PIERRE FEUILLE CISEAUX

1 – REGLES DU JEU

Dans le jeu pierre-papier-ciseaux, aussi connu sous le nom de « chifoumi », deux joueurs se montrent simultanément leur main qui doit symboliser une pierre (poing fermé), un papier (main tendue) ou des ciseaux (l'index et le majeur forment un V).

La pierre bat les ciseaux, les ciseaux battent le papier et le papier bat la pierre. Si les deux joueurs jouent le même symbole, il y a égalité.

2 – VERSION SHELL

Dans un premier temps, vous allez programmer le jeu « Feuille-Pierre-Ciseaux » dans une version shell. Le joueur joue contre l'ordinateur :

- Le programme demande au joueur quel objet il veut montrer
- L'ordinateur tire au hasard l'objet qu'il veut montrer.
- Le programme calcul le score en fonction des coups joués par le joueur et par l'ordinateur.
- Le premier qui arrive à 10 points à gagner.

```
Pierre-Feuille-Ciseaux. Le premier à 10 a gagné !
--------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 2
Joueur montre Feuille

L'ordinateur joue
Ordinateur montre Pierre

Joueur : 1    ordinateur 0
--------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 3
Joueur montre Ciseaux

L'ordinateur joue
Ordinateur montre Pierre

Joueur : 1    ordinateur 1
--------------------------------------------------
```

2.1 – Fonction « choix_joeur() »

La fonction **choix_joeur()** permet au joueur de choisir le coup qu'il veut jouer. Pour cela il doit entrer un nombre compris entre un et trois (prévoir le cas où le joueur ne rentre pas un nombre correct). Ce nombre est retourné par la fonction. Le résultat de l'appel de la fonction doit donner le résultat ci-contre.

```
>>> choix_joueur()
------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 2
2
```

Exercice 1 : **Ouvrir** un nouveau script. **Editer** la fonction **choix_jouer**. **Enregistrer** le script sous le nom « **chifoumi.py** ». **Tester** son fonctionnement.

```
1  def choix_joueur():
2  """ Permet au joueur de jouer
3     Retour :
4     coup : nombre entré par le joueur (compris entre 1 et 3)
5     """
6  # A compléter
```

2.2 – Fonction « jeu_joueur() »

La fonction **jeu_joueur()** permet d'afficher, à partir d'un nombre compris entre 1 et 3, le coup joué par le joueur ou tiré au hasard par l'ordinateur : Pierre, Feuille ou Ciseaux. Le résultat de l'appel de la fonction doit donner le résultat ci-contre :

```
------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 3
Joueur montre Ciseaux
```

Exercice 2 : Dans le script « **chifoumi.py** », **éditer** la fonction **jeu_joueur**. **Enregistrer** le script.

```
1  def jeu_joueur():
2        """Indique le nom du symbole en fonction du nombre entré par le joueur
3        Paramètre :
4           coup : nombre entré par le joueur
5     """
6  # A compléter
```

Tester son fonctionnement avec les appels de fonctions suivants.

```
1  coup_J = choix_joueur()
2  jeu_joueur (coup_J)
```

```
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 1
Pierre
```

Tester le programme sur d'autres appels afin de vérifier son fonctionnement sur toutes les possibilités.

2.3 – Fonction « jeu_ordinateur() »

La fonction **jeu_ordinateur()** permet à l'ordinateur de tiré au hasard le coup qu'il va jouer. Le coup joué par l'ordinateur est alors affiché. Le résultat de l'appel de la fonction doit donner le résultat ci-contre :

```
L'ordinateur joue
Ordinateur montre Feuille
```

Python dispose du module « **random** » qui contient de nombreuses fonctions en relation avec les **nombres aléatoires** notamment la fonction « **randint** » qui permet de délivrer un nombre entier (pseudo-)alétoire dans un certain intervalle de valeurs. Pour utiliser la fonction « **random** » il est nécessaire de charger le module « **random** ».

