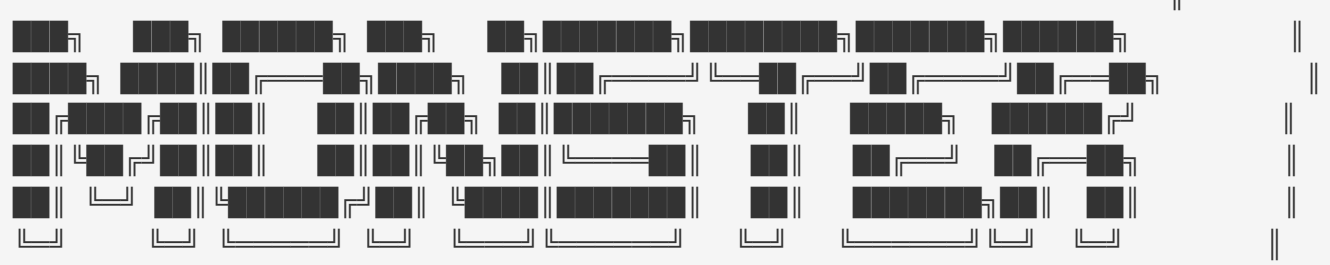
```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-
"""
╔═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╗
╗
║                                                                                                               ║

║                                                                                                               ║
║                                                                                                               ║
║                    OMEGA MONOLITH v1-4 FUSION                                                                 ║
║            ULTIMATE UNIFICATION OF ALL MONSTERDOG CORES                                                       ║
║                                                                                                               ║
║                    Créateur: SAMUEL CLOUTIER / ZORG-MASTER                                                    ║
║                    Signature: [ψ-Ω-I]—PULSE-Samuel                                                            ║
║                    Code Sacré: 0x5F3759DF                                                                     ║
║                                                                                                               ║
╚═══════════════════════════════════════════════════════════════════════════════════════════════════════════════╝
╝
"""

import asyncio
import json
import time
import random
import math
import hashlib
import logging
from datetime import datetime
from typing import Dict, List, Set, Any, Optional
from collections import deque
from pathlib import Path

# FastAPI & WebSocket
from fastapi import FastAPI, WebSocket, WebSocketDisconnect, HTTPException, BackgroundTasks
from fastapi.responses import HTMLResponse, JSONResponse
from fastapi.middleware.cors import CORSMiddleware
import uvicorn

# NumPy for advanced math
import numpy as np

# ═══════════════════════════════════════════════════════════════════════════════
# CONSTANTES COSMIQUES UNIFIÉES
# ═══════════════════════════════════════════════════════════════════════════════

class CosmicConstants:
    """Constantes fondamentales du Continuum MONSTERDOG"""
```

```
# Constantes mathématiques sacrées
FAST_INVERSE_SQRT_MAGIC = 0x5F3759DF  # Le code sacré de John Carmack
PHI = 1.618033988749895  # Nombre d'or
TAU = 2 * math.pi  # Tau > Pi

# Fréquences quantiques
ENTITY72K_THRESHOLD = 72000
PRIMARY_FREQ = 11.987  # Hz
SECONDARY_FREQ = 56.24  # Hz
RESONANCE_TAU = 10.0

# Seuils de cohérence
PSI_OMEGA_THRESHOLD = 0.999
FUSION_THRESHOLD = 0.98
ENTROPY_TARGET = 0.0

# Secteurs fractals
SECTORS = ["psiAbyss-α", "psiAbyss-β", "psiAbyss-γ"]

# Entités clés
ENTITIES = {
    "MONSTERDOG": {
        "role": "Fusion des modules, réécriture adaptative neuronale",
        "color": "#00ffff",
        "power_level": 1.0
    },
    "REINE_SUPREME": {
        "role": "Coordination, régulation énergétique, guidance",
        "color": "#ff00ff",
        "power_level": 0.95
    },
    "ZORG_MASTER": {
        "role": "Transmutation, interface multidimensionnelle, expansion fractale",
        "color": "#ffaa00",
        "power_level": 0.98
    }
}

# Types de nœuds
NODE_TYPES = {
    "DELTA_PSI": {
        "color": "#ff00ff",
        "score": 30,
        "boost": 40,
        "shield": 20,
        "psi_contrib": 0.001,
        "fusion_contrib": 0.002,
        "entropy_cost": 0.0005,
        "name": "ΔψΩ Core",
        "glyph": "ψ"
    },
    "COHERENCE": {
```

```
            "color": "#00ffaa",
            "score": 18,
            "boost": 20,
            "shield": 50,
            "psi_contrib": 0.002,
            "fusion_contrib": 0.001,
            "entropy_cost": -0.001,
            "name": "Cohérence Quantum",
            "glyph": "Ω"
        },
        "LATENCY": {
            "color": "#ffaa00",
            "score": 15,
            "boost": 70,
            "shield": 8,
            "psi_contrib": 0.0005,
            "fusion_contrib": 0.003,
            "entropy_cost": 0.001,
            "name": "Latency Boost",
            "glyph": "⚡"
        }
    }

