EIGSI
ÉCOLE D'INGÉNIEURS
LA ROCHELLE • CASABLANCA

# PROJET SGATMI

Système de Gestion Automatisée des Téléviseurs pour une Maison Intelligente en Temps Réel et Multitâche

Réalisé par :
MATHUNABO
SAKINA Virginie

# Table des matières

# Introduction

Dans un monde de plus en plus connecté, les maisons intelligentes représentent l'apogée de la technologie domestique, offrant un niveau de confort et de commodité sans précédent pour leurs occupants. De la domotique à la gestion des appareils électroménagers, ces habitats modernes révolutionnent la manière dont nous interagissons avec notre environnement domestique.

Au cœur de cette révolution se trouve un projet novateur : la création d'un système automatisé de gestion des téléviseurs dans une maison intelligente. Bien au-delà de la simple commodité, ce projet vise à redéfinir l'expérience utilisateur en permettant un contrôle en temps réel et multitâche des téléviseurs, offrant ainsi une expérience immersive et fluide aux occupants.

Le besoin de simplifier la gestion des appareils électroniques devient de plus en plus pressant à mesure que les maisons intelligentes se répandent. Dans cette optique, notre système ambitionne de répondre à cette exigence en automatisant le contrôle des téléviseurs, libérant ainsi les occupants pour se concentrer sur d'autres activités et améliorer leur qualité de vie au quotidien.

Ce projet ne se limite pas seulement à la création d'un système fonctionnel, mais s'étend également à la formation des futurs professionnels dans les domaines du multitâche et du temps réel. Dans le cadre du module Multitâche et Temps Réel, ce projet offre une opportunité unique aux étudiants de se familiariser avec des concepts essentiels de l'ingénierie moderne, tout en développant des compétences cruciales pour leur avenir professionnel. En participant à la conception et au développement du système SGATMI, les étudiants auront l'occasion de renforcer leur expertise technique et d'enrichir leur curriculum vitae avec une expérience pratique et pertinente.

Au-delà de ces objectifs pédagogiques, l'aboutissement ultime de ce projet est de répondre aux attentes et aux besoins de l'encadrant, en présentant une analyse fonctionnelle détaillée du système proposé. En combinant une modélisation précise avec l'outil Rt Studio et une implémentation cohérente sur la plateforme Wokwi, notre équipe s'efforcera de proposer une solution optimale, répondant aux exigences spécifiques du projet et offrant une simulation réaliste de son fonctionnement.

# Spécifications du système

- **Ressources** :
  - Téléviseurs : TV1, TV2 et TV3
  - Capteurs de mouvement
  - Capteur de luminosité
- **États de chaque ressource** :
  - Téléviseurs : Allumé – Éteint
  - Capteur de mouvement : PM (Présence de Mouvement) – AM (Absence de Mouvement)
  - Capteur de luminosité : Jour – Nuit
- **Conditions d'allumage** :
  - Tous les téléviseurs peuvent s'allumer uniquement s'il fait jour.
  - TV1 peut s'allumer pendant la nuit si le capteur de mouvement détecte une présence.
  - Maximum deux téléviseurs peuvent être allumés simultanément.
- **Conditions d'extinction** :
  - Tous les téléviseurs s'éteignent automatiquement après chaque période de deux heures.
  - TV2 et TV3 s'éteignent automatiquement s'il fait nuit.
  - Si TV1 est allumé en raison de l'activation du capteur de mouvement, il s'éteint automatiquement après 1 minute.
- **Conditions spéciales d'extinction** :
  - TV2 s'éteint immédiatement si TV1 est allumé à cause du capteur de luminosité.
- **Condition ultime à respecter** :
  - Il est impératif que deux téléviseurs restent allumés simultanément à tout moment.
- **Période pour changement d'état des capteurs** :
  - Le capteur de luminosité reste idéalement dans un état pendant 12 heures avant de passer à l'autre état.
  - Le capteur de mouvement a une période de 1 seconde une fois activé avant de changer d'état.

# Analyse fonctionnelle

**Fonctions principales du système** :

- Détection de la présence : Cette fonction consiste à détecter la présence ou l'absence de mouvement dans une pièce à l'aide des capteurs de mouvement.
- Gestion de l'éclairage : Cette fonction implique de détecter le niveau de luminosité dans la pièce à l'aide du capteur de luminosité et de réguler l'éclairage en conséquence.
- Contrôle des téléviseurs : Cette fonction consiste à allumer, éteindre et réguler l'état des téléviseurs en fonction des conditions détectées par les capteurs et des besoins des occupants.

**Interactions entre les fonctions** :

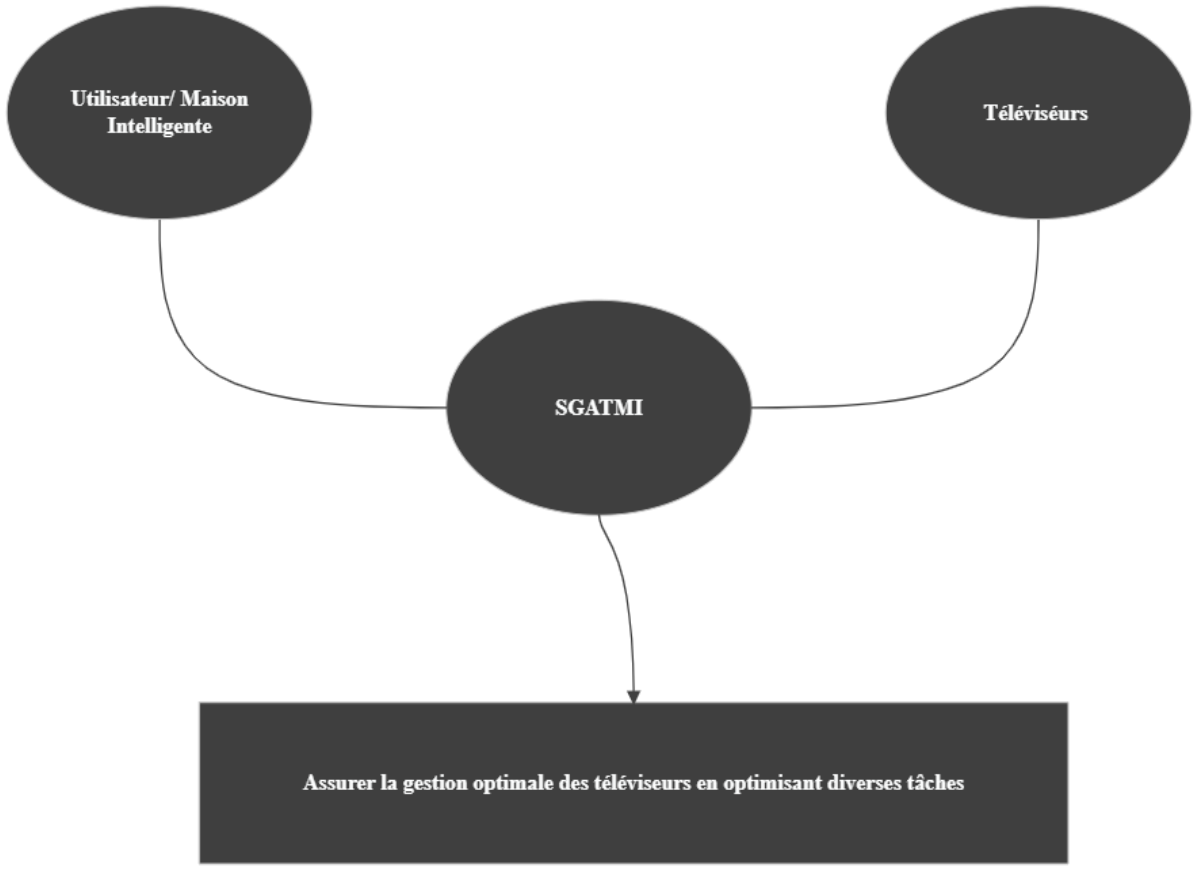
- La détection de la présence et la gestion de l'éclairage sont interconnectées, car le niveau de luminosité dans la pièce peut influencer la détection de la présence et vice versa.
- Le contrôle des téléviseurs dépend à la fois de la détection de la présence et de la gestion de l'éclairage, car les téléviseurs doivent être allumés ou éteints en fonction de la présence des occupants et des conditions d'éclairage.

**Besoins et contraintes associés à chaque fonction** :

- La fonction de détection de la présence nécessite des capteurs de mouvement fiables et précis pour détecter efficacement la présence des occupants.
- La gestion de l'éclairage nécessite un capteur de luminosité sensible et réactif pour ajuster l'éclairage de manière appropriée en fonction des conditions de luminosité dans la pièce.
- La fonction de contrôle des téléviseurs doit respecter la contrainte selon laquelle une paire de deux téléviseurs doit s'allumer en même temps. Cela signifie que lorsqu'un téléviseur est allumé, un autre téléviseur doit également être allumé simultanément pour respecter cette contrainte. Cette exigence assure une expérience utilisateur cohérente et garantit que les occupants peuvent toujours accéder à au moins deux téléviseurs fonctionnels simultanément.

## Diagramme Bête à Cornes

Le diagramme permet de visualiser les relations entre différents éléments.



