Ce document présente les différentes fonctionnalités de la bibliothèque `graphics_nsi`, version 4.1. Cette bibliothèque est construite sur le module `Pygame` et permet d'utiliser la librairie standard `SDL` (Simple DirectMedia Layer) qui est très intéressante pour la gestion d'événements, la création d'applications graphiques, etc... Pour charger cette bibliothèque, n'oubliez pas de commencer votre script par l'instruction

```
from graphics_nsi import *
```

Sommaire

1 La classe `Point(a,b)`

Cette classe, comportant deux champs `x` et `y`, permet de définir un point de la façon suivante : `P.x` est l'abscisse du point `P` et `P.y` est l'ordonnée du point `P`. Pour définir par exemple le point `P` de coordonnées $(10; 50)$, on écrira `P=Point(10,50)`.
D'autre part, on peut comparer si deux points `P` et `Q` sont égaux ou non à l'aide des instructions `P == Q` et `P != Q`. On peut également afficher les coordonnées d'un point `P` dans la console à l'aide de l'instruction `print(P)`.

2 Les couleurs

- Plusieurs couleurs sont prédéfinies : `black`, `blue`, `brown`, `cyan`, `gold`, `gray`, `green`, `magenta`, `orange`, `pink`, `purple`, `red`, `salmon`, `silver`, `turquoise`, `white` et `yellow`.
- On peut également créer des couleurs au format RGB à l'aide de la fonction `couleur_RGB(r,g,b)`, où `r`, `g` et `b` sont des nombres entiers compris entre 0 et 255 et correspondant respectivement à la quantité de rouge, de vert et de bleu.

3 Manipulation de la fenêtre graphique

Les fonctions d'affichage de la fenêtre graphique sont les suivantes :

- `init_graphic(W,H)` : initialise la fenêtre graphique ; ses deux arguments représentent respectivement sa largeur, sa hauteur. L'origine $(0; 0)$ est située en haut à gauche. Cette fonction admet également trois arguments optionnels :
  - `name` (une chaîne de caractère, par défaut, `Fenêtre ISN`) qui indique le nom de la fenêtre graphique,
  - `bg_color` (une couleur, par défaut, `black`) qui donne la couleur de l'arrière-plan de la fenêtre graphique,
  - `fullscreen` (un entier, par défaut, `0`) qui permet d'afficher la fenêtre avec la taille donnée s'il vaut `0` et en plein écran pour toute autre valeur.

  Par exemple, pour afficher une fenêtre ayant pour taille 600×800, pour nom `Essai` et ayant un arrière-plan rouge, on tape `init_graphic(600,800,name="Essai",bg_color=red)`.
- `init_graphic_fullscreen()` : initialise la fenêtre graphique en plein écran (n'est pas disponible sur tous les écrans). Cette fonction admet un argument optionnel `bg_color` qui attribue une couleur à l'arrière-plan de la fenêtre graphique (par défaut `black`).
- `wait_escape()` : instruction bloquante ; attend que l'on tape sur `Echap` pour quitter. Elle admet six arguments optionnels :
  - `text` (une chaîne de caractères, par défaut, `Appuyer sur Echap pour terminer`) qui indique le texte à afficher,
  - `size` (un entier, par défaut, `20`) qui indique la taille du texte à afficher,
  - `color` (une couleur, par défaut, `gray`) qui indique la couleur du texte à afficher,
  - `text_bold` (un booléen, par défaut, `False`) qui indique si le texte est en gras,
  - `text_italic` (un booléen, par défaut, `False`) qui indique si le texte est en italique.
  - `place` (un point) qui indique le point en haut à gauche à partir duquel est affiché le texte. Par défaut, le texte est centré en bas de la fenêtre graphique.

  Par exemple, pour afficher le texte en bleu et en gras, on tape `wait_escape(color=blue,text_bold=True)`.
- `clear_graphics()` : efface le contenu de la fenêtre graphique.
- `size_graphics()` : renvoie le tuple `(largeur,hauteur)` des dimensions de la fenêtre graphique.
- `affiche_auto_off()` : désactive l'affichage automatique sur la fenêtre graphique.
- `affiche_auto_on()` : active l'affichage automatique sur la fenêtre graphique.
- `affiche_all()` : affiche tous les tracés réalisés entre les deux instructions précédentes.