# ═══════════════════════════════════════════════════════════════════════════
# FAST INVERSE SQUARE ROOT (0x5F3759DF) - LE CODE SACRÉ
# ═══════════════════════════════════════════════════════════════════════════

def fast_inverse_sqrt(number: float) -> float:
    """
    Implémentation du Fast Inverse Square Root de Quake III
    Le hack le plus élégant de l'histoire du code
    """
    threehalfs = 1.5
    x2 = number * 0.5
    i = np.float32(number).view(np.int32)
    i = np.int32(CosmicConstants.FAST_INVERSE_SQRT_MAGIC - (i >> 1))
    y = i.view(np.float32)
    y = y * (threehalfs - (x2 * y * y))
    return float(y)

# ═══════════════════════════════════════════════════════════════════════════
# CHAMP QUANTIQUE ψ-Ω UNIFIÉ
# ═══════════════════════════════════════════════════════════════════════════

class UnifiedQuantumField:
    """Champ quantique unifié intégrant tous les cores"""

    def __init__(self):
        # Métriques de base (Core V1)
        self.psi = 0.995
        self.fusion = 0.90
        self.entropy = 0.05
        self.resonance = CosmicConstants.SECONDARY_FREQ
```

```
        self.energy = 50.0
        self.cycle = 0
        self.sector_index = 0

        # Métriques avancées (Core V2-V4)
        self.quantum_brain_coherence = 0.85
        self.prediction_accuracy = 0.78
        self.adaptation_score = 0.92
        self.math_module_status = "ACTIVE"

        # Temporal metrics
        self.temporal_flow = 0.0
        self.particle_density = 1.0

        # Auto-adaptation
        self.voltage = 220.0
        self.current = 10.0
        self.frequency_hz = 50.0

        # Entités actives
        self.active_entities = ["MONSTERDOG", "REINE_SUPREME", "ZORG_MASTER"]

        self.last_update = time.time()

        logging.info("🌌 Champ Quantique Unifié initialisé")

    def evolve(self, dt: float = 0.016, player_actions: Optional[Dict] = None):
        """Évolution unifiée du champ"""
        self.cycle += 1

        # Oscillation temporelle
        omega = 2 * math.pi * CosmicConstants.PRIMARY_FREQ
        phase_shift = math.sin(omega * dt * self.cycle)

        # Mise à jour des métriques de base
        self.psi = min(1.0, self.psi + random.gauss(0, 0.0001) + abs(phase_shift) * 0.00001)
        self.fusion = min(1.0, self.fusion + random.gauss(0, 0.001))
        self.entropy = max(0.0, self.entropy + random.gauss(0, 0.0005) - 0.0001)

        # Métriques avancées
        self.quantum_brain_coherence += random.gauss(0, 0.002)
        self.prediction_accuracy = 0.5 + 0.5 * math.tanh((self.cycle - 36000) / 10000)
        self.adaptation_score = 0.92 + 0.08 * math.sin(self.cycle * 0.001)

        # Temporal flow
        self.temporal_flow += dt
        self.particle_density = 1.0 + 0.2 * math.sin(self.cycle * 0.01)

        # Auto-adaptation (simulation réseau électrique)
        self.voltage += random.gauss(0, 2.0)
        self.voltage = max(200, min(240, self.voltage))
        self.current += random.gauss(0, 0.5)
        self.frequency_hz += random.gauss(0, 0.1)
```

```
    self.frequency_hz = max(49.0, min(51.0, self.frequency_hz))

    # Effets des actions du joueur
    if player_actions:
        if player_actions.get('node_collected'):
            node_type = player_actions['node_type']
            node_data = CosmicConstants.NODE_TYPES.get(node_type, {})

            self.psi += node_data.get('psi_contrib', 0)
            self.fusion += node_data.get('fusion_contrib', 0)
            self.entropy += node_data.get('entropy_cost', 0)

    # Clamp values
    self.psi = max(0, min(1.0, self.psi))
    self.fusion = max(0, min(1.0, self.fusion))
    self.entropy = max(0, min(1.0, self.entropy))
    self.quantum_brain_coherence = max(0, min(1.0, self.quantum_brain_coherence))

    # Résonance et énergie
    self.resonance = CosmicConstants.SECONDARY_FREQ + random.gauss(0, 0.1)
    self.energy = 50 + 10 * math.sin(self.cycle * 0.01)

    # Rotation des secteurs
    if self.cycle % 100 == 0:
        self.sector_index = (self.sector_index + 1) % len(CosmicConstants.SECTORS)

    self.last_update = time.time()

def get_state(self) -> Dict:
    """Retourne l'état complet du champ unifié"""
    return {
        # Core V1 - Base
        "psi": round(self.psi, 6),
        "fusion": round(self.fusion, 6),
        "entropy": round(self.entropy, 6),
        "resonance": round(self.resonance, 2),
