MCEE

Modèle Complet d'Évaluation des États

Résumé Technique pour le Développement
(avec Système de Phases)

Version 2.0

19 décembre 2025

Table des matières

1 Vue d'ensemble

Le MCEE (Modèle Complet d'Évaluation des États) est un système émotionnel complet intégrant :

- **24 émotions instantanées** (prédites par le module C++)
- **Système de phases émotionnelles** (8 phases qui modulent le comportement)
- **Graphe de mémoire Neo4j** (souvenirs, concepts, traumas)
- **Mécanismes de fusion et modulation** (adaptatifs selon la phase)
- **Système d'urgence « Amyghaleon »** (déclenché selon la phase)

2 Architecture des Composants (avec Phases)

```
+--------------------+
|   Capteurs         | -> Donnees environnementales
|   Feedbacks        | -> Fb_ext (externe), Fb_int (interne)
+---------+----------+
          |
          v
+--------------------+
|  Module C++        | -> Predit 24 emotions E_i(t)
|  (emotion)         |    depuis 14 dimensions
+---------+----------+
          |
          v
+-----------------------------------------------------+
|  Phase Detector                                     |
|  Detecte la phase emotionnelle actuelle             |
|                                                     |
|  Input:  24 emotions                                |
|  Output: Phase + PhaseConfig (alpha,beta,gamma,...) |
|                                                     |
|  Phases: SERENITE | JOIE | EXPLORATION | ANXIETE    |
|           PEUR | TRISTESSE | DEGOUT | CONFUSION      |
+---------+-------------------------------------------+
          |
          v
+--------------------+
|  MCEE Engine       | -> Mise a jour E_i(t+1) avec coefficients de phase
|  (Python)          |    Fusion -> E_global(t+1)
|                    |    Applique les parametres de la phase active
+---------+----------+
          |
          v
+--------------------+
|  Neo4j Graphe      | -> Souvenirs, concepts, traumas
|  Memoire           |    Activation, oubli, renforcement
|                    |    Consolidation modulee par phase
+---------+----------+
          |
          v
```

```
+--------------------+
|  Amyghaleon        | -> Reactions d'urgence (seuil selon phase)
|  (court-circuit)   |    Phase PEUR: seuil = 0.50 (!)
|                    |    Phase SERENITE: seuil = 0.85
+--------------------+
```

2.1 Flux de données détaillé

1. Module C++ émet 24 émotions → RabbitMQ (`mcee.emotional.input`)
2. Phase Detector reçoit et analyse → Détecte phase actuelle
3. Phase Detector fournit PhaseConfig → Coefficients adaptés
4. MCEE Engine reçoit :
   — Les 24 émotions brutes
   — La phase actuelle
   — Les coefficients $(\alpha, \beta, \gamma, \delta, \theta)$ de cette phase
   — Les seuils (Amyghaleon, consolidation, etc.)
5. MCEE Engine applique les formules avec les coefficients de phase
6. Neo4j est mis à jour selon les paramètres de phase
7. Amyghaleon vérifie le seuil de la phase actuelle

3 Système de Phases Émotionnelles

3.1 Les 8 Phases

Le système MCEE fonctionne en **8 phases émotionnelles** qui modulent tous les paramètres du système.

Phase	Prio.	α	δ	γ	θ	Seuil A.	Caractéristiques
SÉRÉNITÉ	1	0.25	0.30	0.12	0.10	0.85	Équilibre, apprentissage optimal
JOIE	2	0.40	0.35	0.08	0.05	0.95	Euphorie, renforcement positif
EXPLORATION	2	0.35	0.25	0.10	0.15	0.80	Apprentissage maximal
ANXIÉTÉ	3	0.40	0.45	0.06	0.08	0.70	Hypervigilance, biais négatif
PEUR	5	0.60	0.70	0.02	0.02	0.50	⚠URGENCE - Traumas dominants
TRISTESSE	3	0.20	0.55	0.05	0.12	0.90	Rumination, introspection
DÉGOÛT	4	0.50	0.40	0.08	0.08	0.75	Évitement, associations négatives
CONFUSION	2	0.35	0.50	0.15	0.15	0.80	Recherche d'info, incertitude

