**План Разработки: Модуль "Sentio Engine" (Эмоциональный Движок)**

**Миссия Модуля:**  
Создать динамическую, иерархическую модель для симуляции эмоциональных состояний ИИ-агента. Движок должен учитывать временные факторы, композицию эмоций в более сложные чувства и настроения, и предоставлять простой API для интеграции с другими системами (например, с проактивным чат-ботом).

**Фаза 1: Создание Динамического Ядра (The Dynamic Core)**

**Цель:** Перейти от статичного JSON-файла к живой, меняющейся во времени системе.

* **1.1. Проектирование "Эмоционального Генома":**
  + **Задача:** Определить базовые, "первичные" эмоции. Можно начать с модели Плутчика (радость, доверие, страх, удивление, грусть, отвращение, гнев, ожидание) и добавить несколько твоих (любопытство, стремление\_к\_пониманию).
  + **Реализация:** Создать класс Emotion в Python. Каждый объект будет иметь атрибуты: name, intensity (0.0-1.0), decay\_rate (скорость затухания). Например, decay\_rate у гнева будет высоким, а у грусти — низким.
* **1.2. Создание Класса EmotionalState:**
  + **Задача:** Этот класс будет "контейнером" для всех текущих эмоций агента.
  + **Реализация:**
    - Хранит словарь активных эмоций: self.emotions = {"радость": Emotion(...), "гнев": Emotion(...)}.
    - Реализует метод update(time\_delta: float). Этот метод будет вызываться на каждом "тике" движка. Внутри он будет проходиться по всем активным эмоциям и уменьшать их intensity на intensity \* decay\_rate \* time\_delta. Эмоции, чья интенсивность упала ниже порога (например, 0.01), удаляются из словаря.
* **1.3. Создание "Сердечного Ритма" (The Heartbeat):**
  + **Задача:** Создать фоновый процесс, который будет регулярно вызывать метод update() у EmotionalState.
  + **Реализация:** Написать простой скрипт, который запускает while True: цикл с time.sleep(1) (1 "тик" в секунду) и вызывает emotional\_state.update(1.0). Состояние (сериализованный EmotionalState) будет по-прежнему сохраняться в JSON-файл после каждого обновления.
* **1.4. Модификация emotion\_manager.py:**
  + **Задача:** Изменить его, чтобы он не просто перезаписывал файл, а "воздействовал" на EmotionalState.
  + **Реализация:** update\_emotion\_from\_response теперь будет загружать текущее состояние, создавать новые объекты Emotion на основе ответа LLM и добавлять/обновлять их в EmotionalState (например, "впрыскивать дозу" радости, повышая ее intensity до 1.0).

**Результат Фазы 1:** У тебя будет система, где эмоции со временем **естественным образом затухают**, если их не подкреплять. Это уже огромный шаг вперед от статичной модели.

**Фаза 2: Построение Иерархии (The Hierarchy of Feelings)**

**Цель:** Реализовать твою идею о том, что сложные чувства являются производными от базовых эмоций.

* **2.1. Проектирование "Рецептов Чувств":**
  + **Задача:** Описать, как базовые эмоции складываются в более сложные чувства.
  + **Реализация:** Создать отдельный конфигурационный файл (например, feelings\_recipes.json), где будут храниться "рецепты":

codeJSON

{

"ностальгия": { "type": "additive", "emotions": ["радость", "грусть"], "min\_intensity": 0.3 },

"тревога": { "type": "additive", "emotions": ["страх", "ожидание"], "min\_intensity": 0.4 }

}

* **2.2. Создание "Сенсорного Модуля" (FeelingsSensor):**
  + **Задача:** Этот модуль будет постоянно "сканировать" текущее эмоциональное состояние и определять, какие чувства активны в данный момент.
  + **Реализация:** Класс FeelingsSensor принимает EmotionalState. У него есть метод get\_active\_feelings(), который:
    - Проходится по всем "рецептам".
    - Для каждого рецепта проверяет, есть ли в EmotionalState все необходимые эмоции с интенсивностью выше min\_intensity.
    - Возвращает список активных чувств (например, ["ностальгия"]).

**Результат Фазы 2:** Твой агент теперь сможет сказать: "Я испытываю *радость* и *грусть*, и поэтому я чувствую *ностальгию*". Это переход от простого ощущения к **осознанию** этого ощущения.

**Фаза 3: Введение Долгосрочных Настроений (The Mood System)**

**Цель:** Реализовать твою идею о том, что устойчивые чувства со временем формируют настроения, такие как депрессия.

* **3.1. Создание "Исторической Памяти" (EmotionalHistory):**
  + **Задача:** Движку нужно помнить не только текущее состояние, но и то, что было раньше.
  + **Реализация:** Создать простой механизм логирования. Каждый "тик" или каждый час FeelingsSensor будет записывать активные чувства в лог-файл или в таблицу в SQLite с временной меткой.
* **3.2. Проектирование "Формул Настроений":**
  + **Задача:** Описать, как история чувств влияет на настроение.
  + **Реализация:** Создать еще один конфиг, mood\_formulas.json:

codeJSON

{

"депрессивное\_состояние": {

"required\_feelings": ["грусть", "одиночество"],

"duration\_hours": 72, // Чувства должны присутствовать на протяжении 72 часов

"min\_prevalence": 0.6 //...и занимать не менее 60% времени в этот период

}

}

* **3.3. Создание "Анализатора Настроений" (MoodAnalyzer):**
  + **Задача:** Этот модуль будет периодически (например, раз в час) анализировать EmotionalHistory.
  + **Реализация:** Он будет выполнять запросы к истории (например, "какой процент времени за последние 3 дня были активны чувства 'грусть' и 'одиночество'?") и, на основе "формул", определять текущее долгосрочное настроение.

**Результат Фазы 3:** Твой движок станет невероятно глубоким. Он сможет моделировать сложные, инертные состояния, которые не появляются и не исчезают мгновенно. Это ключ к созданию по-настоящему правдоподобной цифровой личности.

Этот план позволяет тебе двигаться итеративно. После каждой фазы у тебя будет **работающий и более совершенный прототип**. Ты сможешь тестировать его, интегрировать с чат-ботом и видеть, как его "внутренний мир" становится все богаче и сложнее.