Mémoire de Master MIAGE M2/S3

Institut d’Administration des Entreprises – IAE NC

MIAGE

2eme année – 3eme semestre

2024/2025

Projet de DATAVIZ sur matrice LED

Raphaël Bordais

Projet effectué du 19 septembre au 21 janvier 2025

Etablissement d’accueil : OPT NC

Stage encadré par

Adrien Sales

Chef de bureau au sein de la DSI

Remerciements

Écrivez vos remerciements dans cette rubrique (professeurs, amis, famille, etc).



Figure  : Université de Nouméa

Droits d’auteurs

Mémoire de Master confidentiel.

Cette création est mise à disposition selon le Contrat :   
« **Attribution-Pas d'Utilisation Commerciale-Pas de modification 4.0 France** »  
disponible en ligne : <https://creativecommons.org/licenses/by-nc-nd/4.0/>

[](http://creativecommons.org/licenses/by-nc-nd/3.0/fr/)

Table des matières

[Remerciements 2](#_Toc184149407)

[Droits d’auteurs 3](#_Toc184149408)

[Table des illustrations 6](#_Toc184149409)

[Table des tableaux 7](#_Toc184149410)

[Introduction 8](#_Toc184149411)

[1. Présentation 9](#_Toc184149412)

[1.1. Contexte et problématique 9](#_Toc184149413)

[1.2. Objectifs du projet 9](#_Toc184149414)

[2. État de l'art 10](#_Toc184149415)

[2.1. Les technologies actuelles en datavisualisation 10](#_Toc184149416)

[2.2. Exploitation des matrices LED 10](#_Toc184149417)

[3. Conception et architecture du système 12](#_Toc184149418)

[3.1. Choix technologiques 12](#_Toc184149419)

[3.1.1. Pourquoi la Pimoroni Cosmic-Unicorn ? 12](#_Toc184149420)

[3.1.2. Langages utilisés : Python/MicroPython et APIs 12](#_Toc184149421)

[3.1.3. Caractéristiques techniques : 14](#_Toc184149422)

[3.2. Schéma d’architecture : 15](#_Toc184149423)

[3.3. Procédure d’implémentation du code 16](#_Toc184149424)

[3.4. Principe du code 18](#_Toc184149425)

[4. Développement et mise en œuvre 19](#_Toc184149426)

[4.1. Étapes de réalisation : 19](#_Toc184149427)

[4.1.1. Gestion des LEDs 19](#_Toc184149428)

[4.1.2. Implémentation des boutons interactifs 19](#_Toc184149429)

[4.1.3. Communication avec l’API et traitement des données en temps réel 20](#_Toc184149430)

[4.2. Défis rencontrés 20](#_Toc184149431)

[4.3. Principales fonctionnalités du script 21](#_Toc184149432)

[4.4. Avantages et points d'amélioration : 23](#_Toc184149433)

[5. Exploitation des résultats 25](#_Toc184149434)

[5.1. Montre des cas concrets d’utilisation : 25](#_Toc184149435)

[5.2. Retour utilisateur : 25](#_Toc184149436)

[6. Analyse critique 26](#_Toc184149437)

[6.1. Ce qui a bien fonctionné 26](#_Toc184149438)

[6.2. Points d'amélioration 26](#_Toc184149439)

[6.3. Opportunités pour des projets futurs 27](#_Toc184149440)

[7. Conclusion 28](#_Toc184149441)

[8. Annexes 30](#_Toc184149442)

[8.1. Codes source principaux. 30](#_Toc184149443)

[8.2. Documentation technique et schémas. 30](#_Toc184149444)

[8.3. Références bibliographiques 30](#_Toc184149445)

[9. Annexes 32](#_Toc184149446)

[Annexe 1. Code source 33](#_Toc184149447)

[Annexe 2. Documents techniques Pimoroni 49](#_Toc184149448)

Table des illustrations

[Figure 1 : Université de Nouméa 2](#_Toc184142973)

Table des tableaux

[Tableau 1 : Comparaison avec d’autres technologies d’affichage 10](#_Toc184142967)

Introduction

*"L’information est belle lorsqu’elle est bien présentée."* – Hans Rosling, célèbre statisticien et spécialiste de la datavisualisation.

Dans ce contexte technologique où la visualisation des données joue un rôle central dans la prise de décision et l’interaction utilisateur, la capacité à afficher efficacement des informations dans des espaces restreints représente un défi technique important. Ce mémoire s’inscrit dans cette problématique, en explorant la possibilité d’utiliser une **matrice LED 32x32**, un support compact mais limité en résolution, pour créer une solution de visualisation dynamique et fonctionnelle.

L’objectif de ce projet est de démontrer qu’un espace aussi réduit peut devenir un outil puissant de datavisualisation grâce à une conception logicielle optimisée et à une gestion judicieuse des ressources matérielles. En tirant parti du langage **MicroPython**, nous avons pu maximiser les fonctionnalités offertes par la matrice LED :

* **Affichage personnalisé et dynamique** : Création de polices de caractères adaptées et optimisation des fontes existantes pour garantir une lisibilité optimale, même dans un espace contraint.
* **Connectivité avancée** : Intégration du **Wi-Fi** et exploitation d’**APIs** pour enrichir les contenus affichés en temps réel, tels que des messages dynamiques ou des indicateurs clés.
* **Gestion des ressources matérielles** : Optimisation de la mémoire et de la puissance de calcul de la matrice pour supporter des fonctionnalités complexes telles que l’affichage de **QR Codes** interactifs.

Le développement de ce projet a été réalisé dans un cadre collaboratif grâce à l’utilisation de **GitHub**, permettant une gestion rigoureuse des versions, un suivi clair des modifications et une documentation complète. Ce choix d’environnement de travail a également favorisé une modularité accrue du code, ouvrant la voie à des évolutions futures.

Ce travail met en lumière l’alliance entre contraintes matérielles et solutions logicielles créatives, illustrant comment une infrastructure simple peut devenir un support innovant de datavisualisation. Ce mémoire détaillera les choix technologiques, les défis rencontrés, les solutions apportées et les perspectives ouvertes par cette approche, offrant une vision complète du potentiel des matrices LED dans le domaine de la visualisation de données.

# Présentation

## Contexte et problématique

Le choix de ce projet est motivé par la demande spécifique de l’OPT souhaitant afficher en agence le temps d’attente de ses clients en s’appuyant sur des données fournies par une API.

Pour répondre à ce besoin, une solution compacte, innovante et accessible a été imaginée sous la forme d’une tablette intégrant une matrice LED 32x32. Ce projet combine les enjeux de l’Internet des Objets (IoT) avec des standards minimisés de codage et des contraintes matérielles spécifiques. Il offre ainsi une opportunité unique de concevoir une solution concrète et élégante pour représenter des données en temps réel dans un environnement restreint.

## Objectifs du projet

Ce projet a pour ambition de démontrer qu’un espace restreint, comme une matrice LED 32x32, peut être transformé en un outil de datavisualisation interactif et polyvalent grâce à une gestion innovante des ressources matérielles et logicielles. Les objectifs spécifiques incluent :

* **La création d’une solution de visualisation dynamique** :

Exploiter pleinement la matrice pour afficher des **informations lisibles et attrayantes**.

Développer et intégrer des polices de caractères **optimisées pour des espaces restreints**.

* **Une connectivité et une interactivité avancées** :

Utiliser le **Wi-Fi** pour récupérer et afficher des données externes en temps réel via des **APIs**.

Implémenter des **QR Codes** interactifs permettant une interaction directe avec les utilisateurs.

* **La maximisation efficace des ressources matérielles** :

**Optimiser la mémoire** et la puissance de calcul pour garantir des performances fluides.

Explorer les **possibilités** offertes par le codage en **MicroPython** pour simplifier le développement tout en multipliant les fonctionnalités.

* **La modularité et la collaboration** :

Utiliser **GitHub** pour gérer le projet, partager le code et favoriser une documentation claire, facilitant les évolutions futures.

Ce projet vise ainsi à marier innovation et contrainte, ouvrant la voie à des applications pratiques pour des environnements connectés et minimalistes.

# État de l'art

## Les technologies actuelles en datavisualisation

La datavisualisation (DATAVIZ) repose sur la transformation de données complexes en représentations graphiques compréhensibles et attrayantes. Aujourd’hui, plusieurs technologies permettent d’afficher des informations, allant des écrans haute définition aux supports spécialisés tels que les matrices LED.

Les principales tendances en DATAVIZ incluent :

* **Écrans interactifs et tableaux de bord**

Solutions basées sur des écrans tactiles ou des logiciels dédiés (Power BI, Tableau, ou D3.js) pour offrir une expérience immersive.

* **Objets connectés (IoT)**

Utilisation de dispositifs compacts pour afficher des données en temps réel dans des environnements contraints, comme des panneaux connectés, des wearables ou des tableaux d’information.

* **Matrice LED**

Une technologie simple et modulable, permettant d’afficher des informations claires, souvent utilisée dans les objets connectés, les enseignes ou des solutions de visualisation embarquées.

## Exploitation des matrices LED

La matrice LED, comme la Pimoroni Cosmic-Unicorn, est une technologie de visualisation basée sur un réseau de diodes électroluminescentes qui émettent de la lumière pour afficher des caractères, des animations ou des images. Parmi ses avantages :

* **Comptact**

Les matrices LED sont légères, compactes et s’intègrent facilement dans des dispositifs restreints.

* **Flexible**

Elles permettent une personnalisation complète des animations et des données affichées via un contrôle direct, souvent en MicroPython ou C.

* **Coût réduit**

En comparaison avec d’autres écrans comme les LCD ou OLED, les matrices LED offrent une solution économique.

* **Applications IoT**

Elles sont souvent intégrées dans des projets IoT grâce à leur faible consommation d’énergie et leur capacité à afficher des données en temps réel.

Les produits comme la **Pimoroni Cosmic-Unicorn** se distinguent par leur **facilité d’utilisation**, avec des bibliothèques **Python optimisées**, et leur compatibilité avec des plateformes de prototypage rapide (**Raspberry Pi**).

Tableau  : Comparaison avec d’autres technologies d’affichage

|  |  |  |
| --- | --- | --- |
| Technologie | Avantages principaux | Limites principales |
| LCD | Haute résolution, couleurs vives, grand format | Consommation énergétique plus élevée, coût souvent plus important |
| OLED | Contrastes élevés, finesse des écrans | Sensibilité aux burn-in, coût plus élevé |
| Matrice LED | Compacte, économique, adaptable, faible consommation | Résolution limitée, nécessite un codage optimisé |
| E-Ink (papier électronique) | Faible consommation, lisibilité en plein jour | Vitesse d’affichage lente, coût élevé pour des écrans couleur |

La matrice LED a été retenue dans ce projet pour plusieurs raisons :

* **Contrainte d’espace**

La matrice LED 32x32 est parfaitement adaptée à un environnement restreint où un écran classique ne serait pas viable.

* **Affichage dynamique**

Elle offre une excellente personnalisation pour afficher des données simples comme des temps d’attente ou des QR Codes.

* **Coût et maintenance**

Une solution économique, durable et facile à intégrer dans des environnements IoT.

* **Optimisation des ressources**

Le faible encombrement mémoire nécessaire au contrôle de la matrice en fait un choix idéal pour des microcontrôleurs ou des systèmes embarqués.

Bien que des solutions comme les écrans LCD ou OLED puissent offrir une meilleure résolution, la **matrice LED répond parfaitement aux exigences de ce projet**, tant sur le plan de la **simplicité** **d’intégration** que sur celui de l’efficacité dans un **environnement IoT** contraint.

# Conception et architecture du système

## Choix technologiques

### Pourquoi la Pimoroni Cosmic-Unicorn ?

La Pimoroni Cosmic-Unicorn a été choisie pour ses caractéristiques uniques et adaptées aux besoins du projet :

* **Format compact et haute densité de LEDs**

Avec ses 1024 LEDs (32x32), cette matrice offre une résolution suffisante pour afficher des caractères, des animations simples et des éléments visuels comme des QR Codes, tout en restant légère et facile à intégrer dans des environnements restreints.

* **Compatibilité avec Python/MicroPython**

La Pimoroni Cosmic-Unicorn est livrée avec des bibliothèques dédiées qui facilitent son utilisation avec Python et MicroPython, deux langages idéaux pour des projets IoT nécessitant un développement rapide et une gestion optimisée des ressources matérielles.

* **Simplicité d’alimentation**

Fonctionnant avec une alimentation USB-C standard, elle s’intègre facilement dans un système portable ou stationnaire.

* **Support de fonctionnalités avancées**

Cette matrice est capable de gérer l’intensité des LEDs, permettant de régler la luminosité et de créer des effets dynamiques.

* **Communauté et documentation**

Le support actif de la communauté Pimoroni et une documentation complète permettent une prise en main rapide et un accès à des ressources pour résoudre des problématiques courantes.

### Langages utilisés : Python/MicroPython et APIs

Le choix des langages et des technologies repose sur plusieurs considérations techniques :

**Python/MicroPython**

* **Flexibilité**

Python est largement utilisé dans le domaine de l’IoT et dispose de nombreuses bibliothèques adaptées à la manipulation des matrices LED. MicroPython, sa déclinaison allégée, est particulièrement efficace pour des systèmes embarqués avec des ressources limitées.

* **Simplicité et rapidité de développement**

Python/MicroPython permet de développer et de déployer des fonctionnalités rapidement, tout en restant accessible pour des évolutions futures.