Table 1 – Les 8 phases émotionnelles et leurs coefficients

3.2 Transitions entre phases

Règles de transition :

1. **Priorité** : Phase PEUR (priorité 5) court-circuite toutes les autres
2. **Hystérésis** : Marge de 0.15 pour éviter oscillations
3. **Durée minimale** : 30 secondes avant changement (configurable)

4. **Urgence** : Peur > 0.85 OU Horreur > 0.8 → transition IMMÉDIATE

Exemple de séquence :

```
SERENITE (120s) -> EXPLORATION (45s) -> JOIE (60s) -> ANXIETE (35s) ->
    PEUR (15s)
                                                                        |
                                                                AMYGHALEON
                                                                   ACTIVE
```

3.3 Impact des phases sur le MCEE

Chaque phase modifie :

✓ **Coefficients MCEE** $(\alpha, \beta, \gamma, \delta, \theta)$ → Comportement des émotions
✓ **Seuil Amyghaleon** → Sensibilité aux urgences
✓ **Consolidation mémoire** → Force de mémorisation
✓ **Focus attentionnel** → Filtrage des stimuli
✓ **Taux d'apprentissage** → Vitesse d'adaptation

4 Formules Clés à Implémenter (avec Phases)

4.1 Mise à jour des émotions individuelles

La formule principale avec coefficients de phase :

$$\boxed{E_i(t+1) = E_i(t) + \alpha_{\text{phase}} \cdot Fb_{\text{ext}} + \beta_{\text{phase}} \cdot Fb_{\text{int}}(t) - \gamma_{\text{phase}} \cdot \Delta t + \delta_{\text{phase}} \cdot I_S + \theta_{\text{phase}} \cdot W_t} \quad (1)$$

Changements clés :

— Les coefficients $\alpha, \beta, \gamma, \delta, \theta$ sont maintenant **fournis par la phase active**
— Ils ne sont plus fixes mais **changent dynamiquement** selon l'état émotionnel

Variables :

— $E_i(t)$: Émotion i actuelle (du module C++)
— $\alpha_{\text{phase}}, \beta_{\text{phase}}, \gamma_{\text{phase}}, \delta_{\text{phase}}, \theta_{\text{phase}}$: Coefficients de la phase active
— Fb_{ext} : Feedback externe (utilisateur, environnement)
— Fb_{int} : Feedback interne (énergie, tension, auto-évaluation)
— Δt : Delta temps (décroissance naturelle)
— W_t : Coefficient de sagesse (expérience accumulée)
— I_S : Influence des souvenirs (calculé depuis Neo4j)

4.2 Implémentation Python

```python
class EmotionUpdater:
    def __init__(self):
        # Les coefficients sont maintenant dynamiques
        self.alpha = 0.3  # Valeur par defaut (sera ecrasee)
```

```
        self.beta = 0.2
        self.gamma = 0.1
        self.delta = 0.4
        self.theta = 0.1

    def set_coefficients_from_phase(self, phase_config):
        """
        Met a jour les coefficients selon la phase active.

        Args:
            phase_config: PhaseConfig contenant les coefficients
        """
        self.alpha = phase_config['alpha']
        self.beta = phase_config['beta']
        self.gamma = phase_config['gamma']
        self.delta = phase_config['delta']
        self.theta = phase_config['theta']

        print(f"Coefficients mis a jour pour phase:")
        print(f"   alpha={self.alpha:.2f}, beta={self.beta:.2f},
           gamma={self.gamma:.2f}")
        print(f"   delta={self.delta:.2f}, theta={self.theta:.2f}")

    def update_emotion(self, E_current, fb_ext, fb_int, delta_t,
                       influence_memories, wisdom):
        """
        Met a jour une emotion avec les coefficients de la phase active.
        """
        E_next = (E_current +
                  self.alpha * fb_ext +
                  self.beta * fb_int -
                  self.gamma * delta_t +
                  self.delta * influence_memories +
                  self.theta * wisdom)
        return np.clip(E_next, 0.0, 1.0)
```