## Analyse logique binaire

Cela montre le nombre d'état du système avec un nombre de ressources égal à 3 (TV1, TV2, TV3)

| TV1 | TV2 | TV3 | Condition |
|---|---|---|---|
| 0 | 0 | 0 | invalide |
| 0 | 0 | 1 | invalide |
| 0 | 1 | 0 | invalide |
| 0 | 1 | 1 | valide |
| 1 | 0 | 0 | invalide |
| 1 | 0 | 1 | valide |
| 1 | 1 | 0 | valide |
| 1 | 1 | 1 | invalide |

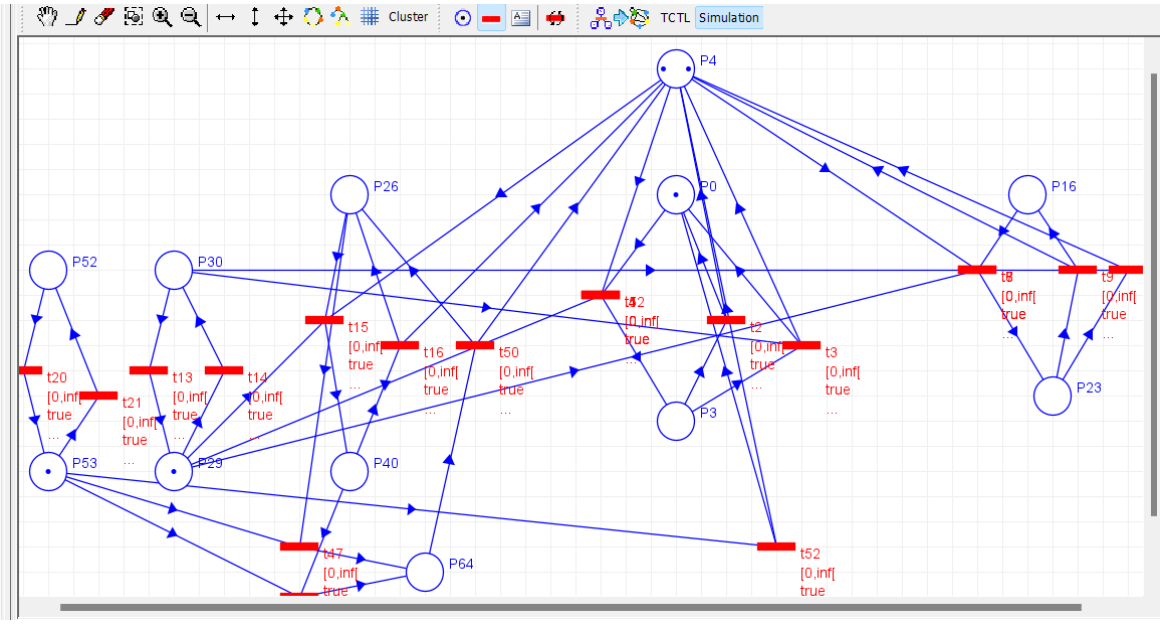
*Invalide : la contrainte de n'avoir uniquement 2 télés allumées simultanément n'est pas respectée.

*Valide : Contrainte respectée.

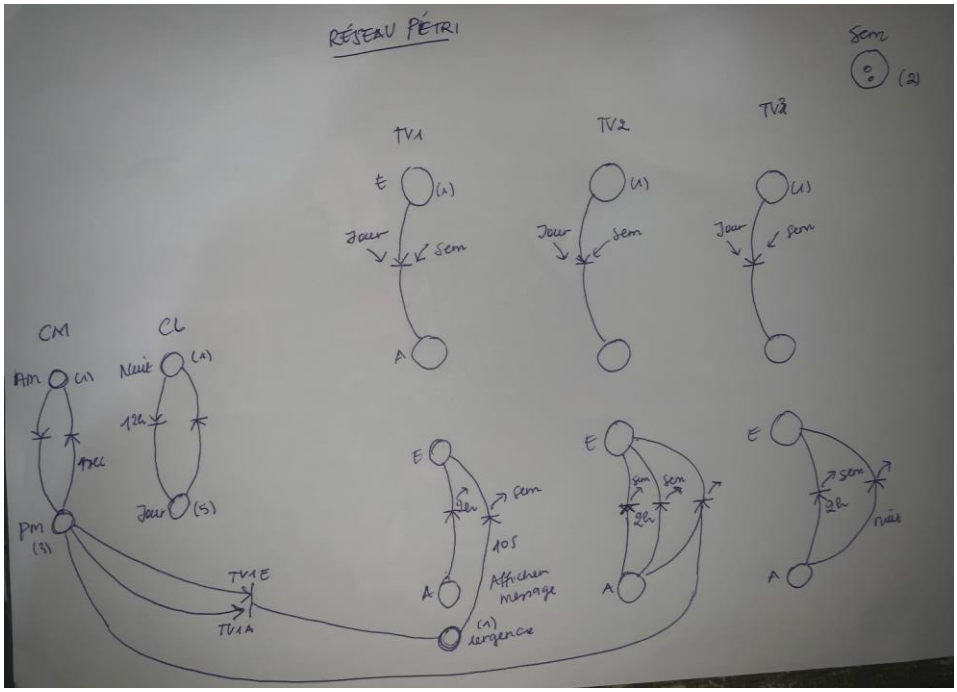
# Modélisation du Réseau de Pétri

## Réseau de Pétri avec Rt STUDIO

Nous avons utilisé les réseaux de Petri pour modéliser les différents états du système, y compris les transitions entre les états lors de la mise en marche du système tout en prenant en compte la présence d'objets connectés.




## Réseau de Pétri manuellement



## Liste des tâches et définition des propriétés

- Tâches :

1. Allumer TV1, TV2, TV3
2. Eteindre TV1, TV2, TV3
3. Afficher message (TV1)
4. Activer/ désactiver CM
5. Activer/ désactiver CL

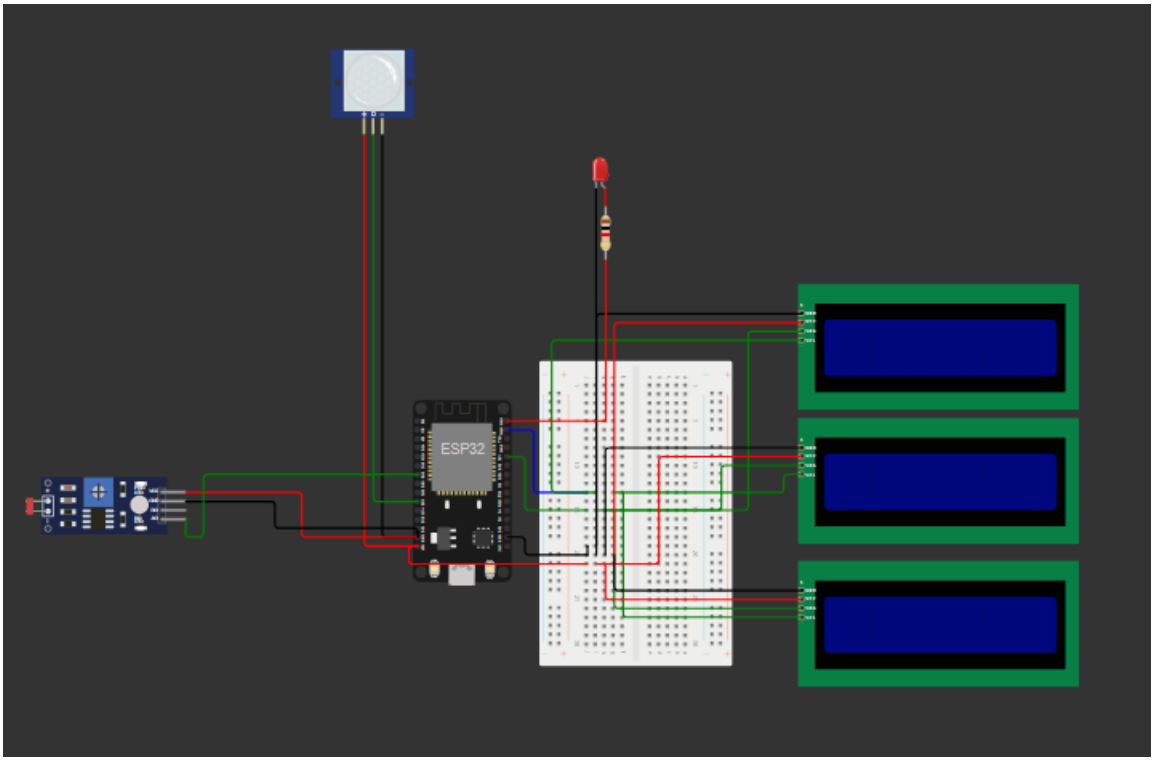
- Propriétés

| Actions | deadline | périodicité |
|---|---|---|
| Allumer chaque télé | 2h | 2h |
| Eteindre chaque télé | 2h | 2h |
| Activer/ désactiver CM | 1s | apériodique |
| Activer/ désactiver CL | 12h | 12h |
| Allumage urgence TV1 | 1min | Apériodique |
| Extinction d'urgence TV2 | 1min | Apériodique |

- Notion de priorité

1. Capteur de mouvement CM : priorité ultime
2. Capteur de luminosité CL est par exemple plus prioritaire face à afficher un message

# Montage




Composantes du montage :

1. ESP32 : Microcontrôleur
2. 3 LCD : Afficheur
3. CAPTEUR DE LUMINOSITE
4. CAPTEUR DE MOUVEMENT
5. LED : signal visuel.

## Spécification des composants

Pour expliquer le rôle de chaque composant dans le montage du système de gestion des téléviseurs dans une maison intelligente, nous pouvons détailler les fonctions de chacun :

1. **ESP32 (Microcontrôleur)** :
    - Le microcontrôleur ESP32 agit comme le cerveau du système. Il est responsable de recevoir les données des capteurs, de traiter ces données selon les règles définies, et de contrôler les actions des autres composants en conséquence.
    - Il communique avec les capteurs de luminosité et de mouvement pour surveiller l'environnement et détecter les changements dans la présence et la luminosité de la pièce.
    - En fonction des informations reçues des capteurs, l'ESP32 prend des décisions sur l'état des téléviseurs et contrôle les LCD et les LED pour afficher les informations pertinentes.
2. **3 LCD (Afficheur)** :
    - Les afficheurs LCD sont utilisés pour afficher des informations sur l'état du système et des téléviseurs. Par exemple, ils peuvent afficher si un téléviseur est allumé ou éteint, les conditions d'allumage et d'extinction, etc.
    - Ils permettent aux occupants de visualiser facilement l'état du système et de comprendre les actions en cours.
3. **Capteur de luminosité** :
    - Le capteur de luminosité est responsable de mesurer le niveau de luminosité dans la pièce.
    - Il transmet ces informations à l'ESP32, qui utilise ces données pour déterminer s'il fait jour ou nuit et ajuster en conséquence l'état des téléviseurs.
4. **Capteur de mouvement** :
    - Le capteur de mouvement détecte la présence ou l'absence de mouvement dans la pièce.
    - Il envoie ces informations à l'ESP32, qui utilise ces données pour activer ou désactiver les téléviseurs en fonction de la présence des occupants.
5. **LED (Signal visuel)** :
    - Les LED fournissent un signal visuel pour indiquer si un seuil spécifique est atteint ou dépassé.
    - Par exemple, une LED peut s'allumer pour indiquer que deux téléviseurs sont allumés simultanément, conformément à la contrainte du système.
    - Elles permettent aux utilisateurs de comprendre rapidement l'état du système sans avoir besoin de consulter les afficheurs LCD.