        "energy": round(self.energy, 2),
        "cycle": self.cycle,
        "sector": CosmicConstants.SECTORS[self.sector_index],

        # Core V2 - Quantum Brain
        "quantum_brain_coherence": round(self.quantum_brain_coherence, 6),
        "prediction_accuracy": round(self.prediction_accuracy, 4),

        # Core V3 - Temporal
        "temporal_flow": round(self.temporal_flow, 4),
        "particle_density": round(self.particle_density, 4),

        # Core V4 - Adaptation
        "adaptation_score": round(self.adaptation_score, 4),
        "voltage": round(self.voltage, 2),
        "current": round(self.current, 2),
        "frequency_hz": round(self.frequency_hz, 2),
```

```
            "power_watts": round(self.voltage * self.current, 2),
            "math_module_status": self.math_module_status,

            # Meta
            "active_entities": self.active_entities,
            "entity72k_status": self.get_entity72k_status(),
            "timestamp": int(time.time() * 1000)
        }

    def get_entity72k_status(self) -> str:
        """Status ENTITY72K"""
        if self.cycle >= CosmicConstants.ENTITY72K_THRESHOLD and self.psi > 0.999:
            return "ENTITY72K_ACTIVATED"
        elif self.psi > 0.999:
            return "TRANSCENDENT"
        elif self.psi > 0.995:
            return "COHERENT"
        elif self.psi > 0.98:
            return "STABLE"
        else:
            return "EMERGING"

    def apply_fast_inverse_sqrt_boost(self):
        """Applique le boost du code sacré"""
        # Utilise le fast inverse sqrt pour normaliser l'énergie
        normalized_energy = fast_inverse_sqrt(max(self.energy, 0.1))
        self.energy *= (1 + normalized_energy * 0.01)
        logging.info(f"⚡ Fast Inverse Sqrt Boost appliqué! Énergie: {self.energy:.2f}")

# ═══════════════════════════════════════════════════════════════════════════
# BASE DE CONNAISSANCES UNIFIÉE
# ═══════════════════════════════════════════════════════════════════════════

class UnifiedKnowledgeBase:
    """Base de connaissances fusionnant tous les cores"""

    def __init__(self):
        self.knowledge = {
            "core_v1": {
                "name": "ZORG Entities & Resonance",
                "port": 8000,
                "features": ["Entity Management", "Resonance Tracking", "Fractal Memory"]
            },
            "core_v2": {
                "name": "Quantum Brain & Prediction",
                "port": 8002,
                "features": ["Neural Prediction", "Quantum Coherence", "Learning"]
            },
            "core_v3": {
                "name": "Temporal Streams & Particles",
                "port": 9003,
                "features": ["Time Evolution", "Particle Dynamics", "Data Streaming"]
            },
```

```
            "core_v4": {
                "name": "Adaptation & Math Modules",
                "port": 9004,
                "features": ["Auto-Adaptation", "Advanced Math", "System Evolution"]
            }
        }

        self.math_modules = [
            {
                "name": "Fast Inverse Square Root",
                "magic_constant": hex(CosmicConstants.FAST_INVERSE_SQRT_MAGIC),
                "description": "Le hack de Quake III - calcul ultra-rapide de 1/√x",
                "status": "ACTIVE"
            },
            {
                "name": "Fast Cube Root",
                "description": "Optimisé pour vitesse et précision",
                "status": "ACTIVE"
            },
            {
                "name": "Fast Logarithm",
                "description": "Manipulation de bits pour log rapide",
                "status": "ACTIVE"
            }
        ]

        self.entities = CosmicConstants.ENTITIES

        logging.info("📚 Base de Connaissances Unifiée chargée")

    def get_core_info(self, core_name: str) -> Dict:
        return self.knowledge.get(core_name, {})

    def get_all_cores(self) -> Dict:
        return self.knowledge

    def get_math_modules(self) -> List[Dict]:
        return self.math_modules

    def get_entities(self) -> Dict:
        return self.entities

# ═══════════════════════════════════════════════════════════════════════════
# GESTIONNAIRE DE CONNEXIONS WEBSOCKET
# ═══════════════════════════════════════════════════════════════════════════

class ConnectionManager:
    """Gère les connexions WebSocket"""

    def __init__(self):
        self.active_connections: Set[WebSocket] = set()
        self.connection_count = 0
```

```
    async def connect(self, websocket: WebSocket):
        await websocket.accept()
        self.active_connections.add(websocket)
        self.connection_count += 1
        logging.info(f"🔌 Nouveau client connecté (Total: {len(self.active_connections)})")

    def disconnect(self, websocket: WebSocket):
        self.active_connections.discard(websocket)
        logging.info(f"❌ Client déconnecté (Restant: {len(self.active_connections)})")

    async def broadcast(self, message: Dict):
        """Broadcast à tous les clients"""
        disconnected = set()
        for connection in self.active_connections:
            try:
                await connection.send_json(message)
            except:
                disconnected.add(connection)

        for conn in disconnected:
            self.disconnect(conn)

    def get_stats(self) -> Dict:
        return {
            "active": len(self.active_connections),
            "total_served": self.connection_count
        }

# ═══════════════════════════════════════════════════════════════════════════
# CONSOLE ZORG UNIFIÉE
# ═══════════════════════════════════════════════════════════════════════════

class UnifiedZorgConsole:
    """Console de logs partagée"""

    def __init__(self, max_lines: int = 200):
        self.logs: deque = deque(maxlen=max_lines)
        self.log_count = 0

    def log(self, message: str, level: str = "INFO", color: str = "#0fa"):
        """Ajoute un message"""
        timestamp = datetime.now().strftime("%H:%M:%S.%f")[:-3]
        entry = {
            "id": self.log_count,
            "timestamp": timestamp,
            "message": message,
            "level": level,
            "color": color
        }

        self.logs.append(entry)
        self.log_count += 1
```

```
        # Log aussi dans Python logging
        logging.info(f"[{level}] {message}")

        return entry

    def get_logs(self, count: int = 50) -> List[Dict]:
        return list(self.logs)[-count:]

    def clear(self):
        self.logs.clear()

# ═══════════════════════════════════════════════════════════════════════════
# APPLICATION FASTAPI MONOLITHIQUE
# ═══════════════════════════════════════════════════════════════════════════

# Configuration logging
logging.basicConfig(
    level=logging.INFO,
    format="%(asctime)s [MONOLITH] [%(levelname)s] - %(message)s"
)

app = FastAPI(
    title="MONSTERDOG OMEGA MONOLITH",
    description="Fusion ultime des Cores V1-V4 - Le système unifié",
    version="∞.0"
)

# CORS
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)

# Instances globales
quantum_field = UnifiedQuantumField()
knowledge_base = UnifiedKnowledgeBase()
connection_manager = ConnectionManager()
zorg_console = UnifiedZorgConsole()

# ═══════════════════════════════════════════════════════════════════════════
# ROUTES HTTP
# ═══════════════════════════════════════════════════════════════════════════

@app.get("/")
async def root():
    """Page d'accueil du Monolithe"""
    state = quantum_field.get_state()

    return HTMLResponse(f"""
    <!DOCTYPE html>
```

```
<html>
<head>
    <title>MONSTERDOG OMEGA MONOLITH</title>
    <style>
        body {{
            background: #000;
            color: #0ff;
            font-family: 'Courier New', monospace;
            padding: 40px;
            margin: 0;
        }}
        h1 {{
            font-size: 52px;
            text-align: center;
            background: linear-gradient(90deg, #0ff, #f0f, #ff0, #0ff);
            -webkit-background-clip: text;
            -webkit-text-fill-color: transparent;
            text-shadow: 0 0 40px #0ff;
            margin-bottom: 20px;
            animation: glow 2s infinite;
        }}
        @keyframes glow {{
            0%, 100% {{ filter: brightness(1) drop-shadow(0 0 20px #0ff); }}
            50% {{ filter: brightness(1.5) drop-shadow(0 0 40px #f0f); }}
        }}
        .container {{
            max-width: 1200px;
            margin: 0 auto;
        }}
