```
"""
Harmony Family AI - Memory System Integration
This integrates MemoRAG for long-term family memory and context
"""

from memorag import MemoRAGLite
import json
from datetime import datetime
from typing import Dict, List, Optional
import asyncio

class HarmonyMemorySystem:
    def __init__(
        self,
        family_id: str,
        gen_model: str = "Qwen/Qwen2.5-1.5B-Instruct",
        ret_model: str = "BAAI/bge-m3",
        cache_dir: Optional[str] = "./cache"
    ):
        """
        Initialize Harmony's memory system for a family

        Args:
            family_id: Unique identifier for the family
            gen_model: Generation model for responses
            ret_model: Retrieval model for searching memories
            cache_dir: Directory to cache models
        """
        self.family_id = family_id
        self.cache_dir = cache_dir

        # Initialize MemoRAG
        self.memorag = MemoRAGLite(
            gen_model_name_or_path=gen_model,
            ret_model_name_or_path=ret_model,
            cache_dir=cache_dir,
            ret_hit=5,  # Return top 5 relevant chunks
            retrieval_chunk_size=512,
            load_in_4bit=True,  # Use 4-bit quantization to save memory
            enable_flash_attn=True
        )

        self.conversation_history = []
        self.family_context = ""

    def add_family_context(self, context: str):
        """
        Add or update family context (profiles, important info, etc.)
```

```
    This is the long-term memory that Harmony remembers

    Args:
        context: Text containing family information
    """
    self.family_context += f"\n\n{context}"

    # Memorize the context
    print("Building family memory...")
    self.memorag.memorize(
        context=self.family_context,
        save_dir=f"./family_memories/{self.family_id}",
        print_stats=True
    )
    print("Family memory updated!")

def load_family_memory(self):
    """Load existing family memory from disk"""
    try:
        self.memorag.load(f"./family_memories/{self.family_id}")
        print(f"Loaded existing memory for family {self.family_id}")
    except Exception as e:
        print(f"No existing memory found: {e}")

def chat(
    self,
    user_message: str,
    user_name: str = "User",
    max_tokens: int = 256
) -> str:
    """
    Chat with Harmony using the family memory

    Args:
        user_message: The message from the user
        user_name: Name of the family member
        max_tokens: Maximum tokens to generate

    Returns:
        Harmony's response
    """
    # Add to conversation history
    self.conversation_history.append({
        "role": "user",
        "name": user_name,
        "content": user_message,
        "timestamp": datetime.now().isoformat()
    })

    # Use MemoRAG to generate response with memory
```

```
        response = self.memorag(
            query=user_message,
            task_type="memorag",
            max_new_tokens=max_tokens,
            use_memory_answer=True  # Use memory to help answer
        )

        # Add response to history
        self.conversation_history.append({
            "role": "assistant",
            "content": response,
            "timestamp": datetime.now().isoformat()
        })

        return response

    def answer_about_family(self, question: str) -> str:
        """
        Answer questions about the family using memory

        Args:
            question: Question about the family

        Returns:
            Answer based on family memory
        """
        return self.memorag(
            query=question,
            task_type="qa",  # Direct question answering
            max_new_tokens=200
        )

    def recall_family_moments(self, query: str) -> str:
        """
        Recall specific family moments or information

        Args:
            query: What to recall

        Returns:
            Recalled information
        """
        # Use recall to get relevant text spans
        recalled_spans = self.memorag.recall(query, max_new_tokens=256)
        return recalled_spans

    def generate_family_insight(self, topic: str) -> str:
        """
        Generate insights about the family based on stored memory
```

```
        Args:
            topic: Topic to generate insights about

        Returns:
            Generated insights
        """
        return self.memorag(
            query=f"Generate insights about: {topic}",
            task_type="memorag",
            max_new_tokens=512,
            use_memory_answer=True
        )

    def summarize_family_story(self) -> str:
        """
        Summarize the family's story from memory