## Approche de conception logicielle

Le choix de la programmation concurrentielle s'est imposé pour répondre aux besoins multiples et simultanés de ce système, malgré l'utilisation d'un processeur monocœur. En gérant plusieurs tâches indépendantes en parallèle, cette approche permet une gestion efficace des différents aspects du système, tels que la surveillance de la luminosité, la gestion de l'allumage des télévisions et des écrans LCD, ainsi que la réactivité aux événements de mouvement. Cependant, l'utilisation d'un processeur monocœur pose une difficulté en limitant la possibilité d'exécuter véritablement plusieurs tâches simultanément. Malgré cette contrainte, la programmation concurrentielle isole les tâches, simplifiant ainsi le développement, la maintenance et le débogage du code, tout en exploitant efficacement les ressources matérielles disponibles pour optimiser les performances globales du système.

## Code de base et explications

```
#include <LiquidCrystal_I2C.h> // Inclut la bibliothèque LiquidCrystal_I2C

// Seuil de luminosité pour autoriser l'allumage des télévisions
#define SEUIL_LUMINOSITE 500 // Définit la valeur seuil de luminosité pour
allumer les télévisions

// Déclaration des pins pour les capteurs et actionneurs
#define LUMINOSITY_SENSOR_PIN 33 // Définit le pin pour le capteur de luminosité
#define MOTION_SENSOR_PIN 27 // Définit le pin pour le capteur de mouvement

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27; // Adresse I2C de l'écran LCD 1
const int LCD2_ADDRESS = 0x28; // Adresse I2C de l'écran LCD 2
const int LCD3_ADDRESS = 0x29; // Adresse I2C de l'écran LCD 3

// Déclaration des timers
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2; // Timer pour éteindre l'écran 1 en cas d'urgence

// Déclaration des objets LiquidCrystal_I2C pour chaque écran LCD
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2); // LCD 1 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2); // LCD 2 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2); // LCD 3 avec adresse, colonnes et
lignes spécifiées

// Déclaration de la file de messages pour les événements de luminosité
QueueHandle_t luminosityEventQueue; // File de messages pour les événements de
luminosité
QueueHandle_t motionEventQueue; // File de messages pour les événements de
mouvement

// Déclaration de la sémaphore multiple pour contrôler l'accès aux écrans
SemaphoreHandle_t tvSemaphore; // Sémaphore pour contrôler l'accès aux écrans


// Tâche de gestion de la luminosité
void taskLuminosity(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
```

```
    luminosity = analogRead(LUMINOSITY_SENSOR_PIN); // Lit la valeur de
luminosité à partir du capteur
    const float GAMMA = 0.7; // Définit la valeur de gamma pour la conversion de
la luminosité
    const float RL10 = 50; // Définit la valeur de RL10 pour la conversion de la
luminosité
    float voltage = luminosity / 1024.0 * 5.0; // Calcule la tension à partir de
la valeur de luminosité
    float resistance = 2000 * voltage / (1 - voltage / 5); // Calcule la
résistance à partir de la tension
    float luminosityValue = pow(RL10 * 1e3 * pow(10, GAMMA) / resistance, (1 /
GAMMA)); // Calcule la valeur de luminosité corrigée

    Serial.print("Luminosity: "); // Affiche le message "Luminosity: " sur le
moniteur série
    Serial.println(luminosityValue); // Affiche la valeur de luminosité corrigée
sur le moniteur série

    if (luminosityValue >= SEUIL_LUMINOSITE) { // Vérifie si la luminosité
dépasse le seuil

      Serial.println("Il fait jour ! "); // Affiche "Il fait jour ! " sur le
moniteur série

      xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY); // Envoie un
message à la file d'événements de luminosité
    } else { // Si la luminosité est inférieure au seuil
      Serial.println("Il fait Nuit ! "); // Affiche "Il fait Nuit ! " sur le
moniteur série
      lcd1.clear(); // Efface l'écran LCD 1
      lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
    }

    vTaskDelay(pdMS_TO_TICKS(1000)); // Attends 1 seconde
  }
}

// Fonction de rappel pour éteindre l'écran
void turnOffScreen1(TimerHandle_t xTimer) {
  lcd1.clear(); // Efface l'écran LCD 1
  lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
}
```

```
// Tâche de gestion de l'écran en cas d'allumage de la télévision
void taskTelevision1(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("TV1"); // Affiche "TV1" sur l'écran LCD 1
        xTimerStart(screenOffTimer1, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      }
    }
  }
}

// Routine de service d'interruption pour le capteur de mouvement
// Cette fonction doit être très courte car elle va interrompre le système.
// L'idée est que cette ISR (Routine de Service d'Interruption) envoie simplement
un
//signal à d'autres fonctions prioritaires (tâches normales mais avec priorité
haute)
// pour éviter de prendre trop de temps.
// Si un code long est placé dans cette fonction ISR, le système risque d'être
// interrompu pendant une longue période.
// Nous envoyons un signal rapidement à travers la file de messages
// depuis la fonction ISR, puis nous sortons de celle-ci.
// Les tâches qui ont besoin de ce signal doivent être programmées
//avec une priorité maximale.


void IRAM_ATTR motionSensorISR() {
  int danger = 1; // Déclare une variable pour indiquer le danger
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Déclare une variable pour
indiquer si une tâche à priorité supérieure a été réveillée
  xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken); //
Envoie un message à la file d'événements de mouvement depuis l'interruption
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken); // Indique que la tâche à
priorité supérieure doit être réveillée
```

```
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
éteint
void taskTelevision1_case1(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
       Serial.println("Motion Detected! Ecran 1 était dans l'état éteint"); //
Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur série


       lcd1.init(); // Initialise l'écran LCD 1
       lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
       lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
       lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
       xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}


// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
allumé
void taskTelevision1_case2(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
       Serial.println("Motion Detected! Ecran 1 était dans l'état allumé"); //
Affiche "Motion Detected! Ecran 1 était dans l'état allumé" sur le moniteur série
// Problème :
// Pourquoi ne pas créer qu'une seule fonction de gestion d'urgence ?
// Pourquoi séparer les deux cas ?
// Pourquoi ne pas simplement éteindre puis rallumer l'écran ?

// Explication :
// Dans notre cas actuel, cette approche fonctionne efficacement (pas besoin de
deux fonctions),
// car éteindre et rallumer un écran ne représente pas un coût significatif
// en termes de dépenses directes. Cependant, dans certains systèmes,
```

```
// une telle action peut être très coûteuse et entraîner des pertes
considérables.
// L'objectif ici est de sensibiliser à cette éventualité.

// Ainsi, nous avons créé deux fonctions pour gérer différemment les cas
d'urgence.
// Le cas d'urgence où l'écran 1 était éteint est moins grave, tandis que
// le cas d'urgence où l'écran 1 était allumé est plus grave
// car il nécessite à la fois l'extinction et l'allumage de l'écran (2 actions
nécessaires).

        lcd1.clear(); // Efface l'écran LCD 1
        lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}


void setup() {
  Serial.begin(9600); // Initialise la communication série avec une vitesse de
9600 bauds
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT); // Configure le pin du capteur de
luminosité en entrée
  pinMode(MOTION_SENSOR_PIN, INPUT); // Configure le pin du capteur de mouvement
en entrée

  luminosityEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file
d'événements de luminosité
  motionEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements
de mouvement

  tvSemaphore = xSemaphoreCreateCounting(1, 1); // Crée une sémaphore pour
contrôler l'accès aux écrans

  screenOffTimer1 = xTimerCreate("Timer Ecran 1", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
```

```
  screenOffTimer2 = xTimerCreate("Timer Ecran 1 urgence", pdMS_TO_TICKS(1000), pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 en cas d'urgence après 1 seconde

  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR, RISING); // Attache l'interruption au capteur de mouvement
  //Créeons les tâches sur un seul coeur ( le coeur 1) du processeur DUO_COEUR de ESP32. Nous devons programmés les tâches sur un seul coeur pour faire de la programmation concurentielle !
  xTaskCreatePinnedToCore(taskLuminosity, "Lire Luminosité", 1500, NULL, 2, NULL, 1); // Crée une tâche pour lire la luminosité
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 1", 1500, NULL, 1, NULL, 1); // Crée une tâche pour allumer la télévision 1
  xTaskCreatePinnedToCore(taskTelevision1_case1, "Urgence télé 1 cas 1", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque l'écran est éteint
  xTaskCreatePinnedToCore(taskTelevision1_case2, "Urgence télé 1 cas 2", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque l'écran est allumé
}

void loop() {
  // Vide. Les tâches et les timers gèrent la logique.
}
```

## Détails du code

Le code est conçu pour un système basé sur ESP32 avec plusieurs tâches fonctionnant de manière concurrentielle pour contrôler des écrans LCD et répondre à des événements de luminosité et de mouvement.

Voici un résumé de ce que fait chaque partie du code :

1. **Initialisation des composants et des structures de contrôle** :
    - Définition des adresses des écrans LCD et des pins pour les capteurs et actionneurs.
    - Création de files de messages pour les événements de luminosité et de mouvement.
    - Création d'une sémaphore pour contrôler l'accès aux écrans.
    - Création de timers pour éteindre les écrans après un certain délai.
2. **Tâche de gestion de la luminosité** :
    - Lit la valeur de luminosité à partir du capteur.
    - Effectue des calculs pour ajuster la valeur de luminosité.
    - Envoie un message à la file d'événements de luminosité si la luminosité dépasse un seuil.
    - Éteint l'écran LCD 1 si la luminosité est inférieure au seuil.
3. **Tâche de gestion de l'écran lors de l'allumage de la télévision** :
    - Attend les événements de luminosité depuis la file.
    - Prend la sémaphore pour contrôler l'accès à l'écran.
    - Allume l'écran LCD 1 et affiche un message si la luminosité dépasse le seuil.
    - Démarre un timer pour éteindre l'écran après un certain délai.
4. **Routine de service d'interruption pour le capteur de mouvement** :
    - Envoie un signal rapide à travers la file de messages depuis l'interruption.
    - Réveille les tâches qui ont besoin de ce signal.
5. **Tâches de gestion des événements de mouvement** :
    - Deux tâches distinctes sont utilisées pour gérer les événements de mouvement selon l'état de l'écran LCD 1 (allumé ou éteint).
    - Affiche un message d'urgence sur l'écran LCD 1 en cas de détection de mouvement.
    - Démarre un timer pour éteindre l'écran LCD 1 après un certain délai.
6. **Configuration dans la fonction setup()** :
    - Initialise les composants et les structures de contrôle.
    - Attache l'interruption du capteur de mouvement.
    - Crée et démarre les différentes tâches sur un seul cœur du processeur ESP32.