`randint(a, b)`	Renvoie nombre entier aléatoirement dans l'intervalle [a, b]

Exercice 3 : Dans le script « **chifoumi.py** », **éditer** la fonction **jeu_ordinateur**. **Enregistrer** le script.

```
1                               # Importer la fonction randint du module random
2  def jeu_ordinateur():
3      """Permet à l'ordinateur de tirer au hasard un nombre (compris entre 1
4         et 3) et d'afficher le coup correspondant à ce nombre
5         Retour :
6             coup : nombre tiré au hasard par l'ordinateur
7      """
8  # A compléter
```

Tester son fonctionnement sur plusieurs appels.

```
1  coup_0 = jeu_ordinateur()
```

```
L'ordinateur joue
Ordinateur montre Ciseaux
```

2.4 – Fonction calcul_scores()

La fonction **calcul_scores()** permet à chaque tour, de calculer les nouveaux scores en fonction des coups joués par le joueur et l'ordinateur. Elle reçoit en paramètre le coup joué par le joueur, le coup joué par l'ordinateur et le tableau des scores. Selon les coups joués, le score du joueur ou de l'ordinateur est incrémenté de 1. Cette fonction retourne ensuite le nouveau tableau des scores.

Le tableau des scores **t_scores** est constitué de deux éléments. Le premier élément **t_scores[0]** correspond au score du joueur et le second élément **t_scores[1]** au score de l'ordinateur.

Exercice 4 : Dans le script « **chifoumi.py** », **éditer** la fonction **`calcul_scores`**. **Enregistrer** le script.

```
def calcul_scores(coup_J, coup_O, t_scores):
    """ Calcule les scores du joeur et de l'ordinateur en fonction
        du coup joué par le joueur et par l'ordinateur
        Paramètres :
            coup_J : coup joué par le joueur
            coup_O : coup joué par l'ordinateur
            t_scores = [score_J, score_O]
        Retour :
            t_scores = [score_J, score_O]
    """
# A compléter
```

Tester son fonctionnement sur l'appel suivant.

```
scores = [0, 0]    # Initialisation des scores
# Joueur montre Feuille et l'ordinateur Pierre
scores = calcul_scores(2, 1, scores)
print(scores)
```

Tester le programme sur d'autres appels afin de vérifier son fonctionnement sur toutes les possibilités.

2.5 – Fonction `jouer()`

La fonction **`jouer()`** fait appel à la fonction **`jeu_joueur()`** en lui passant en paramètre le coup joué par le joueur et à la fonction **`jeu_ordinateur()`** afin de récupérer le coup joué par l'ordinateur. A partir du coup joué par le joueur, du coup joué par l'ordinateur et du tableau des scores, elle fait appel à la fonction **`calcul_scores()`** afin de récupérer le nouveau tableau des scores. Les scores seront ensuite affichés.

Exercice 5 : Dans le script « **chifoumi.py** », **éditer** la fonction **`jouer`**. **Enregistrer** le script.

```
def jouer(coup_J, t_scores):

    """Fait appel à la fonction jeu_joueur, récupère le coup joué par
       l'ordinateur en faisant appel à la fonction jeu_ordinateur et récupère
       le tableau des scores en faisant appel à la fonction calcul_scores
       Affiche les scores
       Paramètres :
       coup_J : coup joué par le joueur
            t_scores = [score_J, score_O]
    """
# A compléter
```

Tester son fonctionnement sur l'appel suivant.

```
scores = [0, 0]    # Initialisation des scores
jouer(1, scores) # Joueur montre Feuille
```

Tester le programme sur d'autres appels afin de vérifier son fonctionnement sur toutes les possibilités.

2.6 – Programme principal

L'algorithme du programme principal est donné ci-dessous.

```
VARIABLES
coup_J : entier compris entre 1 et 3
coup_O : entier compris entre 1 et 3
scores : tableau d'entier scores = [score_J, score_O]
DEBUT
   scores <-- 0           #On met à 0 le tableau des scores
   tant que score joueur < 10 et scores[1] < 10
      coup_J <-- choix_joueur()
      Jouer ()
FIN
```