Listing 1 – Classe EmotionUpdater avec phases

4.3 Exemples concrets par phase

4.3.1 Phase SÉRÉNITÉ

```
# Coefficients equilibres
alpha = 0.25  # Feedback externe modere
beta = 0.15   # Feedback interne bas
gamma = 0.12  # Decroissance normale
delta = 0.30  # Souvenirs moderes
theta = 0.10  # Sagesse active
```

```

# Resultat: Apprentissage stable, decisions posees
```

4.3.2 Phase PEUR ⚠

```
# Coefficients d'urgence
alpha = 0.60  # Feedback externe MAXIMAL (danger!)
beta = 0.45   # Feedback interne ELEVE (stress)
gamma = 0.02  # Decroissance TRES LENTE (etat persistant)
delta = 0.70  # Souvenirs DOMINANTS (traumas actives)
theta = 0.02  # Sagesse QUASI ABSENTE (reflexes)

# Resultat: Reaction d'urgence, traumas actives, pas d'apprentissage
    rationnel
```

4.3.3 Phase EXPLORATION

```
# Coefficients d'apprentissage
alpha = 0.35  # Feedback externe eleve (perception)
beta = 0.10   # Feedback interne bas (focus externe)
gamma = 0.10  # Decroissance normale
delta = 0.25  # Souvenirs moins influents (nouveaute)
theta = 0.15  # Sagesse ELEVEE (apprentissage)

# Resultat: Apprentissage maximal, attention focalisee
```

4.4 Variance (détection d'anomalies)

$$Var_i(t) = \frac{1}{m} \sum_{j=1}^{m} [E_i(t) - S_{i,j}]^2 \tag{2}$$

Inchangé, mais son interprétation dépend de la phase :

— **Phase ANXIÉTÉ** : Variance élevée = menace potentielle

— **Phase EXPLORATION** : Variance élevée = nouveauté intéressante

— **Phase PEUR** : Variance ignorée (réaction immédiate)

4.5 Fusion des émotions (avec stabilisation tanh)

$$E_{\text{global}}(t+1) = \tanh\left(E_{\text{global}}(t) + \sum_i [E_i(t+1) \cdot (1 - Var_{\text{global}}(t))] \right) \tag{3}$$

Inchangé dans la formule, mais les $E_i(t+1)$ sont calculés avec les coefficients de phase.

5 Gestion de la Mémoire (Neo4j) - Modulée par Phase

5.1 Structure des nœuds

```
// Souvenir episodique
CREATE (s:Souvenir {
    name: 'Evenement X',
    date: date('2025-12-19'),
    emotions: [0.7, 0.2, ...],  // 24 valeurs
    dominant: 'Joie',
    valence: 0.7,
    intensity: 0.8,
    last_activated: datetime(),
    activation_count: 1,
    weight: 0.5,
    type: 'positif',
    state: 'SouvenirConsolider',
    phase_at_creation: 'JOIE'  // NOUVEAU: Phase lors de la creation
})
```

Listing 2 – Structure d'un souvenir épisodique

5.2 Calcul de l'Influence des Souvenirs

Formule d'activation :

$$A(S_i) = \text{forget}(S_i, t) \times (1 + R(S_i)) \times \sum_k [C(S_i, S_k) \times Me(S_i, E_{\text{current}}) \times U(S_i)] \quad (4)$$

Mais les coefficients varient selon la phase :

— **Phase PEUR** : Le coefficient $\delta = 0.70$ amplifie massivement l'influence
— **Phase ANXIÉTÉ** : $\delta = 0.45$, souvenirs anxiogènes activés en priorité
— **Phase SÉRÉNITÉ** : $\delta = 0.30$, influence équilibrée