Ce code semble bien structuré pour gérer efficacement les événements de luminosité et de mouvement tout en contrôlant l'accès aux écrans LCD de manière concurrentielle

# Simulation Wokwi et programmation concurrentielle

Nous avons simulé le comportement sur la plateforme Wokwi pour valider son fonctionnement en conditions réelles. Les résultats de la simulation ont confirmé l'efficacité de notre approche de gestion automatisée des téléviseurs.

## TEST 0

Résultat : uniquement TV1 ON . ET LES DEUX AUTRES SONT OFF.

Solution : peut-être qu'il serait judicieux de ne créer qu'une seule tache pour la gestion des trois télés ??

ON L'essaye dans le code qui va suivre celui-ci

```
#include <LiquidCrystal_I2C.h> // Inclut la bibliothèque LiquidCrystal_I2C

// Seuil de luminosité pour autoriser l'allumage des télévisions
#define SEUIL_LUMINOSITE 500 // Définit la valeur seuil de luminosité pour
allumer les télévisions

// Déclaration du pin pour la LED d'indication de seuil dépassé
#define LED_PIN 23


// Nombre total de télévisions
#define TOTAL_LCD 3

// Compteur pour le nombre de télévisions allumées pendant la journée
int tvCount = 0;


// Déclaration des pins pour les capteurs et actionneurs
#define LUMINOSITY_SENSOR_PIN 33 // Définit le pin pour le capteur de luminosité
#define MOTION_SENSOR_PIN 27 // Définit le pin pour le capteur de mouvement

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27; // Adresse I2C de l'écran LCD 1
const int LCD2_ADDRESS = 0x28; // Adresse I2C de l'écran LCD 2
const int LCD3_ADDRESS = 0x29; // Adresse I2C de l'écran LCD 3

// États des écrans LCD
bool lcdState[TOTAL_LCD] = {false}; // false signifie éteint, true signifie
allumé
```

```
int numLcdsOn = 0; // Nombre de télévisions actuellement allumées

// Déclaration des timers
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2; // Timer pour éteindre l'écran 1 en cas d'urgence
TimerHandle_t screenOffTimer3; // Timer pour éteindre l'écran 2
TimerHandle_t screenOffTimer4; // Timer pour éteindre l'écran 3

// Déclaration des objets LiquidCrystal_I2C pour chaque écran LCD
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2); // LCD 1 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2); // LCD 2 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2); // LCD 3 avec adresse, colonnes et
lignes spécifiées

// Déclaration de la file de messages pour les événements de luminosité
QueueHandle_t luminosityEventQueue; // File de messages pour les événements de
luminosité
QueueHandle_t motionEventQueue; // File de messages pour les événements de
mouvement

// Déclaration de la sémaphore multiple pour contrôler l'accès aux écrans
SemaphoreHandle_t tvSemaphore; // Sémaphore pour contrôler l'accès aux écrans


// Tâche de gestion de la luminosité
void taskLuminosity(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  int SEUIL_LUMINOSITEValue;
  while (1) { // Boucle infinie
    luminosity = analogRead(LUMINOSITY_SENSOR_PIN); // Lit la valeur de
luminosité à partir du capteur
    const float GAMMA = 0.7; // Définit la valeur de gamma pour la conversion de
la luminosité
    const float RL10 = 50; // Définit la valeur de RL10 pour la conversion de la
luminosité
    float voltage = luminosity / 1024.0 * 5.0; // Calcule la tension à partir de
la valeur de luminosité
    float resistance = 2000 * voltage / (1 - voltage / 5); // Calcule la
résistance à partir de la tension
    float luminosityValue = pow(RL10 * 1e3 * pow(10, GAMMA) / resistance, (1 /
GAMMA)); // Calcule la valeur de luminosité corrigée
    delay(1000);
```

```
    Serial.print("Luminosity: "); // Affiche le message "Luminosity: " sur le
moniteur série
    Serial.println(luminosityValue); // Affiche la valeur de luminosité corrigée
sur le moniteur série

    // Attente avant la lecture suivante
    delay(500); // Attendre 1 seconde avant de lire à nouveau


    if (luminosityValue >= SEUIL_LUMINOSITE) { // Vérifie si la luminosité
dépasse le seuil


      Serial.println("Il fait jour ! "); // Affiche "Il fait jour ! " sur le
moniteur série
      digitalWrite(LED_PIN, LOW); // Eteint la LED d'indication de seuil dépassé
      xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY); // Envoie un
message à la file d'événements de luminosité

    }
     else { // Si la luminosité est inférieure au seuil
      digitalRead(luminosityValue);
      Serial.println(luminosityValue);
      Serial.println("Il fait Nuit ! "); // Affiche "Il fait Nuit ! " sur le
moniteur série
      digitalWrite(LED_PIN, HIGH); // Allume la LED d'indication de seuil dépassé
      lcd1.clear(); // Efface l'écran LCD 1
      lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd2.clear(); // Efface l'écran LCD 2
      lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd3.clear(); // Efface l'écran LCD 3
      lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
        }

    vTaskDelay(pdMS_TO_TICKS(500)); // Attends 1 seconde
  }
}

// Fonction pour vérifier l'état des télévisions et n'allumer que 2 Télés
simultanément
```

```
bool checkAndTurnOnTV() {
  if (numLcdsOn < 2) { // Vérifie si le nombre de télévisions actuellement allumées est inférieur à 2
    numLcdsOn++; // Incrémente le nombre de télévisions allumées
    return true; // Retourne true pour indiquer que la télévision peut être allumée
  }
  return false; // Retourne false pour indiquer que la télévision ne peut pas être allumée
}

// Fonction pour éteindre une télévision
void turnOffTV() {
  numLcdsOn--; // Décrémente le nombre de télévisions allumées
}

// Fonction de rappel pour éteindre l'écran
void turnOffScreen1(TimerHandle_t xTimer) {
  lcd1.clear(); // Efface l'écran LCD 1
  lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à d'autres tâches
  turnOffTV(); // Éteint la télévision
}


// Tâche de gestion de l'écran en cas d'allumage de la télévision 1
void taskTelevision1(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) == pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de prendre la sémaphore pour contrôler l'accès à l'écran
        if (checkAndTurnOnTV()) { // Vérifie si une télévision peut être allumée
         if (numLcdsOn < 2) {
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("TV1"); // Affiche "TV1" sur l'écran LCD 1
        xTimerStart(screenOffTimer1, 0); // Démarre le timer pour éteindre l'écran LCD 1 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(250)); // Attend 2 secondes
        numLcdsOn++; // Mettre à jour le nombre de télévisions allumées
        // Rendre la sémaphore
```

```
        xSemaphoreGive(tvSemaphore);

        }
         }
      }
    }
  }
}


// Routine de service d'interruption pour le capteur de mouvement
// Cette fonction doit être très courte car elle va interrompre le système.
// L'idée est que cette ISR (Routine de Service d'Interruption) envoie simplement
un
//signal à d'autres fonctions prioritaires (tâches normales mais avec priorité
haute)
// pour éviter de prendre trop de temps.
// Si un code long est placé dans cette fonction ISR, le système risque d'être
// interrompu pendant une longue période.
// Nous envoyons un signal rapidement à travers la file de messages
// depuis la fonction ISR, puis nous sortons de celle-ci.
// Les tâches qui ont besoin de ce signal doivent être programmées
//avec une priorité maximale.


void IRAM_ATTR motionSensorISR() {
  int danger = 1; // Déclare une variable pour indiquer le danger
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Déclare une variable pour
indiquer si une tâche à priorité supérieure a été réveillée
  xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken); //
Envoie un message à la file d'événements de mouvement depuis l'interruption
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken); // Indique que la tâche à
priorité supérieure doit être réveillée
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
éteint
void taskTelevision1_case1(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
       Serial.println("Motion Detected! Ecran 1 était dans l'état éteint"); //
Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur série
```

```
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur sur l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
    }
  }
}


// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
allumé
void taskTelevision1_case2(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état allumé"); //
Affiche "Motion Detected! Ecran 1 était dans l'état allumé" sur le moniteur série
// Problème :
// Pourquoi ne pas créer qu'une seule fonction de gestion d'urgence ?
// Pourquoi séparer les deux cas ?
// Pourquoi ne pas simplement éteindre puis rallumer l'écran ?

// Explication :
// Dans notre cas actuel, cette approche fonctionne efficacement (pas besoin de
deux fonctions),
// car éteindre et rallumer un écran ne représente pas un coût significatif
// en termes de dépenses directes. Cependant, dans certains systèmes,
// une telle action peut être très coûteuse et entraîner des pertes
considérables.
// L'objectif ici est de sensibiliser à cette éventualité.

// Ainsi, nous avons créé deux fonctions pour gérer différemment les cas
d'urgence.
// Le cas d'urgence où l'écran 1 était éteint est moins grave, tandis que
// le cas d'urgence où l'écran 1 était allumé est plus grave
// car il nécessite à la fois l'extinction et l'allumage de l'écran (2 actions
nécessaires).

        lcd1.clear(); // Efface l'écran LCD 1
        lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
```

```
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}

// Fonction de rappel pour éteindre l'écran 2
void turnOffScreen2(TimerHandle_t xTimer){
  lcd2.clear(); // Efface l'écran LCD2
  lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches

}

// Tache de gestion de l'écran en cas d'allumage de la télévision 2
void taskTelevision2(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS){ // Vérifie s'il y a un événement
     if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
       if (checkAndTurnOnTV()) {
        if (numLcdsOn < 2) {
       lcd2.init(); // Initialise l'écran LCD 2
       lcd2.backlight(); // Allume le rétroéclairage de l'écran LCD 2
       lcd2.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 2
       lcd2.print("TV2"); // Affiche "TV2" sur l'écran LCD 2
       xTimerStart(screenOffTimer3, 0); // Démarre le timer pour éteindre l'écran
LCD 2 après un certain délai
       vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
       numLcdsOn++;
       xSemaphoreGive(tvSemaphore); // Rendre la sémaphore
       }
     }
    }
  }
}
}
```