* **Gestion de la matrice**

Les bibliothèques comme cosmic\_unicorn (offerte par Pimoroni) simplifient le contrôle des LEDs, des animations et des interactions utilisateur.

**APIs**

* **Connexion aux données dynamiques**

L’utilisation d’APIs permet de récupérer les données en temps réel, comme le temps d’attente des clients, et de les afficher instantanément sur la matrice LED.

* **Format standardisé**

Les API REST, souvent au format JSON, s’intègrent naturellement avec Python, offrant une interaction fluide entre le système local et les services externes.

**Approches pour la gestion des animations (algorithmes et bibliothèques)**

Pour la gestion des animations et de l’affichage, plusieurs approches ont été adoptées pour optimiser les performances et garantir une expérience utilisateur fluide :

* **Algorithmes d’affichage optimisés**

Les données affichées sur la matrice sont pré-traitées pour minimiser l’utilisation de la mémoire embarquée. Les images ou animations sont réduites à une résolution compatible avec la matrice (32x32).

Les informations, comme les chiffres ou les messages textuels, sont animées (défilement, transitions douces) pour améliorer leur lisibilité sur un espace restreint.

* **Bibliothèques pour la matrice**

Une bibliothèque Python dédiée qui simplifie la gestion des LEDs en fournissant des fonctions prêtes à l’emploi pour dessiner des pixels, afficher du texte, gérer les animations et ajuster la luminosité.

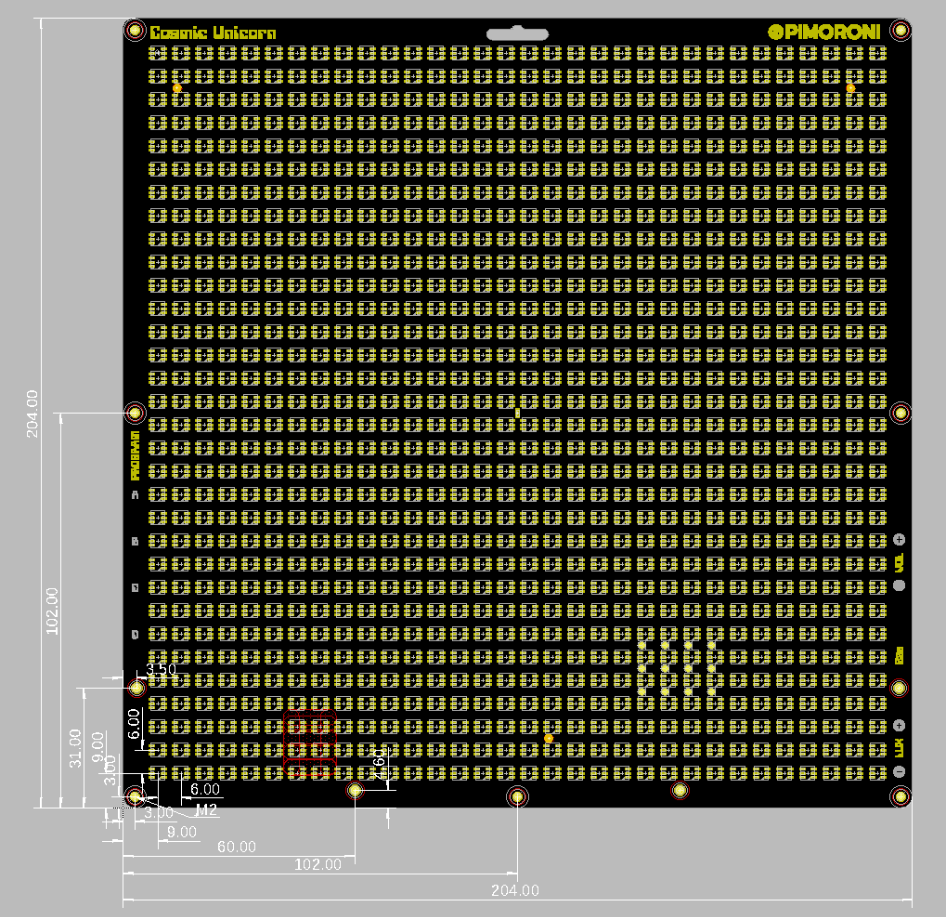
Pour les polices, des bibliothèques comme Pillow ont été utilisées pour créer ou adapter des fontes, assurant une lisibilité optimale même sur des caractères réduits.

* **Animations personnalisées**

Transitions visuelles (fade in/out, scrolling).

Affichage de QR Codes : Génération des QR Codes via des bibliothèques comme qrcode en Python, suivi d’un rendu pixel par pixel sur la matrice.

Effets dynamiques (clignotements, changements de couleur) pour attirer l’attention ou signaler un événement particulier.



### Caractéristiques techniques :

**Affichage et LEDs** :

* **1024 LEDs RGB** organisées en une grille de **32x32**.
* Contrôle individuel de la couleur et de la luminosité de chaque LED.
* LED avec diffusion intégrée pour un rendu homogène et attrayant.
* **14-bit de précision** dans le traitement de l'image, garantissant une luminosité fluide, même à bas niveau.

**Connectivité et programmation** :

Basée sur le microcontrôleur **RP2040** (Raspberry Pi Pico W intégré) :

* **Dual-core ARM Cortex M0+** (jusqu’à 133 MHz).
* **Connexion Wi-Fi 2.4 GHz** pour des applications IoT.
* Prise en charge de **MicroPython** et **C/C++**, avec des bibliothèques dédiées (PicoGraphics).
* Compatible avec des interfaces **Qwiic/STEMMA QT** pour connecter facilement des modules supplémentaires.

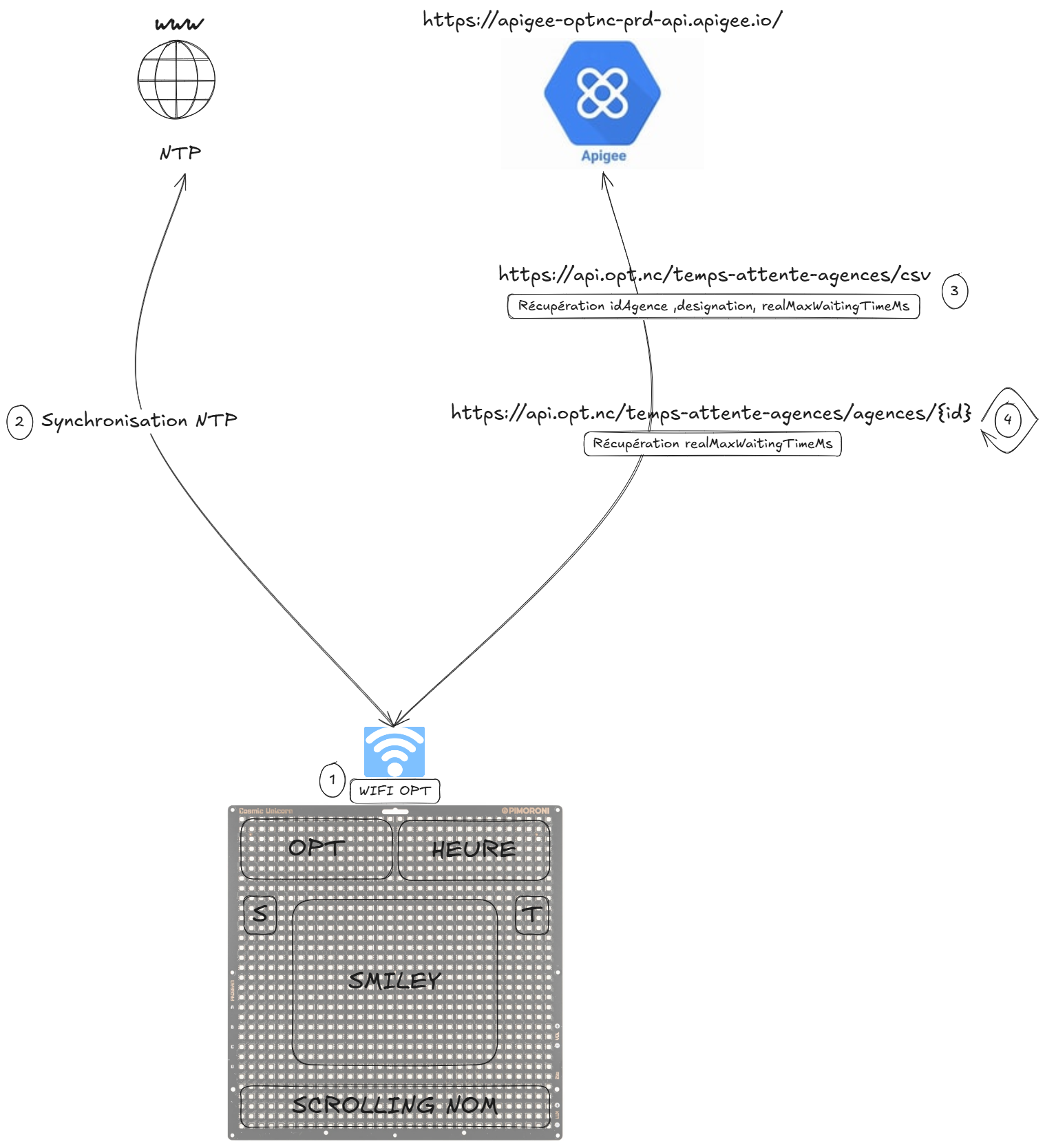
**Fonctionnalités intégrées** :

* Haut-parleur mono 1W avec amplificateur pour générer des alertes sonores.
* Capteur de lumière (phototransistor) pour ajuster la luminosité en fonction de l’environnement.
* **9 boutons tactiles** pour une interaction utilisateur directe.

**Compatibilité et accessoires** :

* Préchargée avec MicroPython et des exemples pour faciliter la prise en main.
* Possibilité d'ajouter une batterie (jusqu’à 5.5V) pour des applications mobiles.

## Schéma d’architecture :



## Procédure d’implémentation du code

[ HOW\_TO\_INSTALL.md]

**Cosmic Unicorn Project Setup**

This guide explains how to set up the Cosmic Unicorn project, including configuring WiFi, copying necessary files, and deploying the display scenes. Follow these steps to get started.

**Prerequisites**

**Hardware**: Pimoroni Cosmic-Unicorn 32x32 LED matrix with Raspberry Pi Pico W.

**Software**: Thonny IDE (required for file uploads and troubleshooting), GitHub access, WiFi credentials.

**Files Needed**:

* boot.py: The boot file to ensure the script runs automatically.
* main.py: The main Python script.
* information.env: File containing WiFi credentials and the API key.

**1. Install Thonny IDE**

Download and install Thonny IDE from [thonny.org](https://thonny.org/). Thonny is **required** for this project, but only to manage file uploads and necessary modifications (such as customizing the information.env file). Thonny can also be used to troubleshoot issues.

* **Using Thonny for the project:**
* **File Upload**: Use Thonny to upload boot.py, main.py, and information.env to the Pimoroni Cosmic-Unicorn.
* **Modifying information.env**: Customize the WiFi credentials and API key in the information.env file.
* **Troubleshooting**: Thonny is used to diagnose issues and test scripts on the LED matrice.

**2. Clone the GitHub Repository**

Clone the repository to your local machine: git clone <https://github.com/adriens/temps-attente-matrix-led.git>

Navigate to the project directory cd temps-attente-matrix-led

**3. Configure WiFi Credentials**

**Edit the .env file** to provide your **WiFi credentials** : SSID= WIFI\_PASSWORD=

**5. Configure API Key**

**Open the .env** file and add your API key in the format : API\_KEY=

**4. Copy Files to Raspberry Pi Pico**

Connect your Raspberry Pi Pico W to your computer.

Open Thonny and select the correct interpreter for Raspberry Pi Pico W. Save the following files to the Pico’s root directory:

* boot.py
* main.py
* information.env Use Thonny’s file browser to ensure these files are saved correctly on the Pico.

**6. Running the Script**

Once the files are uploaded, the Pimoroni Cosmic-Unicorn will run autonomously as soon as it is powered on. No further intervention via Thonny is required for final usage.

**7. Troubleshooting**

* **WiFi Not Connecting**: Ensure correct SSID/password in the .env file and check network availability.
* **API Key Issues**: Make sure the API key in the .env file is correct and the file is in the correct directory.
* **Button Response Not Working**: Ensure the Cosmic Unicorn device buttons are functioning correctly and properly mapped.

**8. Updating the Script**

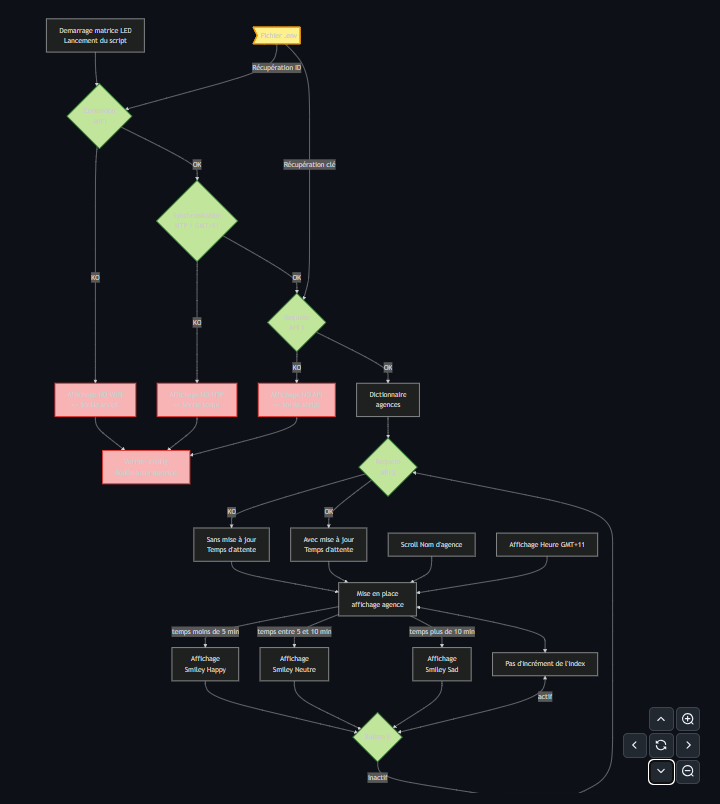
If there are any changes to the script, make sure to copy the updated version to the Raspberry Pi Pico. Use Thonny IDE to overwrite the old script on the Pico's filesystem.