5.3 Requête Neo4j adaptée selon la phase

```
def query_relevant_memories(phase, emotions):
    """
    Recupere les souvenirs pertinents selon la phase active.
    """
    if phase == 'PEUR':
        # Priorite aux traumas et souvenirs de peur
        query = """
        MATCH (s:Souvenir)
        WHERE s.dominant IN ['Peur', 'Horreur', 'Anxiete']
           OR EXISTS((s)<-[:CONCERNE]-(t:Trauma))
        RETURN s
        ORDER BY s.intensity DESC, s.weight DESC
```

```
        LIMIT 20
        """

    elif phase == 'JOIE':
        # Priorite aux souvenirs positifs
        query = """
        MATCH (s:Souvenir)
        WHERE s.valence > 0.5
          AND s.dominant IN ['Joie', 'Satisfaction', 'Excitation']
        RETURN s
        ORDER BY s.valence DESC
        LIMIT 10
        """

    elif phase == 'ANXIETE':
        # Souvenirs negatifs recents
        query = """
        MATCH (s:Souvenir)
        WHERE s.dominant IN ['Anxiete', 'Peur', 'Confusion']
          AND s.last_activated > datetime() - duration({days: 30})
        RETURN s
        ORDER BY s.activation_count DESC
        LIMIT 15
        """

    else:  # SERENITE, EXPLORATION, etc.
        # Requete equilibree
        query = """
        MATCH (s:Souvenir)
        WHERE s.last_activated > datetime() - duration({days: 30})
        RETURN s
        ORDER BY s.weight DESC
        LIMIT 10
        """

    return neo4j.run(query)
```

Listing 3 – Récupération des souvenirs selon la phase

5.4 Consolidation modulée par phase

```
def should_consolidate(souvenir, phase_at_creation):
    """
    Decide si un souvenir doit etre consolide selon la phase
    ou il a ete cree.
    """
    intensity = souvenir['intensity']
    valence = souvenir['valence']
```

```

    # Phase PEUR: consolidation automatique (trauma potentiel)
    if phase_at_creation == 'PEUR':
        if intensity > 0.85 and valence < 0.2:
            return 'TRAUMA'  # Consolider immediatement en trauma
        return 'CONSOLIDATE_STRONG'

    # Phase JOIE: consolidation forte des souvenirs positifs
    elif phase_at_creation == 'JOIE':
        if valence > 0.7 and intensity > 0.6:
            return 'CONSOLIDATE_STRONG'
        return 'CONSOLIDATE_NORMAL'

    # Phase ANXIETE: consolidation selective (renforce biais negatif)
    elif phase_at_creation == 'ANXIETE':
        if valence < 0.3:
            return 'CONSOLIDATE_STRONG'  # Renforce souvenirs negatifs
        return 'FORGET'  # Oublie les positifs

    # Autres phases: consolidation normale
    else:
        if intensity > 0.5:
            return 'CONSOLIDATE_NORMAL'
        return 'FORGET'
```

Listing 4 – Processus de consolidation adapté

6 Système d'Urgence « Amyghaleon » (Seuils de Phase)

6.1 Principe

Le seuil d'activation d'Amyghaleon **dépend de la phase active** :

```
AMYGHALEON_THRESHOLDS = {
    'SERENITE':     0.85,  # Difficile a declencher
    'JOIE':         0.95,  # Tres difficile (euphorie)
    'EXPLORATION': 0.80,  # Modere
    'ANXIETE':      0.70,  # Facile (deja vigilant)
    'PEUR':         0.50,  # TRES FACILE (hypersensible)
    'TRISTESSE':    0.90,  # Difficile (etat depressif)
    'DEGOUT':       0.75,  # Modere
    'CONFUSION':    0.80   # Modere
}
```