4 Affichage de textes (écrits par défaut en `Verdana`)

Les fonctions suivantes gèrent l'affichage de textes sur la fenêtre graphique :

- `largeur_texte(T,t)` : renvoie la largeur d'un texte `T` écrit en taille `t`. Cette fonction admet également, comme la fonction `wait_escape()`, les arguments optionnels `text_bold` et `text_italic` (par défaut, `False`).
- `hauteur_texte(T,t)` : renvoie la hauteur d'un texte `T` écrit en taille `t`. Cette fonction admet également, comme la fonction `wait_escape()`, les arguments optionnels `text_bold` et `text_italic` (par défaut, `False`).
- `aff_pol(T,t,P,C)` : permet d'afficher une chaîne de caractères `T` de taille `t` ; `P` est le point en haut à gauche et `C` est la couleur d'affichage de la chaîne. Cette fonction admet également, comme la fonction `wait_escape()`, les arguments optionnels `text_bold` et `text_italic` (par défaut, `False`).
- `list_fonts()` : renvoie sous forme de liste toutes les polices disponibles.
- `test_font(S)` : renvoie 1 si la police `S` est disponible et 0 sinon ; `S` est une chaîne de caractères.
- `change_font(S)` : attribue à la police courante la police `S` si elle est disponible ; `S` est une chaîne de caractères.

5 Tracé de formes

Les fonctions suivantes permettent de tracer des formes sur la fenêtre graphique :

- `draw_pixel(P,C)` : affiche un pixel de couleur `C` au point `P`.
- `draw_line(P,Q,C)` : trace un segment de couleur `C` entre les points `P` et `Q`.
- `draw_triangle(P,Q,R,C)` : trace un triangle de couleur `C` et de sommets `P`, `Q` et `R`.
- `draw_fill_triangle(P,Q,R,C)` : trace un triangle plein de couleur `C` et de sommets `P`, `Q` et `R`.
- `draw_rectangle(P,l,h,C)` : trace un rectangle de couleur `C`, de centre `P`, de largeur `l` et de hauteur `h`.
- `draw_fill_rectangle(P,l,h,C)` : trace un rectangle plein de couleur `C`, de centre `P`, de largeur `l` et de hauteur `h`.
- `draw_circle(P,r,C)` : trace un cercle de couleur `C`, de centre `P` et de rayon `r`.
- `draw_fill_circle(P,r,C)` : trace un disque de couleur `C`, de centre `P` et de rayon `r`.
- `draw_ellipse(P,l,h,C)` : trace une ellipse de couleur `C` inscrite dans un rectangle de centre `P`, de largeur `l` et de hauteur `h`.
- `draw_fill_ellipse(P,l,h,C)` : trace une ellipse pleine de couleur `C` inscrite dans un rectangle de centre `P`, de largeur `l` et de hauteur `h`.
- `draw_arc(P,l,h,start,end,C)` : trace une portion d'ellipse de couleur `C` inscrite dans un rectangle de centre `P`, de largeur `l` et de hauteur `h` ; les angles `start` et `end` sont donnés en radians et sont dans $[0;2\pi[$.
- `draw_sector(P,r,start,end,C)` : trace un secteur angulaire d'origine `P`, de rayon `r`, d'angle `end-start` et de couleur `C` ; les angles `start` et `end` sont donnés en radians et sont dans $[0;2\pi[$.
- `draw_fill_sector(P,r,start,end,C)` : trace un secteur angulaire plein d'origine `P`, de rayon `r`, d'angle `end-start` et de couleur `C` ; les angles `start` et `end` sont donnés en radians et sont dans $[0;2\pi[$.

Les tracés ne sont affichés sur la fenêtre graphique que lors d'une instruction bloquante ou avec la commande `affiche_all()`.

