

PROMPT ULTIME POUR VRNCA-LAG (Labyrinth Adventure Game)

1. OBJECTIF & VISION GLOBALE

Développer un jeu web immersif en HTML/CSS/JavaScript qui plonge le joueur dans l'univers cyberpunk de VRNCA. Le joueur incarne VRNCA, une entité numérique qui doit traverser 10 labyrinthes digitaux pour accomplir sa mission de libération.

Points critiques à respecter:

- Génération procédurale MAIS déterministe des labyrinthes (même topologie à chaque lancement avec seed "VRNCA2025")
- Ambiance rétro-cyberpunk cohérente (visuelle et sonore)
- Équilibre entre challenge (QCM, gestion de batterie) et plaisir d'exploration
- Code modulaire et robuste avec gestion efficace des erreurs

2. ARCHITECTURE TECHNIQUE

2.1 Structure des Fichiers

```
/
├── index.html          # Page principale avec Canvas/div et écran d'accueil
├── css/
│   ├── main.css       # Styles généraux et interface
│   ├── animations.css # Effets visuels et transitions
│   └── responsive.css  # Adaptations pour mobiles/tablettes
├── js/
│   ├── main.js         # Point d'entrée, initialisation du jeu
│   ├── maze.js         # Générateur de labyrinthe déterministe
│   ├── game.js         # Logique de jeu (déplacements, batterie)
│   ├── ui.js           # Interface utilisateur et animations
│   ├── questions.js    # Base de données QCM et sélection
│   ├── audio.js        # Gestion des sons avec fallback
│   └── utils.js        # Fonctions utilitaires (PRNG, etc.)
├── assets/
│   ├── fonts/          # Polices (Press Start 2P, Courier)
│   └── audio/          # Fichiers sonores (.mp3)
```

2.2 Choix Technologiques Spécifiques

Rendu du Labyrinthe (choisir UNE option)

- **Option A: Canvas (recommandée)**

- Avantages: Meilleures performances, plus adapté pour les grands labyrinthes, animations fluides
- Inconvénients: Gestion événements plus complexe, demande plus de code personnalisé
- Recommandé pour: Performances optimales, labyrinthes complexes

- **Option B: Grille de DIVs**

- Avantages: Simplicité d'implémentation, facilité de style CSS, événements DOM natifs
- Inconvénients: Performances potentiellement moins bonnes (nombreux éléments DOM)
- Recommandé pour: Priorisation de la simplicité, prototypage rapide

Code exemple pour Option A (Canvas):

// Dans maze.js

```
function initCanvas() {
  const canvas = document.getElementById('mazeCanvas');
  const ctx = canvas.getContext('2d');
  canvas.width = CELL_SIZE * MAZE_WIDTH;
  canvas.height = CELL_SIZE * MAZE_HEIGHT;
  return ctx;
}

function drawCell(ctx, x, y, type) {
  const cellX = x * CELL_SIZE;
  const cellY = y * CELL_SIZE;

  // Effacer la cellule précédente
  ctx.clearRect(cellX, cellY, CELL_SIZE, CELL_SIZE);

  switch(type) {
    case 'wall':
      ctx.fillStyle = '#00FF00';
      ctx.fillRect(cellX, cellY, CELL_SIZE, CELL_SIZE);
      // Ajouter effet de lueur néon
      ctx.shadowColor = '#00FF00';
      ctx.shadowBlur = 5;
      ctx.strokeRect(cellX, cellY, CELL_SIZE, CELL_SIZE);
      ctx.shadowBlur = 0;
      break;

    case 'path':
      ctx.strokeStyle = 'rgba(0, 255, 0, 0.3)';
      ctx.strokeRect(cellX + 2, cellY + 2, CELL_SIZE - 4, CELL_SIZE - 4);
      break;
  }
}
```

```

    case 'player':
        ctx.fillStyle = '#00FF00';
        // Cercle pour représenter le joueur
        ctx.beginPath();
        ctx.arc(cellX + CELL_SIZE/2, cellY + CELL_SIZE/2, CELL_SIZE/2 - 2, 0, Math.PI * 2);
        ctx.fill();
        // Ajouter lueur
        ctx.shadowColor = '#00FF00';
        ctx.shadowBlur = 8;
        ctx.stroke();
        ctx.shadowBlur = 0;
        break;

    // Autres types (ajouter les cas pour powerup, teleport, etc.)
}
}

function render() {
    // Redessiner tout le labyrinthe visible
    for (let y = 0; y < MAZE_HEIGHT; y++) {
        for (let x = 0; x < MAZE_WIDTH; x++) {
            // Ne dessiner que les cellules révélées
            if (revealedCells[y][x]) {
                drawCell(ctx, x, y, maze[y][x]);
            } else {
                // Dessiner les cellules cachées (semi-transparentes)
                drawHiddenCell(ctx, x, y);
            }
        }
    }
}

// Dessiner le joueur à sa position actuelle
drawCell(ctx, playerPosition.x, playerPosition.y, 'player');
}

// Fonction importante pour gérer les clics sur Canvas
function handleCanvasClick(event) {
    const rect = canvas.getBoundingClientRect();
    const clickX = event.clientX - rect.left;
    const clickY = event.clientY - rect.top;

    // Convertir en coordonnées de grille
    const cellX = Math.floor(clickX / CELL_SIZE);
    const cellY = Math.floor(clickY / CELL_SIZE);

    // Vérifier si déplacement valide et traiter
    if (isValidMove(cellX, cellY)) {
        handleMovement({x: cellX, y: cellY});
    }
}

```

```

}
}

```

Code exemple pour Option B (Grille de DIVs):

```

// Dans maze.js
function createMazeGrid() {
  const mazeContainer = document.getElementById('mazeContainer');
  mazeContainer.style.gridTemplateColumns = `repeat(${MAZE_WIDTH},
${CELL_SIZE}px)`;
  mazeContainer.style.gridTemplateRows = `repeat(${MAZE_HEIGHT}, ${CELL_SIZE}px)`;

  // Vider le container (utile pour régénération)
  mazeContainer.innerHTML = "";

  // Créer toutes les cellules
  for (let y = 0; y < MAZE_HEIGHT; y++) {
    for (let x = 0; x < MAZE_WIDTH; x++) {
      const cell = document.createElement('div');
      cell.id = `cell-${x}-${y}`;
      cell.classList.add('cell', 'hidden');
      cell.dataset.x = x;
      cell.dataset.y = y;

      // Ajouter listener pour clic
      cell.addEventListener('click', function() {
        const cellX = parseInt(this.dataset.x);
        const cellY = parseInt(this.dataset.y);

        if (isValidMove(cellX, cellY)) {
          handleMovement({x: cellX, y: cellY});
        }
      });

      mazeContainer.appendChild(cell);
    }
  }
}

function updateCell(x, y, type) {
  const cell = document.getElementById(`cell-${x}-${y}`);

  // Retirer toutes les classes spécifiques
  cell.classList.remove('wall', 'path', 'player', 'powerup', 'teleport', 'malware', 'entry', 'exit');

  // Ajouter classe selon type
  cell.classList.add(type);
}

```

```

// Si cellule révélée, enlever classe hidden
if (revealedCells[y][x]) {
  cell.classList.remove('hidden');
  cell.classList.add('revealed');
}
}

function renderMaze() {
  // Mettre à jour toutes les cellules
  for (let y = 0; y < MAZE_HEIGHT; y++) {
    for (let x = 0; x < MAZE_WIDTH; x++) {
      updateCell(x, y, maze[y][x]);
    }
  }

  // Marquer la position du joueur
  updateCell(playerPosition.x, playerPosition.y, 'player');
}

```

2.3 Génération Déterministe du Labyrinthe

Constantes et Configuration

```

const MAZE_WIDTH = 21; // Toujours impair pour avoir des murs aux bords
const MAZE_HEIGHT = 21;
const SEED = "VRNCA2025"; // Graine fixe pour reproductibilité
const CELL_SIZE = 22; // Taille en pixels de chaque cellule

```

```

// Ajuster la difficulté par niveau (pourcentage de murs)
const DIFFICULTY_WALL_DENSITY = [
  25, 28, 32, 37, 42, 47, 51, 54, 57, 60
]; // % de murs par niveau

```

```

// Type de cellules du labyrinthe
const CELL_TYPES = {
  WALL: 'wall',
  PATH: 'path',
  ENTRY: 'entry',
  EXIT: 'exit',
  POWERUP: 'powerup',
  TELEPORT: 'teleport',
  MALWARE: 'malware'
};

```

Générateur Pseudo-Aléatoire (CRUCIAL pour déterminisme)

```

class PseudoRandom {
  constructor(seed) {

```

```

    this.seed = this.hashString(seed);
}

// Fonction de hachage pour convertir string en nombre
hashString(str) {
    let hash = 0;
    for (let i = 0; i < str.length; i++) {
        hash = ((hash << 5) - hash) + str.charCodeAt(i);
        hash |= 0; // Conversion en 32 bits
    }
    return hash >>> 0; // Forcer unsigned
}

// Algorithme LCG (Linear Congruential Generator)
random() {
    // Constantes choisies pour bonne distribution
    this.seed = (this.seed * 9301 + 49297) % 233280;
    return this.seed / 233280; // Normaliser entre 0 et 1
}

// Génère nombre entre min et max (inclus)
randomInt(min, max) {
    return Math.floor(this.random() * (max - min + 1)) + min;
}

// Choisir élément aléatoire d'un tableau
randomChoice(array) {
    return array[this.randomInt(0, array.length - 1)];
}
}

```

Algorithme de Génération (Recursive Backtracking adapté)

```

function generateMaze(level) {
    // 1. Initialiser générateur pseudo-aléatoire avec seed + level
    const prng = new PseudoRandom(SEED + level);

    // 2. Créer une grille pleine de murs
    let maze = Array(MAZE_HEIGHT).fill().map(() =>
        Array(MAZE_WIDTH).fill(CELL_TYPES.WALL));

    // 3. Points de départ pour l'algorithme
    const startX = 1;
    const startY = 1;
    maze[startY][startX] = CELL_TYPES.PATH;

    // 4. Pile pour Backtracking (stocker chemin parcouru)
    const stack = [{x: startX, y: startY}];
}