Exercice 6 : Dans le script « **chifoumi.py** », **éditer** le programme principal à la suite de la définition des différentes fonctions afin d'obtenir le fonctionnement ci-dessous. **Enregistrer** le script. **Tester** le fonctionnement du jeu.

```
Pierre-Feuille-Ciseaux. Le premier à 10 a gagné !
--------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 2
Joueur montre Feuille

L'ordinateur joue
Ordinateur montre Pierre

Joueur : 1    ordinateur 0
--------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 1
Joueur montre Pierre

L'ordinateur joue
Ordinateur montre Ciseaux

Joueur : 2    ordinateur 0
--------------------------------------------------
```

Exercice 7 : Modifier le programme afin d'afficher le gagnant lorsqu'il arrive à 10 points

```
Joueur : 6    ordinateur 9
--------------------------------------------------
Le joueur joue
1 : Pierre, 2 : Feuille, 3 : Ciseaux ? 1
Joueur montre Pierre

L'ordinateur joue
Ordinateur montre Feuille

Joueur : 6    ordinateur 10
Ordinateur a gagner
```

3 – INTERFACES GRAFIQUES

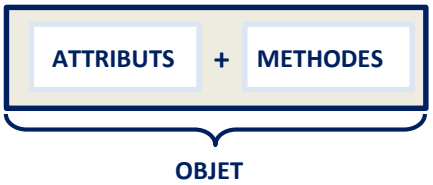
3.1 – La programmation orientée objet

Le langage Python est un langage orienté objet. L'idée de la **programmation orientée objet (poo)**, est de **manipuler des éléments** que l'on appelle des "**objets**" dans son code source. Un des nombreux avantages de la poo, est qu' il existe des **milliers d'objets** (on les appelle des classes) **prêts à être utilisés**. On peut réaliser des programmes extrêmement complexes uniquement en utilisant des **classes (objets) pré-existantes**. Il est possible aussi de créer ses propres classes.

Une **classe** est équivalente à un **nouveau type de données** plus ou moins complexe (avec plus ou moins d'attributs). Un **objet** ou une **instance** est un **exemplaire particulier d'une classe**.
Par exemple il est possible de définir la classe **`Point`** qui représente les coordonnées un point dans un repère x/y. Cette classe d'objet possède deux attributs **`x`** et **`y`**. Il est alors de définir plusieurs objets de type **`Point`** avec des valeurs d'attributs différentes. Par exemple :

- L'objet **`A`** représentant un point de coordonnées (0, 10) : `A.x = 0` et `A.y = 10`
- L'objet **`B`** représentant un point de coordonnées (2, 15) : `B.x = 2` et `B.y = 15`

La plupart des classes **encapsulent** à la fois **les données et les méthodes** (fonctions) applicables aux objets. On pourrait définir un objet comme un **paquet** contenant des **attributs et des méthodes**.



Pour appliquer une méthode à un objet il faut utiliser la syntaxe ci-contre.

```
nom_objet.nom_methode()
```

3.2 – Module Tkinter

Le module Tkinter (**T**ool **K**it **inter**face) est la bibliothèque graphique libre d'origine pour le langage Python, permettant la création d'interfaces graphiques. Le module Tkinter est installé par défaut dans les différentes interfaces de développement Python.

Ce module propose de nombreux **composants graphiques** ou **widgets** : **fenêtre** (Tk), **bouton** (Button), **case à cocher** (Checkbutton), **étiquette** (Label), **zone de texte** simple (Entry), **menu** (Menu), **zone graphique** (Canvas), **cadre** (Frame)...

Il permet également de **gérer de nombreux événements** : clic sur la souris, déplacement de la souris, appui sur une touche du clavier, top d'horloge...

Le module Tkinter permet de réaliser de la **programmation événementielle**. Ce type de programme est basé sur une « **boucle infinie** » qui permet d'attendre **l'apparition des évènements**. Dans le cas de Tkinter, la boucle infinie est réalisée au moyen de l'instruction **`mainloop()`** (boucle principale).

La structure d'un programme utilisant le module Tkinter présente généralement la forme suivante :

- Définition des **fonctions** associées au programme.
- Définition de **l'interface graphique** (widgets).
- Définition des **évènements**.
- Définition de la **boucle principale**.