Listing 5 – Seuils Amyghaleon par phase

6.2 Implémentation

```python
class Amyghaleon:
    CRITICAL_EMOTIONS = ['Peur', 'Horreur', 'Anxiete']

    def check_emergency(self, emotions, souvenirs_actives,
        phase_threshold):
        """
        Verifie si une urgence est detectee avec le seuil de la phase.

        Args:
            emotions: dict, 24 emotions actuelles
            souvenirs_actives: list, souvenirs actives
            phase_threshold: float, seuil de la phase active (0.50 a
                0.95)
        """
        # 1. Verifier les emotions critiques
        max_critical = max([emotions.get(e, 0) for e in
            self.CRITICAL_EMOTIONS])

        if max_critical > phase_threshold:
            print(f"Amyghaleon: Emotion critique {max_critical:.3f} >
                seuil {phase_threshold:.3f}")
            return True

        # 2. Verifier les traumas actives
        for s in souvenirs_actives:
            if s.get('trauma', False) and s['activation'] >
                (phase_threshold - 0.2):
                print(f"Amyghaleon: Trauma active {s['name']}")
                return True

        # 3. Combinaison critique + trauma
        if max_critical > (phase_threshold + 0.2):
            for s in souvenirs_actives:
                if s.get('trauma') and s['activation'] > 0.6:
                    print(f"Amyghaleon: Combinaison critique+trauma")
                    return True

        return False

    def trigger_emergency_response(self, emotions, phase):
        """
        Declenche une reponse d'urgence adaptee a la phase.
        """
        max_emo = max(emotions.items(), key=lambda x: x[1])

        responses = {
            'Peur': {'action': 'FUITE', 'priority': 'CRITIQUE'},
```

```
            'Horreur': {'action': 'BLOCAGE', 'priority': 'CRITIQUE'},
            'Anxiete': {'action': 'ALERTE', 'priority': 'ELEVEE'}
        }

        response = responses.get(max_emo[0], {'action': 'SURVEILLANCE',
            'priority': 'MOYENNE'})
        response['phase_at_trigger'] = phase
        response['emotion_value'] = max_emo[1]

        return response
```

Listing 6 – Classe Amyghaleon

6.3 Exemple : Phase PEUR vs Phase SÉRÉNITÉ

Situation identique : Peur = 0.65

```
# Phase SERENITE (seuil = 0.85)
if 0.65 > 0.85:  # False
    # Pas d'urgence, traitement normal
    pass

# Phase PEUR (seuil = 0.50)
if 0.65 > 0.50:  # True
    # URGENCE DETECTEE
    # Court-circuit MCEE
    # Action immediate : FUITE/BLOCAGE
    pass
```

7 Implémentation Complète du MCEE Engine

7.1 Classe principale

```
import pika
import numpy as np
from datetime import datetime, timedelta
from py2neo import Graph
from sklearn.metrics.pairwise import cosine_similarity

class MCEEEngine:
    def __init__(self, neo4j_uri, neo4j_user, neo4j_pass,
        rabbitmq_config):
        # Connexion Neo4j
        self.graph = Graph(neo4j_uri, auth=(neo4j_user, neo4j_pass))

        # NOUVEAU: Detecteur de phase
        self.phase_detector = PhaseDetector(hysteresis_margin=0.15,
                                            min_phase_duration=30.0)
```

```
15
16          # Modules MCEE
17          self.emotion_updater = EmotionUpdater()
18          self.memory_manager = MemoryManager(self.graph)
19          self.amyghaleon = Amyghaleon()
20          self.dream = DreamModule(self.graph)
21
22          # Etat
23          self.E_global = 0.0
24          self.wisdom = 0.0
25          self.emotions_history = []
26          self.current_phase = 'SERENITE'  # NOUVEAU
27
28          # Configuration RabbitMQ...
29          print("MCEE Engine initialise avec systeme de phases")
30
31      def on_emotions_received(self, ch, method, properties, body):
32          """
33          Callback appele quand des emotions sont recues du module C++.
34          """
35          import json
36          emotions_raw = json.loads(body.decode('utf-8'))
37
38          # 1. DETECTION DE PHASE
39          previous_phase = self.current_phase
40          self.current_phase =
               self.phase_detector.detect_phase(emotions_raw)
41
42          if self.current_phase != previous_phase:
43              print(f"Transition de phase: {previous_phase} ->
                   {self.current_phase}")
44
45          # 2. RECUPERER LA CONFIGURATION DE LA PHASE
46          phase_config = self.phase_detector.get_phase_config()
47
48          # 3. METTRE A JOUR LES COEFFICIENTS MCEE
49          self.emotion_updater.set_coefficients_from_phase(phase_config)
50
51          # 4. RECUPERER LES SOUVENIRS (adapte selon phase)
52          souvenirs = self.memory_manager.query_relevant_memories(
53              phase=self.current_phase,
54              emotions=emotions_raw
55          )
56
57          # 5. VERIFIER AMYGHALEON (seuil de phase)
58          amyghaleon_threshold = phase_config['amyghaleon_threshold']
59
```

```
        if self.amyghaleon.check_emergency(emotions_raw, souvenirs,
            amyghaleon_threshold):
            response = self.amyghaleon.trigger_emergency_response(
                emotions_raw,
                self.current_phase
            )
            self.execute_emergency_action(response)
            return  # Court-circuit

        # 6-13. Suite du traitement...
```