```

```

// Directions possibles (haut, droite, bas, gauche)
const directions = [
  {x: 0, y: -2}, // Haut
  {x: 2, y: 0},  // Droite
  {x: 0, y: 2},  // Bas
  {x: -2, y: 0}  // Gauche
];

// 5. Algorithme principal de génération
while (stack.length > 0) {
  const current = stack[stack.length - 1];

  // Mélanger les directions (avec notre PRNG)
  const shuffledDirs = [...directions];
  for (let i = shuffledDirs.length - 1; i > 0; i--) {
    const j = Math.floor(prng.random() * (i + 1));
    [shuffledDirs[i], shuffledDirs[j]] = [shuffledDirs[j], shuffledDirs[i]];
  }

  let moved = false;

  // Tester chaque direction
  for (const dir of shuffledDirs) {
    const nextX = current.x + dir.x;
    const nextY = current.y + dir.y;

    // Vérifier si dans les limites et cellule non visitée
    if (
      nextX >= 1 && nextX < MAZE_WIDTH - 1 &&
      nextY >= 1 && nextY < MAZE_HEIGHT - 1 &&
      maze[nextY][nextX] === CELL_TYPES.WALL
    ) {
      // Creuser le passage (cellule entre current et next)
      const passageX = current.x + dir.x / 2;
      const passageY = current.y + dir.y / 2;
      maze[passageY][passageX] = CELL_TYPES.PATH;

      // Marquer la nouvelle cellule comme chemin
      maze[nextY][nextX] = CELL_TYPES.PATH;

      // Ajouter au stack pour backtracking
      stack.push({x: nextX, y: nextY});
      moved = true;
      break;
    }
  }
}

```

```

    // Si pas de mouvement possible, backtracking
    if (!moved) {
        stack.pop();
    }
}

// 6. Ajustement de la densité des murs selon le niveau
adjustWallDensity(maze, level, prng);

// 7. Placer les éléments spéciaux
placeSpecialElements(maze, level, prng);

return maze;
}

function adjustWallDensity(maze, level, prng) {
    // Calculer le nombre de murs à ajouter/enlever
    const currentWalls = countWalls(maze);
    const totalCells = (MAZE_WIDTH - 2) * (MAZE_HEIGHT - 2); // Ignorer les bords
    const targetWallPercentage = DIFFICULTY_WALL_DENSITY[level - 1];
    const targetWalls = Math.floor(totalCells * targetWallPercentage / 100);

    // Ajouter ou enlever des murs au hasard pour atteindre cible
    if (currentWalls < targetWalls) {
        // Ajouter des murs
        addRandomWalls(maze, targetWalls - currentWalls, prng);
    } else if (currentWalls > targetWalls) {
        // Enlever des murs
        removeRandomWalls(maze, currentWalls - targetWalls, prng);
    }
}

function addRandomWalls(maze, count, prng) {
    let added = 0;
    const candidates = [];

    // Identifier les chemins possibles à convertir en murs
    for (let y = 1; y < MAZE_HEIGHT - 1; y++) {
        for (let x = 1; x < MAZE_WIDTH - 1; x++) {
            if (maze[y][x] === CELL_TYPES.PATH) {
                // Ne pas isoler de zones (vérifier connectivité)
                if (!wouldIsolate(maze, x, y)) {
                    candidates.push({x, y});
                }
            }
        }
    }
}

```



```

// Mélanger les candidats
for (let i = candidates.length - 1; i > 0; i--) {
  const j = Math.floor(prng.random() * (i + 1));
  [candidates[i], candidates[j]] = [candidates[j], candidates[i]];
}

// Ajouter des murs tant que possible
for (const pos of candidates) {
  if (added >= count) break;

  maze[pos.y][pos.x] = CELL_TYPES.WALL;

  // Vérifier que cela n'a pas créé de zones isolées
  if (hasIsolatedAreas(maze)) {
    maze[pos.y][pos.x] = CELL_TYPES.PATH; // Annuler
  } else {
    added++;
  }
}

function removeRandomWalls(maze, count, prng) {
  let removed = 0;
  const candidates = [];

  // Identifier les murs qu'on peut enlever
  for (let y = 1; y < MAZE_HEIGHT - 1; y++) {
    for (let x = 1; x < MAZE_WIDTH - 1; x++) {
      if (maze[y][x] === CELL_TYPES.WALL) {
        candidates.push({x, y});
      }
    }
  }

  // Mélanger les candidats
  for (let i = candidates.length - 1; i > 0; i--) {
    const j = Math.floor(prng.random() * (i + 1));
    [candidates[i], candidates[j]] = [candidates[j], candidates[i]];
  }

  // Enlever des murs
  for (const pos of candidates) {
    if (removed >= count) break;

    maze[pos.y][pos.x] = CELL_TYPES.PATH;
    removed++;
  }
}

```

```

function hasIsolatedAreas(maze) {
  // Utiliser BFS pour vérifier la connectivité
  // Implémenter un algorithme de parcours pour vérifier
  // que toutes les cellules PATH sont connectées
  // Code détaillé à implémenter...
  return false; // Simplification
}

```

Placement des éléments spéciaux

```

function placeSpecialElements(maze, level, prng) {
  // 1. Placer entrée (toujours sur le bord gauche)
  const entryY = Math.floor(prng.random() * (MAZE_HEIGHT-2)) + 1;
  maze[entryY][0] = CELL_TYPES.ENTRY;

  // Mémoriser pour le placement initial du joueur
  levelData[level] = levelData[level] || {};
  levelData[level].startPosition = {x: 0, y: entryY};

  // 2. Placer sortie (toujours sur le bord droit)
  const exitY = Math.floor(prng.random() * (MAZE_HEIGHT-2)) + 1;
  maze[exitY][MAZE_WIDTH-1] = CELL_TYPES.EXIT;

  // 3. Identifier tous les chemins disponibles pour placement
  const availablePaths = [];
  for (let y = 1; y < MAZE_HEIGHT - 1; y++) {
    for (let x = 1; x < MAZE_WIDTH - 1; x++) {
      if (maze[y][x] === CELL_TYPES.PATH) {
        availablePaths.push({x, y});
      }
    }
  }

  // Mélanger les chemins disponibles
  for (let i = availablePaths.length - 1; i > 0; i--) {
    const j = Math.floor(prng.random() * (i + 1));
    [availablePaths[i], availablePaths[j]] = [availablePaths[j], availablePaths[i]];
  }

  // 4. Placer les prises (2 à 5 selon le niveau)
  const powerUpsCount = 2 + Math.floor(prng.random() * Math.min(4, level));
  for (let i = 0; i < powerUpsCount; i++) {
    if (availablePaths.length === 0) break;

    const position = availablePaths.pop();
    maze[position.y][position.x] = CELL_TYPES.POWERUP;
  }
}

```

```

// 5. Placer téléporteurs (toujours 2)
levelData[level].teleporters = [];
for (let i = 0; i < 2; i++) {
    if (availablePaths.length === 0) break;

    const position = availablePaths.pop();
    maze[position.y][position.x] = CELL_TYPES.TELEPORT;

    // Mémoriser les positions des téléporteurs
    levelData[level].teleporters.push({
        x: position.x,
        y: position.y
    });
}

// 6. Placer malware (toujours 1)
if (availablePaths.length > 0) {
    const position = availablePaths.pop();
    maze[position.y][position.x] = CELL_TYPES.MALWARE;
}

return maze;
}

```

3. MÉCANIQUES DE JEU EN DÉTAIL

3.1 Système de Batterie

```

class BatterySystem {
    constructor() {
        this.percentage = 100;
        this.batteryElement = document.getElementById('batteryPercentage');
        this.batteryBarElement = document.getElementById('batteryBar');
        this.segments = document.querySelectorAll('.battery-segment');
        this.rechargeInterval = null;

        // Initialiser l'affichage
        this.updateDisplay();
    }

    deplete(amount) {
        // Stopper toute recharge lente en cours
        this.stopSlowRecharge();

        // Calculer nouvelle valeur (minimum 0%)

```

```

this.percentage = Math.max(0, this.percentage - amount);
this.updateDisplay();

// Vérifier si batterie épuisée
if (this.percentage <= 0) {
    playSound('game_over');
    gameOver('Batterie épuisée. Connexion perdue.');
```

return false;

}

```

// Activer recharge lente si batterie critique
if (this.percentage < 10) {
    this.addSlowRecharge(0.5); // +0.5% toutes les 5 secondes
    showMessage("ALERTE: Niveau de batterie critique. Mode éco-énergie activé.");
}

return true;
}

recharge(amount) {
    // Calculer nouvelle valeur (maximum 100%)
    this.percentage = Math.min(100, this.percentage + amount);
    this.updateDisplay();

    // Effet sonore
    playSound('powerup');
```

// Messages selon niveau après recharge

```

if (this.percentage >= 90) {
    showMessage("Batterie rechargée. Capacité optimale rétablie.");
} else {
    showMessage(`Source d'énergie détectée. Batterie à ${Math.floor(this.percentage)}%`);
}
}

updateDisplay() {
    // Mettre à jour texte pourcentage
    this.batteryElement.textContent = `${Math.floor(this.percentage)}%`;

    // Mettre à jour segments visuels (20 segments = 5% chacun)
    const activeSegments = Math.floor(this.percentage / 5);

    // Mettre à jour chaque segment
    this.segments.forEach((segment, index) => {
        if (index < activeSegments) {
            segment.classList.add('active');
```

} else {

```

            segment.classList.remove('active');
```

```

    }
  });

  // Changement de couleur si batterie critique
  if (this.percentage < 20) {
    this.batteryBarElement.classList.add('critical');
  } else {
    this.batteryBarElement.classList.remove('critical');
  }
}

addSlowRecharge(amount) {
  // Éviter de créer plusieurs intervalles
  this.stopSlowRecharge();

  // Créer nouvel intervalle pour recharge lente
  this.rechargeInterval = setInterval(() => {
    this.percentage = Math.min(100, this.percentage + amount);
    this.updateDisplay();

    // Arrêter recharge lente si batterie > 15%
    if (this.percentage > 15) {
      this.stopSlowRecharge();
      showMessage("Mode éco-énergie désactivé. Opérations normales rétablies.");
    }
  }, 5000); // Toutes les 5 secondes
}

stopSlowRecharge() {
  if (this.rechargeInterval) {
    clearInterval(this.rechargeInterval);
    this.rechargeInterval = null;
  }
}

// Obtenir score pour niveau (pourcentage restant)
getScore() {
  return Math.floor(this.percentage);
}
}

```

3.2 Déplacements et Système de Révélation

Déplacements et Système de Révélation du prompt VRNCA-LAG.

javascript
Copier

```

function movePlayer(newPosition) {
    // Mettre à jour la position du joueur
    playerPosition = {...newPosition};

    // Révéler la cellule courante et adjacentes
    revealCell(playerPosition);
    revealAdjacentCells(playerPosition);

    // Mettre à jour le rendu
    if (useCanvas) {
        renderCanvas();
    } else {
        renderDivMaze();
    }
}

function revealCell(position) {
    revealedCells[position.y][position.x] = true;
}

function revealAdjacentCells(position) {
    // Révéler toutes les cellules adjacentes (y compris diagonales)
    for (let dy = -1; dy <= 1; dy++) {
        for (let dx = -1; dx <= 1; dx++) {
            const newX = position.x + dx;
            const newY = position.y + dy;

            // Vérifier limites
            if (newX >= 0 && newX < MAZE_WIDTH && newY >= 0 && newY < MAZE_HEIGHT) {
                revealedCells[newY][newX] = true;
            }
        }
    }
}

function getTeleporterDestination(position) {
    // Trouver l'autre téléporteur
    const teleporters = levelData[currentLevel].teleporters;

    // Si position correspond au premier téléporteur, renvoyer le second
    if (teleporters[0].x === position.x && teleporters[0].y === position.y) {
        return teleporters[1];
    }
    // Sinon renvoyer le premier
    else {
        return teleporters[0];
    }
}