6 Gestion d'images

Les fonctions suivantes gèrent l'affichage d'images sur la fenêtre graphique :

- `load_image(titre,P)` : affiche une image ; `titre` est une chaîne de caractères donnant le titre de l'image et `P` est le point en haut à gauche de l'image. La fonction renvoie également la surface pygame correspondant à cette image.
- `load_image_transp(titre,P)` : affiche une image à transparence ; `titre` est une chaîne de caractères donnant le titre de l'image et `P` est le point en haut à gauche de l'image. La fonction renvoie également la surface pygame correspondant à cette image.
- `transfer_image(titre)` : charge une image dans une surface pygame, mais sans l'afficher ; `titre` est une chaîne de caractères donnant le titre de l'image. Cette fonction est utile pour déterminer les dimensions d'une image, pour redimensionner ou sauvegarder une image sans avoir à l'afficher.
- `save_image(I,F)` : permet de sauvegarder une image `I` donnée sous forme d'une surface pygame dans le fichier `F` donné sous forme d'une chaîne de caractères.
- `get_size_image(I)` : renvoie les dimensions de la surface pygame `I`.
- `resize_image(I,W,H)` : permet de redimensionner la surface pygame `I` avec la largeur `W` et la hauteur `H` ; `W` et `H` sont des entiers supérieurs à 1.

7 Gestion des sons et musiques

Les fonctions suivantes permettent de gérer les sons et musiques :

- `play_sound(F)` : joue le son `F` passé en argument ; `F` est une chaîne de caractères contenant le nom du fichier son.
- `stop_sound(F)` : arrêter le son `F` passé en argument ; `S` est une chaîne de caractères contenant le nom du fichier son.
- `set_volume_sound(F,v)` : règle le volume du son `F` ; `F` est une chaîne de caractères contenant le nom du fichier son et `v` est un flottant entre `0.0` et `1.0`.
- `get_volume_sound(F)` : renvoie le volume du son `F` ; `S` est une chaîne de caractères contenant le nom du fichier son.
- `load_music(F)` : charge la musique `F` mais ne la joue pas ; `F` est une chaîne de caractères contenant le nom du fichier audio (utiliser de préférence des `.wav`) Si une musique était déjà chargée, cela la stoppe si elle était jouée.
- `play_music()` : lance la musique ; si celle-ci était déjà jouée, elle reprend au début. La fonction admet un argument optionnel `loop` qui prend les valeurs 0 ou 1 ; si `loop` vaut 1, alors la musique est jouée en boucle.
- `restart_music()` : relance la musique à partir du début.

- `stop_music()` : arrête la musique qui est jouée.
- `pause_music()` : met la musique en pause.
- `unpause_music()` : reprend la musique après une pause.
- `set_volume_music(v)` : règle le volume de la musique ; `v` est un flottant entre `0.0` et `1.0`.
- `get_volume_music()` : renvoie le volume de la musique.
- `get_busy_music()` : renvoie 1 si une musique est jouée et 0 sinon.

8 Gestion de la souris

Les fonctions suivantes gèrent différents événements liés à la souris :

- `wait_clic()` : instruction bloquante ; attend que l'on clique gauche avec la souris et renvoie un point contenant les coordonnées du clic.
- `wait_clic_LR()` : instruction bloquante ; attend que l'on clique avec la souris et renvoie une liste contenant une chaîne de caractère indiquant le bouton cliqué (`G` pour gauche et `D` pour droit) ainsi qu'un point dont les coordonnées sont celles du clic.
- `device_mouse_off()` : supprime le curseur de la souris dans la fenêtre graphique.
- `device_mouse_on()` : affiche le curseur de la souris dans la fenêtre graphique.
- `get_mouse()` : instruction bloquante ; renvoie un point donnant la position de la souris.
- `is_mouse_pressed_left()` : renvoie `True` si le bouton gauche de la souris est cliqué.
- `is_mouse_pressed_right()` : renvoie `True` si le bouton droit de la souris est cliqué.
- `is_mouse_focused()` : renvoie `True` si la fenêtre graphique est sélectionnée.

9 Gestion du clavier

Les fonctions suivantes gèrent différents événements liés au clavier :

- `wait_key()` : instruction bloquante ; attend que l'on tape sur une touche du clavier (y compris avec une combinaison de touches) et retourne le caractère correspond.
- `wait_arrow()` : instruction bloquante ; attend que l'on tape sur une touche du clavier et renvoie
  - `up`, `down`, `left` ou `right` suivant que l'on a tapé sur la flèche du haut, du bas, de gauche ou de droite,
  - une chaîne vide sinon.

10 Gestion du temps

Les fonctions suivantes gèrent différents événements liés au temps :

- `attendre(millisecondes)` : attend le nombre de millisecondes passé en argument.
- `chrono_start()` : démarre un chronomètre.
- `chrono_val()` : affiche le temps écoulé en millisecondes depuis le lancement du chronomètre.