Переходим от теории к практике. Создадим **"Эмоциональный Калькулятор"** — набор классов в Python, который реализует нашу "Таблицу Экспрессии" и временную динамику.

Это будет ядро вашего "Sentio Engine". Мы напишем код, который можно будет "потрогать" и протестировать.

**Концепция: Система с "Эмоциональной Физикой"**

Мы создадим мир со своими законами физики, где:

* **Эмоции** — это частицы с массой (приоритетом) и энергией (интенсивностью).
* **Нейромедиаторы** — это поле, которое меняет свойства пространства.
* **Время** — это реальный фактор, который заставляет энергию рассеиваться (охлаждение).

**Файл 1: engine/emotion\_definitions.py (Конфигурация)**

Здесь мы описываем "атомы" нашей вселенной.

codePython

# engine/emotion\_definitions.py

# Описываем "геном" каждой базовой эмоции

EMOTION\_GENOME = {

"гнев": {

"priority": 10,

"base\_decay\_rate": 0.25, # Быстро вспыхивает, быстро гаснет

"neuromediators\_impact": {"norepinephrine": 0.5, "dopamine": 0.2}

},

"страх": {

"priority": 9,

"base\_decay\_rate": 0.15, # Долго держится

"neuromediators\_impact": {"norepinephrine": 0.6, "serotonin": -0.4}

},

"радость": {

"priority": 8,

"base\_decay\_rate": 0.20,

"neuromediators\_impact": {"dopamine": 0.6, "serotonin": 0.3, "oxytocin": 0.1}

},

"грусть": {

"priority": 7,

"base\_decay\_rate": 0.10, # Очень медленно затухает

"neuromediators\_impact": {"dopamine": -0.5, "serotonin": -0.5}

},

"удивление": {

"priority": 6,

"base\_decay\_rate": 0.80, # Самая быстрая, почти мгновенная

"neuromediators\_impact": {"norepinephrine": 0.3}

},

"доверие": {

"priority": 5,

"base\_decay\_rate": 0.12,

"neuromediators\_impact": {"oxytocin": 0.7, "serotonin": 0.2}

},

"любопытство": {

"priority": 4,

"base\_decay\_rate": 0.18,

"neuromediators\_impact": {"dopamine": 0.4}

}

}

# Описываем "геном" нейромедиаторов

NEUROMEDIATOR\_GENOME = {

"dopamine": {"base\_level": 0.5, "decay\_rate": 0.2},

"serotonin": {"base\_level": 0.5, "decay\_rate": 0.15},

"norepinephrine": {"base\_level": 0.2, "decay\_rate": 0.3},

"oxytocin": {"base\_level": 0.4, "decay\_rate": 0.1}

}```

---

### Файл 2: `engine/sentio\_engine.py` (Основной движок)

Здесь будет вся математика и логика.

```python

# engine/sentio\_engine.py

import time

import math

from typing import Dict, Optional, Tuple

# Импортируем "геномы"

from .emotion\_definitions import EMOTION\_GENOME, NEUROMEDIATOR\_GENOME

class ActiveEmotion:

"""Представляет активную эмоцию с периодом возбуждения и охлаждения."""

def \_\_init\_\_(self, name: str, intensity: float):

self.name = name

self.intensity = max(0.0, min(1.0, intensity))

self.genome = EMOTION\_GENOME[name]

self.priority = self.genome["priority"]

# Период возбуждения (Attack): эмоция не может затухать сразу

self.attack\_phase\_duration = 0.5 # в секундах

self.time\_activated = time.time()

def update(self, delta\_time: float):

"""Обновляет интенсивность эмоции со временем."""

# Пока идет фаза возбуждения, затухания нет

if time.time() - self.time\_activated < self.attack\_phase\_duration:

return

# Формула затухания (экспоненциальное затухание)

decay = self.genome["base\_decay\_rate"] \* self.intensity \* delta\_time

self.intensity = max(0.0, self.intensity - decay)

def \_\_repr\_\_(self):

return f"Emotion({self.name}, Intensity: {self.intensity:.2f})"

class SentioEngine:

"""Эмоциональный калькулятор и таймер."""

def \_\_init\_\_(self):

self.active\_emotions: Dict[str, ActiveEmotion] = {}

self.neuromediators: Dict[str, float] = {

name: config["base\_level"] for name, config in NEUROMEDIATOR\_GENOME.items()

}

self.last\_update\_time = time.time()

def \_update\_physics(self):

"""Основной "тик" движка, обновляющий состояние со временем."""

current\_time = time.time()

delta\_time = current\_time - self.last\_update\_time

if delta\_time < 0.1: # Не обновляем слишком часто

return

# 1. Обновляем (ослабляем) существующие эмоции

emotions\_to\_remove = []

for name, emotion in self.active\_emotions.items():

emotion.update(delta\_time)

if emotion.intensity < 0.01:

emotions\_to\_remove.append(name)

for name in emotions\_to\_remove:

del self.active\_emotions[name]

# 2. Обновляем (нормализуем) уровень нейромедиаторов

for name, level in self.neuromediators.items():

genome = NEUROMEDIATOR\_GENOME[name]

base\_level = genome["base\_level"]

decay\_rate = genome["decay\_rate"]

# Стремление к базовому уровню (возвращение к гомеостазу)

if level > base\_level:

