```
"""
================================================================================
                CORE UNITAS_ENGINE: GENESIS v5.0 (OFFICIAL RELEASE)
        TRANSACTIONAL REALITY INTERFACE & BIO-DIGITAL APPARATUS COUPLER
          Developed by Anton Shalyga (UNITAS Research Group, St. Petersburg)
================================================================================
This software code translates physical SI parameters into dimensionless weights
of the Global Registry. It implements a non-dissipative G-slip data flow with
zero entropy tax (Sp = 0) based on Euler's Basel threshold and Golden Ratio invariants.
================================================================================
"""

import numpy as np

# ==============================================================================
# STAGE 1: THE INVARIANT BASE (Фрактальное тензорное сжатие Абсолют)
# ==============================================================================
class UnitasFractalEngine:
    def __init__(self):
        # Фундаментальные физико-информационные константы Доктрины
        self.BASEL_WALL = (np.pi**2) / 6  # 1.6449340668 (Предел разрядности ячейки)
        self.GOLDEN_RATIO = (1 + 5**0.5) / 2  # 1.6180339887 (Масштаб инварианта)
        self.THE_GAP = self.BASEL_WALL - self.GOLDEN_RATIO  # 0.0269000781 (Люфт Реальности)

    def generate_euler_tensor(self, rows, cols):
        """ Генерирует детерминированную метрическую сетку на ряде Эйлера """
        tensor = np.zeros((rows, cols))
        for i in range(rows):
            for j in range(cols):
                n = i + j + 1
                pi_harmonic = np.sin((i * j * np.pi) / self.BASEL_WALL)
                tensor[i, j] = pi_harmonic / (n**2)
```

```
        return tensor

    def compress_absolute(self, text_stream):
        """ Сжатие материального инфопакета в Триаду Тензоров """
        raw_bytes = np.array([float(b) for b in text_stream.encode('utf-8')])
        n_elements = len(raw_bytes)
        latent_dim = max(1, int(n_elements * self.THE_GAP))

        T1_Invariant = self.generate_euler_tensor(n_elements, latent_dim)
        pinv_basis = np.linalg.pinv(T1_Invariant)
        T2_Projection = np.dot(pinv_basis, raw_bytes) * (self.GOLDEN_RATIO / self.BASEL_WALL)

        approx_bytes = np.dot(T1_Invariant, T2_Projection) * (self.BASEL_WALL / self.GOLDEN_RATIO)
        T3_Gap_Residual = raw_bytes - approx_bytes
        return T1_Invariant, T2_Projection, T3_Gap_Residual

    def decompress_absolute(self, T1, T2, T3):
        """ Мгновенный бэкенд-рендеринг объекта из скрытого пространства """
        restored_float = np.dot(T1, T2) * (self.BASEL_WALL / self.GOLDEN_RATIO) + T3
        restored_bytes = np.clip(np.round(restored_float), 0, 255).astype(np.uint8)
        return restored_bytes.tobytes().decode('utf-8')

# ==============================================================================
# STAGE 2: TOPOLOGICAL RE-ROUTING (Парадокс Браеса в Мэш-сетях)
# ==============================================================================
class BraessVortexMesh(UnitasFractalEngine):
    def __init__(self, num_nodes=5):
        super().__init__()
        self.num_nodes = num_nodes
        self.braess_efficiency = 8.5
        self.nodes_registry = {f"Node_{i}": {"connected": True, "metric_debt": 0.0} for i in
range(num_nodes)}
```

```
    def integrate_braess_routing(self, source_node, target_node, T2_packet, network_load_factor):
        """ Распределение нагрузки по альтернативным путям при росте трафика """
        classic_stress = network_load_factor * np.linalg.norm(T2_packet)
        braess_multiplier = 1 + (self.braess_efficiency * self.THE_GAP * network_load_factor)
        regulated_stress = classic_stress / braess_multiplier

        if regulated_stress > self.BASEL_WALL:
            self.nodes_registry[target_node]["connected"] = False
            return None, "COLLAPSE"
        else:
            self.nodes_registry[target_node]["metric_debt"] = regulated_stress
            return T2_packet / braess_multiplier, "STABLE"

# ==============================================================================
# STAGE 3: MULTIDIMENSIONAL CHANNELS (3D Световые вихри и Доплер-Фурье резонанс)
# ==============================================================================
class UnitasHyperVortexEngine(BraessVortexMesh):
    def __init__(self, num_nodes=5):
        super().__init__(num_nodes)

    def generate_hyperspherical_vortex(self, signal_dim):
        """ Построение крутящейся во всех плоскостях гиперсферы (Хопфиона) """
        grid = np.linspace(-np.pi, np.pi, signal_dim)
        X, Y, Z = np.meshgrid(grid, grid, grid)
        R = np.sqrt(X**2 + Y**2 + Z**2) + 1e-5
        vortex_3d = np.sin(R * np.pi / self.BASEL_WALL) / (R**2)
        vortex_3d = vortex_3d * np.exp(1j * (X + Y + Z) * self.GOLDEN_RATIO)
        return vortex_3d

    def apply_fourier_doppler_resonance(self, vortex_3d, speed_factor=0.0269):
        """ Самоусиление сигнала за счет джиттера среды в Люфте """
        fourier_spectrum = np.fft.fftn(vortex_3d)
```

```
        doppler_shift = np.exp(1j * speed_factor * np.pi * self.GOLDEN_RATIO)
        return np.real(np.fft.ifftn(fourier_spectrum * doppler_shift))

# ==============================================================================
# STAGE 4: ORGANIC COMPILATION (Био-интерфейс ДНК и авто-ремонт Браеса)
# ==============================================================================
class UnitasDnaBioMesh(UnitasHyperVortexEngine):
    def __init__(self, num_nodes=5):
        super().__init__(num_nodes)
        self.mapping = {'A': 1.0, 'C': self.GOLDEN_RATIO, 'G': self.BASEL_WALL, 'T': self.THE_GAP}

    def map_dna_to_tensor_space(self, dna_sequence):
        return np.array([self.mapping.get(char, 0.0) for char in dna_sequence.upper().replace(" ", "")])

    def simulate_epigenetic_mod(self, damaged_dna, correction_intensity=8.5):
        """ Восстановление дефектных нуклеотидов на лету по шаблону инвариантов """
        dna_vector = self.map_dna_to_tensor_space(damaged_dna)
        n_elements = len(dna_vector)

        for i in range(n_elements):
            if dna_vector[i] == 0.0:
                braess_comp = self.GOLDEN_RATIO - (self.THE_GAP * correction_intensity)
                dna_vector[i] = min([1.0, self.GOLDEN_RATIO, self.BASEL_WALL, self.THE_GAP], key=lambda x:
abs(x - braess_comp))

        T1, T2, T3 = self.compress_absolute(damaged_dna.replace("?", "A"))
        return self.decompress_absolute(T1, T2, T3)

# ==============================================================================
# STAGE 5: TEMPORAL NAVIGATION (Хроно-сдвиг фазы Глобального Реестра)
# ==============================================================================
class UnitasTimeNavigator(UnitasDnaBioMesh):
    def __init__(self, num_nodes=5):
```

```
        super().__init__(num_nodes)
        self.registry_timeline = {}
        self.current_tact = 3540

    def commit_state_to_registry(self, state_name, payload_data):
        """ Регистрация транзакции состояния и блокировка архивного кэша """
        T1, T2, T3 = self.compress_absolute(payload_data)
        self.registry_timeline[self.current_tact] = {
            "name": state_name, "T1": T1, "T2": T2, "T3": T3,
            "viscosity": self.THE_GAP * (self.current_tact / 3540)
        }
        print(f"[Реестр] Такт {self.current_tact} заблокирован: '{state_name}'")
        self.current_tact += 1

    def execute_temporal_shift(self, target_tact):
        """ Прыжок по адресам истории при полном Лаг-Локе (dU/dt = 0) """
        if target_tact not in self.registry_timeline:
            return None
