ERES PlayNAC "KERNEL" Codebase (v6.0 Draft)

🚀 Project Overview

ERES PlayNAC (New Age Cybernetic Game Theory) is a comprehensive, modular framework uniting empirical simulations, decentralized consensus, and immersive voice-driven interfaces to advance human–machine collaboration. The **KERNEL** at its heart orchestrates:

- **EarnedPath (EP):** Lifelong learning & merit-based progression.
- **GiantERP (GERP):** Global Earth Resource Planning & optimization.
- **Bio-Energetic Economy (BEE):** Ecological health scoring & proof-of-work.
- **BERC:** Bio-Electric Ratings Codex & decentralized consensus.
- **NBERS:** Neural Blockchain Economic Reasoning System for AI-driven forecasts.
- **GCF:** Gracechain with Meritcoin for value exchange.
- **CARE:** Core property management & Protect-Enrich scoring.
- **GEO:** Goal-Oriented Design with geographic anchoring & NPR remediation.
- **SOMT:** Solid-State Sustainability snapshots via GEAR.
- **PlayNAC Game Engine:** Real-world simulation scenarios & collective reasoning.
- **Voice Navigation (VERTECA):** Hands-free 4D VR commands & NLP routing.

Version **6.0** synchronizes all modules into a unified codebase, ensures parity with documented drafts (v2–v5), and extends the architecture to include CARE, GEO, SOMT, NBERS, and GCF as first-class packages.

📦 Repository Structure

```
├── docs/                # Design docs, architecture diagrams, whitepapers
│   ├── architecture/    # UML, sequence, data-flow schematics
│   └── api/             # Auto-generated Sphinx HTML
├── src/                 # Core Python packages
│   ├── kernel/          # Orchestration & config management
│   ├── earnedpath/      # Simulation engine + merit calculus
│   ├── gianterp/        # GiantERP API client & models
│   ├── bee/             # BioPoW algorithms & aura tools
│   ├── berc/            # Consensus protocol & codex logic
│   ├── nbers/           # Neural-economic modeling & forecasts
│   ├── gcf/             # Gracechain & Meritcoin interfaces
│   ├── care/            # Protect-Enrich property management
│   ├── geo/             # Goal-Oriented Design & NPR remediation
│   ├── somt/            # Sustainability snapshots & GEAR integration
│   ├── media/           # RT Media ingestion & adaptive transforms
│   ├── nav/             # Voice/NLP interface & VERTECA router
│   └── utils/           # Exceptions, logging, helpers, retry/cache
├── examples/            # Sample scripts, Jupyter notebooks
├── tests/               # pytest unit & integration tests
├── .github/             # CI/CD workflows (lint, mypy, tests)
├── Dockerfile           # Production-ready container build
├── requirements.txt     # Pinned Python dependencies
├── pyproject.toml       # Build metadata & entry points
```

├── CHANGELOG.md # Version history & highlights

└── README.md # This file

⚙️ Installation & Quick Start

Clone & venv
git clone https://github.com/ERES-Institute-for-New-Age-Cybernetics/PlayNAC-KERNEL.git

cd PlayNAC-KERNEL

1. python3 -m venv venv && source venv/bin/activate

Install
pip install --upgrade pip

2. pip install -r requirements.txt

Env vars
export WEB3_RPC_URL="https://gracechain-node.example.com"

3. export BEE_SECRET_KEY="<your-secret>"
4. **Run tests**
   pytest --maxfail=1 --disable-warnings -q
5. **Demo**
   python -m examples/demo_kernel.py

🏗️ Architecture & Modules

classDiagram

%% Core Kernel

PlayNACKernel <|-- ConfigManager : configures

PlayNACKernel --> BioPoW : generates EP

PlayNACKernel --> SimulationEngine : runs scenarios

PlayNACKernel --> JASConsensus : links tasks

```
%% EarnedPath Module

EPNode <-- MeritCalculator : calculates

SimulationEngine --> EPNode : updates state


%% Governance Modules

PlayNACKernel --> GiantERPClient : fetches grids

PlayNACKernel --> CAREManager : computes PE

PlayNACKernel --> GODRouter : routes geo

PlayNACKernel --> StateRecorder : snapshots sustainability


%% Blockchain Modules

PlayNACKernel --> GracechainClient : distributes Meritcoin

PlayNACKernel --> JASConsensus : issues credits

ForecastEngine <|-- EconomicModel : predicts


%% Media & Navigation

PlayNACKernel --> MediaProcessor : processes media

PlayNACKernel --> ASRClient : listens

DialogueManager --> IntentParser : parses

MandalaTranslator <-- DialogueManager : maps gestures
```

Below is an in-depth breakdown of the **Mandala-VERTECA Symbolic Layer**, which maps biometric gestures and concentric mandala zones to system commands, enabling intuitive, hands-free control.