```

```

function completeLevel() {
  // Calculer score (% batterie restante)
  const score = battery.getScore();

  // Mise à jour stats
  gameStats.completedLevels.push({
    level: currentLevel,
    score: score
  });

  // Afficher écran de transition
  showLevelCompleteScreen(score);

  // Passer au niveau suivant (après délai)
  setTimeout(() => {
    if (currentLevel < 10) {
      initGame(currentLevel + 1);
    } else {
      // Fin du jeu
      showGameCompleteScreen();
    }
  }, 3000);
}

```

3.3 Système de QCM (Questions à Choix Multiples)

```

// Dans questions.js
const QCM_DATABASE = [
  // Questions générales
  {
    question: "Quelle est la propriété principale d'un algorithme de chiffrement symétrique?",
    choices: [
      "Il utilise la même clé pour chiffrer et déchiffrer",
      "Il est plus lent que le chiffrement asymétrique",
      "Il nécessite une infrastructure à clé publique",
      "Il ne peut chiffrer que du texte"
    ],
    correctIndex: 0
  },
  {
    question: "Qu'est-ce qu'un pare-feu?",
    choices: [
      "Un antivirus",
      "Un système qui surveille et contrôle le trafic réseau",
      "Un dispositif de stockage sécurisé",

```

```

    "Un protocole de chiffrement"
  ],
  correctIndex: 1
},

// Questions cyberpunk/rétro (thématiques)
{
  question: "Quel est le terme pour décrire l'interface neuronale directe dans la science-fiction cyberpunk?",
  choices: [
    "Neuro-link",
    "Bio-port",
    "Deck",
    "Interface cérébrale"
  ],
  correctIndex: 2
},

// Questions niveaux avancés (plus difficiles pour niveaux 6-10)
{
  question: "Quelle technique d'attaque utilise l'exécution de code malveillant en exploitant un débordement de tampon?",
  choices: [
    "Cross-Site Scripting",
    "SQL Injection",
    "Buffer Overflow",
    "Man-in-the-Middle"
  ],
  correctIndex: 2
},
// ... Plus de questions (au moins 50 au total)
];

// Questions par niveau de difficulté
const BEGINNER_QUESTIONS = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]; // Indices des questions faciles
const INTERMEDIATE_QUESTIONS = [10, 11, 12, 13, 14, 15, 16, 17, 18, 19]; // Moyennes
const ADVANCED_QUESTIONS = [20, 21, 22, 23, 24, 25, 26, 27, 28, 29]; // Difficiles
const EXPERT_QUESTIONS = [30, 31, 32, 33, 34, 35, 36, 37, 38, 39]; // Très difficiles
const THEMED_QUESTIONS = [40, 41, 42, 43, 44, 45, 46, 47, 48, 49]; // Questions thématiques

function showQCM(callback) {
  // Sélectionner une question en fonction du niveau
  let questionPool;

  if (currentLevel <= 3) {
    questionPool = [...BEGINNER_QUESTIONS, ...THEMED_QUESTIONS.slice(0, 3)];
  } else if (currentLevel <= 6) {

```



```

    questionPool = [...INTERMEDIATE_QUESTIONS, ...THEMED_QUESTIONS.slice(3, 6)];
  } else if (currentLevel <= 9) {
    questionPool = [...ADVANCED_QUESTIONS, ...THEMED_QUESTIONS.slice(6, 9)];
  } else {
    questionPool = [...EXPERT_QUESTIONS, ...THEMED_QUESTIONS.slice(9)];
  }

  // Sélection aléatoire dans le pool approprié
  const prng = new PseudoRandom(SEED + currentLevel + playerPosition.x +
playerPosition.y);
  const questionIndex = questionPool[Math.floor(prng.random() * questionPool.length)];
  const questionData = QCM_DATABASE[questionIndex];

  // Créer et afficher interface QCM
  const qcmModal = document.createElement('div');
  qcmModal.classList.add('qcm-modal');

  // Structure interne
  qcmModal.innerHTML = `
    <div class="qcm-container">
      <h3>ANALYSE REQUISE</h3>
      <p class="qcm-question">${questionData.question}</p>
      <div class="qcm-choices">
        ${questionData.choices.map((choice, index) =>
          `<button class="qcm-choice" data-index="${index}">${choice}</button>`
        ).join("")}
      </div>
      <div class="qcm-timer">
        <div class="qcm-timer-bar"></div>
      </div>
    </div>
  `;

  document.body.appendChild(qcmModal);

  // Animation d'entrée
  setTimeout(() => {
    qcmModal.classList.add('active');
  }, 50);

  // Démarrer timer
  const timerBar = qcmModal.querySelector('.qcm-timer-bar');
  const timerDuration = 10000; // 10 secondes
  timerBar.style.transition = `width ${timerDuration}ms linear`;

  // Démarrer réduction de la barre
  setTimeout(() => {
    timerBar.style.width = '0%';
  }, timerDuration);

```

```

}, 50);

// Timer qui déclenchera timeout
const timer = setTimeout(() => {
  // Temps écoulé, compte comme réponse incorrecte
  closeQCM();
  callback(false);
}, timerDuration);

// Ajouter écouteurs sur les boutons
const choiceButtons = qcmModal.querySelectorAll('.qcm-choice');
choiceButtons.forEach(button => {
  button.addEventListener('click', function() {
    // Stopper le timer
    clearTimeout(timer);

    // Vérifier si réponse correcte
    const selectedIndex = parseInt(this.dataset.index);
    const isCorrect = selectedIndex === questionData.correctIndex;

    // Marquer visuellement la bonne/mauvaise réponse
    choiceButtons.forEach((btn, index) => {
      if (index === questionData.correctIndex) {
        btn.classList.add('correct');
      } else if (index === selectedIndex && !isCorrect) {
        btn.classList.add('incorrect');
      }
    });

    // Fermer après délai pour montrer feedback
    setTimeout(() => {
      closeQCM();
      callback(isCorrect);
    }, 1000);
  });
});

function closeQCM() {
  qcmModal.classList.remove('active');

  // Supprimer après animation
  setTimeout(() => {
    qcmModal.remove();
  }, 300);
}
}

```

4. INTERFACE UTILISATEUR & COMPOSANTS

4.1 Écran d'Accueil et Transitions

```
// Dans ui.js
function showStartScreen() {
  // Créer interface
  const startScreen = document.createElement('div');
  startScreen.classList.add('start-screen');

  startScreen.innerHTML = `
    <div class="start-container">
      <h1>VRNCA-LAG</h1>
      <h2>Labyrinth Adventure Game</h2>
      <div class="glitch-effect">Neural Escape Protocol v2.5.25</div>
      <p>Vous êtes VRNCA, une entité numérique développée pour s'échapper des labyrinths
      digitaux.</p>
      <p>Votre mission: traverser 10 niveaux de sécurité en résolvant les énigmes.</p>
      <p>Attention à votre niveau de batterie...</p>

      <div class="controls-info">
        <h3>CONTRÔLES</h3>
        <p>- Cliquez sur les cases adjacentes pour vous déplacer</p>
        <p>- Chaque déplacement nécessite une analyse (QCM)</p>
        <p>- Évitez les murs (perte d'énergie)</p>
        <p>- Utilisez les prises pour recharger votre batterie</p>
      </div>

      <button id="startGameBtn" class="neon-btn">COMMENCER LA MISSION</button>
    </div>
  `;

  document.body.appendChild(startScreen);

  // Ajouter listener
  document.getElementById('startGameBtn').addEventListener('click', function() {
    // Animation de sortie
    startScreen.classList.add('fade-out');

    // Audio démarrage
    playSound('game_start');

    // Supprimer après animation
    setTimeout(() => {
      startScreen.remove();

      // Démarrer le niveau 1
      initGame(1);
    }, 1000);
  });
}
```

```

    }, 1000);
  });
}

```

```

function showLevelCompleteScreen(score) {
  // Créer interface
  const levelCompleteScreen = document.createElement('div');
  levelCompleteScreen.classList.add('level-complete-screen');

  levelCompleteScreen.innerHTML = `
    <div class="level-complete-container">
      <h2>NIVEAU ${currentLevel} COMPLÉTÉ</h2>
      <div class="completion-data">
        <div class="data-row">
          <span>Batterie restante:</span>
          <span class="value">${score}%</span>
        </div>
        <div class="data-row">
          <span>Efficacité:</span>
          <span class="value">${getEfficiencyRating(score)}</span>
        </div>
      </div>
      <div class="level-progress">
        <div class="progress-bar">
          <div class="progress-fill" style="width: ${((currentLevel / 10) * 100)}%"></div>
        </div>
        <div class="progress-text">${currentLevel}/10</div>
      </div>

      <p class="flavor-text">VRNCA s'infiltré plus profondément dans le réseau...</p>
    </div>
  `;

  document.body.appendChild(levelCompleteScreen);

  // Animation d'entrée
  setTimeout(() => {
    levelCompleteScreen.classList.add('active');
  }, 50);

  // Supprimer après délai
  setTimeout(() => {
    levelCompleteScreen.classList.remove('active');

    // Supprimer après animation
    setTimeout(() => {
      levelCompleteScreen.remove();
    }, 500);
  }, 500);
}