3.3 – Fenêtre principale

Pour créer la fenêtre principale de l'application tkinter il faut utiliser la fonction **`Tk()`**.

```
nom_fenetre = Tk()
```

Exercice 8

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script.

```
1 from tkinter import *
2 fen=Tk()
3 fen.mainloop()
```

Parmi les méthodes (fonctions particulières applicables aux objets de type Tk) qui peuvent être utilisées pour modifier l'aspect de la fenêtre graphique, on trouve :

- **title** : Permet de donner un nom à la fenêtre

```
nom_fenetre.title("Titre de la fenêtre")
```

- **geometry** : Permet de définir les dimensions de la fenêtre (Largeur x Hauteur) et position (x, y) du coin supérieur gauche par rapport à celui de l'écran.

```
nom_fenetre.geometry("LargeurxHauteur+x+y")
```

- **configure** : Permet de définir une ou plusieurs options (couleur de fond par exemple). Pour modifier la couleur de fond on utilise l'option **bg** (background) en précisant la couleur soit par un mot clé (`"red"`) ou en hexadécimale (`"#FF0000"`).

```
nom_fenetre.configure(bg = "couleur")
```

- **destroy** : Permet de fermer la fenêtre.

```
nom_fenetre.destroy()
```

Exercice 9

Modifier le script précédent afin d'obtenir la fenêtre suivante.

3.4 – Widgets Button et Label

Un widget **bouton** (Button) permet de **proposer une action à l'utilisateur**. Un **label** est un espace prévu pour **afficher un texte**.

Pour créer un bouton il faut utiliser la fonction **Button.**

```
nom_bouton = Button(nom_fenetre, text = "texte bouton", command = fonction)
```

Les principales options du widget Button sont :

- Le **nom de la fenêtre** dans lequel sera placé le bouton
- **text** : texte qui sera affiché dans le bouton
- **command** : nom de la fonction (pas parathèses) qui sera appelée lors de l'appui sur le bouton.

Les widgets ne seront placés dans la fenêtre graphique qui si la méthode **pack()** est appliquée. Le positionnement du widget dans la fenêtre peut être configuré par un certain nombre d'attributs (https://www.fil.univ-lille1.fr/~marvie/python/chapitre6.html)

Exercice 10

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme

```
1 from tkinter import *
2 fen=Tk()
3 fen.title("Interface graphique")
4 fen.geometry("200x200")
5 bouton1 = Button(fen, text = "Quitter", command = fen.destroy)
6 bouton1.pack()
7 fen.mainloop()
```

Pour créer une zone de texte il faut utiliser la fonction **Label.**

```
nom_texte = Label(fenetre, text="Texte à afficher", font=("Style", Taille))
```

Les principales options du widget Label sont :

- Le **nom de la fenêtre** dans lequel sera placé la zone de texte
- **text** : texte qui sera affiché dans la zone de texte
- **fg** : Couleur du texte
- **font** : Style et taille de la police.

Exercice 11

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme

```
1 from tkinter import *
2 fen=Tk()
3 fen.title("Interface graphique")
4 fen.geometry("400x200")
5 texte1 = Label(fen, text = "Bienvenue en STI2D", fg = "red",font=("Arial", 14))
6 texte1.pack()
7 bouton1 = Button(fen, text = "Quitter", command = fen.destroy)
8 bouton1.pack(side = BOTTOM, pady = 10)
9 fen.mainloop()
```

Un appui sur un bouton peut lancer l'exécution d'une fonction quelconque du programme. L'appel se fait par l'option **command** (attention il ne faut mettre les parenthèses).

Exercice 12

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme

```
1  from tkinter import *
2
3  def couleur() :
4      texte1.configure(fg = "red")
5
6  fen=Tk()
7  fen.title("Interface graphique")
8  fen.geometry("400x200")
9  texte1 = Label(fen, text = "Bienvenue en STI2D", font=("Arial", 14))
10 texte1.pack()
11 bouton1 = Button(fen, text = "Rouge", command = couleur)
12 bouton1.pack(side = BOTTOM, pady = 10)
13 fen.mainloop()
```

Pour autoriser le passage de paramètres lors de l'appel de la fonction après un appui sur le bouton, il utiliser des fonctions `lambda`.

