Fiche Présentatives AxHelper

Objectif : AxHelper

Description : Dans le cadre de mon BTS SIO option SLAM, j'ai développé une application nommée AxHelper, destinée à assister les personnes souffrant de forts tremblements en améliorant la précision de l'utilisation de la souris. AxHelper est une solution innovante qui stabilise les mouvements du curseur, déplace un tracker vers les icônes les plus proches et propose une confirmation de clic via une touche du clavier. Elle est spécialement conçue pour faciliter et rendre plus précise l'interaction avec l'ordinateur pour les utilisateurs ayant des tremblements importants.

Fonctionnalités principales :

- **Stabilisation du curseur** : Réduction des vibrations indésirables de la souris en ajustant automatiquement la position du curseur pour qu'il se fixe sur les icônes les plus proches.

- **Tracker intelligent** : Le tracker se déplace vers les icônes les plus proches et se bloque sur elles, permettant une interaction précise avec les éléments de l'interface utilisateur.

- **Confirmation de clic** : Possibilité de confirmer les clics à l'aide d'une touche du clavier, évitant ainsi les clics accidentels et offrant une meilleure maîtrise.

- **Interface conviviale** : Interface utilisateur simple et intuitive, permettant une configuration facile de l'application selon les besoins spécifiques de l'utilisateur.

Technologies utilisées :

- Langage de programmation : Python

- Bibliothèques utilisées : PyAutoGUI pour la gestion des mouvements de la souris et des clics, Tkinter pour l'interface utilisateur, et OpenCV pour la détection des icônes.

Objectifs pédagogiques : Ce projet m'a permis de développer et de renforcer mes compétences en programmation Python ainsi qu'en :

- Interaction et manipulation des périphériques d'entrée (souris, clavier).

- Développement d'algorithmes de traitement de signal pour la stabilisation des mouvements.

- Utilisation de la vision par ordinateur pour la détection et le suivi des icônes.

- Conception et développement d'interfaces utilisateurs avec Tkinter.

- Gestion des événements et des interactions en temps réel.

Perspectives d'amélioration :

- Ajout d'options de personnalisation avancées pour ajuster la sensibilité et le comportement de l'application en fonction des besoins individuels des utilisateurs.

- Intégration de l'apprentissage automatique pour mieux prédire et ajuster les mouvements en fonction des habitudes de chaque utilisateur.

- Développement de versions compatibles avec différents systèmes d'exploitation.

- Amélioration continue de l'interface utilisateur pour une meilleure expérience et accessibilité.

Nous avions une semaine pour terminer le projet. Afin d'assurer un rendu de toutes les missions, nous avons créé un Trello sur lequel nous avons ajouté toutes les missions. Grâce à ce Trello, nous avons pu planifier nos activités et également savoir où nous en étions

Extrait de code :

```python
if len(Speeds) >= 2 and int(Speeds[-2]) != 0:
    velocity_ratio = int(newSpeed) / int(Speeds[-2])
else:
    velocity_ratio = 0

# Efface la ligne précédente en imprimant des espaces pour garantir une longueur uniforme
print('\r', end='')

# Formatage de la ligne de sortie avec une longueur fixe
output_line = f"Position : {x}, {y} | Vitesse : {newSpeed} pixels/secondes | Ratio de vitesse : {int(velocity_ratio)}".ljust(80)
#print(output_line, end='')
X=x
Y=y
# Capture a screenshot of the entire screen
#screenshot = pyautogui.screenshot('desktop_screenshot.png')

from PIL import Image

# Load the screenshot
#screenshot = Image.open('calc-screen.png')

# Load the template image (icon image)
icon_chrome = Image.open('7-calc.png')
# Create a pool of processes
pool = multiprocessing.Pool()

# Use the pool to parallelize icon detection
results = [pool.apply_async(detect_icon, args=(image_path, X, Y)) for image_path in Image_list]

# Close the pool of processes
pool.close()

# Wait for all processes to finish
pool.join()

# Process the results
for result in zip(results, Image_list):
    icon_position = result.get()
    if icon_position is not None:
        # Define a threshold for cursor proximity
        proximity_threshold = 30  # Adjust as needed

        # Check if the cursor is within the proximity threshold of the icon
        if abs(x - icon_position[0]) <= proximity_threshold and abs(y - icon_position[1]) <= proximity_threshold:
            # Move the cursor to the center of the icon
            pyautogui.moveTo(icon_position[0], icon_position[1], duration=0.5)
            is_cursor_over_icon = True
        else:
            # Cursor is not over the icon, release the mouse button
            if is_cursor_over_icon:
                pyautogui.mouseUp()
                is_cursor_over_icon = False
```

```python
while True:
    x, y = pyautogui.position()

    current_time = time.time()
    time_elapsed = current_time - prev_time

    if time_elapsed > 0:
        delta_x = x - prev_x
        delta_y = y - prev_y

        speed_x = delta_x / time_elapsed
        speed_y = delta_y / time_elapsed
        newSpeed = int(math.hypot(speed_x, speed_y))
        Speeds.append(str(newSpeed))

        prev_x, prev_y = x, y
        prev_time = current_time

        if len(Speeds) >= 2 and int(Speeds[-2]) != 0:
            velocity_ratio = int(newSpeed) / int(Speeds[-2])
        else:
            velocity_ratio = 0

        # Efface la ligne précédente en imprimant des espaces pour garantir une longueur uniforme
        print('\r', end='')

        # Formatage de la ligne de sortie avec une longueur fixe
        output_line = f"Position : {x}, {y} | Vitesse : {newSpeed} pixels/secondes".ljust(80)
        print(output_line, end='')
        X=x
        Y=y

    time.sleep(0.1)
```