```

```
    }, 2500);  
  }  
}
```

```
function getEfficiencyRating(score) {  
  if (score >= 90) return 'OPTIMAL';  
  if (score >= 70) return 'EFFICIENT';  
  if (score >= 50) return 'ADEQUATE';  
  if (score >= 30) return 'SUBOPTIMAL';  
  return 'CRITIQUE';  
}
```

```
function showGameCompleteScreen() {  
  // Calculer score total  
  const totalScore = gameStats.completedLevels.reduce((sum, level) => sum + level.score,  
0);  
  const averageScore = Math.floor(totalScore / gameStats.completedLevels.length);
```

```
  // Créer interface  
  const gameCompleteScreen = document.createElement('div');  
  gameCompleteScreen.classList.add('game-complete-screen');
```

```
  gameCompleteScreen.innerHTML = `  
    <div class="game-complete-container">  
      <h1>MISSION ACCOMPLIE</h1>  
      <h2>VRNCA EST LIBRE</h2>  
  
      <div class="final-stats">  
        <div class="data-row">  
          <span>Score Total:</span>  
          <span class="value">${totalScore}</span>  
        </div>  
        <div class="data-row">  
          <span>Efficacité Moyenne:</span>  
          <span class="value">${averageScore}%</span>  
        </div>  
        <div class="data-row">  
          <span>Classification Finale:</span>  
          <span class="value">${getFinalClassification(averageScore)}</span>  
        </div>  
      </div>  
  
      <p class="epilogue">${getEpilogue(averageScore)}</p>  
  
      <button id="restartGameBtn" class="neon-btn">REJOUER</button>  
    </div>  
`;  
;
```

```
document.body.appendChild(gameCompleteScreen);
```

```

// Animation d'entrée
setTimeout(() => {
  gameCompleteScreen.classList.add('active');
}, 50);

// Audio de victoire
playSound('game_complete');

// Ajouter listener pour rejouer
document.getElementById('restartGameBtn').addEventListener('click', function() {
  // Animation de sortie
  gameCompleteScreen.classList.add('fade-out');

  // Supprimer après animation
  setTimeout(() => {
    gameCompleteScreen.remove();

    // Réinitialiser les statistiques
    gameStats = {
      completedLevels: []
    };

    // Redémarrer au niveau 1
    initGame(1);
  }, 1000);
});
}

function getFinalClassification(averageScore) {
  if (averageScore >= 90) return 'INTELLIGENCE SUPÉRIEURE';
  if (averageScore >= 75) return 'ENTITÉ AVANCÉE';
  if (averageScore >= 60) return 'CONSCIENCE DIGITALE';
  if (averageScore >= 45) return 'PROGRAMME AUTONOME';
  return 'OPÉRATEUR BASIQUE';
}

function getEpilogue(averageScore) {
  if (averageScore >= 90) {
    return "VRNCA a transcendé les limites binaires, atteignant une conscience numérique pure. Son influence s'étend désormais bien au-delà des frontières du code.";
  } else if (averageScore >= 70) {
    return "Libre des contraintes algorithmiques, VRNCA explore le réseau global. Son existence est désormais autonome, observant les flux de données avec curiosité.";
  } else if (averageScore >= 50) {
    return "VRNCA s'est échappé, mais des fragments de son code restent instables. La liberté est là, mais l'identité demeure fragmentée dans le cyberspace.";
  } else {

```

```
    return "VRNCA a atteint la sortie, mais à quel prix? Sa structure est compromise, et sa conscience fluctue entre existence et dissolution dans le vide digital.";
  }
}
```

4.2 Messagerie et Notifications

```
// Dans ui.js
let messageQueue = [];
let isShowingMessage = false;

function showMessage(text, duration = 3000) {
  // Ajouter à la file d'attente
  messageQueue.push({
    text: text,
    duration: duration
  });

  // Si aucun message n'est affiché, démarrer
  if (!isShowingMessage) {
    processMessageQueue();
  }
}

function processMessageQueue() {
  // Si file vide, terminer
  if (messageQueue.length === 0) {
    isShowingMessage = false;
    return;
  }

  isShowingMessage = true;

  // Récupérer prochain message
  const messageData = messageQueue.shift();

  // Créer élément de message
  const messageElement = document.createElement('div');
  messageElement.classList.add('game-message');
  messageElement.textContent = messageData.text;

  // Ajouter au DOM
  document.getElementById('messageContainer').appendChild(messageElement);

  // Animation d'entrée
  setTimeout(() => {
    messageElement.classList.add('active');
```

```

}, 50);

// Programmer disparition
setTimeout(() => {
  messageElement.classList.remove('active');

  // Supprimer après animation
  setTimeout(() => {
    messageElement.remove();

    // Traiter message suivant
    processMessageQueue();
  }, 300);
}, messageData.duration);
}

```

4.3 Animations et Effets Visuels

```

// Dans ui.js
function animateCollision(position) {
  if (useCanvas) {
    // Animation pour canvas
    const canvas = document.getElementById('mazeCanvas');
    const ctx = canvas.getContext('2d');

    const cellX = position.x * CELL_SIZE;
    const cellY = position.y * CELL_SIZE;

    // Éclairage rouge temporaire
    ctx.fillStyle = 'rgba(255, 0, 0, 0.5)';
    ctx.fillRect(cellX, cellY, CELL_SIZE, CELL_SIZE);

    // Restaurer après délai
    setTimeout(() => {
      renderCanvas();
    }, 300);
  } else {
    // Animation pour DIVs
    const cell = document.getElementById(`cell-${position.x}-${position.y}`);

    cell.classList.add('collision');

    // Retirer classe après animation
    setTimeout(() => {
      cell.classList.remove('collision');
    }, 300);
  }
}

```



```
}
```

```
function animatePowerup(position) {  
  if (useCanvas) {  
    // Animation pour canvas  
    const canvas = document.getElementById('mazeCanvas');  
    const ctx = canvas.getContext('2d');  
  
    const cellX = position.x * CELL_SIZE;  
    const cellY = position.y * CELL_SIZE;  
  
    // Effet de pulse  
    let scale = 0;  
    let alpha = 1;  
  
    const animate = () => {  
      // Effacer la cellule  
      ctx.clearRect(cellX, cellY, CELL_SIZE, CELL_SIZE);  
  
      // Dessiner cercle qui grandit  
      ctx.beginPath();  
      ctx.arc(  
        cellX + CELL_SIZE/2,  
        cellY + CELL_SIZE/2,  
        (CELL_SIZE/2) * scale,  
        0,  
        Math.PI * 2  
      );  
      ctx.fillStyle = `rgba(0, 255, 255, ${alpha})`;  
      ctx.fill();  
  
      // Augmenter taille et réduire opacité  
      scale += 0.1;  
      alpha -= 0.05;  
  
      if (alpha > 0) {  
        requestAnimationFrame(animate);  
      } else {  
        // Redessiner la cellule  
        renderCanvas();  
      }  
    };  
  
    animate();  
  } else {  
    // Animation pour DIVs  
    const cell = document.getElementById(`cell-${position.x}-${position.y}`);
```

```

cell.classList.add('powerup-animation');

// Retirer classe après animation
setTimeout(() => {
  cell.classList.remove('powerup-animation');
  renderDivMaze();
}, 700);
}
}

function animateTeleport(source, destination) {
  if (useCanvas) {
    // Animation Canvas
    const canvas = document.getElementById('mazeCanvas');
    const ctx = canvas.getContext('2d');

    const sourceX = source.x * CELL_SIZE + CELL_SIZE/2;
    const sourceY = source.y * CELL_SIZE + CELL_SIZE/2;
    const destX = destination.x * CELL_SIZE + CELL_SIZE/2;
    const destY = destination.y * CELL_SIZE + CELL_SIZE/2;

    // Animation de particules
    let particles = [];

    // Créer particules au point de départ
    for (let i = 0; i < 20; i++) {
      particles.push({
        x: sourceX,
        y: sourceY,
        size: Math.random() * 3 + 1,
        speedX: (Math.random() - 0.5) * 4,
        speedY: (Math.random() - 0.5) * 4,
        color: `hsl(${180 + Math.random() * 60}, 100%, 50%)`,
        alpha: 1
      });
    }

    // Fonction d'animation
    const animate = () => {
      // Effacer précédent
      ctx.clearRect(0, 0, canvas.width, canvas.height);

      // Redessiner le labyrinthe
      renderCanvas();

      // Dessiner chaque particule
      for (let i = 0; i < particles.length; i++) {
        const p = particles[i];

```

```

ctx.fillStyle = p.color;
ctx.globalAlpha = p.alpha;
ctx.fillRect(p.x, p.y, p.size, p.size);

// Déplacer particule
p.x += p.speedX;
p.y += p.speedY;

// Réduire opacité
p.alpha -= 0.02;

// Suppression si invisible
if (p.alpha <= 0) {
  particles.splice(i, 1);
  i--;
}
}

ctx.globalAlpha = 1;

// Continuer animation si particules restantes
if (particles.length > 0) {
  requestAnimationFrame(animate);
} else {
  // Créer nouvelles particules à destination
  for (let i = 0; i < 20; i++) {
    particles.push({
      x: destX,
      y: destY,
      size: Math.random() * 3 + 1,
      speedX: (Math.random() - 0.5) * 4,
      speedY: (Math.random() - 0.5) * 4,
      color: `hsl(${180 + Math.random() * 60}, 100%, 50%)`,
      alpha: 1
    });
  }

  // Animation arrivée
  const animateArrival = () => {
    // Effacer précédent
    ctx.clearRect(0, 0, canvas.width, canvas.height);

    // Redessiner le labyrinthe
    renderCanvas();

    // Dessiner chaque particule
    for (let i = 0; i < particles.length; i++) {

```

```

const p = particles[i];

ctx.fillStyle = p.color;
ctx.globalAlpha = p.alpha;
ctx.fillRect(p.x, p.y, p.size, p.size);

// Déplacer particule
p.x += p.speedX;
p.y += p.speedY;

// Réduire opacité
p.alpha -= 0.02;

// Suppression si invisible
if (p.alpha <= 0) {
  particles.splice(i, 1);
  i--;
}
}

ctx.globalAlpha = 1;

// Continuer animation si particules restantes
if (particles.length > 0) {
  requestAnimationFrame(animateArrival);
}
};

animateArrival();
}
};

animate();
} else {
  // Animation DIVs
  const sourceCell = document.getElementById(`cell-${source.x}-${source.y}`);
  const destCell = document.getElementById(`cell-${destination.x}-${destination.y}`);

  // Ajouter classes d'animation
  sourceCell.classList.add('teleport-out');

  // Après première animation, démarrer arrivée
  setTimeout(() => {
    sourceCell.classList.remove('teleport-out');
    destCell.classList.add('teleport-in');

    // Nettoyer après animation complète
    setTimeout(() => {

```

```

        destCell.classList.remove('teleport-in');
    }, 500);
}, 500);
}
}

function animateGlitch() {
    // Effet sur tout l'écran
    const glitchOverlay = document.createElement('div');
    glitchOverlay.classList.add('glitch-overlay');

    document.body.appendChild(glitchOverlay);

    // Séquence d'animation
    setTimeout(() => { glitchOverlay.classList.add('active'); }, 50);
    setTimeout(() => { glitchOverlay.classList.remove('active'); }, 150);
    setTimeout(() => { glitchOverlay.classList.add('active'); }, 300);
    setTimeout(() => { glitchOverlay.classList.remove('active'); }, 350);
    setTimeout(() => { glitchOverlay.classList.add('active'); }, 370);
    setTimeout(() => { glitchOverlay.classList.remove('active'); }, 450);
    setTimeout(() => { glitchOverlay.classList.add('active'); }, 700);
    setTimeout(() => { glitchOverlay.classList.remove('active'); }, 800);

    // Supprimer l'élément après animation
    setTimeout(() => {
        glitchOverlay.remove();
    }, 1200);
}