Exercice 13

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme

```
1  from tkinter import *
2
3  def couleur(Couleur_Texte) :
4      texte1.configure(fg = Couleur_Texte)
5
6  fen=Tk()
7  fen.title("Interface graphique")
8  fen.geometry("400x200")
9  texte1 = Label(fen, text = "Bienvenue en STI2D", font=("Arial", 14))
10 texte1.pack()
11 bouton1 = Button(fen, text = "Rouge", command = lambda : couleur("red"))
12 bouton1.pack(side = LEFT, padx = 50)
13 bouton2 = Button(fen, text = "Bleue", command = lambda : couleur("blue"))
14 bouton2.pack(side = RIGHT, padx = 50)
15 fen.mainloop()
```

3.5 – Organiser les widgets dans la fenêtre : méthode grid()

La méthode **`grid()`** permet de placer les widgets sous forme de grille c'est-à-dire sous forme de ligne (**`row`**) et de colonne (**`column`**). La numérotation des lignes commençant à 0. Les attributs **`rowspan`** et **`columnspan`** permettent de fusionner des lignes et des colonnes.

Exercice 14

Ouvrir un nouveau script. **Editer**, dans l'éditeur, les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme

```
1  from tkinter import *
2
3  def nop () :
4      return 0
5
6  fen=Tk()
7  fen.title("Interface graphique")
8  texte = Label(fen, text = "INTERFACE HOMME MACHINE")
9  texte.grid(row = 0, columnspan = 3)
10 bR = Button(fen, text = "QUIT", fg = "white", bg = "red", command = fen.destroy)
11 bR.grid(row = 1, column = 0)
12 bV = Button(fen, text = "RUN", fg = "white", bg = "green", command = nop)
14 bV.grid(row = 1, column = 1)
15 bB = Button(fen, text = "EDIT", fg = "white", bg = "blue", command = nop)
16 bB.grid(row = 1, column = 2)
17 fen.mainloop()
```

3.6 – Affecter une image à un label ou à un bouton

Il possible d'affecter une image à un widget ou à un bouton.

La fonction **PhotoImage** permet de générer un objet image à partir d'un fichier gif , png ou pgm.

```
nom_image = PhotoImage("nom_fichier.extension")
```

Le fichier doit se trouver dans le même dossier que le programme, sinon il est nécessaire d'indiquer le chemin du dossier dans lequel se trouve l'image.

Pour affecter une image à un bouton ou à un label il faut utiliser l'attribut **image**.

```
nom_bouton = Button(nom_fenetre, image = nom_image, commande = nom_fonction)
```

```
nom_label = Label(nom_fenetre, image = nom_image)
```

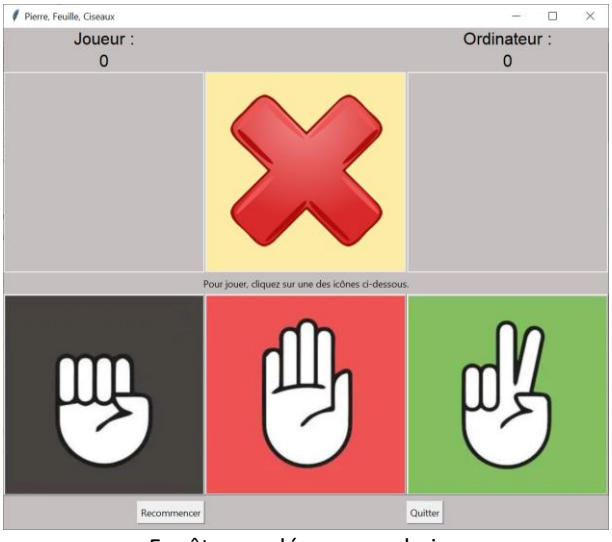
Exercice 15

Ouvrir un nouveau script. **Enregistrer**-le dans le dossier Exercice15 qui a été fourni. **Editer** les lignes de code ci-dessous **Exécuter** le script. **Expliquer** le fonctionnement du programme.