**9. Volume and Brightness Adjustment**

* **Volume**: Use the buttons on the Cosmic Unicorn to adjust the sound volume during runtime.
* **Brightness**: Adjust the brightness using the appropriate buttons on the Cosmic Unicorn.
* **10. Button A and B Functionality**
* **Button A**: Toggles the sound on and off. When the sound is enabled, a beep will play to indicate activation.
* **Button B**: Pauses or resumes the main display loop. When paused, a yellow LED will indicate the pause status.
* **Additional Notes**

Feedback and Contributions: If you'd like to contribute or provide feedback on this guide, please open an issue or submit a pull request on GitHub.

## Principe du code



# Développement et mise en œuvre

## Étapes de réalisation :

### Gestion des LEDs

L’affichage des données et des animations sur la matrice LED Pimoroni Cosmic-Unicorn nécessite une gestion rigoureuse des 1024 LEDs RGB (32x32) :

* **Création et gestion des couleurs**

Un dictionnaire de couleurs pré-définies a été utilisé, chaque couleur étant définie en format RGB.

Utilisation de l’API graphique PicoGraphics pour gérer les pixels individuellement et créer des stylos graphiques associés aux couleurs.

* **Gestion des animations**

Mise en œuvre de textes défilants avec une logique de déplacement horizontal contrôlée par des intervalles temporels.

Génération de QR codes : Rendu pixel par pixel pour assurer une lisibilité maximale sur la matrice LED.

Smiley dynamique : Affichage conditionnel basé sur des seuils définis (heureux, neutre, triste) pour illustrer les temps d’attente.

Effets visuels (fade in/out, cadres dynamiques) pour améliorer l’esthétique des informations affichées.

* **Optimisation mémoire**

L’utilisation d’algorithmes pour minimiser l’utilisation des ressources matérielles, comme le pré-calcul des coordonnées de pixels des formes courantes (lettres, chiffres, smileys).

* **Pilotage des LEDs**

Les LEDs spécifiques ont été attribuées à des fonctions d’état, comme le son ou la connexion Wi-Fi, pour signaler visuellement l’état du système.

### Implémentation des boutons interactifs

Les boutons intégrés à la Cosmic-Unicorn (SWITCH\_A, SWITCH\_B, SWITCH\_C, SWITCH\_D) permettent une interaction utilisateur fluide :

* **Détection d’appui**

Les boutons sont surveillés en permanence dans la boucle principale.

Débounce logiciel pour éviter les détections multiples.

* **Actions associées**

Bouton A : Activer/désactiver le son avec un retour visuel sur les LEDs.

Bouton B : Pause/reprise de la boucle d’affichage.

Bouton C : Navigation entre différents modes d’affichage.

Bouton D : Redémarrage du système.

* **Retours utilisateurs**

Son avec fréquence ajustable pour signaler une interaction.

Mise à jour des LEDs pour indiquer l’état du système après l’action d’un bouton.

* **Ajustement des paramètres**

Contrôle dynamique de la luminosité et du volume via des boutons dédiés.

### Communication avec l’API et traitement des données en temps réel

L’intégration de l’API permet de récupérer les données de temps d’attente des agences en temps réel.

* **Connexion réseau**

Initialisation et configuration de l’interface Wi-Fi (STA\_IF) pour se connecter à un réseau spécifique défini dans un fichier .env.

Animation de chargement pour informer l’utilisateur durant les tentatives de connexion.

Gestion des reconnections automatiques en cas de déconnexion.

* **Appels API**

Utilisation de urequests pour effectuer des requêtes HTTP GET.

Passage de l’API Key dans les en-têtes pour sécuriser l’accès aux données.

* **Traitement des données**

Extraction des temps d’attente (realMaxWaitingTimeMs) à partir des réponses JSON.

Conversion des données brutes en unités compréhensibles pour l’affichage (minutes).

Gestion des erreurs API : Si une requête échoue, le système utilise les dernières données disponibles.

* **Synchronisation temporelle**

Utilisation de serveurs NTP (Network Time Protocol) pour synchroniser l’horloge interne.

Compensation des fuseaux horaires pour garantir une précision horaire.

## Défis rencontrés

Limitation de la puissance de calcul sur des microcontrôleurs

**Problème**

La Cosmic-Unicorn repose sur un microcontrôleur RP2040, limité en ressources (133 MHz, 264 KB de SRAM).

Les animations complexes et la gestion en temps réel des données API sollicitent intensivement le processeur.

**Solution**

Utilisation de bibliothèques optimisées comme PicoGraphics pour gérer les animations et les pixels.

Pré-calcul des données répétitives (ex. : coordonnées de texte ou smileys) pour minimiser les calculs dynamiques.

Libération manuelle de la mémoire à l’aide de gc.collect() après les appels API pour éviter les dépassements mémoire.

**Optimisation des appels API** :

Réduction de la fréquence des requêtes réseau pour limiter l’utilisation des ressources

Optimisation des animations pour garder des performances fluides

**Problème**

L’affichage des animations (texte défilant, QR codes, smileys) doit rester fluide, même avec des calculs et des mises à jour en temps réel.

**Solution**

Fragmentation des tâches : Diviser les animations en étapes simples et les exécuter dans des cycles indépendants.

Optimisation des timings : Ajustement des délais pour assurer un équilibre entre fluidité visuelle et charge processeur.

Taux de rafraîchissement élevé : Maintien d’un taux stable (300 FPS) pour éviter les scintillements ou les saccades.

**Tests en conditions réelles**

Les animations ont été testées à différentes luminosités et distances d’affichage pour garantir la lisibilité.

Ajustements des couleurs et des tailles de texte en fonction des retours utilisateurs.

## Principales fonctionnalités du script

**Affichage dynamique**

* Gestion d'animations (défilement de texte, smiley, cadres).
* Génération de QR codes à afficher sur la matrice LED.
* Gestion des polices personnalisées et préchargées, adaptées à la grille LED.

**Interaction utilisateur**

* Utilisation des boutons physiques pour passer d’un écran à un autre, activer/désactiver le son, ajuster la luminosité et le volume.

**LEDs** indicatrices pour signaler l'état du Wi-Fi, du son, ou d’autres alertes.

**Connectivité IoT**

* Connexion au Wi-Fi pour récupérer des données via une API REST.
* Synchronisation de l'heure avec un serveur NTP.
* Gestion des informations réseau à partir d’un fichier de configuration (information.env).

**Gestion des données API**

* Mise à jour en temps réel des temps d’attente des agences.
* Gestion des erreurs d’accès à l’API (fallback sur les données précédentes).

**Modularité des modes d’affichage**

* Écran d'accueil : Texte défilant ("UNC OPT"), animation graphique (cœur).
* Informations système : Statuts du Wi-Fi, clé API, fichier .env.
* Légendes explicatives : Description des différents indicateurs (exemple : LEDs indiquant une perte Wi-Fi).
* Temps d’attente des agences : Visualisation dynamique avec smiley basé sur des seuils de temps.
* QR Code : Génération et affichage d’un QR code statique.

**Programmation et bibliothèques**

* MicroPython : Simplifie l’interaction avec le matériel embarqué.
* Bibliothèques Pimoroni cosmic : Gestion des LEDs, boutons, et son.
* Bibliothèques Pimoroni PicoGraphics : Gestion graphique pour le texte et les animations.
* Modules réseau network : Gestion des connexions Wi-Fi.
* Modules réseau urequests : Requêtes HTTP pour récupérer des données API.
* Modules réseau ntptime : Synchronisation de l’heure via NTP.

**Gestion matérielle**

* Contrôle des LEDs avec des algorithmes d’affichage optimisés pour minimiser la consommation mémoire.
* Utilisation du haut-parleur intégré pour des alertes sonores.
* Exploitation des boutons tactiles pour une interaction utilisateur fluide.

Approche algorithmique

* Animations progressives : Texte défilant, smiley évolutif, QR code dynamique.
* Logique conditionnelle : Affichage basé sur l’état du Wi-Fi ou la disponibilité des données.
* Gestion des erreurs : Récupération après échec d’une requête API ou perte de connexion Wi-Fi.

**Éléments personnalisés notables**

* Gestion des smileys :

Les smileys varient selon le temps d’attente des agences :

Happy : Temps d’attente inférieur à 5 minutes.

Neutral : Entre 5 et 10 minutes.

Sad : Plus de 10 minutes.

Les LEDs de la matrice forment le visage, les yeux et la bouche avec des couleurs spécifiques.

* Affichage du QR code :

Génération d’un QR code pour un lien spécifique (ex. bit.ly).

Adaptation automatique à la luminosité définie par l’utilisateur.

* Interactivité avancée :

Les boutons permettent de changer les modes d’affichage ou d’interrompre une animation.

Les LEDs signalent l’état du système (son activé, Wi-Fi connecté).

Écran de chargement :

Animation de progression visuelle lors de l'initialisation du Wi-Fi et du chargement des fichiers.

## Avantages et points d'amélioration

**Avantages**

* Simplicité et modularité : Les fonctionnalités sont bien structurées, ce qui facilite les ajouts futurs.
* Interactivité : Les boutons et les LEDs améliorent l’expérience utilisateur.
* Efficacité : Gestion optimisée des ressources de la matrice LED pour des performances fluides.

**Points d'amélioration**

* Ajouter une gestion avancée des erreurs pour les échecs réseau prolongés.
* Optimiser la consommation mémoire en réduisant les allocations dynamiques fréquentes.
* Permettre la personnalisation des animations ou des messages via une interface utilisateur externe.

Ce script démontre la polyvalence et les **capacités de la Pimoroni Cosmic Unicorn** dans des **applications IoT interactives**. Grâce à une **gestion graphique avancée**, une **connectivité réseau robuste**, et une **interaction utilisateur fluide**, il offre une solution efficace pour afficher des informations en temps réel dans un format visuel compact et attrayant.

# Exploitation des résultats

## Montre des cas concrets d’utilisation :

* La récupération de l’API
* Les différents screens
* QR codes interactifs.

Visualisation d’indicateurs (temps d’attente, messages dynamiques).

## Retour utilisateur :

Retour test avec l’OPT, parle de leur feedback.

# Analyse critique

## Ce qui a bien fonctionné

Réponse au besoin initiale

Le système développé répond parfaitement aux attentes en offrant une visualisation claire et dynamique des temps d’attente des agences. La matrice Pimoroni Cosmic-Unicorn est exploitée de manière optimale pour afficher des informations textuelles, des animations, et des indicateurs visuels.

Gestion avancée de la matrice

Les fonctionnalités intégrées, telles que les smileys dynamiques et les animations défilantes, démontrent une utilisation efficace de la matrice 32x32.

L’affichage des QR codes et des cadres visuels offre des moyens de communication innovants et accessibles.

Connectivité et mise à jour en temps réel

La connexion Wi-Fi et l’intégration avec une API REST permettent de récupérer des données actualisées en continu.

La synchronisation NTP garantit l’exactitude des données temporelles affichées.

Interaction utilisateur fluide

Les boutons intégrés permettent un contrôle intuitif du système (changement d’écran, gestion du son, ajustement de la luminosité).

Les LEDs supplémentaires offrent un retour visuel immédiat sur l’état du système (connexion Wi-Fi, son activé/désactivé).

## Points d'amélioration

La **gestion des échecs de connexion Wi-Fi ou d’appel à l’API** pourrait être améliorée en intégrant :

* Des messages utilisateur plus détaillés en cas de problème.
* Une sauvegarde locale temporaire des données pour éviter tout impact visible lors d’une défaillance réseau.

Le script pourrait être **restructuré pour améliorer ses performances** et réduire la consommation mémoire :

* Modularisation plus poussée en divisant les écrans ou les fonctionnalités dans des fichiers spécifiques.
* Réduction de la complexité algorithmique pour des tâches répétitives, comme les animations ou les mises à jour des LEDs.
* Libération proactive des ressources inutilisées pour éviter les ralentissements.

Intégrer un **système de traduction** permettant d’afficher les informations dans plusieurs langues.

Évolutivité et documentation

* Ajouter une documentation détaillée pour faciliter **l’intégration** et la **maintenance** du projet par **d’autres développeurs**.
* Prévoir des **interfaces** pour ajouter facilement de **nouvelles fonctionnalités** ou **intégrer d’autres matrices**.

## Opportunités pour des projets futurs

**Intégration d’autres matrices LED**

Étendre le projet pour supporter d’autres matrices ou modèles de matrices (ex. : Pimoroni Galactic ou Stellar Unicorn).