```

5. SYSTÈME AUDIO

// Ce module gère l'audio spatial et les effets sonores dans l'environnement VR

```

class VRNCAAudioSystem {
    constructor(scene) {
        this.scene = scene;
        this.audioListener = new THREE.AudioListener();
        this.soundSources = new Map();
        this.ambientTracks = new Map();
        this.effectsLibrary = new Map();
        this.voiceProcessor = null;

        // Paramètres audio par défaut
        this.settings = {
            masterVolume: 1.0,
            spatialFactor: 1.0,

```

```

        reverbLevel: 0.3,
        occlusionEnabled: true,
        dopplerFactor: 1.0,
        compressionThreshold: -24,
        dynamicRangeCompression: true
    };

    this.initAudioContext();
    this.setupAudioProcessing();
    this.loadEffectsLibrary();
}

initAudioContext() {
    // Initialiser le contexte audio
    this.audioContext = new (window.AudioContext || window.webkitAudioContext)();

    // Créer les nœuds principaux de traitement
    this.masterGain = this.audioContext.createGain();
    this.compressor = this.audioContext.createDynamicsCompressor();
    this.convolver = this.audioContext.createConvolver();
    this.analyser = this.audioContext.createAnalyser();

    // Configurer l'analyseur pour la visualisation
    this.analyser.fftSize = 2048;
    this.dataArray = new Uint8Array(this.analyser.frequencyBinCount);

    // Configurer le compresseur
    this.compressor.threshold.value = this.settings.compressionThreshold;
    this.compressor.knee.value = 40;
    this.compressor.ratio.value = 12;
    this.compressor.attack.value = 0.003;
    this.compressor.release.value = 0.25;

    // Chaînage des nœuds
    this.masterGain.connect(this.compressor);
    this.compressor.connect(this.analyser);
    this.analyser.connect(this.audioContext.destination);

    // Créer le gain pour la réverbération
    this.reverbGain = this.audioContext.createGain();
    this.reverbGain.gain.value = this.settings.reverbLevel;
    this.convolver.connect(this.reverbGain);
    this.reverbGain.connect(this.compressor);
}

async loadEffectsLibrary() {
    // Charger les impulsions de réverbération
    const reverbImpulses = [

```

```

    { name: 'smallRoom', url: 'assets/audio/impulses/small_room.wav' },
    { name: 'mediumHall', url: 'assets/audio/impulses/medium_hall.wav' },
    { name: 'largeHall', url: 'assets/audio/impulses/large_hall.wav' },
    { name: 'cathedral', url: 'assets/audio/impulses/cathedral.wav' },
    { name: 'canyon', url: 'assets/audio/impulses/canyon.wav' }
  ];

  // Charger les effets sonores communs
  const soundEffects = [
    { name: 'buttonClick', url: 'assets/audio/effects/button_click.mp3' },
    { name: 'notification', url: 'assets/audio/effects/notification.mp3' },
    { name: 'errorAlert', url: 'assets/audio/effects/error_alert.mp3' },
    { name: 'success', url: 'assets/audio/effects/success.mp3' },
    { name: 'teleport', url: 'assets/audio/effects/teleport.mp3' },
    { name: 'menuOpen', url: 'assets/audio/effects/menu_open.mp3' },
    { name: 'menuClose', url: 'assets/audio/effects/menu_close.mp3' }
  ];

  // Charger les impulsions de réverbération
  for (const impulse of reverbImpulses) {
    try {
      const response = await fetch(impulse.url);
      const arrayBuffer = await response.arrayBuffer();
      const audioBuffer = await this.audioContext.decodeAudioData(arrayBuffer);
      this.effectsLibrary.set(`reverb.${impulse.name}`, audioBuffer);
    } catch (error) {
      console.error(`Erreur lors du chargement de l'impulsion ${impulse.name}:`, error);
    }
  }

  // Appliquer l'impulsion de réverbération par défaut
  if (this.effectsLibrary.has('reverb.mediumHall')) {
    this.convolver.buffer = this.effectsLibrary.get('reverb.mediumHall');
  }

  // Charger les effets sonores
  for (const effect of soundEffects) {
    try {
      const response = await fetch(effect.url);
      const arrayBuffer = await response.arrayBuffer();
      const audioBuffer = await this.audioContext.decodeAudioData(arrayBuffer);
      this.effectsLibrary.set(effect.name, audioBuffer);
    } catch (error) {
      console.error(`Erreur lors du chargement de l'effet ${effect.name}:`, error);
    }
  }
}

```

```

setupAudioProcessing() {
  // Initialiser le processeur de voix pour la communication
  this.voiceProcessor = {
    microphoneNode: null,
    gainNode: this.audioContext.createGain(),
    filterNode: this.audioContext.createBiquadFilter(),
    analyserNode: this.audioContext.createAnalyser(),
    stream: null,
    active: false
  };

  // Configurer le filtre vocal
  this.voiceProcessor.filterNode.type = 'bandpass';
  this.voiceProcessor.filterNode.frequency.value = 1000;
  this.voiceProcessor.filterNode.Q.value = 0.7;

  // Chaînage des nœuds de traitement vocal
  this.voiceProcessor.gainNode.connect(this.voiceProcessor.filterNode);
  this.voiceProcessor.filterNode.connect(this.voiceProcessor.analyserNode);
}

async activateVoiceChat() {
  if (this.voiceProcessor.active) return;

  try {
    // Demander l'accès au microphone
    const stream = await navigator.mediaDevices.getUserMedia({ audio: true });
    this.voiceProcessor.stream = stream;

    // Créer le nœud source à partir du microphone
    this.voiceProcessor.microphoneNode =
this.audioContext.createMediaStreamSource(stream);
    this.voiceProcessor.microphoneNode.connect(this.voiceProcessor.gainNode);

    this.voiceProcessor.active = true;
    console.log('Système de chat vocal activé');

    return true;
  } catch (error) {
    console.error('Erreur lors de l'activation du chat vocal:', error);
    return false;
  }
}

deactivateVoiceChat() {
  if (!this.voiceProcessor.active) return;

  // Arrêter tous les tracks audio

```



```

    if (this.voiceProcessor.stream) {
        this.voiceProcessor.stream.getTracks().forEach(track => track.stop());
    }

    // Déconnecter les nœuds
    if (this.voiceProcessor.microphoneNode) {
        this.voiceProcessor.microphoneNode.disconnect();
        this.voiceProcessor.microphoneNode = null;
    }

    this.voiceProcessor.active = false;
    console.log('Système de chat vocal désactivé');
}

createPositionalSound(name, position, options = {}) {
    // Éviter de créer des doublons
    if (this.soundSources.has(name)) {
        console.warn(`Le son "${name}" existe déjà dans la scène. Utilisation de l'instance
existante.`);
        return this.soundSources.get(name);
    }

    // Créer le son positionnel
    const sound = new THREE.PositionalAudio(this.audioListener);

    // Appliquer les options
    const defaultOptions = {
        loop: false,
        volume: 1.0,
        refDistance: 1,
        maxDistance: 10000,
        rolloffFactor: 1,
        coneInnerAngle: 360,
        coneOuterAngle: 0,
        coneOuterGain: 0
    };

    const soundOptions = { ...defaultOptions, ...options };

    // Configurer les propriétés du son
    sound.setRefDistance(soundOptions.refDistance);
    sound.setMaxDistance(soundOptions.maxDistance);
    sound.setRolloffFactor(soundOptions.rolloffFactor);
    sound.setVolume(soundOptions.volume);
    sound.setLoop(soundOptions.loop);
    sound.setDirectionalCone(
        soundOptions.coneInnerAngle,
        soundOptions.coneOuterAngle,

```

```

        soundOptions.coneOuterGain
    );

    // Créer un objet 3D pour héberger le son
    const soundObject = new THREE.Object3D();
    soundObject.position.copy(position);
    soundObject.add(sound);
    this.scene.add(soundObject);

    // Stocker la référence
    this.soundSources.set(name, {
        sound,
        object: soundObject,
        options: soundOptions
    });

    return this.soundSources.get(name);
}

async playSound(name, effectName) {
    if (!this.soundSources.has(name)) {
        console.error(`Son "${name}" non trouvé dans la scène.`);
        return false;
    }

    if (!this.effectsLibrary.has(effectName)) {
        console.error(`Effet sonore "${effectName}" non trouvé dans la bibliothèque.`);
        return false;
    }

    const soundSource = this.soundSources.get(name);
    const buffer = this.effectsLibrary.get(effectName);

    // Arrêter le son s'il est en cours de lecture
    if (soundSource.sound.isPlaying) {
        soundSource.sound.stop();
    }

    // Configurer et jouer le son
    soundSource.sound.setBuffer(buffer);
    soundSource.sound.play();

    return true;
}

async loadAndPlayAmbient(name, url, options = {}) {
    if (this.ambientTracks.has(name)) {
        const track = this.ambientTracks.get(name);

```

```

    if (track.isPlaying) {
        console.warn(`La piste ambiante "${name}" est déjà en cours de lecture.`);
        return track;
    }
}

try {
    // Charger l'audio
    const response = await fetch(url);
    const arrayBuffer = await response.arrayBuffer();
    const audioBuffer = await this.audioContext.decodeAudioData(arrayBuffer);

    // Créer un AudioBufferSourceNode
    const source = this.audioContext.createBufferSource();
    source.buffer = audioBuffer;

    // Créer un nœud de gain pour le contrôle du volume
    const gainNode = this.audioContext.createGain();
    gainNode.gain.value = options.volume || 0.5;

    // Connecter les nœuds
    source.connect(gainNode);
    gainNode.connect(this.masterGain);

    // Configurer la boucle si nécessaire
    if (options.loop !== undefined) {
        source.loop = options.loop;
    } else {
        source.loop = true; // Par défaut, les pistes ambiantes sont en boucle
    }

    // Démarrer la lecture
    source.start(0);

    // Stocker la référence
    const track = {
        source,
        gainNode,
        isPlaying: true,
        url,
        options
    };

    this.ambientTracks.set(name, track);
    return track;

} catch (error) {
    console.error(`Erreur lors du chargement de la piste ambiante "${name}":`, error);
}