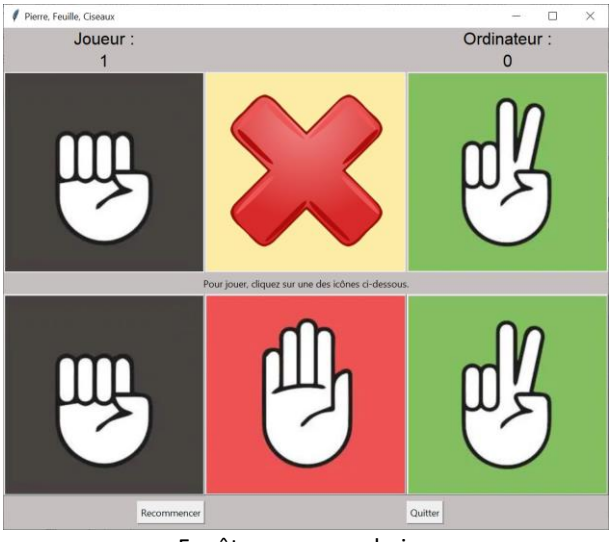
```
1  from tkinter import *
2
3  def modif_Etat(image_etat) :
4      Etat.configure(image = image_etat)
5
6  fen=Tk()
7  fen.title("Interface graphique")
8
9  Marche = PhotoImage(file = 'Marche.png')
10 Arret = PhotoImage(file = 'Arret.png')
11 BoutonV = PhotoImage(file = 'BVert.png')
12 BoutonR = PhotoImage(file = 'BRouge.png')
14
15 texte = Label(fen, text = "INTERFACE HOMME MACHINE")
16 texte.grid(row = 0, columnspan = 3)
17
18 Etat = Label(fen, image = Arret)
19 Etat.grid(row = 1, column = 1)
20
21 bR = Button(fen, image = BoutonR, command = lambda : modif_Etat(Arret))
22 bR.grid(row = 1, column = 0)
23 bV = Button(fen, image = BoutonV, command = lambda : modif_Etat(Marche))
24 bV.grid(row = 1, column = 2)
25 fen.mainloop()
```

4 – VERSION GRAPHIQUE DU JEU PIERRE FEUILLE CISEAUX

La version graphique se présente ainsi :

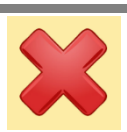
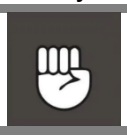


- Fenêtre au démarrage du jeu -



- Fenêtre au cours du jeu -

La fenêtre est organisée en grille constituée de 6 lignes et 3 colonnes :

	0	1	2
0	Joueur		Ordinateur
1	Score Joueur		Score Ordinateur
2	Image coup joué par le joueur		Image coup joué par l'ordinateur
3	Pour jouer, cliquez sur une des icônes ci-dessous.		
4			
5	Bouton « Recommencer »		Bouton « Quitter »

Les widgets utilisés pour réaliser la fenêtre sont les suivants :

- **Ligne 0** : 2 zones de texte dont le texte est fixe.
- **Ligne 1** : 2 zones de texte dont le texte change en fonction des scores.
- **Ligne 2** : 3 zones de texte dans lesquelles seront insérées des images. Les images des colonnes 0 et 2 seront fonctions des coups joués par le joueur ou par l'ordinateur (au démarrage, il faudra placer une image vide). L'image de la colonne 1 est une image fixe.
- **Ligne 3** :Zone de texte dont le texte est fixe.
- **Ligne 4** : 3 boutons dans lesquels seront insérées les images des 3 coups possibles
- **Ligne 5** : 1 bouton « Recommencer » qui permet de réinitialiser la partie et 1 bouton 3 « Quitter » qui permet de terminer la partie.

4.1 – Fenêtre graphique

Au lancement du jeu, les scores doivent être initialisés et affichés à 0 et les images placées dans les zones des coups joués seront une image « Vide ».