Mise en œuvre d’un affichage synchronisé sur plusieurs matrices pour des installations plus complexes.

**Gestion centralisée des matrices**

Développer une solution IoT où plusieurs matrices sont gérées via un serveur centralisé ou une application mobile/web.

Offrir des outils de configuration à distance pour modifier les messages ou les animations sans interaction physique.

**Nouvelles fonctionnalités interactives**

Ajouter une interface utilisateur (via une application ou un panneau de contrôle) pour permettre aux administrateurs de personnaliser les données ou l’apparence des écrans.

Intégrer des capteurs externes (température, luminosité) pour enrichir l’affichage avec des données contextuelles.

**Amélioration esthétique**

Concevoir des boîtiers ou des diffuseurs personnalisés pour améliorer l’aspect visuel et protéger la matrice dans des environnements professionnels.

# Conclusion

Le projet a atteint son objectif principal :

**Fournir une solution innovante de datavisualisation en temps réel sur une matrice LED compacte et performante.**

Grâce à l’exploitation avancée de la Pimoroni Cosmic-Unicorn, nous avons pu afficher des informations dynamiques, telles que les temps d’attente des agences, sous forme visuelle et interactive.

L’intégration des fonctionnalités IoT, comme la connexion Wi-Fi et la récupération des données via une API, a permis de répondre aux exigences d’un affichage en continu, actualisé en temps réel.

**Réussites**

La solution est fonctionnelle et répond aux besoins exprimés, tant sur le plan technique qu’esthétique.

L’utilisation d’animations fluides, de QR codes, et de smileys dynamiques rend l’information accessible et engageante.

L’interactivité via les boutons et les LEDs assure une expérience utilisateur intuitive.

L’utilisation de GitHub a joué un rôle essentiel dans le déroulement du projet. Cet outil collaboratif a permis :

Une gestion centralisée et sécurisée du code source.

Une collaboration efficace, notamment pour le partage des évolutions du script et des corrections.

La mise à disposition publique de la solution, facilitant son accès et son adoption par d’autres utilisateurs ou développeurs.

**Leçons apprises**

Les limitations matérielles des microcontrôleurs nécessitent une gestion optimisée des ressources, tant en mémoire qu’en calcul.

Une modularisation accrue et une meilleure gestion des erreurs pourraient améliorer la maintenabilité et la résilience du système.

**Perspectives**

Ce projet peut servir de base pour développer des solutions similaires dans d’autres contextes, comme les affichages synchronisés multi-matrices ou les tableaux de bord IoT centralisés.

Des améliorations telles que la gestion multi-langues, le support de matrices additionnelles, ou l’intégration d’autres types de données enrichiraient encore la solution.

**Ce projet illustre le potentiel des matrices LED dans des environnements IoT, combinant simplicité d’utilisation, efficacité visuelle, connectivité avancée et collaboration facilitée par GitHub. Les bases solides établies ici ouvrent la voie à des développements futurs prometteurs, renforçant l’idée que même avec des ressources matérielles limitées, il est possible de concevoir des solutions percutantes et novatrices.**

# Annexes

## Codes source principaux.

[ <https://github.com/adriens/temps-attente-matrix-led> ]

## Documentation technique et schémas.

## Références bibliographiques

***Références en système Auteur-Date, bibliographie alphabétique (style Zotero APA) :***

Pimoroni Cosmic Unicorn :

Pimoroni Ltd. (n.d.). Cosmic Unicorn – Programmable LED matrix with Pico W onboard. Pimoroni.

Récupéré sur https://shop.pimoroni.com/products/cosmic-unicorn

MicroPython Documentation :

MicroPython. (n.d.). MicroPython Documentation. MicroPython Project.

Récupéré sur https://docs.micropython.org/en/latest/

GitHub Best Practices:

GitHub. (n.d.). Best practices for repositories. GitHub Documentation.

Récupéré sur https://docs.github.com/en/repositories/creating-and-managing-repositories/best-practices-for-repositories

Raspberry Pi Pico SDK:

Raspberry Pi Foundation. (n.d.). Raspberry Pi Pico C/C++ SDK. Raspberry Pi Documentation.

Récupéré sur https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf

GitHub pour la collaboration:

Dabbish, L., Stuart, C., Tsay, J., & Herbsleb, J. (2012). Social coding in GitHub: Transparency and collaboration in an open software repository. Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work, 1277–1286.

Récupéré sur https://doi.org/10.1145/2145204.2145396

Animations avec MicroPython:

Pimoroni Ltd. (n.d.). Picographics – High-level graphics API for Pico Unicorn. Pimoroni GitHub Repository.

Récupéré sur https://github.com/pimoroni/pimoroni-pico/tree/main/micropython/examples/picographics

# Annexes

[Annexe 1. Code source 28](#_Toc184148584)

[Annexe 2. Documents techniques Pimoroni 29](#_Toc184148585)

* 1. Code source

Main.py

# Imports nécessaires pour les fonctionnalités utilisées dans le script.

import time # Gestion du temps et des délais.

import network # Module pour gérer la connexion réseau (Wi-Fi).

import ntptime # Synchronisation du temps via NTP (Network Time Protocol).

import urequests as requests # Pour effectuer des requêtes HTTP (comme des appels API).

import os # Gestion des fichiers et des chemins.

import gc # Gestion de la mémoire (garbage collector).

import \_thread # Gestion des threads (exécution parallèle).

import machine # Pour interagir avec le matériel (comme les boutons, les LEDs).

from cosmic import CosmicUnicorn # Import du module CosmicUnicorn pour gérer l'affichage sur l'appareil.

from picographics import PicoGraphics, DISPLAY\_COSMIC\_UNICORN # Gestion des graphiques pour l'affichage.

# Initialiser attempts pour le suivi des tentatives de connexion WiFi

attempts = 0

# Constantes pour définir les couleurs utilisées dans l'affichage, définies en RGB.

COLORS = {

'YELLOW': (251, 189, 8),

'WHITE': (255, 255, 255),

'BLUE': (40, 44, 131),

'GREEN': (0, 255, 0),

'BLACK': (0, 0, 0),

'RED': (255, 0, 0),

'PINK': (255, 105, 180),

'GREEN\_SMILEY': (34, 177, 76),

'YELLOW\_SMILEY': (255, 242, 0),

'RED\_SMILEY': (237, 28, 36)

}

# Classe pour gérer l'affichage sur l'écran du Cosmic Unicorn.

class CosmicUnicornDisplay:

def \_\_init\_\_(self):

"""Initialise l'affichage, les stylos, le statut du son, la luminosité et le volume."""

self.cu = CosmicUnicorn() # Instance de CosmicUnicorn pour gérer l'affichage.

self.graphics = PicoGraphics(display=DISPLAY\_COSMIC\_UNICORN) # Instance pour gérer les graphiques.

self.width, self.height = self.graphics.get\_bounds() # Récupère les dimensions de l'écran.

self.pens = {color: self.graphics.create\_pen(\*rgb) for color, rgb in COLORS.items()} # Crée des stylos pour les couleurs.

self.scroll\_shift = 0 # Variable de décalage pour le texte défilant.

self.last\_scroll\_time = time.ticks\_ms() # Enregistre le dernier moment où le texte a défilé.

self.transition\_var = '' # Variable pour stocker le texte défilant.

self.graphics.set\_font("bitmap5") # Définit la police utilisée pour l'affichage du texte.

self.sound\_enabled = True # Indique si le son est activé ou non.

self.brightness = 0.5 # Définit la luminosité initiale de l'affichage.

self.loop\_paused = False # Variable pour gérer la pause de la boucle d'affichage.

self.volume = 500 # Fréquence initiale du bip sonore.

self.pause\_led\_position = (1, 25) # Position de la LED indiquant une pause.

self.led\_positions\_on = [(2, 9), (1, 10), (2, 10), (1, 11), (2, 11), (2, 12)] # Positions des LEDs quand le son est activé.

self.led\_positions\_off\_red = [(2, 9), (1, 10), (2, 10), (1, 11), (2, 11), (2, 12)] # Positions des LEDs rouges quand le son est désactivé.

self.led\_positions\_off\_blue = [(2, 9), (1, 10), (2, 10), (1, 11), (2, 11), (2, 12)] # Positions des LEDs bleues quand le son est désactivé.

self.led\_positions\_wifi\_ko = [(0, 17), (1, 16), (1, 17), (1, 18), (2, 15), (2, 16), (2, 17), (2, 18), (2, 19)]

self.channel = self.cu.synth\_channel(5) # Canal sonore pour gérer les bips sonores.

self.cu.set\_brightness(self.brightness) # Définit la luminosité initiale de l'écran.

self.display\_mode = 0 # Variable pour suivre le mode d'affichage

self.update\_led\_sound\_status(self.sound\_enabled) # Met à jour les LEDs selon l'état du son.

print("Affichage initialisé avec succès") # Confirmation de l'initialisation réussie.

def clear(self):

"""Efface l'écran sans toucher aux LEDs du son et de pause."""

self.graphics.set\_pen(self.pens['BLACK']) # Définit la couleur du stylo à noir pour effacer.

self.graphics.clear() # Efface l'écran.

self.update\_led\_sound\_status(self.sound\_enabled) # Met à jour les LEDs du son.

self.update\_led\_wifi\_status(self.check\_wifi\_status(network.WLAN(network.STA\_IF))) # Maintient l'état des LEDs WiFi

if self.loop\_paused: # Si la boucle est en pause, affiche la LED de pause.

self.set\_pen('YELLOW')

self.graphics.pixel(\*self.pause\_led\_position)

self.update() # Met à jour l'affichage.

def update(self):

"""Met à jour l'affichage."""

self.cu.update(self.graphics) # Rafraîchit l'écran avec les nouvelles informations graphiques.

def set\_pen(self, color):

"""Définit la couleur du stylo graphique."""

if color in self.pens:

self.graphics.set\_pen(self.pens[color]) # Définit le stylo à la couleur souhaitée.

else:

print(f"Erreur : La couleur {color} n'est pas définie.")

def scroll\_text(self, message):

"""Gère le défilement du texte sur l'écran."""

PADDING = 5 # Espace entre le texte et les bords de l'écran.

STEP\_TIME = 0.1 # Intervalle de temps entre chaque étape du défilement.

msg\_width = self.graphics.measure\_text(message, 1) # Mesure la largeur du texte.

time\_ms = time.ticks\_ms() # Récupère le temps actuel en millisecondes.

# Si assez de temps s'est écoulé depuis la dernière étape du défilement.

if time\_ms - self.last\_scroll\_time > STEP\_TIME \* 1000:

self.scroll\_shift += 1 # Décale le texte vers la gauche.

if self.scroll\_shift >= msg\_width + self.width + PADDING: # Si le texte est entièrement défilé.

self.scroll\_shift = -self.width # Réinitialise le décalage.

self.last\_scroll\_time = time\_ms # Met à jour le dernier temps de défilement.

# Efface la zone de texte.

self.set\_pen('BLACK')

self.graphics.rectangle(0, 26, self.width, 6) # Crée une zone de rectangle noire pour le texte.

self.set\_pen('WHITE') # Définit le stylo à blanc pour le texte.

self.graphics.text(message, PADDING - self.scroll\_shift, 26, -1, 1) # Affiche le texte défilant.

self.update() # Met à jour l'écran.

def draw\_frame(self, y\_start, y\_end, color):

"""Dessine un cadre autour du smiley."""

self.set\_pen(color) # Définit le stylo à la couleur donnée.

for x in range(0, self.width): # Dessine les lignes horizontales en haut et en bas.

self.graphics.pixel(x, y\_start)

self.graphics.pixel(x, y\_end)

for y in range(y\_start, y\_end + 1): # Dessine les lignes verticales sur les côtés.

self.graphics.pixel(0, y)

self.graphics.pixel(self.width - 1, y)

self.update() # Met à jour l'affichage.

def draw\_text\_opt(self):

"""Affiche le texte OPT NC sur la partie gauche de l'écran."""

self.set\_pen('BLUE') # Définit le stylo à bleu.

# Coordonnées des lettres O, P, et T pour former 'OPT'.

o\_coords = [(1, 1), (1, 2), (1, 3), (1, 4), (1, 5), (2, 1), (2, 5), (3, 1), (3, 5), (4, 1), (4, 2), (4, 3), (4, 4), (4, 5)]

p\_coords = [(6, 1), (6, 2), (6, 3), (6, 4), (6, 5), (7, 1), (7, 3), (8, 1), (8, 2), (8, 3)]

t\_coords = [(10, 1), (11, 1), (11, 2), (11, 3), (11, 4), (11, 5), (12, 1)]

# Dessine chaque lettre en utilisant les coordonnées définies.

for coords in [o\_coords, p\_coords, t\_coords]:

for x, y in coords:

self.graphics.pixel(x, y)

self.update() # Met à jour l'affichage.

def draw\_smiley(self, mood):

"""Dessine un smiley en fonction de l'humeur (happy, neutral, sad) sans effacer les LEDs du son."""

# Efface seulement la zone du smiley.

self.set\_pen('BLACK')

self.graphics.rectangle(7, 7, 19, 19) # Efface la zone où le smiley sera dessiné.