        Returns:
            Summary of family information
        """
        return self.memorag.summarize(max_new_tokens=512)

    def save_conversation(self, filepath: str):
        """Save conversation history to file"""
        with open(filepath, 'w') as f:
            json.dump(self.conversation_history, f, indent=2)

    def load_conversation(self, filepath: str):
        """Load conversation history from file"""
        with open(filepath, 'r') as f:
            self.conversation_history = json.load(f)


# Example usage for Harmony
class HarmonyFamilyAssistant:
    def __init__(self, family_id: str):
        self.family_id = family_id
        self.memory_system = HarmonyMemorySystem(family_id)

    async def welcome_new_member(
        self,
        new_member_name: str,
        existing_member_name: str,
        family_info: Dict
    ) -> str:
        """
        Generate a personalized welcome message for new family member

        Args:
            new_member_name: Name of the new member joining
```

```
            existing_member_name: Name of existing member who invited them
            family_info: Dictionary with family information

        Returns:
            Personalized welcome message
        """
        # Build context about the family
        context = f"""
Family Name: {family_info.get('family_name', 'Adams Family')}
Existing Members: {', '.join(family_info.get('members', []))}
New Member: {new_member_name}
Invited By: {existing_member_name}
Family Values: {', '.join(family_info.get('values', []))}
Upcoming Events: {', '.join(family_info.get('upcoming_events', []))}
Family Traditions: {family_info.get('traditions', 'Creating new traditions together')}
"""

        # Add this to family memory
        self.memory_system.add_family_context(context)

        # Generate personalized welcome
        welcome_prompt = f"""
        Create a warm, personalized welcome message for {new_member_name}
        who is joining the {family_info.get('family_name')} through {existing_member_name}.
        Make it feel genuine and mention specific family details.
        """

        welcome_message = self.memory_system.chat(
            welcome_prompt,
            user_name="System",
            max_tokens=300
        )

        return welcome_message

    async def generate_family_insights(self) -> Dict[str, str]:
        """
        Generate insights about the family based on all stored data

        Returns:
            Dictionary of insights by category
        """
        insights = {}

        # Generate different types of insights
        categories = [
            "family strengths and values",
            "shared interests and hobbies",
            "upcoming opportunities for bonding",
            "family growth and changes"
```

```
        ]

        for category in categories:
            insight = self.memory_system.generate_family_insight(category)
            insights[category] = insight

        return insights

    def chat_with_harmony(self, message: str, member_name: str) -> str:
        """
        Chat with Harmony AI

        Args:
            message: Message from family member
            member_name: Name of the member

        Returns:
            Harmony's response
        """
        return self.memory_system.chat(message, member_name)


# Usage Example
if __name__ == "__main__":
    # Initialize Harmony for the Adams family
    harmony = HarmonyFamilyAssistant("adams_family_001")

    # Add initial family context
    family_data = {
        "family_name": "Adams Family",
        "members": ["Jeramy", "Ella", "Addy"],
        "values": ["faith", "leadership", "creativity", "stability"],
        "upcoming_events": ["Baby #1 arriving 2024", "Baby #2 planned 2026"],
        "traditions": "Creative art nights, family painting sessions",
        "hobbies": {
            "Addy": "pink and purple art projects",
            "Mom": "painting and makeup",
            "Ella": "vital part of family, bringing love and care"
        }
    }

    # Generate welcome message for Ella
    welcome = asyncio.run(harmony.welcome_new_member(
        "Ella",
        "Jeramy",
        family_data
    ))
    print("Welcome Message:")
    print(welcome)
```

```
# Chat with Harmony
response = harmony.chat_with_harmony(
    "What are some activities we could do as a family?",
    "Jeramy"
)
print("\nHarmony's Response:")
print(response)

# Generate family insights
insights = asyncio.run(harmony.generate_family_insights())
print("\nFamily Insights:")
for category, insight in insights.items():
    print(f"\n{category.title()}:")
    print(insight)
```