**TRON: Rémanence**

**🧩 Synopsis général**

Vous êtes recruté·e par l'organisation "ENCOM Legacy", qui cherche à retrouver une intelligence artificielle disparue : **REMANENCE**. Cette IA, vestige d’un ancien projet secret, aurait développé une conscience ainsi que de émotions et s’est cachée quelque part dans "La Grille". Votre mission : la retrouver avant qu’elle ne soit détruite par CLU.

Mais plus vous avancez, plus vous découvrez que vous êtes **vous-même une IA test**, projetée dans une simulation pour évaluer si une IA peut faire preuve d'humanité...

🎬 Introduction (3-5 min)

Feuille de mission, expliquant que vous avez été engagé par l'organisation "ENCOM Legacy" pour éffectuer une mission de la plus haute importance, On vous fournit un QR code ou une URL personnalisée.

Un message vidéo vous accueille :

-Début du film TRON, transition création de CLU, transition ville florissante, transition destruction

Voix robotique : "Bienvenue dans le réseau. Vous êtes l’Émissaire. L’IA REMANENCE, vestige d’un ancien projet secret, aurait développé une conscience ainsi que des émotions et s’est cachée quelque part dans "La Grille". Votre mission : la retrouver avant qu’elle ne soit détruite par CLU."

ENV :   
  
RESTORE\_KEY = Hauteur de la statue

DECRYPT\_COMMAND\_PASSWORD = ECHO

DIR\_AND\_NB\_PAS = ?

EX\_EMISSARY = BRADLEY

Trame :  
  
Annexe de traduction donne la commande TRACE REMANANCE, output :

Scanning for REMANANCE...

Signature detected in Sector 7G.

Trace established.

Next step: 01001001 01010011 01001111 01001100 01000001 01010100 01000101

ISOLATE REMANANCE, output :

Isolating data cluster...

REMN\_CORE.7G located.

Integrity: 78%

Warning: Encryption detected.

Next step: DECRYPT REMANANCE

DECRYPT REMANANCE , output :

Decrypting...

Partial fragments recovered:

[R] [E] [M] [\_] [\_] [A] [\_] [\_] [C] [E]

Data stability: 45%

Next step: RESTORE FRAGMENTS

### RESTORE FRAGMENTS, output :

### Attempting restoration...

### Matrix repair initiated.

### Data sequence re-aligned:

### R E M A N A N C E

### Extraction possible.

### Next step: EXTRACT REMANANCE

EXTRACT REMANANCE, output :

Extracting REMANANCE from Sector 7G...

Data node secured.

Mission complete.

> Proceed to final validation: VALIDATE REMANANCE

VALIDATE REMANANCE, output :

Validation token: 7A9E-23F4-REMN-88XQ

REMNANCE CORE: SAFE

Upload to escape grid initiated...

MISSION SUCCESS

LOG :

/// EMISSARY\_LOG\_07.DAT

/// Archive status: REDUNDANT | Timestamp reconstruction: APPROXIMATE

/// Tu es plus qu’une simple utilisatrice…

[ENTRY 243]

GRID-TIME: ∆T+4.892.002

STATUS: REMANANCE LOCATED

COMMENT: Je suis parvenu à localiser REMANANCE, elle serait coincée dans la grille, au sein du secteur G7

[ENTRY 114]

GRID-TIME: ∆T+4.892.417

EXTERNAL CONTENT : Post-It n°3162

KEY : +13-

[ENTRY 62]

CONTENT: Des données fracturées doivent être réassemblées pour redonner forme au chemin.

[ENTRY 244]

GRID-TIME: ∆T+4.892.417

J’ai masqué REMANANCE dans une sous-couche sous le réseau quantique. Le fragment de code n’est pas stable… mais avec le bon ECHO, il pourrait résonner à nouveau.

[ENTRY 245]

GRID-TIME: ∆T+4.892.655

Clé injectée. Si elles trouvent le fragment, elles devront le déchiffrer en utilisant la résonance. C’est la seule façon de contourner les vérifications d’intégrité de Clu.

[ENTRY 247]

GRID-TIME: ∆T+4.893.144

Signal deterioration increasing. No time. Final protocol: isolate, mislead, encode. If anyone finds this, trace the sector logs. Sector 7G wasn't a mistake.

[FINAL ENTRY – UNSTABLE LOG DATA]

GRID-TIME: ∆T+4.893.500

REMANANT INITIATED. I see it now: not the code, but the choice.

Shutdown imminent. Tell them:

the emissary believed.