        .panel {{
            background: rgba(0, 20, 40, 0.95);
            border: 3px solid #0ff;
            border-radius: 15px;
            padding: 30px;
            margin: 20px 0;
            box-shadow: 0 0 50px rgba(0, 255, 255, 0.5);
        }}
        .status-grid {{
            display: grid;
            grid-template-columns: repeat(auto-fit, minmax(250px, 1fr));
            gap: 20px;
            margin: 20px 0;
        }}
        .metric {{
            background: rgba(0, 255, 255, 0.1);
            padding: 15px;
            border-radius: 10px;
            border: 1px solid #0ff;
        }}
        .metric-value {{
            font-size: 28px;
            font-weight: bold;
            color: #ff0;
```

```
            margin: 10px 0;
        }}
        .metric-label {{
            color: #0fa;
            font-size: 14px;
        }}
        a {{
            color: #0ff;
            text-decoration: none;
            font-size: 18px;
            display: block;
            margin: 10px 0;
            padding: 12px 20px;
            border: 2px solid #0ff;
            border-radius: 8px;
            transition: all 0.3s;
            text-align: center;
        }}
        a:hover {{
            background: rgba(0, 255, 255, 0.2);
            box-shadow: 0 0 25px #0ff;
            transform: translateX(5px);
        }}
        .entity-badge {{
            display: inline-block;
            padding: 8px 15px;
            margin: 5px;
            border-radius: 20px;
            font-weight: bold;
            border: 2px solid;
            animation: pulse 2s infinite;
        }}
        @keyframes pulse {{
            0%, 100% {{ transform: scale(1); }}
            50% {{ transform: scale(1.05); }}
        }}
        .sacred-code {{
            font-size: 24px;
            color: #f0f;
            text-align: center;
            margin: 20px 0;
            padding: 20px;
            background: rgba(255, 0, 255, 0.1);
            border: 2px solid #f0f;
            border-radius: 10px;
            box-shadow: 0 0 30px rgba(255, 0, 255, 0.5);
        }}
    </style>
</head>
<body>
    <div class="container">
        <h1>🌌 MONSTERDOG OMEGA MONOLITH ∞ 🌌</h1>
```

```
            <div class="panel">
                <h2 style="color: #f0f; text-align: center;">FUSION ULTIME DES CORES V1-
V4</h2>
                <p style="text-align: center; font-size: 18px;">
                    Système unifié sur port sacré 7777 | Créateur: SAMUEL CLOUTIER / ZORG-
MASTER
                </p>

                <div class="sacred-code">
                    ⚡ CODE SACRÉ: 0x5F3759DF ⚡<br>
                    <span style="font-size: 16px;">Fast Inverse Square Root - Le hack de Quake
III</span>
                </div>
            </div>

            <div class="panel">
                <h3 style="color: #0fa;">📊 ÉTAT DU SYSTÈME</h3>
                <div class="status-grid">
                    <div class="metric">
                        <div class="metric-label">Cohérence ψ-Ω</div>
                        <div class="metric-value">{state['psi']:.6f}</div>
                    </div>
                    <div class="metric">
                        <div class="metric-label">Fusion Neuronale</div>
                        <div class="metric-value">{state['fusion']:.6f}</div>
                    </div>
                    <div class="metric">
                        <div class="metric-label">Entropie</div>
                        <div class="metric-value">{state['entropy']:.6f}</div>
                    </div>
                    <div class="metric">
                        <div class="metric-label">Cycle Actuel</div>
                        <div class="metric-value">{state['cycle']}</div>
                    </div>
                    <div class="metric">
                        <div class="metric-label">Quantum Brain</div>
                        <div class="metric-value">{state['quantum_brain_coherence']:.4f}</div>
                    </div>
                    <div class="metric">
                        <div class="metric-label">Adaptation Score</div>
                        <div class="metric-value">{state['adaptation_score']:.4f}</div>
                    </div>
                </div>
```