🔮 Mandala-VERTECA Symbolic Mapping (src/nav/mandala.py)

- **Purpose:** Translates multi-modal biometric inputs (gestures, auras, zones) into discrete commands within the 4D VR environment.
- **Core Concepts & Symbols:**
    1. **Central Zone (Self):** Thumb-to-palm gesture represents system “Home” (root context).
        - Symbol: 🜁 (Alchemical symbol for Air)
        - Command: `home()`
    2. **Inner Ring (Family):** Index-finger Tibetan mudra denotes “Back” (previous context).
        - Symbol: 🜂 (Alchemical symbol for Fire)
        - Command: `navigate_back()`
    3. **Middle Ring (Community):** Middle-finger chakra press maps to “Select” (confirm).
        - Symbol: 🜄 (Alchemical symbol for Water)
        - Command: `select_item()`
    4. **Outer Ring (Nation):** Ring-finger aura swirl triggers “Menu” (context options).
        - Symbol: 🜃 (Alchemical symbol for Earth)
        - Command: `open_menu()`
    5. **Periphery (Universe):** Pinky-wave pattern invokes “Voice Input” (activate ASR).
        - Symbol: ☿ (Mercury)
        - Command: `start_voice_control()`
- **Implementation Snippet:**

```
# src/nav/mandala.py

from typing import Tuple



class MandalaTranslator:

    """

    Maps biometric gesture data to PlayNAC commands via Mandala-VERTECA symbols.

    """

    SYMBOL_MAP = {

        'thumb_palm': ('🜁', 'home'),

        'index_mudra': ('🜂', 'back'),
```

```
    'middle_press': ('∀', 'select'),
    'ring_swirl': ('∇', 'menu'),
    'pinky_wave': ('☿', 'voice'),
}

def translate(self, gesture: str) -> Tuple[str, str]:
    """
    Returns (symbol, command) for a given gesture identifier.
    """
    symbol, cmd = self.SYMBOL_MAP.get(gesture, ('', ''))
    return symbol, cmd

def execute(self, symbol: str, kernel):
    """
    Executes the mapped command on the kernel instance.
    """
    cmd_map = {
        'home': kernel.home,
        'back': kernel.back,
        'select': kernel.select,
        'menu': kernel.menu,
        'voice': kernel.start_voice_control,
    }
    if symbol and cmd_map.get(symbol[1]):
```

```
    cmd_map[symbol[1]]()
```

This layer ensures that five biometric gestures—aligned with concentric Mandala zones—are symbolically mapped to core navigation and control functions, blending intuitive metaphors (Air, Fire, Water, Earth, Mercury) with cybernetic commands.

🖐 5-Finger to QWERTY Command Mapping

To facilitate direct integration with standard keyboards, each Mandala-VERTECA gesture also maps to a QWERTY key, enabling hybrid control between voice, gesture, and keyboard for Next-Generation RT Media workflows:

Finger/Gesture	Symbol	Command	QWERTY Key	Function Description
Thumb-palm	🜂	`home()`	`H`	Return to home context / reset media view
Index-mudra	🜁	`navigate_back()`	`B`	Step back in navigation or timeline
Middle-press	🜄	`select_item()`	`S`	Confirm selection or start playback
Ring-swirl	🜃	`open_menu()`	`M`	Open contextual menu or settings overlay
Pinky-wave	☿	`start_voice_control()`	`V`	Activate voice input for command dictation

By unifying biometric gestures with familiar keyboard shortcuts, developers and power users can seamlessly transition between input modalities when building Next-Generation RT Media applications within the PlayNAC ecosystem.

📢 Green Box Environment (Hands-Free Voice Navigation)

The **Green Box** is a **simulation environment** within the Mandala-VERTECA framework, designed for Hands-Free Voice Navigation (HFVN). It leverages the core PlayNAC subsystems—**EarnedPath**, **GiantERP (GERP)**, and **Vacationomics**—to render dynamic real-time simulations. While in HFVN mode, user gestures and voice commands seamlessly interact with ongoing simulations of learning pathways, planetary resource planning, and vacationomics scenarios.

Key Characteristics

- **Visual Overlay**: Semi-transparent “green box” border around the viewport indicates HFVN mode is active.
- **Spatial Audio Cues**: 3D audio sources aligned with mandala zones guide user attention.
- **Context Layers**: Five concentric zones (Self → Family → Community → Nation → Universe) rendered as colored rings with interactive hotspots.
- **State Indicators**: Dynamic icons at each ring edge reflect current command mappings and system status (e.g., listening, processing, error).

Core Classes (src/nav/hfvn.py)

```
class GreenBoxEnvironment:
    """
    Manages VR/AR rendering and state for Hands-Free Voice Navigation "green box" mode.
    """
    def __init__(self, renderer, audio_engine, mandala_translator):
        self.renderer = renderer          # 3D/AR rendering engine instance
        self.audio = audio_engine         # Spatial audio manager
        self.translator = mandala_translator
        self.active = False
```

```
    def activate(self):
        """Enable green box visuals and audio cues"""
        self.active = True
        self.renderer.show_border(color='green')
        self.audio.play_loop('hfvn_background')

    def deactivate(self):
        """Disable HFVN mode"""
        self.active = False
        self.renderer.hide_border()
        self.audio.stop('hfvn_background')

    def on_gesture(self, gesture_data):
        """Handle biometric gesture within HFVN context"""
        symbol, cmd = self.translator.translate(gesture_data)
        self.renderer.highlight_zone(symbol)
        return cmd

    def on_voice(self, text_command, kernel):
        """Process recognized speech as PlayNAC commands"""
        intent, params = kernel.nav.intent_parser.parse(text_command)
        self.renderer.flash_zone(intent)
        return kernel.nav.dialogue_manager.handle(intent, params)
```

Usage Example

```
from nav.hfvn import GreenBoxEnvironment


env = GreenBoxEnvironment(renderer, audio_engine, MandalaTranslator())
env.activate()


# In main loop:
if gesture_detected:
    cmd = env.on_gesture(gesture_id)
    kernel.execute(cmd)


if speech_text:
    response = env.on_voice(speech_text, kernel)
    print(response)


# To exit HFVN:
env.deactivate()
```

This integration ensures that when the **Green Box** is active, all user inputs—voice or gesture—are contextually mapped via Mandala-VERTECA, with immersive visual and audio feedback guiding seamless, hands-free operation.