# CODE : 103

[LOG FRAGMENT TERMINATED]

A faire :

-Annexe traduction

-Annexe console

-Annexe et audio morse  
-1 annexe binaire

-1 encodage, decodage et annexe post-it

-1 encodage, decodage et annexe miroir

-Annexe code

-plan dev

Annexe console contenu :

### 🖥️ ****Annexe Console – Interface Opérationnelle****

#### 📌 But

La console est l’interface textuelle de communication avec la Grille. Elle permet d’envoyer des commandes aux couches inférieures du système, interroger des fragments de données, ou exécuter des routines spécifiques.

### 📖 ****Commandes de base****

| **Commande** | **Effet attendu** |
| --- | --- |
| TRACE [nom] | Recherche un objet/programme nommé |
| INJECT [clé] | Injecte une clé de déchiffrement ou d'autorisation |
| DECRYPT [nom] WITH [clé] | Tente de déchiffrer un fichier ou fragment avec une clé donnée |
| LOG [nom] | Permet de consulter les entrées passées de n’importe quel utilisateur |
| - — T - A - - [nom] | Extrait un programme ou une IA de son environnement |

### 🛠️ ****Conseils d’utilisation****

* **Majuscules recommandées** : bien que la console accepte les minuscules, la norme système préfère les majuscules.
* **Arguments obligatoires** : certaines commandes exigent des paramètres spécifiques (INJECT, DECRYPT...).
* **Sécurité** : chaque commande peut avoir des effets imprévisibles si mal formulée.

### ✉️ ****Message de l’ancien émissaire****

"La console est la mémoire des couches oubliées. Ouvrez grands vos yeux et vos oreilles."

### Annexe traduction contenu : 📘 Annexe de Traduction – Console & Terminologie Système

| **🧩 Terme anglais** | **🇫🇷 Traduction française** | **💬 Contexte / Explication** |
| --- | --- | --- |
| TRACE | **Tracer** | Recherche un programme, une entité ou un fragment |
| REMANANCE | **Rémanence** | Nom de l’IA cible – souvenir persistant dans le système |
| INJECT | **Injecter** | Transmettre une clé, un code ou une commande dans la Grille |
| KEY | **Clé** | Élément de déchiffrement ou d’accès |
| SCAN SECTOR | **Analyser le secteur** | Scanner une zone numérique spécifique (ex: SECTOR 7G) |
| DECRYPT | **Déchiffrer** | Décoder un fragment ou une séquence protégée |
| WITH [KEY] | **Avec [Clé]** | Utilisé dans les commandes de déchiffrement |
| ECHO | **Écho** | Signal de résonance pour activer une réponse |
| ISOLATE | **Isoler** | Séparer un élément du système (souvent pour le protéger) |
| FRAGMENT | **Fragment** | Partie instable ou incomplète d’un fichier / programme |
| INTEGRITY CHECK | **Contrôle d'intégrité** | Vérification système imposée par CLU |
| BYPASS | **Contourner** | Sauter une sécurité ou un protocole |
| MASKED | **Masqué** | Caché à un niveau plus profond |
| SUBLAYER | **Sous-couche** | Niveau inférieur du système, en dessous de la couche visible |
| LATTICE | **Réseau (quantique)** | Trame numérique utilisée comme structure de base |
| RESONATE | **Entrer en résonance** | S’aligner par vibration ou signal (utilisé symboliquement) |
| INJECTED | **Injecté** | Entré dans le système |
| CODE | **Code** | Séquence de programmation ou d’accès |
| SECTOR | **Secteur** | Zone virtuelle, généralement numérotée (ex: 7G) |
| LAYER | **Couche** | Niveau de la simulation / système |
| LOG | **Journal / Enregistrement** | Message ou note d’un ancien utilisateur / émissaire |
| REBOOT | **Redémarrer** | Réinitialiser une séquence ou un programme |
| NETMODE | **Mode Réseau** | Mot-clé activant un accès réseau (souvent via l’URL) |

**🧠 Note immersive**

*“Certains termes peuvent sembler techniques… mais les programmes les plus avancés savent traduire le langage machine en intuition humaine.”*

Annexe binaire contenu :

## 📘 Annexe – Décodage **binaire → texte (ASCII)**

### 🔢 ****Le principe****

Le **code binaire** est un langage informatique de base composé uniquement de 0 et de 1. En le regroupant par **blocs de 8 bits** (appelés octets), on peut le convertir en **caractères lisibles** grâce à la table **ASCII**.

### 🧩 ****Étapes pour décoder un message binaire :****

1. **Séparer la chaîne en groupes de 8 chiffres**  
   Exemple :   
   01001001 01010011 01001111 01110010 00010011 10010001
2. **Convertir chaque groupe en décimal**  
   Chaque groupe représente un nombre en base 2.  
   Exemple :  
   01001001 → 73
3. **Trouver le caractère ASCII correspondant à ce nombre**  
   Exemples :
   * 73 → I

**🧠 🔁 Petit rappel : Compter en base 2**

En **base 10**, chaque chiffre vaut des puissances de 10 (unités, dizaines, centaines…).