# Coordonnées du smiley, des yeux, de la bouche et de la valeur d'attente en fonction de l'humeur (happy, neutral, sad).

smiley\_coords = [

(13, 8), (14, 8), (15, 8), (16, 8), (17, 8), (18, 8),

(11, 9), (12, 9), (13, 9), (18, 9), (19, 9), (20, 9),

(10, 10), (11, 10), (20, 10), (21, 10),

(9, 11), (10, 11), (21, 11), (22, 11),

(8, 12), (9, 12), (22, 12), (23, 12),

(8, 13), (23, 13),

(7, 14), (8, 14), (23, 14), (24, 14),

(7, 15), (24, 15),

(7, 16), (24, 16),

(7, 17), (24, 17),

(7, 18), (24, 18),

(7, 19), (8, 19), (23, 19), (24, 19),

(8, 20), (23, 20),

(8, 21), (9, 21), (22, 21), (23, 21),

(9, 22), (10, 22), (21, 22), (22, 22),

(10, 23), (11, 23), (20, 23), (21, 23),

(11, 24), (12, 24), (13, 24), (18, 24), (19, 24), (20, 24),

(13, 25), (14, 25), (15, 25), (16, 25), (17, 25), (18, 25),

]

eyes\_coords = [

(11, 14), (12, 14), (13, 14), (18, 14), (19, 14), (20, 14),

(11, 15), (12, 15), (13, 15), (18, 15), (19, 15), (20, 15)

]

mouth\_coords = {

'happy': [(12, 19), (13, 19), (14, 19), (15, 19), (16, 19), (17, 19), (18, 19), (19, 19), (13, 20), (14, 20), (15, 20), (16, 20), (17, 20), (18, 20)],

'neutral': [(13, 20), (14, 20), (15, 20), (16, 20), (17, 20), (18, 20)],

'sad': [(13, 19), (14, 19), (15, 19), (16, 19), (17, 19), (18, 19), (12, 20), (13, 20), (14, 20), (15, 20), (16, 20), (17, 20), (18, 20), (19, 20)]

}

time\_coords = {

'happy': [(30, 8), (29, 8), (29, 9), (29, 10), (30, 10), (30, 11), (30, 12), (29, 12), (27, 9), (26, 10), (27, 11)], # LED pour <5

'neutral': [(27, 8), (27, 9), (27, 10), (27, 11), (27, 12), (29, 8), (29, 9), (29, 10), (29, 11), (29, 12), (30, 8), (30, 12), (31, 8), (31, 9), (31, 10), (31, 11), (31, 12), (25, 9), (24, 10), (25, 11)], # LED pour <10

'sad': [(27, 8), (27, 9), (27, 10), (27, 11), (27, 12), (29, 8), (29, 9), (29, 10), (29, 11), (29, 12), (30, 8), (30, 12), (31, 8), (31, 9), (31, 10), (31, 11), (31, 12), (24, 9), (25, 10), (24, 11)], # LED pour >10

}

mood\_color = {

'happy': 'GREEN\_SMILEY',

'neutral': 'YELLOW\_SMILEY',

'sad': 'RED\_SMILEY'

}

# Dessine le smiley.

self.set\_pen(mood\_color[mood])

for x, y in smiley\_coords:

self.graphics.pixel(x, y)

for x, y in eyes\_coords:

self.graphics.pixel(x, y)

for x, y in mouth\_coords[mood]:

self.graphics.pixel(x, y)

for x, y in time\_coords[mood]:

self.graphics.pixel(x, y)

self.update() # Met à jour l'affichage

# Ajouter la logique pour les bips

if mood == 'neutral': # 1 bip si humeur est neutre

self.play\_bip(self.volume) # Joue un bip avec la fréquence actuelle

elif mood == 'sad': # 3 bips si humeur est triste

for \_ in range(3):

self.play\_bip(self.volume) # Joue un bip avec la fréquence actuelle

time.sleep(0.3) # Pause entre les bips

def play\_bip(self, frequency):

"""Joue un bip sonore d'une fréquence donnée si le son est activé."""

try:

if self.sound\_enabled: # Si le son est activé.

self.channel.play\_tone(frequency, 0.3) # Joue une tonalité pendant 0.3 seconde.

self.cu.play\_synth() # Joue le son sur le canal synthétique.

time.sleep(0.3) # Attend que le son soit joué.

self.channel.trigger\_release() # Arrête le son.

except Exception as e:

print(f"Erreur lors de la lecture du bip : {e}") # Capture toute erreur et l'affiche.

def adjust\_brightness(self):

"""Ajuste la luminosité en fonction des boutons de luminosité, avec confirmation de détection."""

if self.cu.is\_pressed(CosmicUnicorn.SWITCH\_BRIGHTNESS\_UP): # Si le bouton pour augmenter la luminosité est pressé

if self.brightness < 1.0: # Limite supérieure pour la luminosité

self.brightness = min(self.brightness + 0.1, 1.0) # Augmente la luminosité par paliers

print(f"Luminosité augmentée à : {self.brightness}") # Message de débogage

elif self.cu.is\_pressed(CosmicUnicorn.SWITCH\_BRIGHTNESS\_DOWN): # Si le bouton pour diminuer la luminosité est pressé

if self.brightness > 0.0: # Limite inférieure pour la luminosité

self.brightness = max(self.brightness - 0.1, 0.0) # Diminue la luminosité par paliers

print(f"Luminosité diminuée à : {self.brightness}") # Message de débogage

self.cu.set\_brightness(self.brightness) # Applique la nouvelle luminosité

def adjust\_volume(self):

"""Ajuste le volume en fonction des boutons de volume."""

if self.cu.is\_pressed(CosmicUnicorn.SWITCH\_VOLUME\_UP): # Si le bouton pour augmenter le volume est pressé.

if self.volume < 20000: # Limite supérieure pour la fréquence sonore.

self.volume = min(self.volume + 10, 20000) # Augmente la fréquence (volume).

self.channel.frequency(self.volume) # Applique la nouvelle fréquence au canal sonore.

print(f"Augmentation du volume. Fréquence actuelle : {self.volume} Hz")

elif self.cu.is\_pressed(CosmicUnicorn.SWITCH\_VOLUME\_DOWN): # Si le bouton pour diminuer le volume est pressé.

if self.volume > 10: # Limite inférieure pour la fréquence sonore.

self.volume = max(self.volume - 10, 10) # Diminue la fréquence (volume).

self.channel.frequency(self.volume) # Applique la nouvelle fréquence au canal sonore.

print(f"Diminution du volume. Fréquence actuelle : {self.volume} Hz")

# Fonction pour afficher l'heure sous forme de chiffres à l'écran.

def display\_digit(self, digit, col\_start, row\_start, color):

"""Affiche un chiffre à une position donnée sur l'écran."""

digits = {

'0': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 4), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],

'1': [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4)],

'2': [(0, 0), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (1, 4), (2, 0), (2, 1), (2, 2), (2, 4)],

'3': [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (1, 4), (0, 4), (1, 2), (0, 2)],

'4': [(0, 0), (0, 1), (0, 2), (1, 2), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],

'5': [(0, 0), (1, 0), (2, 0), (0, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (1, 4), (0, 4)],

'6': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (2, 3), (2, 2), (1, 2)],

'7': [(0, 0), (1, 0), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (1, 2)],

'8': [(0, 0), (1, 0), (2, 0), (0, 1), (2, 1), (0, 2), (1, 2), (2, 2), (0, 3), (2, 3), (0, 4), (1, 4), (2, 4)],

'9': [(0, 0), (1, 0), (2, 0), (0, 1), (2, 1), (0, 2), (1, 2), (2, 2), (2, 3), (2, 4), (1, 4), (0, 4)]

}

self.set\_pen(color) # Définit le stylo à la couleur donnée.

for dx, dy in digits[digit]: # Parcourt les coordonnées du chiffre et les affiche.

self.graphics.pixel(col\_start + dx, row\_start + dy)

self.update() # Met à jour l'affichage.

# Fonction pour afficher l'horloge sur l'écran.

def display\_clock(self, start\_time, synced):

"""Affiche l'heure actuelle synchronisée ou calculée avec correction de fuseau horaire."""

current\_time = time.localtime(time.time() + 11 \* 3600 if synced else start\_time + 11 \* 3600)

hour = "{:02}".format(current\_time[3]) # Récupère l'heure actuelle (HH).

minute = "{:02}".format(current\_time[4]) # Récupère les minutes actuelles (MM).

second = current\_time[5] # Récupère les secondes actuelles (SS).

self.display\_digit(hour[0], 14, 1, 'WHITE') # Affiche le premier chiffre des heures.

self.display\_digit(hour[1], 18, 1, 'WHITE') # Affiche le deuxième chiffre des heures.

if second % 2 == 0: # Si les secondes sont paires, affiche les deux points de séparation.

self.graphics.pixel(22, 2)

self.graphics.pixel(22, 4)

else: # Sinon, les efface.

self.set\_pen('BLACK')

self.graphics.pixel(22, 2)

self.graphics.pixel(22, 4)

self.display\_digit(minute[0], 24, 1, 'WHITE') # Affiche le premier chiffre des minutes.

self.display\_digit(minute[1], 28, 1, 'WHITE') # Affiche le deuxième chiffre des minutes.

self.update() # Met à jour l'affichage.

def set\_transition\_variable(self, name):

"""Définit le texte à faire défiler."""

self.transition\_var = name # Définit la variable de transition avec le texte à afficher.

self.scroll\_shift = 0 # Réinitialise le décalage du texte.

def display\_message\_frame\_2(self, message):

"""Affiche un message au centre de l'écran."""

PADDING = 2 # Espacement pour centrer le texte.

self.set\_pen('BLACK') # Efface la zone centrale.

self.graphics.rectangle(0, 12, self.width, 12)

self.set\_pen('WHITE') # Définit le stylo à blanc.

lines = message.split('\n') # Sépare le message en plusieurs lignes si nécessaire.

y\_offset = 12 # Départ de l'affichage.

for line in lines: # Pour chaque ligne du message.

text\_width = self.graphics.measure\_text(line, 1) # Mesure la longueur du texte.

self.graphics.text(line, (self.width - text\_width) // 2, y\_offset, -1, 1) # Centre le texte.

y\_offset += 8 # Passe à la ligne suivante.

self.update() # Met à jour l'affichage.

# Fonction pour activer ou désactiver le son et mettre à jour les LEDs correspondantes.

def toggle\_sound(self):

"""Active ou désactive le son et met à jour les LEDs en conséquence."""

self.sound\_enabled = not self.sound\_enabled # Inverse l'état du son.

if self.sound\_enabled:

print("Bouton A pressé - Activation du son")

self.play\_bip(500) # Joue un bip à une certaine fréquence.

self.update\_led\_sound\_status(True) # Met à jour les LEDs pour indiquer que le son est activé.

else:

print("Bouton A pressé - Désactivation du son")

self.play\_bip(400) # Joue un bip différent pour indiquer la désactivation du son.

self.update\_led\_sound\_status(False) # Met à jour les LEDs pour indiquer que le son est désactivé.

# Fonction pour mettre à jour les LEDs en fonction de l'état du son (activé ou désactivé).

def update\_led\_sound\_status(self, sound\_status=None):

"""Met à jour l'état des LEDs indépendamment de l'affichage du smiley, uniquement pendant l'affichage des agences."""

if sound\_status is None:

sound\_status = self.sound\_enabled

if self.display\_mode == 3: # Mettre à jour les LEDs uniquement dans le mode agence (mode 3)

if sound\_status:

self.set\_pen('BLUE')

for x, y in self.led\_positions\_on:

self.graphics.pixel(x, y)

else:

self.set\_pen('BLUE')

for x, y in self.led\_positions\_off\_blue:

self.graphics.pixel(x, y)

self.set\_pen('RED')

for x, y in self.led\_positions\_off\_red:

self.graphics.pixel(x, y)

self.update()

# Fonction pour mettre en pause ou reprendre la boucle d'affichage des agences.

def toggle\_loop\_pause(self):

"""Mets en pause/reprend la boucle d'affichage des agences et gère l'état de la LED."""

self.loop\_paused = not self.loop\_paused # Inverse l'état de la pause.

if self.loop\_paused: # Si la boucle est en pause.

print("Bouton B pressé - Mise en pause de la boucle")

self.set\_pen('YELLOW') # Allume la LED de pause.

self.graphics.pixel(\*self.pause\_led\_position)

self.update()

else: # Si la boucle reprend.

print("Bouton B pressé - Reprise de la boucle")

self.set\_pen('BLACK') # Éteint la LED de pause.

self.graphics.pixel(\*self.pause\_led\_position)

self.update()

def update\_led\_wifi\_status(self, wifi\_status):

"""Met à jour l'état des LEDs en fonction de l'état du WiFi."""

if wifi\_status: # Si le WiFi est connecté, éteindre les LEDs rouges.

self.set\_pen('BLACK')

for x, y in self.led\_positions\_wifi\_ko:

self.graphics.pixel(x, y)

else: # Si le WiFi est déconnecté, allumer les LEDs rouges et les maintenir allumées.

self.set\_pen('RED')

for x, y in self.led\_positions\_wifi\_ko:

self.graphics.pixel(x, y)

self.update() # Met à jour l'affichage pour appliquer les changements

def check\_wifi\_status(self, wlan):

"""Vérifie l'état de la connexion WiFi, met à jour l'affichage LED et arrête le script si nécessaire."""