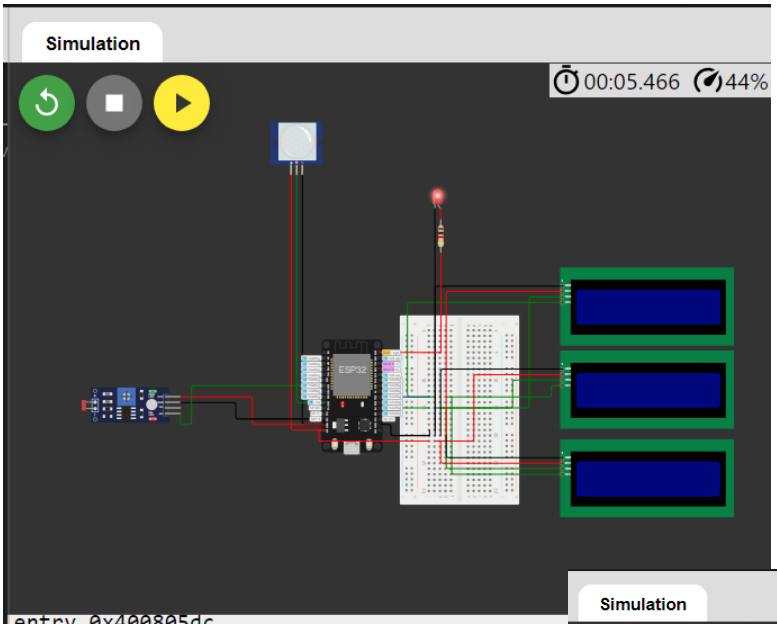
```
// Fonction de rappel pour éteindre l'écran 3
void turnOffScreen3(TimerHandle_t xTimer){
  lcd3.clear(); // Efface l'écran LCD 3
  lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches


}

// Tache de gestion de l'écran en cas d'allumage de la télévision 3
void taskTelevision3(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
   if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) == pdPASS)
{ // Vérifie s'il y a un événement
    if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
      if (checkAndTurnOnTV()) {
        if (numLcdsOn < 2) {
      lcd3.init(); // Initialise l'écran LCD 3
      lcd3.backlight(); // Allume le rétroéclairage de l'écran LDC 3
      lcd3.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 3
      lcd3.print("TV3"); // Affiche "TV3" sur l'écran LCD 3
      xTimerStart(screenOffTimer4, 0); // Démarre le timer pour éteindre l'écran
LCD 3 après un certain délai
      vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
      numLcdsOn++;
      xSemaphoreGive(tvSemaphore); // Rendre la sémaphore
    }
   }
  }
 }
}
}

void setup() {
  Serial.begin(9600); // Initialise la communication série avec une vitesse de
9600 bauds
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT); // Configure le pin du capteur de
luminosité en entrée
  pinMode(MOTION_SENSOR_PIN, INPUT); // Configure le pin du capteur de mouvement
en entrée
  pinMode(LED_PIN, OUTPUT); // Configure le pin pour la LED en sortie
```

```
  luminosityEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file
d'événements de luminosité
  motionEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements
de mouvement

  // Initialisation de la sémaphore avec deux tokens
  tvSemaphore = xSemaphoreCreateCounting(2, 2); // Crée une sémaphore pour
contrôler l'accès aux écrans

  screenOffTimer1 = xTimerCreate("Timer Ecran 1", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
  screenOffTimer3 = xTimerCreate("Timer Ecran 2", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen2); // Crée un timer pour éteindre l'écran LCD 2 après 5 secondes
  screenOffTimer4 = xTimerCreate("Timer Ecran 3", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen3); // Crée un timer pour éteindre l'écran LCD 3 après 5 secondes
  screenOffTimer2 = xTimerCreate("Timer Ecran 1 urgence", pdMS_TO_TICKS(1000),
pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 en cas
d'urgence après 1 seconde

  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR,
RISING); // Attache l'interruption au capteur de mouvement
  //Créeons les tâches sur un seul coeur ( le coeur 1) du processeur DUO_COEUR de
ESP32. Nous devons programmés les tâches sur un seul coeur pour faire de la
programmation concurentielle !
  xTaskCreatePinnedToCore(taskLuminosity, "Lire Luminosité", 1500, NULL, 2, NULL,
1); // Crée une tâche pour lire la luminosité
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 1", 1500, NULL, 1, NULL,
1); // Crée une tâche pour allumer la télévision 1
  xTaskCreatePinnedToCore(taskTelevision1_case1, "Urgence télé 1 cas 1", 1500,
NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement
lorsque l'écran est éteint
  xTaskCreatePinnedToCore(taskTelevision1_case2, "Urgence télé 1 cas 2", 1500,
NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement
lorsque l'écran est allumé
}

void loop() {
  // Vide. Les tâches et les timers gèrent la logique.
 // Le programme ne fait rien dans la boucle loop() car la logique est gérée par
les tâches
}
```

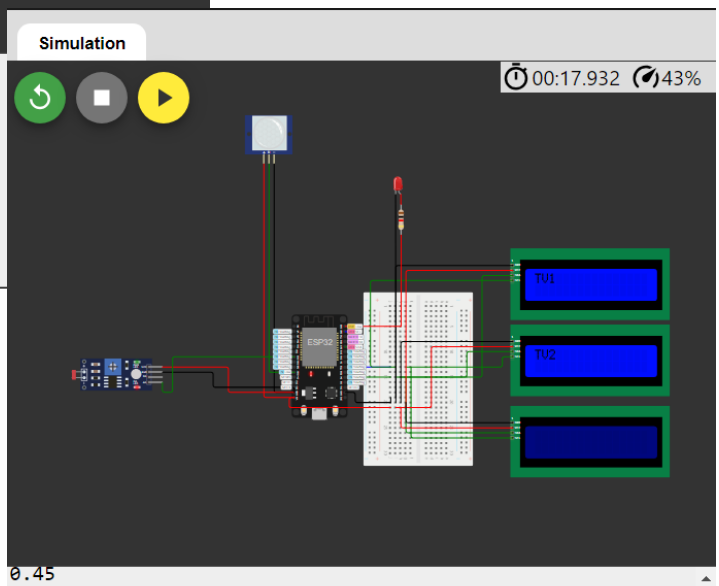
## TEST 1

Ici uniquement deux écrans s'allument de façon décalée.

ETAT 1 : IL FAIT NUIT, LED DE SIGNALISATION ALLUME




```
entry 0x400805dc
Luminosity: 0.45
0.45
Il fait Nuit !
Luminosity: 0.45
0.45
Il fait Nuit !
```




```
0.45
Il fait Nuit !
Luminosity: 4887.37
Il fait jour !
Luminosity: 3322.19
Il fait jour !
Luminosity: 3322.19
```

