcentage_vie.set(self.pourcentage_vie)
           # Mise du slider sur la valeur passé en paramètre
        self.bouton_pourcentage_vie.pack(side = "bottom")

        self.bouton_taille_grille =
           tk.Scale(self.menu_cote, orient = "horizontal",
           from_= 5, to = 100, length = 180, variable =
           taille_grille, bg = "#f0f0f0", relief = "flat",
           highlightthickness = 0, label = "Taille de la
           grille", fg = "blue", command =
           self.slider_taille_grille)
        self.bouton_taille_grille.set(self.taille_grille) #
           Mise du slider sur la valeur passé en paramètre
        self.bouton_taille_grille.pack(side = "bottom")
```

Listing 8: Classe `Jeu_de_la_vie` - Constructeur - Partie 3

Les *scale*s de taille de grille, pourcentage de vie initial et vitesse d'animation sont associées au bonnes méthodes qui permet de mettre à jour la valeur des variables.

```
# Boucle interne du jeu
    def boucle(self):
        if (self.statut == True):
            self.grille.mise_a_jour_grille(self.taille_grille)
            self.dessiner_grille()
            self.root.after(self.pas, self.boucle)
```

Listing 9: Classe `Jeu_de_la_vie` - Constructeur - Partie 3

La boucle du jeu est lancée si et seulement si le booléen statut est vrai. On met à jour la grille, puis on dessine le tableau.

```
# Méthode de dessin de l'application
    def dessiner_grille(self):
        # On supprime l'ancien tableau
        self.canevas.delete("all")

        # Initialisation de la position du pointeur
        position_x = 0
        position_y = 0
        # On parcourt le canvas ligne par ligne et on
           dessine rectangle par rectangle
        for i in range(self.taille_grille):
            position_x = 0
            for j in range(self.taille_grille):
                ### Si la case associée dans le tableau est
                   vivante, on la remplit en rouge
                if (self.grille.tableau[i][j] == True):
                    self.canevas.create_rectangle(position_x,
                       position_y, position_x +
                       self.taille_bloc, position_y +
                       self.taille_bloc, fill =
                       self.couleur_cellule)
                ### Sinon, on ne la remplit pas
                else:
                    self.canevas.create_rectangle(position_x,
                       position_y, position_x +
                       self.taille_bloc, position_y +
                       self.taille_bloc, fill =
                       self.couleur_fond)
                position_x += self.taille_bloc
            ## On passe à la ligne suivante
            position_y += self.taille_bloc
```

Listing 10: Classe `Jeu_de_la_vie` - Méthode `dessiner_grille`

Pour le dessin, on parcourt le tableau ligne par ligne et on dessine un rectangle vide si la cellule est morte; un rectangle rouge si la cellule est vivante. On utilise pour cela la taille d'un bloc calculée à partir de la taille de la fenêtre et de la taille de la grille.

```
# Méthode du bouton <Lancer>
    def lancer_jeu(self):
        # Changement du statut de l'application
        self.statut = True

        # Modification des boutons pour les désactiver
        self.bouton_taille_grille.config(state =
           "disabled", label = "Taille de la grille
           (DESACTIVE)", length = 220)
        self.bouton_pourcentage_vie.config(state =
           "disabled", label = "% de Vie (DESACTIVE)",
           length = 220)
        self.bouton_vitesse.config(length = 220)

        # Lancement de la boucle du jeu
        self.boucle()
```

Listing 11: Classe `Jeu_de_la_vie` - Méthode `lancer_jeu`

Cette méthode permet de mettre à jour le statut du jeu. On change ensuite les *Scales* pour désactiver celui de la taille de la taille et du pourcentage de vie initial. On ne veut pas pouvoir changer la taille de la grille, ni le pourcentage initial lorsque le jeu est en cours. On appelle ensuite la boucle du jeu pour démarrer.

```
# Méthode du bouton <Arrêter>
    def arreter_jeu(self):
        # Changement du statut de l'application
        ## Le changement de statut est suffisant pour
           arrêter la boucle du jeu
        self.statut = False

        # Modification des boutons pour les réactiver
        self.bouton_taille_grille.config(state = "active",
           label = "Taille de la grille", length = 180)
        self.bouton_pourcentage_vie.config(state =
           "active", label = "% de Vie", length = 180)
        self.bouton_vitesse.config(length = 180)
```

Listing 12: Classe `Jeu_de_la_vie` - Méthode `arreter_jeu`

On met à jour le statut du jeu pour arrêter le jeu. On remet les boutons à leur apparence initial.

```
# Méthode du bouton <initialisation>
    def initialiser_jeu(self):
        # On appelle la méthode de l'objet grille
        self.grille.initialiser_tableau(self.taille_grille,
            self.pourcentage_vie)

        # On appelle la méthode de dessin
        self.dessiner_grille()
```

Listing 13: Classe `Jeu_de_la_vie` - Méthode `initialiser_jeu`

Le jeu est initialisé. On dessine ensuite le tableau généré.

```
# Méthode du slider <pourcentage de vie>
    def slider_pourcentage_vie(self, nouvelle_valeur):
        # On met à jour la valeur
        self.pourcentage_vie = float(nouvelle_valeur)

    # Méthode du slider <vitesse d'animation>
    def slider_vitesse_animation(self, nouvelle_valeur):
        # On met à jour la valeur
        self.vitesse_animation = int(nouvelle_valeur)
        # On recalcule le pas utilisé pour animer le jeu
            dans la boucle pour
        self.pas = int(1000/self.vitesse_animation)

    # Méthode du slider <Taille de la grille>
    def slider_taille_grille(self, nouvelle_valeur):
        # On met à jour la valeur
        self.taille_grille = int(nouvelle_valeur)
        # On recalcule la taille d'un bloc pour les afficher
        self.taille_bloc =
            self.taille_fenetre/self.taille_grille
        # On réinitialise les variables tableaux à la bonne
            taille
        self.grille.__init__(self.taille_grille)
```

Listing 14: Classe `Jeu_de_la_vie` - Méthodes pour les sliders

Les trois méthodes correspondants aux *Scale*s de l'application mettent à jour les variables de la classe. Particulièrement, celui de la taille de la grille met à jour la taille de la grille, et réinitialise le tableau lui-même. C'est ce comportement qui oblige à désactiver le *Scale* de la taille de la grille.