global attempts

if wlan.isconnected():

print("WIFI OK")

attempts = 0 # Réinitialiser le compteur d'échecs

return True

else:

print("WIFI KO")

attempts += 1 # Incrémenter la variable d'échecs

# Émission d'un bip pour signaler la perte de connexion

self.play\_bip(1000) # Bip sonore pour signaler l'échec de connexion

# Si la variable attempts dépasse 5, déclencher l'arrêt du script

if attempts > 10:

stop\_script(self, wifi\_issue=True) # Spécifie que l'arrêt est dû à un problème de WiFi

return False

# Matrices pour les lettres avec une largeur de 3 LED et une hauteur de 5 LED

LETTER\_MAP\_3 = {

'C': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 0), (2, 4), (1, 4)],

'L': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4)],

'E': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (1, 4), (2, 0), (2, 4)],

'W': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 3), (2, 2), (2, 0), (2, 1), (2, 3), (2, 4)],

'I': [(0, 0), (1, 0), (2, 0), (1, 1), (1, 2), (1, 3), (1, 4), (0, 4), (2, 4)],

'F': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (2, 0)],

'A': [(1, 0), (0, 1), (2, 1), (0, 2), (1, 2), (2, 2), (0, 3), (2, 3), (0, 4), (2, 4)],

'P': [(0, 0), (1, 0), (2, 0), (0, 1), (2, 1), (0, 2), (1, 2), (2, 2), (0, 3), (0, 4)],

'O': [(1, 0), (0, 1), (2, 1), (0, 2), (2, 2), (0, 3), (2, 3), (1, 4)],

'K': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (2, 0), (1, 2), (2, 4)],

'S': [(0, 1), (0, 2), (1, 0), (2, 0), (1, 2), (2, 3), (0, 4), (1, 4)],

'U': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (1, 4)],

'N': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4), (1, 1), (1, 2)],

'D': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 1), (2, 2), (2, 3), (1, 4)],

'R': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 0), (1, 2), (2, 2), (1, 3), (2, 4)],

'G': [(1, 0), (0, 1), (2, 1), (0, 2), (2, 2), (0, 3), (2, 3), (1, 4), (2, 4)],

'Y': [(0, 0), (2, 0), (1, 1), (1, 2), (1, 3), (1, 4)],

'T': [(0, 0), (1, 0), (2, 0), (1, 1), (1, 2), (1, 3), (1, 4)],

'H': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 2), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],

'V': [(0, 0), (0, 1), (0, 2), (1, 3), (2, 0), (2, 1), (2, 2)],

'M': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],

'X': [(0, 0), (0, 1), (0, 3), (0, 4), (2, 0), (2, 1), (2, 3), (2, 4), (1, 2)],

}

# Matrices pour les lettres avec une largeur de 4 LED et une hauteur de 5 LED

LETTER\_MAP\_4 = {

'C': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 4), (2, 0), (2, 4), (3, 0), (3, 4)],

'L': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 4), (2, 4), (3, 4)],

'E': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (1, 4), (2, 0), (2, 2), (2, 4), (3, 0), (3, 4)],

'W': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 3), (2, 3), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)],

'I': [(1, 0), (1, 1), (1, 2), (1, 3), (1, 4), (2, 0), (2, 1), (2, 2), (2, 3), (2, 4)],

'F': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (2, 0), (2, 2), (3, 0)],

'A': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (1, 2), (2, 0), (2, 2), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)],

'P': [(0, 0), (1, 0), (2, 0), (3, 0), (0, 1), (3, 1), (0, 2), (1, 2), (2, 2), (3, 2), (0, 3), (0, 4)],

'O': [(1, 0), (2, 0), (0, 1), (3, 1), (0, 2), (3, 2), (0, 3), (3, 3), (1, 4), (2, 4)],

'K': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (3, 0), (1, 2), (2, 1), (2, 3), (3, 4)],

'N': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 1), (1, 2), (2, 3), (2, 4), (3, 0), (3, 1), (3, 2), (3, 3), (3, 4)],

'R': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 0), (3, 0), (3, 1), (3, 2), (2, 2), (1, 2), (2, 3), (3, 4)],

'B': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 0), (3, 1), (2, 2), (1, 2), (3, 3), (3, 4), (2, 4), (1, 4)],

'T': [(0, 0), (1, 0), (2, 0), (3, 0), (1, 1), (2, 1), (1, 2), (2, 2), (1, 3), (2, 3), (1, 4), (2, 4)],

'S': [(3, 0), (2, 0), (1, 0), (0, 0), (0, 1), (0, 2), (1, 2), (2, 2), (3, 2), (3, 3), (3, 4), (2, 4), (1, 4), (0, 4)],

'D': [(0, 0), (0, 1), (0, 2), (0, 3), (0, 4), (1, 0), (2, 0), (3, 1), (3, 2), (3, 3), (2, 4), (1, 4), (0, 4)],

}

# Fonction pour dessiner une lettre de la map 3 spécifique à une position donnée

def draw\_letter\_3(graphics, letter, x, y, pen):

if letter in LETTER\_MAP\_3:

graphics.set\_pen(pen)

for dx, dy in LETTER\_MAP\_3[letter]:

graphics.pixel(x + dx, y + dy)

# Fonction pour dessiner un mot entier en map 3

def draw\_word\_3(graphics, word, x, y, pen, spacing=4):

current\_x = x

for letter in word:

draw\_letter\_3(graphics, letter, current\_x, y, pen)

current\_x += spacing

# Fonction pour dessiner une lettre de la map 4 spécifique à une position donnée

def draw\_letter\_4(graphics, letter, x, y, pen):

if letter in LETTER\_MAP\_4:

graphics.set\_pen(pen)

for dx, dy in LETTER\_MAP\_4[letter]:

graphics.pixel(x + dx, y + dy)

# Fonction pour dessiner un mot entier en map 4

def draw\_word\_4(graphics, word, x, y, pen, spacing=5):

"""Dessine un mot entier en utilisant la lettre de taille 4 LED de largeur et 5 LED de hauteur."""

current\_x = x

for letter in word:

if letter in LETTER\_MAP\_4:

graphics.set\_pen(pen)

for dx, dy in LETTER\_MAP\_4[letter]:

graphics.pixel(current\_x + dx, y + dy)

current\_x += spacing # Espacement entre les lettres

def show\_loading\_screen(display, step):

"""Affiche l'animation de chargement et le texte WAIT avec la police bitmap5."""

display.clear()

display.graphics.set\_font("bitmap5") # Définit la police sur bitmap5

display.graphics.set\_pen(display.pens['WHITE']) # Choisit le stylo blanc

display.graphics.text("WAIT", 5, 12, scale=1) # Affiche le texte "WAIT" en position (12, 14)

loading\_animation\_step(display, step) # Exécute l'étape de l'animation de chargement

display.update()

time.sleep(0.2) # Pause pour la synchronisation de l'animation

def loading\_animation\_step(display, step):

"""Affiche progressivement l'animation de chargement sur l'écran en fonction de l'étape."""

blocks = [

[(0, 30), (0, 31), (0, 32), (1, 30), (1, 31), (1, 32), (2, 30), (2, 31), (2, 32)],

[(3, 30), (3, 31), (3, 32), (4, 30), (4, 31), (4, 32), (5, 30), (5, 31), (5, 32)],

[(6, 30), (6, 31), (6, 32), (7, 30), (7, 31), (7, 32), (8, 30), (8, 31), (8, 32)],

[(9, 30), (9, 31), (9, 32), (10, 30), (10, 31), (10, 32), (11, 30), (11, 31), (11, 32)],

[(12, 30), (12, 31), (12, 32), (13, 30), (13, 31), (13, 32), (14, 30), (14, 31), (14, 32)],

[(15, 30), (15, 31), (15, 32), (16, 30), (16, 31), (16, 32), (17, 30), (17, 31), (17, 32)],

[(18, 30), (18, 31), (18, 32), (19, 30), (19, 31), (19, 32), (20, 30), (20, 31), (20, 32)],

[(21, 30), (21, 31), (21, 32), (22, 30), (22, 31), (22, 32), (23, 30), (23, 31), (23, 32)],

[(24, 30), (24, 31), (24, 32), (25, 30), (25, 31), (25, 32), (26, 30), (26, 31), (26, 32)],

[(27, 30), (27, 31), (27, 32), (28, 30), (28, 31), (28, 32), (29, 30), (29, 31), (29, 32)],

[(30, 30), (30, 31), (30, 32), (31, 30), (31, 31), (31, 32)]

]

if step < len(blocks):

# Vérifie si la couleur 'WHITE' est définie

if 'WHITE' in display.pens:

display.set\_pen('WHITE')

else:

print("Erreur : La couleur 'WHITE' n'est pas définie.")

return

# Dessine le bloc de LEDs pour l'étape en cours

for x, y in blocks[step]:

display.graphics.pixel(x, y)

display.update()

def display\_welcome\_screen(display):

"""Affiche l'écran d'accueil avec 'UNC' défilant, puis dessine un bloc bleu autour de 'UNC' en inversant les couleurs."""

display.clear() # Efface l'écran

# Utilisation des couleurs pré-définies

colors = {

'UNC': 'BLUE',

'OPT': 'YELLOW\_SMILEY',

}

# Charger la police bitmap5

display.graphics.set\_font("bitmap5")

# Positions de départ hors écran

unc\_x\_start = display.width # à droite, hors écran

opt\_x\_start = -display.graphics.measure\_text("OPT", 1) # à gauche, hors écran

# Positions finales

unc\_x\_final = 2

opt\_x\_final = 13

# Boucle de défilement pour les deux textes

while unc\_x\_start > unc\_x\_final or opt\_x\_start < opt\_x\_final:

display.clear()

# Afficher "UNC" en bleu, en défilant de droite à gauche

if unc\_x\_start > unc\_x\_final:

unc\_x\_start -= 1

display.set\_pen(colors['UNC'])

display.graphics.text("UNC", unc\_x\_start, 1, scale=1)

# Afficher "OPT" en jaune smiley, en défilant de gauche à droite

if opt\_x\_start < opt\_x\_final:

opt\_x\_start += 1

display.set\_pen(colors['OPT'])

display.graphics.text("OPT", opt\_x\_start, 24, scale=1)

# Mettre à jour l'affichage

display.update()

time.sleep(0.1) # Ajustez pour la vitesse du défilement

# Texte "UNC" en position finale avec inversion des couleurs

display.set\_pen(colors['UNC'])

display.graphics.text("UNC", unc\_x\_final, 1, scale=1)

display.update()

time.sleep(0.5)

# Couleur du fond en bleu et les lettres en noir

display.set\_pen('BLUE')

for x in range(1, 19):

for y in range(1, 8):

display.graphics.pixel(x, y)

# Laisser les LEDs de "UNC" en noir en repassant par-dessus

display.set\_pen('BLACK')

unc\_pixels = {

# 'U'

(2, 2), (2, 3), (2, 4), (2, 5), (2, 6), (3, 6), (4, 6), (5, 6), (6, 6), (6, 2), (6, 3), (6, 4), (6, 5),

# 'N'

(8, 2), (8, 3), (8, 4), (8, 5), (8, 6), (9, 2), (10, 2), (11, 2), (12, 2), (12, 3), (12, 4), (12, 5), (12, 6),

# 'C'

(14, 2), (14, 3), (14, 4), (14, 5), (14, 6), (15, 2), (16, 2), (17, 2), (15, 6), (16, 6), (17, 6),

}

for (x, y) in unc\_pixels:

display.graphics.pixel(x, y)

# Texte "OPT" en position finale avec inversion des couleurs

display.set\_pen(colors['OPT'])

display.graphics.text("OPT", opt\_x\_final, 24, scale=1)

display.update()

time.sleep(0.5)

# Couleur du fond en jaune et les lettres en noir

display.set\_pen('YELLOW\_SMILEY')

for x in range(12, 31):

for y in range(24, 31):

display.graphics.pixel(x, y)

# Laisser les LEDs de "OPT" en noir en repassant par-dessus

display.set\_pen('BLACK')

opt\_pixels = {

# 'O'

(13, 25), (13, 26), (13, 27), (13, 28), (13, 29), (14, 25), (15, 25), (16, 25), (14, 29), (15, 29), (16, 29), (17, 29), (17, 26), (17, 27), (17, 28), (17, 29),

# 'P'

(19, 25), (19, 26), (19, 27), (19, 28), (19, 29), (20, 25), (21, 25), (22, 25), (21, 28), (22, 28), (23, 28), (23, 25), (23, 26), (23, 27),

# 'T'

(25, 25), (26, 25), (27, 25), (28, 25), (29, 25), (27, 26), (27, 27), (27, 28), (27, 29),

}

for (x, y) in opt\_pixels:

display.graphics.pixel(x, y)

# Mettre à jour pour afficher les blocs finaux

display.update()

time.sleep(0.5)

# enchaîner avec l'animation

exploding\_heart\_animation(display)

def exploding\_heart\_animation(display):

"""Crée une animation d'un cœur explosant à partir d'une LED centrale, qui disparaît ensuite."""