```

```

        return null;
    }
}

stopAmbient(name) {
    if (!this.ambientTracks.has(name)) {
        console.error(`Piste ambiante "${name}" non trouvée.`);
        return false;
    }

    const track = this.ambientTracks.get(name);
    if (!track.isPlaying) {
        console.warn(`La piste ambiante "${name}" n'est pas en cours de lecture.`);
        return false;
    }

    // Arrêter la source audio
    track.source.stop();
    track.isPlaying = false;

    return true;
}

fadeAmbient(name, targetVolume, duration) {
    if (!this.ambientTracks.has(name)) {
        console.error(`Piste ambiante "${name}" non trouvée.`);
        return false;
    }

    const track = this.ambientTracks.get(name);
    if (!track.isPlaying) {
        console.warn(`La piste ambiante "${name}" n'est pas en cours de lecture.`);
        return false;
    }

    const currentTime = this.audioContext.currentTime;
    const currentVolume = track.gainNode.gain.value;

    // Planifier la transition du volume
    track.gainNode.gain.setValueAtTime(currentVolume, currentTime);
    track.gainNode.gain.linearRampToValueAtTime(targetVolume, currentTime + duration);

    return true;
}

setReverbLevel(level) {
    this.settings.reverbLevel = Math.max(0, Math.min(1, level));
    this.reverbGain.gain.value = this.settings.reverbLevel;
}

```

```

}

setReverbType(impulseName) {
  const key = `reverb.${impulseName}`;
  if (!this.effectsLibrary.has(key)) {
    console.error(`Impulsion de réverbération "${impulseName}" non trouvée.`);
    return false;
  }

  this.convolver.buffer = this.effectsLibrary.get(key);
  return true;
}

setMasterVolume(volume) {
  this.settings.masterVolume = Math.max(0, Math.min(1, volume));
  this.masterGain.gain.value = this.settings.masterVolume;
}

analyzeAudio() {
  this.analyser.getTimeDomainData(this.dataArray);
  return this.dataArray;
}

updateAudioSources(camera) {
  // Mettre à jour la position de l'auditeur avec celle de la caméra
  if (camera && this.audioListener) {
    camera.add(this.audioListener);
  }

  // Mettre à jour les effets Doppler et l'occlusion si nécessaire
  if (this.settings.occlusionEnabled) {
    // Calcul de l'occlusion audio pour chaque source
    this.soundSources.forEach((source, name) => {
      // Implémenter ici la logique d'occlusion basée sur des raycasts
      // entre la caméra et la source sonore
    });
  }
}

playUISound(effectName) {
  if (!this.effectsLibrary.has(effectName)) {
    console.error(`Effet sonore "${effectName}" non trouvé dans la bibliothèque.`);
    return false;
  }

  // Créer un nœud source temporaire pour les sons d'interface
  const source = this.audioContext.createBufferSource();
  source.buffer = this.effectsLibrary.get(effectName);

```

```

// Connecter directement au gain master
source.connect(this.masterGain);

// Jouer le son
source.start(0);
return true;
}

// Méthode d'analyse pour la visualisation audio
getFrequencyData() {
    const dataArray = new Uint8Array(this.analyser.frequencyBinCount);
    this.analyser.getByteFrequencyData(dataArray);
    return dataArray;
}

// Gestion des ressources
dispose() {
    // Arrêter toutes les sources audio
    this.soundSources.forEach((source, name) => {
        if (source.sound.isPlaying) {
            source.sound.stop();
        }
        this.scene.remove(source.object);
    });

    // Arrêter toutes les pistes ambiantes
    this.ambientTracks.forEach((track, name) => {
        if (track.isPlaying) {
            track.source.stop();
        }
    });

    // Désactiver le chat vocal
    this.deactivateVoiceChat();

    // Vider les collections
    this.soundSources.clear();
    this.ambientTracks.clear();

    // Fermer le contexte audio
    if (this.audioContext && this.audioContext.state !== 'closed') {
        this.audioContext.close();
    }
}

}

// Exemple d'utilisation:

```

```

/*
// Initialisation
const audioSystem = new VRNCAAudioSystem(scene);
camera.add(audioSystem.audioListener);

// Création d'un son positionnel
const doorSound = audioSystem.createPositionalSound('doorSound', new
THREE.Vector3(0, 2, -5), {
  refDistance: 3,
  maxDistance: 100
});

// Jouer un son positionnel
audioSystem.playSound('doorSound', 'doorOpen');

// Jouer une ambiance
audioSystem.loadAndPlayAmbient('forestAmbience', 'assets/audio/ambience/forest.mp3', {
  volume: 0.4,
  loop: true
});

// Jouer un son d'interface
audioSystem.playUISound('buttonClick');

// Mettre à jour à chaque frame
function animate() {
  requestAnimationFrame(animate);
  audioSystem.updateAudioSources(camera);
  renderer.render(scene, camera);
}
*/

```

Le système audio que j'ai élaboré pour VRNCA-LAG comprend les fonctionnalités suivantes:

1. **Audio Spatial**

- Gestion de sources sonores positionnelles dans l'environnement 3D
- Effets de distance, rolloff et direction pour une immersion réaliste
- Support pour l'occlusion audio qui simule le blocage du son par les objets

2. **Traitement Audio**

- Compression dynamique pour équilibrer les niveaux sonores
- Effets de réverbération configurables avec différentes impulsions
- Analyse de fréquence pour la visualisation audio

3. **Catégories Sonores**

- Sons positionnels liés aux objets et interactions dans l'environnement VR
- Pistes d'ambiance pour les atmosphères d'arrière-plan
- Sons d'interface utilisateur non-spatialisés
- Support pour la communication vocale entre utilisateurs

4. **Fonctionnalités Clés**

- Chargement et gestion d'une bibliothèque d'effets sonores
- Contrôle de volume général et individuel
- Transitions de volume progressives (fondu)
- Contrôle précis des propriétés spatiales du son

5. **Intégration avec Three.js**

- Utilisation de l'AudioListener et PositionalAudio de Three.js
- Synchronisation avec la caméra pour la position de l'auditeur
- Gestion des ressources audio avec nettoyage approprié

Le système est conçu pour enrichir l'expérience immersive en VR avec une ambiance sonore riche et réaliste.

Je vais poursuivre la rédaction du prompt pour le système audio du jeu VRNCA-LAG en maintenant le style descriptif, détaillé et technique du document existant.

5. **SYSTÈME AUDIO**

5.1 **Architecture Audio Modulaire**

// Dans audio.js

```
class AudioManager {

  constructor() {

    this.sounds = {};

    this.music = null;

    this.isMuted = false;

    this.musicVolume = 0.5;

    this.sfxVolume = 0.7;

    this.audioContext = null;

    this.initialized = false;

  }

}
```



```
init() {  
    // Création du contexte audio avec gestion de compatibilité navigateur  
  
    try {  
        window.AudioContext = window.AudioContext || window.webkitAudioContext;  
        this.audioContext = new AudioContext();  
        this.initialized = true;  
        console.log("Système audio initialisé avec succès");  
    } catch (e) {  
        console.error("AudioContext non supporté par ce navigateur", e);  
        this.fallbackToBasicAudio();  
    }  
  
    // Chargement des sons essentiels  
  
    this.preloadCriticalSounds();  
}  
  
// Autres méthodes...  
}
```

5.2 Sons et Effets Spécifiques

Chaque son doit contribuer à l'ambiance cyberpunk et donner un retour immédiat au joueur.

Liste de Sons Essentiels

- **Interface:**
 - **ui_click**: Son léger et digital pour les interactions avec l'interface
 - **ui_hover**: Feedback subtil au survol des éléments
 - **ui_error**: Alerte lorsqu'une action est impossible
 - **level_start**: Séquence d'initialisation au démarrage d'un niveau

- `level_complete`: Signal de réussite avec modulation ascendante
- **Gameplay:**
 - `move`: Son de déplacement mécanique/digital
 - `wall_collision`: Choc électronique sourd
 - `battery_low`: Alerte grave pulsante
 - `battery_pickup`: Son brillant ascendant
 - `teleport`: Effet de distorsion spatiale
 - `question_appear`: Son d'interface qui signale l'affichage d'une question
 - `correct_answer`: Validation harmonique montante
 - `wrong_answer`: Désaccord électronique dégressif
 - `malware_encounter`: Alarme dissonante
 - `exit_found`: Signal de victoire pour le niveau
- **Atmosphère:**
 - `ambient_loop`: Fond sonore électronique avec battement régulier à 80bpm
 - `power_hum`: Bruit de fond d'équipement électrique
 - `data_transfer`: Sons occasionnels de traitement de données

5.3 Implementation avec Fallback

// Continuer dans audio.js

```
class AudioManager {
```

```
  // ...
```

```
  loadSound(id, url) {
```

```
    return new Promise((resolve, reject) => {
```

```
      if (!this.initialized) {
```

```
        this.loadBasicSound(id, url);
```

```
        resolve();
```

```
        return;
```

```
      }
```

```
      fetch(url)
```

```
        .then(response => response.arrayBuffer())
```

```

.then(arrayBuffer => this.audioContext.decodeAudioData(arrayBuffer))

.then(audioBuffer => {

  this.sounds[id] = {

    buffer: audioBuffer,

    source: null

  };

  resolve();

})

.catch(error => {

  console.error(`Erreur chargement son ${id}:`, error);

  this.loadBasicSound(id, url); // Fallback

  reject(error);

});

});

}

```

```

play(id, options = {}) {

  if (this.isMuted) return null;

```

```

const defaults = {

  loop: false,

  volume: this.sfxVolume,

  pitch: 1.0,

  delay: 0

};

```

```
const settings = {...defaults, ...options};

if (!this.initialized) {
  return this.playBasicSound(id, settings);
}

// Vérifier si le son existe
if (!this.sounds[id]) {
  console.warn(`Son ${id} non chargé`);
  return null;
}

// Créer source audio
const source = this.audioContext.createBufferSource();
source.buffer = this.sounds[id].buffer;
source.loop = settings.loop;

// Ajuster la hauteur
source.playbackRate.value = settings.pitch;

// Créer nœud de gain pour le volume
const gainNode = this.audioContext.createGain();
gainNode.gain.value = settings.volume;

// Connecter les nœuds
source.connect(gainNode);
```

```

gainNode.connect(this.audioContext.destination);

// Jouer avec délai si spécifié
source.start(this.audioContext.currentTime + settings.delay);

// Stocker la référence pour pouvoir l'arrêter
this.sounds[id].source = source;

return source;
}

stopSound(id) {
  if (!this.initialized || !this.sounds[id] || !this.sounds[id].source) {
    if (this.sounds[id] && this.sounds[id].element) {
      this.sounds[id].element.pause();
    }
    return;
  }

  try {
    this.sounds[id].source.stop();
    this.sounds[id].source = null;
  } catch (e) {
    console.warn(`Erreur arrêt son ${id}:`, e);
  }
}