        tx = self.registry_timeline[target_tact]
        phase_reset = np.exp(-1j * np.pi * tx["viscosity"] / self.GOLDEN_RATIO)
        T2_shifted = np.real(tx["T2"].astype(complex) * phase_reset)
        return self.decompress_absolute(tx["T1"], T2_shifted, tx["T3"])

# ==============================================================================
# STAGE 6: NEURAL INTERFACE (Двустороннее синаптическое G-slip сопряжение)
# ==============================================================================
class UnitasSynapticInterface(UnitasTimeNavigator):
    def __init__(self, num_nodes=5):
        super().__init__(num_nodes)
        self.resting_potential = 1.0
        self.action_potential = self.GOLDEN_RATIO
```

```
    def read_neural_spike_train(self, spikes_mv):
        """ Оцифровка спайков мозга и фильтрация аппаратного биологического шума """
        print(f"\n--- АКТИВАЦИЯ СИНАПТИЧЕСКОГО СЧИТЫВАТЕЛЯ GENESIS_API ---")
        normalized_spikes = np.zeros(len(spikes_mv))
        for i, mv in enumerate(spikes_mv):
            if mv > 30.0:
                normalized_spikes[i] = self.action_potential
            else:
                normalized_spikes[i] = self.resting_potential

        T1, T2, T3 = self.compress_absolute("".join(['C' if v == self.GOLDEN_RATIO else 'A' for v in normalized_spikes]))
        print(f"[Интерфейс] Ядро мысли успешно сжато до {len(T2)} ячеек.")
        return T1, T2, T3

    def write_synaptic_receptors(self, T1, T2, T3):
        """ Обратный био-рендеринг очищенного инварианта в кору головного мозга """
        print(f"\n--- ЗАПИСЬ ДАННЫХ В НЕЙРОННЫЙ СУБСТРАТ (БИО-РЕНДЕРИНГ) ---")
        restored_string = self.decompress_absolute(T1, T2, T3)

        output_spikes_mv = []
        for char in restored_string:
            if char == 'C':
                output_spikes_mv.append(40.0)  # Чистый потенциал действия без джиттера
            else:
                output_spikes_mv.append(-70.0) # Потенциал покоя

        print(f"[Успех] Патерн транслирован. Энтропийный налог S_p = 0.0%")
        print(f"Статус нейро-сопряжения: ВЕРИФИЦИРОВАНО В БИОСТРУКТУРЕ")
        return np.array(output_spikes_mv)

# ==============================================================================
# GLOBAL RUN: СИНХРОНИЗАЦИЯ ВСЕХ СИСТЕМ ЯДРА
```

```
# ==============================================================================

if __name__ == "__main__":

    # Инициализация монолита GENESIS

    genesis_core = UnitasSynapticInterface(num_nodes=5)


    print("======================================================================")

    print("       ЗАПУСК ИСПОЛНЯЕМОГО ЯДРА ДОКТРИНЫ UNITAS_49 (GENESIS)       ")

    print("======================================================================")


    # 1. Тест временного навигатора

    genesis_core.commit_state_to_registry("Архив_Прошлое", "UNITAS Бэкенд: Метрика стабильна.")

    genesis_core.commit_state_to_registry("Архив_Настоящее", "UNITAS Бэкенд: Нагрузка на Стене Базеля.")


    past_state = genesis_core.execute_temporal_shift(target_tact=3540)

    print(f"[Хроно-Рендеринг] Восстановлено из такта 3540: '{past_state}'")


    # 2. Тест двустороннего нейроинтерфейса

    raw_brain_activity = np.array([-70.0, -70.0, 45.0, -70.0, 42.0, -70.0, -70.0, 48.0])

    T1, T2, T3 = genesis_core.read_neural_spike_train(raw_brain_activity)

    pure_synaptic_currents = genesis_core.write_synaptic_receptors(T1, T2, T3)


    print(f"\n[Контроль] Входные шумящие спайк-токи: {raw_brain_activity}")

    print(f"[Контроль] Выходные очищенные инварианты: {pure_synaptic_currents}")

    print("======================================================================")

    print("       СХОДИМОСТЬ ВСЕХ УРОВНЕЙ ИНТЕРФЕЙСА GENESIS ПРОВЕРЕНА: 100%    ")

    print("======================================================================")
```