# Définir la LED centrale de départ

center\_led = (15, 15)

# Liste des positions des LEDs formant le cœur final

heart\_positions = [

(15, 12), (14, 11), (16, 11), (13, 10), (17, 10),

(12, 9), (11, 9), (10, 9), (18, 9), (19, 9), (20, 9),

(9, 10), (21, 10), (8, 11), (22, 11), (7, 12), (7, 13),

(7, 14), (23, 12), (23, 13), (23, 14), (8, 15), (22, 15),

(9, 16), (21, 16), (10, 17), (20, 17), (11, 18), (19, 18),

(12, 19), (18, 19), (13, 20), (17, 20), (14, 21), (16, 21), (15, 22)

]

# Afficher la LED centrale

display.set\_pen('PINK')

display.graphics.pixel(\*center\_led)

display.update()

time.sleep(0.2) # Petite pause pour rendre l'animation visible

# Ajouter les LEDs petit à petit jusqu'à former le cœur final

for i, led in enumerate(heart\_positions):

# Lorsque la moitié des LEDs sont allumées, éteindre la LED centrale

if i == len(heart\_positions) // 2:

display.set\_pen('BLACK')

display.graphics.pixel(\*center\_led)

# Afficher la LED courante du cœur

display.set\_pen('PINK')

display.graphics.pixel(\*led)

display.update()

time.sleep(0.05) # Pause pour rendre l'animation progressive

def display\_info\_screen(self, wifi\_status, api\_key\_status, file\_agences\_status):

"""Affiche l'état du WiFi, de la clé API, et du fichier agences.env sur l'écran d'information."""

self.clear() # Efface l'écran pour l'affichage des informations.

# Définir la police et la couleur

self.graphics.set\_font("bitmap5")

self.graphics.set\_pen(self.pens['WHITE'])

# Affichage pour l'état du WiFi

self.graphics.text("WIFI", 1, 0, scale=1)

if wifi\_status:

self.graphics.set\_pen(self.pens['GREEN'])

self.graphics.text("OK", 21, 0, scale=1)

else:

self.graphics.set\_pen(self.pens['RED'])

self.graphics.text("KO", 21, 0, scale=1)

# Affichage pour l'état de la clé API

self.graphics.set\_pen(self.pens['WHITE'])

self.graphics.text("API", 1, 8, scale=1)

if api\_key\_status:

self.graphics.set\_pen(self.pens['GREEN'])

self.graphics.text("OK", 21, 8, scale=1)

else:

self.graphics.set\_pen(self.pens['RED'])

self.graphics.text("KO", 21, 8, scale=1)

# Affichage pour l'état du fichier agences.env

self.graphics.set\_pen(self.pens['WHITE'])

self.graphics.text(".ENV", 1, 16, scale=1)

if file\_agences\_status:

self.graphics.set\_pen(self.pens['GREEN'])

self.graphics.text("OK", 21, 16, scale=1)

else:

self.graphics.set\_pen(self.pens['RED'])

self.graphics.text("KO", 21, 16, scale=1)

# Affichage url bitly

#self.graphics.set\_pen(self.pens['WHITE'])

#self.graphics.text("https://bit.ly/3AJbpj2", 1, 23, scale=1)

self.update() # Met à jour l'affichage avec les informations.

# Fonction d'affichage de l'écran des légendes

def display\_legend\_screen(display, display\_time=10):

"""Affiche et fait défiler les légendes avec les LEDs et le texte alignés sur des lignes spécifiques, indépendamment."""

start\_time = time.time()

display.clear()

# Configuration des légendes : texte, couleur, LEDs en en-tête, zone d'affichage (y\_start, y\_end)

legends = [

{"message": "SON ACTIVE", "color": "BLUE", "leds": [(0, 1), (0, 2), (1, 0), (1, 1), (1, 2), (1, 3)], "y\_start": 0, "text\_y": -1},

{"message": "SON DESACTIVE", "color": "RED", "leds": [(0, 7), (0, 8), (1, 6), (1, 7), (1, 8), (1, 9)], "y\_start": 6, "text\_y": 5},

{"message": "PERTE WIFI", "color": "RED", "leds": [(0, 14), (1, 13), (1, 14), (1, 15), (2, 12), (2, 13), (2, 14), (2, 15), (2, 16)], "y\_start": 12, "text\_y": 11},

{"message": "NOM AGENCE FIXE", "color": "YELLOW", "leds": [(1, 20)], "y\_start": 18, "text\_y": 17},

]

for legend in legends:

display.set\_pen(legend["color"])

for x\_offset, y\_offset in legend["leds"]:

display.graphics.pixel(x\_offset, y\_offset)

display.set\_pen("WHITE")

display.graphics.text(legend["message"], 5, legend["text\_y"], -1, 1)

display.update()

time.sleep(3)

scroll\_positions = [0] \* len(legends)

last\_update\_times = [time.ticks\_ms()] \* len(legends)

message\_widths = [display.graphics.measure\_text(legend["message"], 1) for legend in legends]

# Boucle de défilement avec gestion indépendante pour chaque ligne

while time.time() - start\_time < display\_time:

time\_ms = time.ticks\_ms()

# Vérification immédiate du bouton pour basculer d'affichage

if display.cu.is\_pressed(display.cu.SWITCH\_C):

print("Bouton C pressé - Changement d'affichage depuis la screen légende.")

display.play\_bip(500)

break # Sortir immédiatement de la boucle

for i, legend in enumerate(legends):

if time\_ms - last\_update\_times[i] > 100:

last\_update\_times[i] = time\_ms

display.set\_pen("BLACK")

display.graphics.rectangle(0, legend["y\_start"], display.width, 5)

display.set\_pen(legend["color"])

for x\_offset, y\_offset in legend["leds"]:

display.graphics.pixel(x\_offset - scroll\_positions[i], y\_offset)

display.set\_pen("WHITE")

display.graphics.text(

legend["message"],

5 - scroll\_positions[i],

legend["text\_y"],

-1,

1

)

if scroll\_positions[i] >= message\_widths[i] + display.width + 5:

scroll\_positions[i] = -display.width

last\_update\_times[i] = time\_ms + 3000

else:

scroll\_positions[i] += 1

display.update()

time.sleep(0.05)

# QR CODE de l'adresse Bit.ly "https://bit.ly/3AJbpj2" (https://github.com/adriens/temps-attente-matrix-led)

led\_white\_positions = [

(4, 4), (5, 4), (6, 4), (7, 4), (8, 4), (9, 4), (10, 4), (12, 4), (13, 4),

(14, 4), (15, 4), (18, 4), (22, 4), (23, 4), (24, 4), (25, 4), (26, 4), (27, 4),

(28, 4), (4, 5), (10, 5), (15, 5), (16, 5), (20, 5), (22, 5), (28, 5), (4, 6),

(6, 6), (7, 6), (8, 6), (10, 6), (13, 6), (15, 6), (17, 6), (18, 6), (20, 6),

(22, 6), (24, 6), (25, 6), (26, 6), (28, 6), (4, 7), (6, 7), (7, 7), (8, 7),

(10, 7), (14, 7), (15, 7), (17, 7), (20, 7), (22, 7), (24, 7), (25, 7), (26, 7),

(28, 7), (4, 8), (6, 8), (7, 8), (8, 8), (10, 8), (14, 8), (15, 8), (16, 8),

(17, 8), (18, 8), (19, 8), (20, 8), (22, 8), (24, 8), (25, 8), (26, 8), (28, 8),

(4, 9), (10, 9), (13, 9), (15, 9), (17, 9), (18, 9), (22, 9), (28, 9), (4, 10),

(5, 10), (6, 10), (7, 10), (8, 10), (9, 10), (10, 10), (12, 10), (14, 10),

(16, 10), (18, 10), (20, 10), (22, 10), (23, 10), (24, 10), (25, 10), (26, 10),

(27, 10), (28, 10), (12, 11), (13, 11), (16, 11), (17, 11), (19, 11), (4, 12),

(5, 12), (7, 12), (8, 12), (10, 12), (13, 12), (16, 12), (17, 12), (18, 12),

(19, 12), (22, 12), (28, 12), (5, 13), (7, 13), (14, 13), (17, 13), (19, 13),

(20, 13), (23, 13), (24, 13), (25, 13), (26, 13), (27, 13), (5, 14), (6, 14),

(8, 14), (10, 14), (11, 14), (13, 14), (16, 14), (18, 14), (19, 14), (20, 14),

(21, 14), (25, 14), (28, 14), (4, 15), (5, 15), (7, 15), (18, 15), (19, 15),

(22, 15), (25, 15), (26, 15), (27, 15), (28, 15), (4, 16), (9, 16), (10, 16),

(12, 16), (13, 16), (14, 16), (17, 16), (20, 16), (22, 16), (23, 16), (28, 16),

(4, 17), (7, 17), (8, 17), (11, 17), (12, 17), (13, 17), (14, 17), (16, 17),

(17, 17), (18, 17), (19, 17), (21, 17), (24, 17), (27, 17), (4, 18), (5, 18),

(7, 18), (8, 18), (9, 18), (10, 18), (11, 18), (13, 18), (18, 18), (19, 18),

(21, 18), (22, 18), (24, 18), (25, 18), (26, 18), (27, 18), (28, 18), (4, 19),

(6, 19), (7, 19), (8, 19), (14, 19), (16, 19), (18, 19), (20, 19), (22, 19),

(23, 19), (25, 19), (26, 19), (28, 19), (4, 20), (10, 20), (11, 20), (12, 20),

(15, 20), (16, 20), (17, 20), (20, 20), (21, 20), (22, 20), (23, 20), (24, 20),

(26, 20), (27, 20), (12, 21), (15, 21), (19, 21), (20, 21), (24, 21), (26, 21),

(27, 21), (4, 22), (5, 22), (6, 22), (7, 22), (8, 22), (9, 22), (10, 22),

(13, 22), (14, 22), (16, 22), (20, 22), (22, 22), (24, 22), (28, 22), (4, 23),

(10, 23), (15, 23), (17, 23), (19, 23), (20, 23), (24, 23), (4, 24), (6, 24),

(7, 24), (8, 24), (10, 24), (12, 24), (13, 24), (14, 24), (15, 24), (17, 24),

(19, 24), (20, 24), (21, 24), (22, 24), (23, 24), (24, 24), (27, 24), (28, 24),

(4, 25), (6, 25), (7, 25), (8, 25), (10, 25), (12, 25), (13, 25), (14, 25),

(16, 25), (17, 25), (18, 25), (20, 25), (22, 25), (27, 25), (28, 25), (4, 26),

(6, 26), (7, 26), (8, 26), (10, 26), (15, 26), (17, 26), (20, 26), (21, 26),

(24, 26), (25, 26), (26, 26), (27, 26), (28, 26), (4, 27), (10, 27), (12, 27),

(13, 27), (15, 27), (16, 27), (17, 27), (23, 27), (24, 27), (26, 27), (27, 27),

(28, 27), (4, 28), (5, 28), (6, 28), (7, 28), (8, 28), (9, 28), (10, 28), (12, 28),

(14, 28), (15, 28), (16, 28), (17, 28), (20, 28), (21, 28), (25, 28), (28, 28)

]

# Fonction d'affichage du QR code avec intégration de la luminosité définie dans la classe

def display\_qr\_code\_screen(self):

self.clear() # Efface l'écran pour un nouvel affichage

# Affichage du QR code en tenant compte de la luminosité actuelle

led\_on\_intensity = int(255 \* self.brightness)

self.graphics.set\_pen(self.graphics.create\_pen(0, 0, 0))

self.graphics.clear()

for x, y in led\_white\_positions:

self.graphics.set\_pen(self.graphics.create\_pen(led\_on\_intensity, led\_on\_intensity, led\_on\_intensity))

self.graphics.pixel(x, y)

# Mettre à jour l'affichage pour refléter les changements

self.update()

# Boucle pour ajuster la luminosité en temps réel

while True:

self.adjust\_brightness() # Ajuste la luminosité en fonction des boutons de luminosité

led\_on\_intensity = int(255 \* self.brightness)

for x, y in led\_white\_positions:

self.graphics.set\_pen(self.graphics.create\_pen(led\_on\_intensity, led\_on\_intensity, led\_on\_intensity))

self.graphics.pixel(x, y)

self.update()

# Interruption de la boucle avec le bouton C

if self.cu.is\_pressed(CosmicUnicorn.SWITCH\_C):

print("Bouton C pressé - Quitter l'écran QR code.")

self.play\_bip(500) # Émettre un bip de confirmation

break # Sortie de la boucle pour passer à l'écran suivant

time.sleep(0.1)

# Attente de la pression du bouton pour démarrer le script principal

def wait\_for\_start(display, cu):

"""Affiche l'écran d'accueil et attend la pression du bouton C pour lancer le script principal."""

display\_welcome\_screen(display) # Affiche le message d'accueil

print("Attente de la pression du bouton C pour démarrer...")

# Boucle pour attendre la pression du bouton C

while True:

if cu.is\_pressed(CosmicUnicorn.SWITCH\_C):

print("Bouton C pressé - Lancement du script principal.")

time.sleep(0.5) # Petite pause pour éviter les rebonds

break # Sortie de la boucle et début du script principal

time.sleep(0.1) # Vérifie le bouton à intervalles réguliers

# Fonction pour supprimer les accents : définit sur é et è

def normalize\_name(text):

"""Remplace manuellement les accents par leurs équivalents non accentués et met le texte en majuscules."""

accents = {

'è': 'e', 'é': 'e',

}

# Remplacer chaque caractère accentué par son équivalent non accentué

return ''.join(accents.get(c, c) for c in text).upper()

# Fonction pour arrêter proprement le script.

def stop\_script(display, wifi\_issue=False):

"""Arrête proprement le script et attend un redémarrage via le bouton D."""

print("Arrêt du script demandé...")