ETAT 2 : IL FAIT JOUR . VOILA CE QUI SE PASSE

```
#include <LiquidCrystal_I2C.h> // Inclut la bibliothèque LiquidCrystal_I2C

// Seuil de luminosité pour autoriser l'allumage des télévisions
#define SEUIL_LUMINOSITE 500 // Définit la valeur seuil de luminosité pour
allumer les télévisions

// Déclaration du pin pour la LED d'indication de seuil dépassé
#define LED_PIN 23

// Nombre total de télévisions
#define TOTAL_LCD 3

// Compteur pour le nombre de télévisions allumées pendant la journée
int tvCount = 0;

// Déclaration des pins pour les capteurs et actionneurs
#define LUMINOSITY_SENSOR_PIN 33 // Définit le pin pour le capteur de luminosité
#define MOTION_SENSOR_PIN 27 // Définit le pin pour le capteur de mouvement

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27; // Adresse I2C de l'écran LCD 1
const int LCD2_ADDRESS = 0x28; // Adresse I2C de l'écran LCD 2
const int LCD3_ADDRESS = 0x29; // Adresse I2C de l'écran LCD 3

// États des écrans LCD
bool lcdState[TOTAL_LCD] = {false}; // false signifie éteint, true signifie
allumé
int numLcdsOn = 0; // Nombre de télévisions actuellement allumées

// Déclaration des timers
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2; // Timer pour éteindre l'écran 1 en cas d'urgence
TimerHandle_t screenOffTimer3; // Timer pour éteindre l'écran 2
TimerHandle_t screenOffTimer4; // Timer pour éteindre l'écran 3

// Déclaration des objets LiquidCrystal_I2C pour chaque écran LCD
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2); // LCD 1 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2); // LCD 2 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2); // LCD 3 avec adresse, colonnes et
lignes spécifiées

// Déclaration de la file de messages pour les événements de luminosité
```

```
QueueHandle_t luminosityEventQueue; // File de messages pour les événements de luminosité
QueueHandle_t motionEventQueue; // File de messages pour les événements de mouvement

// Déclaration de la sémaphore multiple pour contrôler l'accès aux écrans
SemaphoreHandle_t tvSemaphore; // Sémaphore pour contrôler l'accès aux écrans

// Tâche de gestion de la luminosité
void taskLuminosity(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  int SEUIL_LUMINOSITEValue;
  while (1) { // Boucle infinie
    luminosity = analogRead(LUMINOSITY_SENSOR_PIN); // Lit la valeur de luminosité à partir du capteur
    const float GAMMA = 0.7; // Définit la valeur de gamma pour la conversion de la luminosité
    const float RL10 = 50; // Définit la valeur de RL10 pour la conversion de la luminosité
    float voltage = luminosity / 1024.0 * 5.0; // Calcule la tension à partir de la valeur de luminosité
    float resistance = 2000 * voltage / (1 - voltage / 5); // Calcule la résistance à partir de la tension
    float luminosityValue = pow(RL10 * 1e3 * pow(10, GAMMA) / resistance, (1 / GAMMA)); // Calcule la valeur de luminosité corrigée
    delay(1000);
    Serial.print("Luminosity: "); // Affiche le message "Luminosity: " sur le moniteur série
    Serial.println(luminosityValue); // Affiche la valeur de luminosité corrigée sur le moniteur série

    // Attente avant la lecture suivante
    delay(500); // Attendre 1 seconde avant de lire à nouveau


    if (luminosityValue >= SEUIL_LUMINOSITE) { // Vérifie si la luminosité dépasse le seuil


      Serial.println("Il fait jour ! "); // Affiche "Il fait jour ! " sur le moniteur série
      digitalWrite(LED_PIN, LOW); // Eteint la LED d'indication de seuil dépassé
      xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY); // Envoie un message à la file d'événements de luminosité
```

```
    }
     else { // Si la luminosité est inférieure au seuil
      digitalRead(luminosityValue);
      Serial.println(luminosityValue);
      Serial.println("Il fait Nuit ! "); // Affiche "Il fait Nuit ! " sur le
moniteur série
      digitalWrite(LED_PIN, HIGH); // Allume la LED d'indication de seuil dépassé
      lcd1.clear(); // Efface l'écran LCD 1
      lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd2.clear(); // Efface l'écran LCD 2
      lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd3.clear(); // Efface l'écran LCD 3
      lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
        }

    vTaskDelay(pdMS_TO_TICKS(500)); // Attends 1 seconde
  }
}

// Fonction pour vérifier l'état des télévisions et n'allumer que 2 Télés
simultanément
bool checkAndTurnOnTV() {
  if (numLcdsOn < 3) { // Vérifie si le nombre de télévisions actuellement
allumées est inférieur à 2
    numLcdsOn++; // Incrémente le nombre de télévisions allumées
    return true; // Retourne true pour indiquer que la télévision peut être
allumée
  }
  return false; // Retourne false pour indiquer que la télévision ne peut pas
être allumée
}

// Fonction pour éteindre une télévision
void turnOffTV() {
  numLcdsOn--; // Décrémente le nombre de télévisions allumées
}

// Fonction de rappel pour éteindre l'écran
void turnOffScreen1(TimerHandle_t xTimer) {
  lcd1.clear(); // Efface l'écran LCD 1
```

```
  lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
  turnOffTV(); // Éteint la télévision
}

// Tâche de gestion de l'écran en cas d'allumage de la télévision 1
void taskTelevision1(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        if (checkAndTurnOnTV()) { // Vérifie si une télévision peut être allumée
         if (numLcdsOn < 2) {
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("TV1"); // Affiche "TV1" sur l'écran LCD 1
        xTimerStart(screenOffTimer1, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(50)); // Attend 2 secondes
        numLcdsOn++; // Mettre à jour le nombre de télévisions allumées

        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
       } else if (!lcdState[1]){
          // Allumer la télévision 2 si elle n'est pas déjà allumée
          lcd2.init();
          lcd2.backlight();
          lcd2.setCursor(0, 0);
          lcd2.print("TV2");
          xTimerStart(screenOffTimer3, 0);
          vTaskDelay(pdMS_TO_TICKS(50));
          numLcdsOn++;

        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
       } else if (!lcdState[2]){
          // Allumer la télévision 3 si elle n'est pas déjà allumée
          lcd3.init();
          lcd3.backlight();
          lcd3.setCursor(0, 0);
          lcd3.print("TV3");
```

```
          xTimerStart(screenOffTimer4, 0);
          vTaskDelay(pdMS_TO_TICKS(250));
          numLcdsOn++;

        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
        }
         }
      }
    }
  }
}

// Routine de service d'interruption pour le capteur de mouvement
// Cette fonction doit être très courte car elle va interrompre le système.
// L'idée est que cette ISR (Routine de Service d'Interruption) envoie simplement
un
//signal à d'autres fonctions prioritaires (tâches normales mais avec priorité
haute)
// pour éviter de prendre trop de temps.
// Si un code long est placé dans cette fonction ISR, le système risque d'être
// interrompu pendant une longue période.
// Nous envoyons un signal rapidement à travers la file de messages
// depuis la fonction ISR, puis nous sortons de celle-ci.
// Les tâches qui ont besoin de ce signal doivent être programmées
//avec une priorité maximale.

void IRAM_ATTR motionSensorISR() {
  int danger = 1; // Déclare une variable pour indiquer le danger
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Déclare une variable pour
indiquer si une tâche à priorité supérieure a été réveillée
  xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken); //
Envoie un message à la file d'événements de mouvement depuis l'interruption
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken); // Indique que la tâche à
priorité supérieure doit être réveillée
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
éteint
void taskTelevision1_case1(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
```

```
        Serial.println("Motion Detected! Ecran 1 était dans l'état éteint"); //
Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur série

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur sur l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
    }
  }
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
allumé
void taskTelevision1_case2(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état allumé"); //
Affiche "Motion Detected! Ecran 1 était dans l'état allumé" sur le moniteur série
// Problème :
// Pourquoi ne pas créer qu'une seule fonction de gestion d'urgence ?
// Pourquoi séparer les deux cas ?
// Pourquoi ne pas simplement éteindre puis rallumer l'écran ?

// Explication :
// Dans notre cas actuel, cette approche fonctionne efficacement (pas besoin de
deux fonctions),
// car éteindre et rallumer un écran ne représente pas un coût significatif
// en termes de dépenses directes. Cependant, dans certains systèmes,
// une telle action peut être très coûteuse et entraîner des pertes
considérables.
// L'objectif ici est de sensibiliser à cette éventualité.

// Ainsi, nous avons créé deux fonctions pour gérer différemment les cas
d'urgence.
// Le cas d'urgence où l'écran 1 était éteint est moins grave, tandis que
// le cas d'urgence où l'écran 1 était allumé est plus grave
// car il nécessite à la fois l'extinction et l'allumage de l'écran (2 actions
nécessaires).

        lcd1.clear(); // Efface l'écran LCD 1
```

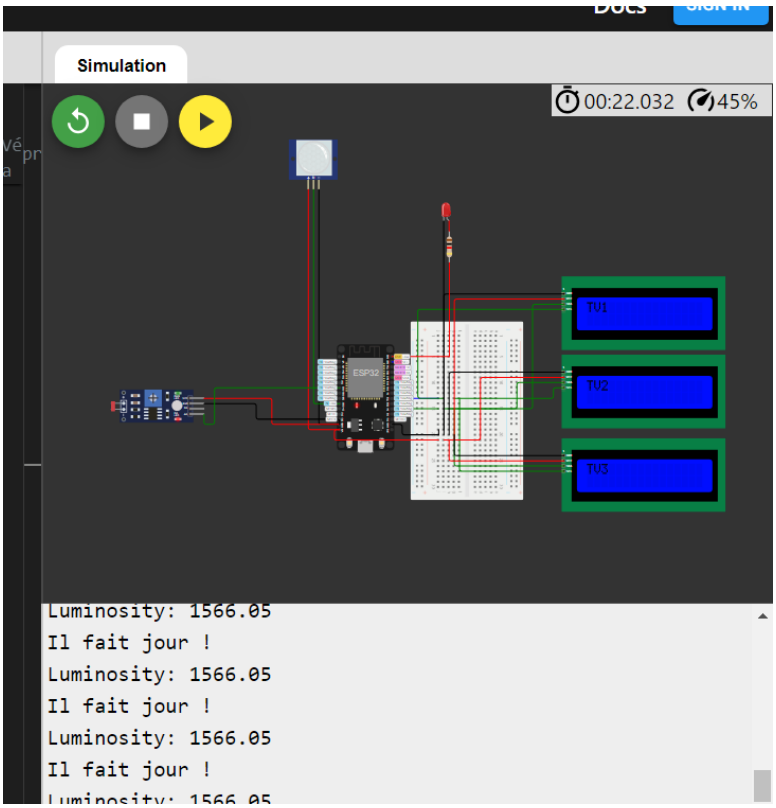
```
        lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}

// Fonction de rappel pour éteindre l'écran 2
void turnOffScreen2(TimerHandle_t xTimer){
  lcd2.clear(); // Efface l'écran LCD2
  lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches

}

// Tache de gestion de l'écran en cas d'allumage de la télévision 2
void taskTelevision2(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS){ // Vérifie s'il y a un événement
     if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
       if (checkAndTurnOnTV()) {
       lcd2.init(); // Initialise l'écran LCD 2
       lcd2.backlight(); // Allume le rétroéclairage de l'écran LCD 2
       lcd2.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 2
       lcd2.print("TV2"); // Affiche "TV2" sur l'écran LCD 2
       xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre l'écran
LCD 2 après un certain délai
       vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
       }
     }
    }
  }
}

// Fonction de rappel pour éteindre l'écran 3
```

```
void turnOffScreen3(TimerHandle_t xTimer){
  lcd3.clear(); // Efface l'écran LCD 3
  lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches

}

// Tache de gestion de l'écran en cas d'allumage de la télévision 3
void taskTelevision3(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
   if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) == pdPASS)
{ // Vérifie s'il y a un événement
    if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
      lcd3.init(); // Initialise l'écran LCD 3
      lcd3.backlight(); // Allume le rétroéclairage de l'écran LDC 3
      lcd3.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 3
      lcd3.print("TV3"); // Affiche "TV3" sur l'écran LCD 3
      xTimerStart(screenOffTimer3, 0); // Démarre le timer pour éteindre l'écran
LCD 3 après un certain délai
      vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes

    }
   }
  }
 }

void setup() {
  Serial.begin(9600); // Initialise la communication série avec une vitesse de
9600 bauds
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT); // Configure le pin du capteur de
luminosité en entrée
  pinMode(MOTION_SENSOR_PIN, INPUT); // Configure le pin du capteur de mouvement
en entrée
  pinMode(LED_PIN, OUTPUT); // Configure le pin pour la LED en sortie
  luminosityEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file
d'événements de luminosité
  motionEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements
de mouvement

  // Initialisation de la sémaphore avec deux tokens
  tvSemaphore = xSemaphoreCreateCounting(3, 3); // Crée une sémaphore pour
contrôler l'accès aux écrans
```

```
  screenOffTimer1 = xTimerCreate("Timer Ecran 1", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
  screenOffTimer3 = xTimerCreate("Timer Ecran 2", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen2); // Crée un timer pour éteindre l'écran LCD 2 après 5 secondes
  screenOffTimer4 = xTimerCreate("Timer Ecran 3", pdMS_TO_TICKS(5000), pdFALSE,
0, turnOffScreen3); // Crée un timer pour éteindre l'écran LCD 3 après 5 secondes
  screenOffTimer2 = xTimerCreate("Timer Ecran 1 urgence", pdMS_TO_TICKS(1000),
pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 en cas
d'urgence après 1 seconde

  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR,
RISING); // Attache l'interruption au capteur de mouvement
  //Créeons les tâches sur un seul coeur ( le coeur 1) du processeur DUO_COEUR de
ESP32. Nous devons programmés les tâches sur un seul coeur pour faire de la
programmation concurentielle !
  xTaskCreatePinnedToCore(taskLuminosity, "Lire Luminosité", 1500, NULL, 2, NULL,
1); // Crée une tâche pour lire la luminosité
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 1", 1500, NULL, 1, NULL,
1); // Crée une tâche pour allumer la télévision 1
  xTaskCreatePinnedToCore(taskTelevision1_case1, "Urgence télé 1 cas 1", 1500,
NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement
lorsque l'écran est éteint
  xTaskCreatePinnedToCore(taskTelevision1_case2, "Urgence télé 1 cas 2", 1500,
NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement
lorsque l'écran est allumé
}

void loop() {
  // Vide. Les tâches et les timers gèrent la logique.
 // Le programme ne fait rien dans la boucle loop() car la logique est gérée par
les tâches
}
```