self.neuromediators[name] -= (level - base\_level) \* decay\_rate \* delta\_time

else:

self.neuromediators[name] += (base\_level - level) \* decay\_rate \* delta\_time

# 3. Эмоции влияют на нейромедиаторы

for emotion in self.active\_emotions.values():

for mediator, impact in emotion.genome["neuromediators\_impact"].items():

# Влияние пропорционально интенсивности эмоции

change = impact \* emotion.intensity \* delta\_time

self.neuromediators[mediator] = max(0.0, min(1.5, self.neuromediators[mediator] + change))

self.last\_update\_time = current\_time

def inject\_stimulus(self, stimulus: Dict[str, float]):

"""

Основной метод для внешнего воздействия.

Пример стимула: {"радость": 1.0, "удивление": 0.5}

"""

self.\_update\_physics()

for emotion\_name, intensity in stimulus.items():

if emotion\_name in EMOTION\_GENOME:

if emotion\_name in self.active\_emotions:

# Если эмоция уже активна, мы ее "подкрепляем"

self.active\_emotions[emotion\_name].intensity = max(

self.active\_emotions[emotion\_name].intensity, intensity

)

self.active\_emotions[emotion\_name].time\_activated = time.time() # Перезапускаем фазу возбуждения

else:

# Создаем новую активную эмоцию

self.active\_emotions[emotion\_name] = ActiveEmotion(emotion\_name, intensity)

def get\_emotional\_state\_report(self) -> Dict:

"""Возвращает полный отчет о текущем состоянии для LLM."""

self.\_update\_physics()

if not self.active\_emotions:

dominant\_emotion = None

else:

# Находим доминантную эмоцию по приоритету \* интенсивность

dominant\_emotion = max(

self.active\_emotions.values(),

key=lambda e: e.priority \* e.intensity

)

secondary\_emotions = sorted(

[e for e in self.active\_emotions.values() if e is not dominant\_emotion],

key=lambda e: e.priority \* e.intensity,

reverse=True

)

return {

"dominant\_emotion": dominant\_emotion.name if dominant\_emotion else "Neutral",

"dominant\_intensity": dominant\_emotion.intensity if dominant\_emotion else 0.0,

"secondary\_emotions": {e.name: e.intensity for e in secondary\_emotions[:2]},

"neuromediator\_balance": self.neuromediators,

"full\_emotional\_state": {name: emo.intensity for name, emo in self.active\_emotions.items()}

}

**Файл 3: main\_test.py (Пример использования)**

codePython

# main\_test.py

import time

from engine.sentio\_engine import SentioEngine

if \_\_name\_\_ == "\_\_main\_\_":

engine = SentioEngine()

print("--- НАЧАЛЬНОЕ СОСТОЯНИЕ ---")

print(engine.get\_emotional\_state\_report())

print("\n")

print("--- СОБЫТИЕ 1: Пользователь хвалит бота ---")

engine.inject\_stimulus({"радость": 1.0, "удивление": 0.6, "доверие": 0.5})

print(engine.get\_emotional\_state\_report())

print("\n")

print("--- ПРОШЛО 3 СЕКУНДЫ (ОХЛАЖДЕНИЕ) ---")

time.sleep(3)

print(engine.get\_emotional\_state\_report())

print("\n")

print("--- СОБЫТИЕ 2: Пользователь критикует бота ---")

engine.inject\_stimulus({"гнев": 0.9, "грусть": 0.7})

print(engine.get\_emotional\_state\_report())

print("\n")

print("--- ПРОШЛО 2 СЕКУНДЫ (Гнев доминирует) ---")

time.sleep(2)

# Радость и доверие уже почти угасли, а гнев и грусть еще сильны

print(engine.get\_emotional\_state\_report())

print("\n")

print("--- ПРОШЛО ЕЩЕ 5 СЕКУНД (Возвращение к норме) ---")

time.sleep(5)

print(engine.get\_emotional\_state\_report())

print("\n")

**Что делает этот код:**

1. **Конфигурируемость (emotion\_definitions.py):** Вы можете легко "тюнинговать" характер вашего бота, меняя приоритеты и скорости затухания в одном файле.
2. **Эмоциональный Таймер (ActiveEmotion):** Каждая эмоция имеет фазу "возбуждения", во время которой она не может угаснуть, и фазу "охлаждения" с экспоненциальным затуханием. Это имитирует реальную психологическую инерцию.
3. **Эмоциональный Калькулятор (SentioEngine):**
   * \_update\_physics(): Это "сердцебиение" системы. Оно постоянно симулирует течение времени, ослабляя эмоции и возвращая "химию мозга" к норме (гомеостаз).
   * inject\_stimulus(): Это API для внешнего мира. LLM-парсер (ваш emotion\_manager.py) будет вызывать этот метод, чтобы "впрыснуть" эмоции в систему.
   * get\_emotional\_state\_report(): Это API для вашего проактивного бота. Он вызывает этот метод, чтобы получить четкую, структурированную сводку о текущем "настроении", на основе которой он будет строить свой ответ, как мы и описали в "Таблице Экспрессии".

Этот код — это не просто пример. Это **работающий прототип** ядра вашего "Sentio Engine", который реализует все сложные концепции, которые мы обсуждали.