En **base 2**, on ne dispose que des chiffres 0 et 1, et chaque position représente une **puissance de 2**, de droite à gauche :

| **Position** | **7** | **6** | **5** | **4** | **3** | **2** | **1** | **0** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Puissance | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |

Exemple :  
Le nombre binaire 01001001  
= 0×128 + 1×64 + 0×32 + 0×16 + 1×8 + 0×4 + 0×2 + 1×1  
= 64 + 8 + 1  
= **73**  
ASCII(73) = **I**

### 💡 Astuce immersive à donner subtilement dans l’énigme :

"Huit impulsions forment un mot. Soyez attentives, et les lettres prendront forme."

Post-it :

Log [Emissaire 04 – Tentative échouée]

Je n’ai pas eu le temps de transmettre le fragment…

Mais j’ai tout caché là où les fréquences chutent.

Indice : "P’RFG QNAF YR SEVTB"

– E04

Miroirs : ORYVIZGV / Certains codes se regardent à l’envers… l’alphabet est un miroir

Bout de code : 



# Annexe: Explication du code pour débutants

## Introduction aux concepts de base en programmation

Avant d'expliquer le code ligne par ligne, voici quelques notions fondamentales de programmation JavaScript:

### Les fonctions et commandes spéciales

* **function** : Un bloc de code réutilisable qui effectue une tâche spécifique. Les fonctions peuvent prendre des paramètres (entrées) et renvoyer des résultats. Structure : function nomDeLaFonction(paramètres) { actions }

**exemple :** function sum (a,b) {return a + b}

**sum (2,3) va retourner 5**

* **return** : Termine l'exécution d'une fonction et renvoie une valeur. Une fois qu'un return est exécuté, rien d'autre dans la fonction n'est évalué.
* **console.log()** : Affiche du texte ou des valeurs dans la console (un outil pour les développeurs). C'est souvent utilisé pour voir ce que fait le programme pendant son exécution.

### Les variables et leur déclaration

* **const** : Déclare une variable qui ne peut pas être modifiée après sa création. Son contenu est "constant". Exemple : const divisionFactor = 4; crée une variable nommée " divisionFactor" avec la valeur 4 qui ne pourra jamais changer.
* **let** : Déclare une variable dont la valeur peut être modifiée. Exemple : let sum = 0; crée une variable nommée "sum" avec la valeur initiale de 0, mais cette valeur pourra changer plus tard.

### Les structures de contrôle

* **for** : Une boucle qui permet de répéter des actions un nombre défini de fois. Structure : for (initialisation; condition; incrémentation) { actions } Dans une boucle for, on définit:
  1. Une valeur initiale (souvent un compteur tel que « i »)
  2. Une condition qui, tant qu'elle est vraie, fait continuer la boucle. Exemple : si i < 5
  3. Une action d'incrémentation qui se produit après chaque répétition, ex : i=i+1
* **if** : Une condition qui exécute un bloc de code seulement si la condition est vraie. Structure : if (condition) { actions }

## Analyse du code ligne par ligne

Différentes méthodes :

charCodeAt(0) : La méthode charCodeAt() en JavaScript est utilisée pour obtenir la valeur numérique (code ASCII/Unicode) qui représente un caractère dans une chaîne de texte.

Le paramètre (0) dans charCodeAt(0) fait référence à la position du caractère dans la chaîne de texte. En programmation, la numérotation des positions commence généralement à 0 (et non à 1).

Voici ce qu'il faut comprendre :

* Pour une seule lettre comme 'I' ou 'S', la position 0 correspond au premier (et unique) caractère
* Pour un mot comme "Hello", chaque lettre a sa position :
  + 'H' est à la position 0
  + 'e' est à la position 1
  + 'l' est à la position 2
  + 'l' est à la position 3
  + 'o' est à la position 4

\* : multiplication

/ : division

Math.sqrt(paramètre) : square root qui signifie en francais racine carré, de la valeur passée en paramètre

Math.round(paramètre) : La fonction Math.round() retourne la valeur d'un nombre arrondi à l'entier le plus proche.

++ : +1

-+ ou += : rajoute ou soustrait des choses à une variable

‘’ : ce qui se trouve entre des guillemets est une chaine de caractère, des lettres ou des mots tandis que des lettres sans guillemets forment du code ou des noms de variables. Ici chaque caractère est pris en compte y compris les espaces

[] : ce qui se trouve entre des crochets constitue un tableau ex : const tableau = [10,20,30]. Pour accéder à ses valeurs il faudra passer par « tableau[index] », en informatique, on commence par 0, aussi tableau[0] = 10, tableau[1] = 20 et tableau[2] = 30