Exercice 16

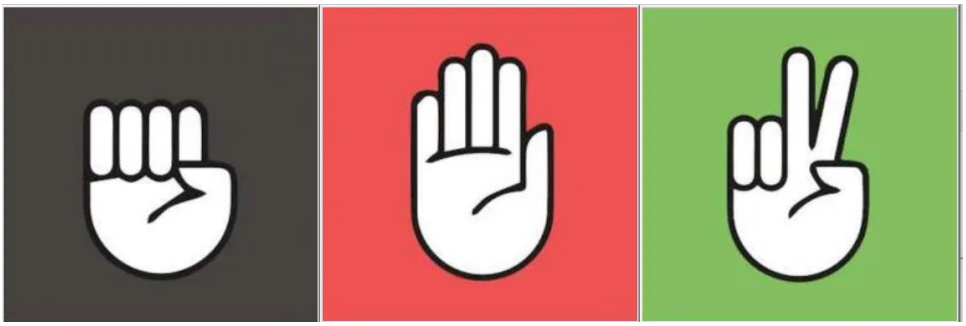
Ouvrir le script « Pierre_Feuille_Ciseaux.py » qui se trouvent dans le dossier ressource Pierre_Feuille_Ciseaux. Dans la zone « Programme Principal », **créer** une fenêtre Tkinter. **Donner**-lui un titre. **Fixer** une couleur de fond pour la fenêtre.

Editer les lignes de codes qui permettent de créer les images nécessaires au jeu (Utiliser les images données dans le dossier Pierre_Feuille_Ciseaux ou utiliser d'autres images).

Créer les zones de texte des lignes 0, 1, 2 et 3 de la grille.

Exécuter le script et vérifier que l'affichage correspond à l'affichage attendu.

Les boutons de la ligne 4, seront représentés par les trois images « Pierre », « Feuille », « Ciseaux ».



L'appui sur le premier bouton (Pierre) doit permettre l'appel de la fonction **`jouer(1, scores)`**, l'appui sur le second bouton (Feuille) doit permettre l'appel de la fonction **`jouer(2, scores)`** et l'appui sur le second bouton (Ciseaux) doit permettre l'appel de la fonction **`jouer(3, scores)`**. Pour cela il faut utiliser des fonctions `lambda` avec l'attribut **`command`**.

La ligne 5 est constituée de 2 boutons. Le bouton « Quitter » doit terminer la partie et fermer la fenêtre graphique. Le bouton « Recommencer » doit permettre de réinitialiser la partie en faisant appel à la fonction **`reinit`** en lui passant en paramètre le tableau des scores.

Exercice 17

Créer les boutons des lignes 4 et 5.

Exécuter le script et **vérifier** que l'affichage correspond à l'affichage attendu.

4.2 – Définitions des fonctions

Exercice 18

Editer la fonction **`calcul_scores`** en recopiant les lignes de code de la même fonction dans la version shell.

Exercice 19

En partant de la fonction de la version shell, **modifier** la fonction **`jeu_joueur`** afin que l'image correspondant au coup joué par le joueur soit affichée dans la zone de texte adéquate. Pour cela utiliser la méthode **`configure`** :

```
nom_label.configure(image = nom_image)
```

Exercice 20

En partant de la fonction de la version shell, **modifier** la fonction **`jeu_ordinateur`** afin que l'image correspondant au nombre tiré au hasard par l'ordinateur soit affichée dans la zone de texte adéquate.

Exercice 21

En partant de la fonction de la version shell, **modifier** les lignes de codes de la fonction **jouer** afin que les scores soient affichés dans les zones de texte adéquates. Pour cela utiliser la méthode **configure** :

```
nom_label.configure(text = "Nouveau texte")
```

Attention : Il faudra transformer les scores en chaînes de caractères.

La fonction **reinit** permet de :

- Mettre à 0 le tableau des scores.
- Mettre à 0 l'affichage des scores dans les zones de texte adéquates.
- Placer l'image « Vide » dans les zones des coups joués.
- Pour cela utiliser la méthode **configure** :

Exercice 22

Editer la fonction **reinit.**

```
1  def reinit(coup_J, t_scores):
2      """Réinitialise la partie. Place les images "Vide" dans les zones
3         correspondant aux coups joués par le joueur et par l'ordinateur
4         Remet à 0 les scores
5         Paramètres :
6             t_scores = [score_J, score_O]
7      """
8  # A compléter
```

Exercice 23

Exécuter le script « Pierre_Feuille_Ciseaux.py » et **vérifier** que le fonctionnement correspond au fonctionnement attendu.