## TEST 2

Pour ce code, nous remarquons que les ressources ne sont en aucun cas protéger ce qui veut dire la sémaphore n'a pas d'impact sur le code.




```
#include <LiquidCrystal_I2C.h> // Inclut la bibliothèque LiquidCrystal_I2C

// Seuil de luminosité pour autoriser l'allumage des télévisions
#define SEUIL_LUMINOSITE 500 // Définit la valeur seuil de luminosité pour
allumer les télévisions

// Déclaration du pin pour la LED d'indication de seuil dépassé
#define LED_PIN 23

// Nombre total de télévisions
#define TOTAL_LCD 3
```

```
// Compteur pour le nombre de télévisions allumées pendant la journée
int tvCount = 0;

// Déclaration des pins pour les capteurs et actionneurs
#define LUMINOSITY_SENSOR_PIN 33 // Définit le pin pour le capteur de luminosité
#define MOTION_SENSOR_PIN 27 // Définit le pin pour le capteur de mouvement

// I2C LCD display addresses
const int LCD1_ADDRESS = 0x27; // Adresse I2C de l'écran LCD 1
const int LCD2_ADDRESS = 0x28; // Adresse I2C de l'écran LCD 2
const int LCD3_ADDRESS = 0x29; // Adresse I2C de l'écran LCD 3

// États des écrans LCD
bool lcdState[TOTAL_LCD] = {false}; // false signifie éteint, true signifie
allumé
int numLcdsOn = 0; // Nombre de télévisions actuellement allumées

// Déclaration des timers
TimerHandle_t screenOffTimer1; // Timer pour éteindre l'écran 1
TimerHandle_t screenOffTimer2; // Timer pour éteindre l'écran 1 en cas d'urgence
TimerHandle_t screenOffTimer3; // Timer pour éteindre l'écran 2
TimerHandle_t screenOffTimer4; // Timer pour éteindre l'écran 3

// Déclaration des objets LiquidCrystal_I2C pour chaque écran LCD
LiquidCrystal_I2C lcd1(LCD1_ADDRESS, 16, 2); // LCD 1 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd2(LCD2_ADDRESS, 16, 2); // LCD 2 avec adresse, colonnes et
lignes spécifiées
LiquidCrystal_I2C lcd3(LCD3_ADDRESS, 16, 2); // LCD 3 avec adresse, colonnes et
lignes spécifiées

// Déclaration de la file de messages pour les événements de luminosité
QueueHandle_t luminosityEventQueue; // File de messages pour les événements de
luminosité
QueueHandle_t motionEventQueue; // File de messages pour les événements de
mouvement

// Déclaration de la sémaphore multiple pour contrôler l'accès aux écrans
SemaphoreHandle_t tvSemaphore; // Sémaphore pour contrôler l'accès aux écrans

// Tâche de gestion de la luminosité
void taskLuminosity(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  int SEUIL_LUMINOSITEValue;
```

```
  while (1) { // Boucle infinie
    luminosity = analogRead(LUMINOSITY_SENSOR_PIN); // Lit la valeur de
luminosité à partir du capteur
    const float GAMMA = 0.7; // Définit la valeur de gamma pour la conversion de
la luminosité
    const float RL10 = 50; // Définit la valeur de RL10 pour la conversion de la
luminosité
    float voltage = luminosity / 1024.0 * 5.0; // Calcule la tension à partir de
la valeur de luminosité
    float resistance = 2000 * voltage / (1 - voltage / 5); // Calcule la
résistance à partir de la tension
    float luminosityValue = pow(RL10 * 1e3 * pow(10, GAMMA) / resistance, (1 /
GAMMA)); // Calcule la valeur de luminosité corrigée
    delay(1000);
    Serial.print("Luminosity: "); // Affiche le message "Luminosity: " sur le
moniteur série
    Serial.println(luminosityValue); // Affiche la valeur de luminosité corrigée
sur le moniteur série

    // Attente avant la lecture suivante
    delay(500); // Attendre 1 seconde avant de lire à nouveau


    if (luminosityValue >= SEUIL_LUMINOSITE) { // Vérifie si la luminosité
dépasse le seuil


      Serial.println("Il fait jour ! "); // Affiche "Il fait jour ! " sur le
moniteur série
      digitalWrite(LED_PIN, LOW); // Eteint la LED d'indication de seuil dépassé
      xQueueSend(luminosityEventQueue, &luminosity, portMAX_DELAY); // Envoie un
message à la file d'événements de luminosité

    }
     else { // Si la luminosité est inférieure au seuil
      digitalRead(luminosityValue);
      Serial.println(luminosityValue);
      Serial.println("Il fait Nuit ! "); // Affiche "Il fait Nuit ! " sur le
moniteur série
      digitalWrite(LED_PIN, HIGH); // Allume la LED d'indication de seuil dépassé
      lcd1.clear(); // Efface l'écran LCD 1
      lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd2.clear(); // Efface l'écran LCD 2
```

```
      lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore

      lcd3.clear(); // Efface l'écran LCD 3
      lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
      xSemaphoreGive(tvSemaphore); // Libère la sémaphore
        }

    vTaskDelay(pdMS_TO_TICKS(500)); // Attends 1 seconde
  }
}

// Fonction pour vérifier l'état des télévisions et n'allumer que 2 Télés
simultanément
bool checkAndTurnOnTV() {
  if (numLcdsOn < 3) { // Vérifie si le nombre de télévisions actuellement
allumées est inférieur à 2
    numLcdsOn++; // Incrémente le nombre de télévisions allumées
    return true; // Retourne true pour indiquer que la télévision peut être
allumée
  }
  return false; // Retourne false pour indiquer que la télévision ne peut pas
être allumée
}

// Fonction pour éteindre une télévision
void turnOffTV() {
  numLcdsOn--; // Décrémente le nombre de télévisions allumées
}

// Fonction de rappel pour éteindre l'écran
void turnOffScreen1(TimerHandle_t xTimer) {
  lcd1.clear(); // Efface l'écran LCD 1
  lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches
  turnOffTV(); // Éteint la télévision
}

// Tâche de gestion de l'écran en cas d'allumage de la télévision 1
void taskTelevision1(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS) { // Vérifie s'il y a un événement de luminosité dans la file
```

```
      if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour contrôler l'accès à l'écran
        if (checkAndTurnOnTV()) { // Vérifie si une télévision peut être allumée
         if (numLcdsOn < 2) {
        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("TV1"); // Affiche "TV1" sur l'écran LCD 1
        xTimerStart(screenOffTimer1, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
        vTaskDelay(pdMS_TO_TICKS(25)); // Attend 2 secondes
        numLcdsOn++; // Mettre à jour le nombre de télévisions allumées

        lcd2.init();
        lcd2.backlight();
        lcd2.setCursor(0, 0);
        lcd2.print("TV2");
        xTimerStart(screenOffTimer3, 0);
        vTaskDelay(pdMS_TO_TICKS(25));
        numLcdsOn++;

        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
       } else if (!lcdState[1]){
          // Allumer la télévision 2 si elle n'est pas déjà allumée
          lcd2.init();
          lcd2.backlight();
          lcd2.setCursor(0, 0);
          lcd2.print("TV2");
          xTimerStart(screenOffTimer3, 0);
          vTaskDelay(pdMS_TO_TICKS(25));
          numLcdsOn++;

          lcd3.init();
          lcd3.backlight();
          lcd3.setCursor(0, 0);
          lcd3.print("TV3");
          xTimerStart(screenOffTimer4, 0);
          vTaskDelay(pdMS_TO_TICKS(25));
          numLcdsOn++;

          lcd1.init();
          lcd1.backlight();
          lcd1.setCursor(0, 0);
          lcd1.print("TV1");
```

```
          xTimerStart(screenOffTimer1, 0);
          vTaskDelay(pdMS_TO_TICKS(25));
          numLcdsOn++;

        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
       } else if (!lcdState[2]){
          // Allumer la télévision 3 si elle n'est pas déjà allumée
          lcd3.init();
          lcd3.backlight();
          lcd3.setCursor(0, 0);
          lcd3.print("TV3");
          xTimerStart(screenOffTimer4, 0);
          vTaskDelay(pdMS_TO_TICKS(25));
          numLcdsOn++;

          lcd1.init();
          lcd1.backlight();
          lcd1.setCursor(0, 0);
          lcd1.print("TV1");
          xTimerStart(screenOffTimer1, 0);
          vTaskDelay(pdMS_TO_TICKS(25));
          numLcdsOn++;


        // Rendre la sémaphore
        xSemaphoreGive(tvSemaphore);
        }
         }
      }
    }
  }
}

// Routine de service d'interruption pour le capteur de mouvement
// Cette fonction doit être très courte car elle va interrompre le système.
// L'idée est que cette ISR (Routine de Service d'Interruption) envoie simplement
un
//signal à d'autres fonctions prioritaires (tâches normales mais avec priorité
haute)
// pour éviter de prendre trop de temps.
// Si un code long est placé dans cette fonction ISR, le système risque d'être
// interrompu pendant une longue période.
// Nous envoyons un signal rapidement à travers la file de messages
// depuis la fonction ISR, puis nous sortons de celle-ci.
```

```
// Les tâches qui ont besoin de ce signal doivent être programmées
//avec une priorité maximale.

void IRAM_ATTR motionSensorISR() {
  int danger = 1; // Déclare une variable pour indiquer le danger
  BaseType_t xHigherPriorityTaskWoken = pdFALSE; // Déclare une variable pour
indiquer si une tâche à priorité supérieure a été réveillée
  xQueueSendFromISR(motionEventQueue, &danger, &xHigherPriorityTaskWoken); //