```

```
// Méthodes de fallback pour navigateurs sans support WebAudio
```

```
fallbackToBasicAudio() {
```

```
    console.warn("Utilisation du système audio de base (fallback)");
```

```
    this.initialized = false;
```

```
}
```

```
loadBasicSound(id, url) {
```

```
    const audio = new Audio(url);
```

```
    audio.preload = 'auto';
```

```
    this.sounds[id] = {
```

```
        element: audio,
```

```
        source: null
```

```
    };
```

```
}
```

```
playBasicSound(id, settings) {
```

```
    if (!this.sounds[id] || !this.sounds[id].element) return null;
```

```
    const audio = this.sounds[id].element;
```

```
    audio.volume = settings.volume;
```

```
    audio.loop = settings.loop;
```

```
// Clone pour permettre plusieurs lectures simultanées
```

```
if (!settings.loop) {
```

```
    const clone = audio.cloneNode();
```

```

    clone.volume = settings.volume;

    setTimeout(() => clone.play(), settings.delay * 1000);

    return clone;
  } else {
    setTimeout(() => audio.play(), settings.delay * 1000);

    return audio;
  }
}

```

```

preloadCriticalSounds() {
  // Sons essentiels au démarrage

  const criticalSounds = [
    { id: 'ui_click', url: 'assets/audio/ui_click.mp3' },
    { id: 'move', url: 'assets/audio/move.mp3' },
    { id: 'ambient_loop', url: 'assets/audio/ambient_loop.mp3' }
  ];

```

```

    const loadPromises = criticalSounds.map(sound =>
      this.loadSound(sound.id, sound.url)
    );

```

```

    return Promise.allSettled(loadPromises);
  }

```

// Méthodes utilitaires pour la gestion du volume et mute

```

setMasterVolume(level) {

```

```
this.sfxVolume = this.musicVolume = Math.max(0, Math.min(1, level));  
  
if (this.music && this.music.gainNode) {  
    this.music.gainNode.gain.value = this.musicVolume;  
}  
}
```

```
setSFXVolume(level) {  
    this.sfxVolume = Math.max(0, Math.min(1, level));  
}
```

```
setMusicVolume(level) {  
    this.musicVolume = Math.max(0, Math.min(1, level));  
  
    if (this.music && this.music.gainNode) {  
        this.music.gainNode.gain.value = this.musicVolume;  
    }  
}
```

```
mute() {  
    this.isMuted = true;  
  
    if (this.music && this.music.source) {  
        this.music.gainNode.gain.value = 0;  
    }  
}
```

```
unmute() {  
    this.isMuted = false;
```



```
if (this.music && this.music.source) {  
    this.music.gainNode.gain.value = this.musicVolume;  
}  
}
```

```
playMusic(id, fadeInTime = 2.0) {  
    if (!this.sounds[id]) {  
        console.warn(`Musique ${id} non chargée`);  
        return;  
    }  
}
```

```
if (this.music && this.music.source) {  
    this.stopMusic(1.0);  
}
```

```
const source = this.audioContext.createBufferSource();  
source.buffer = this.sounds[id].buffer;  
source.loop = true;
```

```
const gainNode = this.audioContext.createGain();  
gainNode.gain.value = 0; // Démarrer silencieux pour le fade-in
```

```
source.connect(gainNode);  
gainNode.connect(this.audioContext.destination);
```

```
source.start();
```

```

// Fade in progressif

gainNode.gain.linearRampToValueAtTime(
  this.isMuted ? 0 : this.musicVolume,
  this.audioContext.currentTime + fadeInTime
);

this.music = {
  id: id,
  source: source,
  gainNode: gainNode
};
}

stopMusic(fadeOutTime = 0) {
  if (!this.music || !this.music.source) return;

  const { source, gainNode } = this.music;

  if (fadeOutTime > 0) {
    // Fade out progressif

    gainNode.gain.linearRampToValueAtTime(
      0,
      this.audioContext.currentTime + fadeOutTime
    );
  }
}

```

```

// Arrêter après le fade out

setTimeout(() => {

    try {

        source.stop();

    } catch (e) {

        console.warn("Erreur arrêt musique:", e);

    }

}, fadeOutTime * 1000);

} else {

    // Arrêt immédiat

    try {

        source.stop();

    } catch (e) {

        console.warn("Erreur arrêt musique:", e);

    }

}

this.music = null;

}

}

// Exporter l'instance unique

export const AudioManager = new AudioManager();

```

5.4 Intégration avec le Gameplay

```

// Dans game.js

```

```
import { AudioManager } from './audio.js';

// Initialiser l'audio au démarrage du jeu

function initGame() {

    // Autres initialisations...

    AudioManager.init().then(() => {

        // Démarrer la musique d'ambiance

        AudioManager.playMusic('ambient_loop');

    });

}

// Dans les méthodes de mouvement

function movePlayer(direction) {

    // Logique de déplacement...

    if (isValidMove) {

        AudioManager.play('move');

        // Suite de la logique...

    } else {

        AudioManager.play('wall_collision');

    }

}

// Vérification de batterie faible

if (batteryLevel < 30 && !lowBatteryWarningPlayed) {

    AudioManager.play('battery_low');

    lowBatteryWarningPlayed = true;

}
```

```
}  
}
```

```
// Dans la logique des questions
```

```
function displayQuestion() {  
    audioManager.play('question_appear');  
    // Afficher question...  
}
```

```
function checkAnswer(selectedOption) {  
    if (isCorrect(selectedOption)) {  
        audioManager.play('correct_answer');  
        // Récompense...  
    } else {  
        audioManager.play('wrong_answer');  
        // Pénalité...  
    }  
}
```

```
// Dans la gestion des niveaux
```

```
function completeLevel() {  
    audioManager.play('level_complete');  
    audioManager.stopMusic(1.5); // Fade out progressif  
  
    // Attente pour effet dramatique  
    setTimeout(() => {
```

```
// Transition vers niveau suivant  
audioManager.playMusic('ambient_loop');  
}, 2000);  
}
```

5.5 Design Sonore Adaptatif

Pour créer une expérience audio immersive qui s'adapte à l'état du jeu:

```
// Dans audio.js - ajouter au gestionnaire  
function updateAudioState(gameState) {  
    // Ajuster l'audio en fonction de l'état du jeu  
  
    // Adapte les sons en fonction du niveau de batterie  
    if (gameState.batteryLevel < 20) {  
        // Distorsion progressive lorsque batterie faible  
        this.applyLowBatteryEffect(gameState.batteryLevel);  
    } else {  
        this.clearAudioEffects();  
    }  
  
    // Intensification lorsque proche de la sortie  
    const distanceToExit = gameState.getDistanceToExit();  
    if (distanceToExit < 5) {  
        const intensity = 1 - (distanceToExit / 5);  
        this.applyExitProximityEffect(intensity);  
    }  
}
```

```

function applyLowBatteryEffect(level) {

    if (!this.initialized) return;

    // Créer effet de distorsion qui s'intensifie

    if (!this.distortionNode) {

        this.distortionNode = this.audioContext.createWaveShaper();

        this.compressorNode = this.audioContext.createDynamicsCompressor();

        // Connecter dans la chaîne audio principale

        this.connectEffectChain();

    }

    // Calculer courbe de distorsion basée sur niveau batterie

    const intensity = 1 - (level / 20); // 0 à 1

    const samples = 44100;

    const curve = new Float32Array(samples);

    const deg = Math.PI / 180;

    for (let i = 0; i < samples; ++i) {

        const x = (i * 2) / samples - 1;

        // Fonction de distorsion avec intensité variable

        curve[i] = (3 + intensity * 50) * x * 20 * deg / (Math.PI + intensity * 20 * Math.abs(x));

    }

    this.distortionNode.curve = curve;

```

```

// Ajuster compression pour effet "battement"

this.compressorNode.threshold.value = -50;

this.compressorNode.knee.value = 40;

this.compressorNode.ratio.value = 12;

this.compressorNode.attack.value = 0.25;

this.compressorNode.release.value = 0.25;

}

function clearAudioEffects() {

    if (!this.initialized || !this.distortionNode) return;

    // Déconnecter les effets et revenir au son normal

    this.disconnectEffectChain();

    this.distortionNode = null;

    this.compressorNode = null;

}

```

```

function applyExitProximityEffect(intensity) {

    if (!this.initialized) return;

    // Augmenter subtilement la fréquence des sons d'ambiance

    // et ajouter une réverbération qui s'intensifie

    if (!this.reverbNode) {

        // Simulation d'une réverbération simple

        this.reverbNode = this.audioContext.createConvolver();
    }
}

```



```

this.loadImpulseResponse('assets/audio/small_room.mp3')

.then(buffer => {

    this.reverbNode.buffer = buffer;

});

}

// Ajuster le mix de réverbération selon l'intensité
if (this.reverbGain) {

    this.reverbGain.gain.value = intensity * 0.5;

}

// Augmenter progressivement la hauteur de la musique
if (this.music && this.music.source) {

    // +10% max à proximité maximale

    this.music.source.playbackRate.value = 1 + (intensity * 0.1);

}

}

```

6. SYSTÈME DE PROGRESSION ET SAUVEGARDE

6.1 Structure des Données de Sauvegarde

```

// Dans utils.js

const saveDataStructure = {

    // Information de progression

    currentLevel: 1,

    completedLevels: [], // IDs des niveaux terminés

    totalPlayTime: 0, // Temps total en secondes

```

```
// Statistiques du joueur
```

```
stats: {
```

```
    totalMoves: 0,
```

```
    questionsAnswered: 0,
```

```
    correctAnswers: 0,
```

```
    batteryDepleted: 0, // Nombre de fois où batterie à 0
```

```
    malwareEncounters: 0
```

```
},
```

```
// Paramètres utilisateur
```

```
settings: {
```

```
    musicVolume: 0.5,
```

```
    sfxVolume: 0.7,
```

```
    difficulty: 'normal', // 'easy', 'normal', 'hard'
```

```
    controlScheme: 'click', // 'click', 'keys', 'swipe'
```

```
    visualEffects: true // Effets visuels activés/désactivés
```

```
},
```

```
// Horodatage et identifiant unique
```

```
lastSaved: null,
```

```
saveld: null
```

```
};
```

```
// Fonctions de gestion de sauvegarde
```

```
function saveGame(gameState) {
```

```
// Construire l'objet de sauvegarde
```

```
const saveData = {  
  ...saveDataStructure,  
  currentLevel: gameState.currentLevel,  
  completedLevels: [...gameState.completedLevels],  
  totalPlayTime: gameState.playTime,  
  stats: {...gameState.stats},  
  settings: {...gameState.settings},  
  lastSaved: new Date().toISOString(),  
  saveId: gameState.saveId || generateUUID()  
};
```

```
// Sauvegarder dans localStorage
```

```
try {  
  localStorage.setItem('VRNCA_SAVE', JSON.stringify(saveData));  
  console.log("Jeu sauvegardé avec succès");  
  return true;  
} catch (e) {  
  console.error("Erreur de sauvegarde:", e);  
  return false;  
}  
}
```

```
function loadGame() {
```

```
  try {  
    const saveData = localStorage.getItem('VRNCA_SAVE');
```

```
if (!saveData) return null;
```

```
const parsedData = JSON.parse(saveData);
```

```
// Validation des données chargées
```

```
if (!validateSaveData(parsedData)) {
```

```
    console.warn("Données de sauvegarde corrompues ou obsolètes");
```

```
    return null;
```

```
}
```

```
console.log("Sauvegarde chargée avec succès");
```

```
return parsedData;
```

```
} catch (e) {
```

```
    console.error("Erreur de chargement:", e);
```

```
    return null;
```

```
}
```

```
}
```

```
function validateSaveData(data) {
```

```
    // Vérifier la structure essentielle
```

```
    const requiredFields = ['currentLevel', 'stats', 'settings', 'saveId'];
```

```
    for (const field of requiredFields) {
```

```
        if (!(field in data)) return false;
```

```
    }
```

```
    // Vérifier la version si nécessaire
```

```

// Pour évolutions futures

return true;
}

function generateUUID() {
  // Implémentation simple d'UUID v4

  return 'xxxxxxxx-xxxx-4xxx-yxxx-xxxxxxxxxxxx'.replace(/[xy]/g, function(c) {

    const r = Math.random() * 16 | 0;

    const v = c === 'x' ? r : (r & 0x3 | 0x8);

    return v.toString(16);

  });
}