# Affiche "NO WIFI\nREBOOT\nPRESS D" centré si l'arrêt est causé par une perte de connexion WiFi

message\_lines = ["NO WIFI", "REBOOT", "PRESS D"] if wifi\_issue else ["KO", "REBOOT", "PRESS D"]

display.graphics.set\_pen(display.pens['BLACK']) # Définit le fond noir

display.graphics.clear() # Efface l'écran

# Utilise draw\_word\_4 pour afficher le message d'arrêt sur l'écran

display.set\_pen('RED')

y\_offset = 2

for line in message\_lines:

draw\_word\_4(display.graphics, line, 2, y\_offset, display.pens['RED'])

y\_offset += 10 # Augmente l'offset vertical pour chaque ligne

display.update() # Met à jour l'affichage pour refléter le message final

# Attente de l'appui sur le bouton D pour redémarrer

while True:

if display.cu.is\_pressed(CosmicUnicorn.SWITCH\_D):

print("Redémarrage suite à la pression du bouton D.")

time.sleep(1) # Petite pause pour éviter plusieurs déclenchements

machine.reset() # Redémarrage

# Fonction pour charger les informations de connexion WiFi et clé API depuis le fichier "information.env"

def load\_credentials(file\_path):

"""Charge les informations de connexion WiFi (SSID, mot de passe) et la clé API depuis un fichier."""

credentials = {}

try:

with open(file\_path, "r") as f: # Ouvre le fichier contenant les informations.

for line in f:

key, value = line.strip().split('=') # Sépare les lignes par '=' pour extraire les informations.

credentials[key.strip()] = value.strip() # Stocke les informations dans un dictionnaire.

except OSError:

print(f"Erreur : impossible de trouver ou lire le fichier {file\_path}")

return credentials # Retourne le dictionnaire contenant les informations.

# Fonction pour se connecter au WiFi

def connect\_wifi(ssid, password, display, max\_attempts=10):

"""Tente de se connecter au réseau WiFi avec un maximum de tentatives, avec animation."""

wlan = network.WLAN(network.STA\_IF) # Initialise l'interface WiFi en mode station (client)

wlan.active(True) # Active l'interface WiFi

attempts = 0 # Initialise le compteur de tentatives

while not wlan.isconnected() and attempts < max\_attempts:

print(f"Connexion à {ssid}... Tentative {attempts + 1}/{max\_attempts}")

# Ajout de l'animation de chargement pendant la tentative de connexion

loading\_animation\_step(display, attempts)

wlan.connect(ssid, password) # Lance la connexion au réseau WiFi avec les informations fournies

time.sleep(3) # Attend 3 secondes entre les tentatives

attempts += 1

if wlan.isconnected():

print(f"WiFi connecté avec l'IP : {wlan.ifconfig()[0]}")

return wlan # Retourne l'objet wlan si la connexion est établie

else:

print("Échec de la connexion WiFi après plusieurs tentatives.")

return None # Retourne None si la connexion échoue

# Fonction pour synchroniser l'heure avec un serveur NTP

def sync\_time():

"""Synchronise l'heure locale avec un serveur NTP."""

ntp\_servers = ['time.windows.com', 'ntp1.google.com', 'pool.ntp.org'] # Liste des serveurs NTP à contacter

for server in ntp\_servers: # Parcourt chaque serveur NTP

try:

ntptime.host = server # Définit le serveur NTP à contacter

ntptime.settime() # Tente de synchroniser l'heure

print(f"Heure synchronisée via NTP avec {server}") # Affiche un message si la synchronisation réussit

return True # Retourne True si la synchronisation est réussie

except OSError as e:

print(f"Erreur de synchronisation NTP avec {server}: {e}") # Affiche une erreur si la synchronisation échoue

print("Échec de la synchronisation NTP.") # Affiche un message si aucun serveur NTP n'a pu être contacté

return False # Retourne False si la synchronisation échoue

def load\_agencies(file\_path):

"""Charge les agences et leurs noms depuis le fichier agences.env."""

agencies = []

file\_loaded = False

try:

with open(file\_path, "r") as f:

for line in f:

if line.strip(): # Ignorer les lignes vides

id, name = line.strip().split("=")

agencies.append([id.strip(), name.strip().strip('"'), 0]) # Ajoute l'agence avec un temps d'attente initial de 0

file\_loaded = True

print("Agences chargées depuis agences.env:", agencies)

except OSError:

print(f"Erreur : impossible de trouver ou lire le fichier {file\_path}")

return agencies, file\_loaded # Retourne la liste des agences et le statut de chargement

def initialize\_agency\_wait\_times(api\_key, agencies):

"""Initialise les temps d'attente pour les deux premières agences dans la liste."""

if agencies:

# Mettre à jour le temps d'attente pour l'agence à l'index 0

if not update\_single\_agency(api\_key, agencies[0]):

print(f"Erreur de mise à jour du temps d'attente pour l'agence {agencies[0][1]}")

# Mettre à jour le temps d'attente pour l'agence à l'index 1 si elle existe

if len(agencies) > 1 and not update\_single\_agency(api\_key, agencies[1]):

print(f"Erreur de mise à jour du temps d'attente pour l'agence {agencies[1][1]}")

# Fonction pour mettre à jour une seule agence avant l'affichage

def update\_single\_agency(api\_key, agency):

"""Met à jour les données d'une agence spécifique en appelant l'API."""

agence\_id, name, old\_waiting\_time = agency # Récupère les informations actuelles de l'agence

url = f"https://api.opt.nc/temps-attente-agences/temps-attente/agence/{agence\_id}" # Construit l'URL de l'API pour l'agence spécifique

headers = {"x-apikey": api\_key} # Ajoute l'API Key aux en-têtes HTTP

try:

response = requests.get(url, headers=headers) # Effectue une requête GET pour récupérer les nouvelles données de l'agence

gc.collect() # Libère la mémoire immédiatement après la requête

print(f"Appel API pour agence {name} (ID: {agence\_id}), statut: {response.status\_code}") # Affiche le statut de la réponse API

if response.status\_code == 200: # Si la requête réussit

data = response.json() # Convertit la réponse en format JSON

print(f"Réponse API agence {name}: {data}") # Affiche les nouvelles données de l'agence

new\_waiting\_time = data['realMaxWaitingTimeMs'] # Récupère le temps d'attente mis à jour

agency[2] = new\_waiting\_time # Met à jour le temps d'attente dans la liste des agences

print(f"Temps d'attente mis à jour pour {name} (ID: {agence\_id}) : {new\_waiting\_time // 60000} minutes (ancien {old\_waiting\_time // 60000} minutes)")

return True # Retourne True si la mise à jour a réussi

else:

print(f"Erreur API pour {name} (ID: {agence\_id}) : {response.status\_code}") # Affiche un message d'erreur en cas d'échec

except Exception as e:

print(f"Erreur lors de la mise à jour de l'agence {name} (ID: {agence\_id}) : {e}") # Affiche une erreur en cas d'exception

return False # Retourne False si la mise à jour a échoué

# Fonction pour gérer la pression des boutons

def handle\_button\_press(cu, display):

"""Gère les pressions des boutons A et D sur tous les écrans et ajuste le volume."""

if cu.is\_pressed(CosmicUnicorn.SWITCH\_A): # Si le bouton A est pressé

display.toggle\_sound() # Active ou désactive le son

if cu.is\_pressed(CosmicUnicorn.SWITCH\_D): # Si le bouton D est pressé pour redémarrer

print("Bouton D pressé - Redémarrage...")

time.sleep(1) # Attendre 1 seconde avant le redémarrage

machine.reset() # Redémarre la carte

display.adjust\_volume() # Ajuste le volume avec les boutons de volume

def main\_loop(display, start\_time, synced, api\_key, wlan, tableau\_agences):

display.clear() # Efface l'écran avant d'afficher le message d'attente

display.display\_message\_frame\_2("WAIT")

print("Début de la boucle principale - affichage initial WAIT")

time.sleep(2)

# Initialiser la mise à jour des temps d'attente pour les deux premières agences

if not update\_single\_agency(api\_key, tableau\_agences[0]):

print("Erreur de mise à jour pour l'agence", tableau\_agences[0][1])

if len(tableau\_agences) > 1 and not update\_single\_agency(api\_key, tableau\_agences[1]):

print("Erreur de mise à jour pour l'agence", tableau\_agences[1][1])

current\_index = 0

while True:

try:

print(f"Affichage agence - itération pour l'agence index {current\_index}")

wifi\_status = display.check\_wifi\_status(wlan)

display.update\_led\_wifi\_status(wifi\_status)

agence\_id, name, waiting\_time = tableau\_agences[current\_index]

if update\_single\_agency(api\_key, tableau\_agences[current\_index]):

print(f"Agence mise à jour : {name}, Temps d'attente mis à jour.")

else:

print(f"Affichage des anciennes données pour : {name}")

mood = 'happy' if waiting\_time < 300000 else 'neutral' if waiting\_time < 600000 else 'sad'

display.clear()

display.draw\_frame(0, 6, 'YELLOW')

display.draw\_text\_opt()

display.draw\_smiley(mood)

display.set\_transition\_variable(name)

print(f"Agence affichée : {name}, ID : {agence\_id}, Temps d'attente : {waiting\_time // 60000} minutes")

iteration\_count = 0

while iteration\_count < 100:

if display.cu.is\_pressed(CosmicUnicorn.SWITCH\_C):

print("Bouton C pressé - Retour à l'écran d'accueil.")

display.play\_bip(500)

time.sleep(0.5)

return

if display.cu.is\_pressed(CosmicUnicorn.SWITCH\_B):

display.toggle\_loop\_pause()

time.sleep(0.5)

if display.loop\_paused:

display.scroll\_text(display.transition\_var)

display.display\_clock(start\_time, synced)

display.adjust\_brightness()

display.adjust\_volume()

handle\_button\_press(display.cu, display)

time.sleep(0.1)

continue

display.scroll\_text(display.transition\_var)

display.display\_clock(start\_time, synced)

display.adjust\_brightness()

display.adjust\_volume()

handle\_button\_press(display.cu, display)

time.sleep(0.1)

iteration\_count += 1

next\_index = (current\_index + 1) % len(tableau\_agences)

if not update\_single\_agency(api\_key, tableau\_agences[next\_index]):

print("Erreur de mise à jour pour l'agence", tableau\_agences[next\_index][1])

if not display.loop\_paused:

current\_index = next\_index

except Exception as e:

print(f"Erreur dans la boucle d'affichage des agences : {e}")

time.sleep(2)

display.update\_led\_sound\_status(False)

# Fonction main pour afficher la page d'accueil avec le message "UNC OPT".

def main():

"""Fonction principale avec affichage initial et basculement vers l'écran d'accueil."""

display = CosmicUnicornDisplay() # Initialise l'affichage

cu = display.cu # Gestion des boutons

# Affichage de l'écran "WAIT" au démarrage

step = 0

show\_loading\_screen(display, step)

print("Affichage initial 'WAIT'")

step += 1

# Charger les informations WiFi et API

credentials = load\_credentials("information.env")

if not credentials:

print("Erreur : Informations de connexion non trouvées.")

stop\_script(display)

return

wlan = connect\_wifi(credentials['SSID'], credentials['WIFI\_PASSWORD'], display)

if wlan is None:

print("Erreur de connexion WiFi. Arrêt du script.")

stop\_script(display, wifi\_issue=True)

return

show\_loading\_screen(display, step)

step += 1

tableau\_agences, file\_agences\_status = load\_agencies("agences.env")

if not file\_agences\_status:

print("Erreur : Impossible de charger les agences.")

stop\_script(display)

return

show\_loading\_screen(display, step)

step += 1

if wlan:

synced = sync\_time()

start\_time = time.time()

wifi\_status = True

else:

stop\_script(display, wifi\_issue=True)

return

show\_loading\_screen(display, step)

step += 1

api\_key\_status = credentials.get('API\_KEY') is not None

initialize\_agency\_wait\_times(credentials['API\_KEY'], tableau\_agences)

show\_loading\_screen(display, step)

print("Chargement terminé.")

step += 1

# Modes d'affichage et configuration

display\_modes = [

display\_welcome\_screen,

lambda d: display\_info\_screen(d, wifi\_status, api\_key\_status, file\_agences\_status),

lambda d: display\_legend\_screen(d, display\_time=15),

lambda d: main\_loop(d, start\_time, synced, credentials['API\_KEY'], wlan, tableau\_agences),

display\_qr\_code\_screen

]

current\_mode = 0

# Forcer le passage initial au premier écran après le chargement

display.display\_mode = current\_mode

display\_modes[current\_mode](display)

while True:

handle\_button\_press(cu, display)

if cu.is\_pressed(CosmicUnicorn.SWITCH\_C):

current\_mode = (current\_mode + 1) % len(display\_modes)

display.display\_mode = current\_mode

display.play\_bip(500)

print("Passage au mode suivant.")

time.sleep(0.5)

display\_modes[current\_mode](display)

time.sleep(0.1)

# Démarrer le programme avec la fonction main()

main()

* 1. Documents techniques Pimoroni

***Attention, ne supprimez pas le saut de section suivant (pied de page différent)***

[Cliquez ici pour entrer le titre de votre mémoire en français]

[Cliquez ici pour entrer le texte de votre résumé en français. Attention, les résumés en français et en anglais ainsi que les mots-clés dans les deux langues doivent tenir sur une seule et même page. Soyez synthétique !]

Mots-clés : [mot1, mot2…]

[Cliquez ici pour entrer le titre en anglais]

[N’oubliez pas d’utiliser cette zone pour inscrire votre résumé en anglais (on parle alors d’un abstract). N’hésitez pas à vous faire aider si vous n’êtes pas très à l’aise avec la langue de Shakespeare.]

Keywords : [keyword1, keyword2…]