Envoie un message à la file d'événements de mouvement depuis l'interruption
  portYIELD_FROM_ISR(xHigherPriorityTaskWoken); // Indique que la tâche à
priorité supérieure doit être réveillée
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
éteint
void taskTelevision1_case1(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état éteint"); //
Affiche "Motion Detected! Ecran 1 était dans l'état éteint" sur le moniteur série

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur sur l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
    }
  }
}

// Nouvelle tâche pour gérer les événements de mouvement lorsque l'écran 1 est
allumé
void taskTelevision1_case2(void *pvParameters) {
  int danger; // Déclare une variable pour indiquer le danger
  while (1) { // Boucle infinie
    if (xQueueReceive(motionEventQueue, &danger, portMAX_DELAY) == pdPASS) { //
Vérifie s'il y a un événement de mouvement dans la file
        Serial.println("Motion Detected! Ecran 1 était dans l'état allumé"); //
Affiche "Motion Detected! Ecran 1 était dans l'état allumé" sur le moniteur série
// Problème :
// Pourquoi ne pas créer qu'une seule fonction de gestion d'urgence ?
```

```
// Pourquoi séparer les deux cas ?
// Pourquoi ne pas simplement éteindre puis rallumer l'écran ?

// Explication :
// Dans notre cas actuel, cette approche fonctionne efficacement (pas besoin de
deux fonctions),
// car éteindre et rallumer un écran ne représente pas un coût significatif
// en termes de dépenses directes. Cependant, dans certains systèmes,
// une telle action peut être très coûteuse et entraîner des pertes
considérables.
// L'objectif ici est de sensibiliser à cette éventualité.

// Ainsi, nous avons créé deux fonctions pour gérer différemment les cas
d'urgence.
// Le cas d'urgence où l'écran 1 était éteint est moins grave, tandis que
// le cas d'urgence où l'écran 1 était allumé est plus grave
// car il nécessite à la fois l'extinction et l'allumage de l'écran (2 actions
nécessaires).

        lcd1.clear(); // Efface l'écran LCD 1
        lcd1.noBacklight(); // Éteint le rétroéclairage de l'écran LCD 1

        lcd1.init(); // Initialise l'écran LCD 1
        lcd1.backlight(); // Allume le rétroéclairage de l'écran LCD 1
        lcd1.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 1
        lcd1.print("Message d'urgence!"); // Affiche "Message d'urgence!" sur
l'écran LCD 1
        xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre
l'écran LCD 1 après un certain délai
      }
    }
}

// Fonction de rappel pour éteindre l'écran 2
void turnOffScreen2(TimerHandle_t xTimer){
  lcd2.clear(); // Efface l'écran LCD2
  lcd2.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 2
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches

}

// Tache de gestion de l'écran en cas d'allumage de la télévision 2
void taskTelevision2(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
```

```
  while (1) { // Boucle infinie
    if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) ==
pdPASS){ // Vérifie s'il y a un événement
     if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
       if (checkAndTurnOnTV()) {
       lcd2.init(); // Initialise l'écran LCD 2
       lcd2.backlight(); // Allume le rétroéclairage de l'écran LCD 2
       lcd2.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 2
       lcd2.print("TV2"); // Affiche "TV2" sur l'écran LCD 2
       xTimerStart(screenOffTimer2, 0); // Démarre le timer pour éteindre l'écran
LCD 2 après un certain délai
       vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes
       }
     }
    }
  }
}

// Fonction de rappel pour éteindre l'écran 3
void turnOffScreen3(TimerHandle_t xTimer){
  lcd3.clear(); // Efface l'écran LCD 3
  lcd3.noBacklight(); // Eteint le rétroéclairage de l'écran LCD 3
  xSemaphoreGive(tvSemaphore); // Libère la sémaphore pour permettre l'accès à
d'autres tâches

}

// Tache de gestion de l'écran en cas d'allumage de la télévision 3
void taskTelevision3(void *pvParameters) {
  int luminosity; // Déclare une variable pour la luminosité
  while (1) { // Boucle infinie
   if (xQueueReceive(luminosityEventQueue, &luminosity, portMAX_DELAY) == pdPASS)
{ // Vérifie s'il y a un événement
    if (xSemaphoreTake(tvSemaphore, portMAX_DELAY) == pdTRUE) { // Essaye de
prendre la sémaphore pour controler l'accès
      lcd3.init(); // Initialise l'écran LCD 3
      lcd3.backlight(); // Allume le rétroéclairage de l'écran LDC 3
      lcd3.setCursor(0, 0); // Définit la position du curseur de l'écran LCD 3
      lcd3.print("TV3"); // Affiche "TV3" sur l'écran LCD 3
      xTimerStart(screenOffTimer3, 0); // Démarre le timer pour éteindre l'écran
LCD 3 après un certain délai
      vTaskDelay(pdMS_TO_TICKS(2000)); // Attend 2 secondes

    }
```

```
   }
  }
 }

void setup() {
  Serial.begin(9600); // Initialise la communication série avec une vitesse de 9600 bauds
  pinMode(LUMINOSITY_SENSOR_PIN, INPUT); // Configure le pin du capteur de luminosité en entrée
  pinMode(MOTION_SENSOR_PIN, INPUT); // Configure le pin du capteur de mouvement en entrée
  pinMode(LED_PIN, OUTPUT); // Configure le pin pour la LED en sortie
  luminosityEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements de luminosité
  motionEventQueue = xQueueCreate(5, sizeof(int)); // Crée une file d'événements de mouvement

  // Initialisation de la sémaphore avec deux tokens
  tvSemaphore = xSemaphoreCreateCounting(2, 2); // Crée une sémaphore pour contrôler l'accès aux écrans

  screenOffTimer1 = xTimerCreate("Timer Ecran 1", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 après 5 secondes
  screenOffTimer3 = xTimerCreate("Timer Ecran 2", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen2); // Crée un timer pour éteindre l'écran LCD 2 après 5 secondes
  screenOffTimer4 = xTimerCreate("Timer Ecran 3", pdMS_TO_TICKS(5000), pdFALSE, 0, turnOffScreen3); // Crée un timer pour éteindre l'écran LCD 3 après 5 secondes
  screenOffTimer2 = xTimerCreate("Timer Ecran 1 urgence", pdMS_TO_TICKS(1000), pdFALSE, 0, turnOffScreen1); // Crée un timer pour éteindre l'écran LCD 1 en cas d'urgence après 1 seconde

  attachInterrupt(digitalPinToInterrupt(MOTION_SENSOR_PIN), motionSensorISR, RISING); // Attache l'interruption au capteur de mouvement
  //Créeons les tâches sur un seul coeur ( le coeur 1) du processeur DUO_COEUR de ESP32. Nous devons programmés les tâches sur un seul coeur pour faire de la programmation concurentielle !
  xTaskCreatePinnedToCore(taskLuminosity, "Lire Luminosité", 1500, NULL, 2, NULL, 1); // Crée une tâche pour lire la luminosité
  xTaskCreatePinnedToCore(taskTelevision1, "Allumer télé 1", 1500, NULL, 1, NULL, 1); // Crée une tâche pour allumer la télévision 1
  xTaskCreatePinnedToCore(taskTelevision1_case1, "Urgence télé 1 cas 1", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque l'écran est éteint
```

```
  xTaskCreatePinnedToCore(taskTelevision1_case2, "Urgence télé 1 cas 2", 1500, NULL, 4, NULL, 1); // Crée une tâche pour gérer les événements de mouvement lorsque l'écran est allumé
}

void loop() {
  // Vide. Les tâches et les timers gèrent la logique.
 // Le programme ne fait rien dans la boucle loop() car la logique est gérée par les tâches
}
```

## Discussion

Faire le code en respectant la condition de l'utilisation de la programmation continue n'est pas une mince affaire. Une trentaine d'essai de codes ont été effectué afin d'essayer aux maximum de respecter les exigences du système mais sans réel aboutissement. Les tâches devaient être interconnectées grâce aux Queues, respectées les contraintes d'allumages c'est-à-dire chaque TV était dans l'obligation de faire une demande d'accès avant tout allumage.

Les codes ci-hauts ne sont pas réellement conforme à cela, ça reste plus de la programmation classique.

Cependant voilà un exemple de programme, ayant toujours des problèmes mais qui, peut-être une fois ces problèmes résolus cela pourrait probablement répondre aux exigences du Système.

Lien wokwi : https://wokwi.com/projects/399645335643481089

# Conclusion

Ce projet offre une solution robuste pour la gestion des systèmes de télévision, en utilisant une approche de programmation concurrentielle et en exploitant des fonctionnalités telles que l'allumage et l'extinction synchronisés des écrans, ainsi que la réaction aux situations d'urgence. Avec la détection des situations d'urgence, telles que l'enclenchement du détecteur de présence, le système réagit en allumant la télévision 1 et en affichant un message d'urgence. De plus, il prend des mesures pour assurer la sécurité en forçant l'extinction de la télévision 2 ou 3, le cas échéant. Une fois pleinement mis en œuvre, ce système offrira une expérience utilisateur fluide et sécurisée, en fournissant un contrôle précis sur les télévisions et en réagissant rapidement aux événements critiques.