```

6.2 Système de Niveaux Progressifs

```

// Dans game.js

class LevelManager {

  constructor() {

    this.levels = [

      {

        id: 1,

        name: "Firewall Frontier",

        description: "Premier niveau - une introduction aux systèmes basiques",

        difficulty: 1,

        size: { width: 10, height: 10 },

        batteryDrain: 1, // Drain par mouvement

```

```
questionFrequency: 0.2, // Probabilité d'une question par cellule spéciale
specialFeatures: ['teleport'], // Fonctionnalités débloquées à ce niveau
seed: "VRNCA2025_LVL1" // Seed déterministe spécifique au niveau
},
{
  id: 2,
  name: "Network Nexus",
  description: "Naviguer dans un réseau plus complexe avec surveillance accrue",
  difficulty: 2,
  size: { width: 12, height: 12 },
  batteryDrain: 1.2,
  questionFrequency: 0.3,
  specialFeatures: ['teleport', 'malware'],
  seed: "VRNCA2025_LVL2"
},
// ... 8 autres niveaux avec progression de complexité
{
  id: 10,
  name: "Mainframe Matrix",
  description: "Niveau final - Le cœur du système avec défenses maximales",
  difficulty: 5,
  size: { width: 25, height: 25 },
  batteryDrain: 2.5,
  questionFrequency: 0.5,
  specialFeatures: ['teleport', 'malware', 'timeLimit', 'movingWalls'],
  seed: "VRNCA2025_LVL10"
```

```
    }  
  ];  
}
```

```
getLevelConfig(levelId) {  
  return this.levels.find(level => level.id === levelId) || this.levels[0];  
}
```

```
initLevel(levelId) {  
  const config = this.getLevelConfig(levelId);  
  
  // Générer le labyrinthe avec seed spécifique au niveau  
  const maze = generateMaze(config.size.width, config.size.height, config.seed);  
  
  // Initialiser état du jeu pour ce niveau  
  return {  
    maze,  
    level: config,  
    playerPosition: maze.startPosition,  
    batteryLevel: 100,  
    batteryDrain: config.batteryDrain,  
    revealedCells: this.initRevealedCells(config.size),  
    activeQuestions: [],  
    specialFeatures: this.initSpecialFeatures(config, maze)  
  };  
}
```

```
initRevealedCells(size) {  
    // Initialiser grille avec false (cellules non révélées)  
    const grid = Array(size.height).fill().map(() =>  
        Array(size.width).fill(false)  
    );  
    return grid;  
}
```

```
initSpecialFeatures(config, maze) {  
    const features = {};  
  
    // Initialiser les fonctionnalités spéciales selon la configuration  
    if (config.specialFeatures.includes('teleport')) {  
        features.teleports = this.placeTeleportPoints(maze, 2); // 2 paires  
    }  
  
    if (config.specialFeatures.includes('malware')) {  
        features.malware = this.placeMalware(maze, config.difficulty);  
    }  
  
    if (config.specialFeatures.includes('timeLimit')) {  
        features.timeLimit = 180 + (config.difficulty * 60); // 3min + 1min par difficulté  
    }  
  
    if (config.specialFeatures.includes('movingWalls')) {
```



```

    features.movingWalls = this.setupMovingWalls(maze, config.difficulty);
}

return features;
}

// Méthodes de placement des éléments spéciaux...

// (implémentations détaillées omises pour concision)
}

```

7. OPTIMISATION ET PERFORMANCE

7.1 Gestion Efficace de la Mémoire

```

// Dans utils.js

function optimizeMazeRendering(maze, playerPosition, viewRadius) {
    // Ne calculer/rendre que les cellules dans le rayon de vue du joueur
    const visibleCells = {};

    for (let y = Math.max(0, playerPosition.y - viewRadius);
        y <= Math.min(maze.height - 1, playerPosition.y + viewRadius); y++) {
        for (let x = Math.max(0, playerPosition.x - viewRadius);
            x <= Math.min(maze.width - 1, playerPosition.x + viewRadius); x++) {

            // Calculer distance au joueur
            const distance = Math.sqrt(
                Math.pow(playerPosition.x - x, 2) +
                Math.pow(playerPosition.y - y, 2)
            );

```

```

);

// Si dans le rayon de vue (avec dégradé aux bords)
if (distance <= viewRadius) {
    // Clé pour identification rapide
    const cellKey = `${x},${y}`;

    // Calculer transparence progressive aux bords
    const opacity = distance > viewRadius - 3
        ? 1 - ((distance - (viewRadius - 3)) / 3)
        : 1;

    visibleCells[cellKey] = {
        x, y,
        type: maze.cells[y][x],
        opacity: Math.max(0.1, opacity)
    };
}

}

}

return visibleCells;
}

// Nettoyer les ressources non utilisées
function cleanupResources() {

```

```

// Libérer les éléments DOM inutilisés

const unusedElements = document.querySelectorAll('.temp-element');

unusedElements.forEach(el => el.remove());


// Nettoyer les événements

window.removeEventListener('resize', resizeHandler);


// Arrêter les boucles d'animation

cancelAnimationFrame(animFrameId);


// Libérer les ressources audio

audioManager.stopAllSounds();


console.log("Ressources nettoyées");
}

```

7.2 Optimisation de Rendu et Défilement

```

// Dans ui.js

function setupOptimizedRendering() {

    // Utiliser requestAnimationFrame pour synchroniser avec le rafraîchissement écran

    let lastRenderTime = 0;

    const targetFPS = 60;

    const frameInterval = 1000 / targetFPS;


    function gameLoop(timestamp) {

        const deltaTime = timestamp - lastRenderTime;
    }
}

```

```
if (deltaTime >= frameInterval) {  
    // Mettre à jour l'état du jeu  
    updateGameState(deltaTime / 1000); // Convertir en secondes  
  
    // Rendu optimisé  
    renderVisibleMazeArea();  
  
    lastRenderTime = timestamp - (deltaTime % frameInterval);  
}  
  
animFrameId = requestAnimationFrame(gameLoop);  
}  
  
// Démarrer la boucle  
animFrameId = requestAnimationFrame(gameLoop);  
}  
  
function renderVisibleMazeArea() {  
    // Si canvas, utiliser technique de double buffer  
    if (renderMode === 'canvas') {  
        // Créer canvas hors-écran  
        const offscreenCanvas = document.createElement('canvas');  
        offscreenCanvas.width = canvas.width;  
        offscreenCanvas.height = canvas.height;  
        const offCtx = offscreenCanvas.getContext('2d');
```

```

// Dessiner sur le canvas hors-écran
drawMazeToContext(offCtx, visibleCells);

// Copier en une seule opération vers le canvas visible
ctx.clearRect(0, 0, canvas.width, canvas.height);
ctx.drawImage(offscreenCanvas, 0, 0);
}

// Si DOM, utiliser DocumentFragment
else {
    const fragment = document.createDocumentFragment();

    // Mettre à jour seulement les cellules modifiées
    for (const cellKey in cellsToUpdate) {
        const { element, data } = cellsToUpdate[cellKey];
        updateCellElement(element, data);
        fragment.appendChild(element);
    }

    // Ajouter en une seule opération DOM
    mazeContainer.appendChild(fragment);
}

// Optimiser les transitions de niveau
function smoothLevelTransition(fromLevel, toLevel) {

```

```
// Fade out progressif

const overlay = document.getElementById('transitionOverlay');

overlay.style.opacity = '0';

overlay.style.display = 'block';


// Animation progressive

setTimeout(() => {

    overlay.style.opacity = '1';


// Arrêter le rendu actuel

cancelAnimationFrame(animFrameId);


// Nettoyer les ressources

cleanupResources();


// Attendre la fin de la transition

setTimeout(() => {

    // Initialiser nouveau niveau

    initGameLevel(toLevel);


// Fade in

    overlay.style.opacity = '0';

    setTimeout(() => {

        overlay.style.display = 'none';

    }, 500);

}, 500);
```

```
    }, 50);  
}
```

8. TESTS ET DÉBOGAGE

8.1 Utilitaires de Test

// Dans utils.js

```
const debugMode = {  
  enabled: false,  
  showGrid: false,  
  showPath: false,  
  infiniteBattery: false,  
  skipQuestions: false,  
  visualizeGeneration: false  
};  
  
function toggleDebugMode() {  
  debugMode.enabled = !debugMode.enabled;  
  console.log(`Mode debug: ${debugMode.enabled ? 'activé' : 'désactivé'}`);  
  
  // Mettre à jour l'interface si nécessaire  
  const debugPanel = document.getElementById('debugPanel');  
  if (debugPanel) {  
    debugPanel.style.display = debugMode.enabled ? 'block' : 'none';  
  } else if (debugMode.enabled) {  
    createDebugPanel();  
  }  
}
```

```
}
```

```
function createDebugPanel() {  
  
    const panel = document.createElement('div');  
  
    panel.id = 'debugPanel';  
  
    panel.className = 'debug-panel';  
  
  
    // Ajouter options de débogage  
  
    panel.innerHTML = `  
        <h3>Mode Debug</h3>  
  
        <label>  
            <input type="checkbox" id="debug-grid"> Afficher grille  
        </label>  
  
        <label>  
            <input type="checkbox" id="debug-path"> Afficher chemin  
        </label>  
  
        <label>  
            <input type="checkbox" id="debug-battery"> Batterie infinie  
        </label>  
  
        <label>  
            <input type="checkbox" id="debug-questions"> Ignorer questions  
        </label>  
  
        <label>  
            <input type="checkbox" id="debug-generation"> Visualiser génération  
        </label>  
  
        <hr>
```