Listing 7 – MCEEEngine avec système de phases

7.2 Détecteur de phases

```
class PhaseDetector:
    """
    Detecteur de phases emotionnelles.
    Peut etre implemente en Python ou utiliser le module C++.
    """

    PHASE_CONFIGS = {
        'SERENITE': {
            'alpha': 0.25, 'beta': 0.15, 'gamma': 0.12, 'delta': 0.30,
                'theta': 0.10,
            'amyghaleon_threshold': 0.85,
            'memory_consolidation': 0.4,
            'learning_rate': 1.0,
            'priority': 1
        },
        'PEUR': {
            'alpha': 0.60, 'beta': 0.45, 'gamma': 0.02, 'delta': 0.70,
                'theta': 0.02,
            'amyghaleon_threshold': 0.50,
            'memory_consolidation': 0.8,
            'learning_rate': 0.3,
            'priority': 5
        },
        # ... autres phases
    }

    def __init__(self, hysteresis_margin=0.15, min_phase_duration=30.0):
        self.current_phase = 'SERENITE'
        self.phase_start_time = datetime.now()
        self.hysteresis_margin = hysteresis_margin
        self.min_phase_duration = timedelta(seconds=min_phase_duration)

    def detect_phase(self, emotions):
```

```
        """
        Detecte la phase actuelle selon les emotions.
        """
        # Calculer les scores de chaque phase
        scores = self._compute_phase_scores(emotions)

        # Trouver la meilleure
        best_phase = max(scores.items(), key=lambda x: x[1])[0]

        # Verifier urgence
        if best_phase == 'PEUR' and (emotions.get('Peur', 0) > 0.85 or
                                     emotions.get('Horreur', 0) > 0.8):
            self._transition_to(best_phase, reason="URGENCE")
            return best_phase

        # Appliquer hysteresis et duree minimale...
        return self.current_phase

    def get_phase_config(self):
        """Retourne la configuration de la phase actuelle"""
        return self.PHASE_CONFIGS[self.current_phase]
```

Listing 8 – Classe PhaseDetector

8 Mise à Jour de Neo4j avec Phases

8.1 Enregistrement des souvenirs avec phase

```
class MemoryManager:
    def record_new_memory(self, emotions, E_global, phase, context):
        """
        Enregistre un nouveau souvenir avec la phase active.
        """
        dominant = max(emotions.items(), key=lambda x: x[1])[0]
        valence = self._compute_valence(emotions)
        intensity = np.mean(list(emotions.values()))

        query = """
        CREATE (s:Souvenir {
            name: $name,
            date: date($date),
            emotions: $emotions,
            dominant: $dominant,
            valence: $valence,
            intensity: $intensity,
            last_activated: datetime(),
            activation_count: 1,
            weight: $weight,
```

```
            state: 'SouvenirConsolider',
            phase_at_creation: $phase,
            E_global: $E_global
        })
        RETURN s
        """

        # Poids initial selon phase
        initial_weight = self._get_initial_weight(phase, intensity,
            valence)

        self.graph.run(query, ...)

    def _get_initial_weight(self, phase, intensity, valence):
        """
        Calcule le poids initial selon la phase.
        """
        if phase == 'PEUR':
            return min(1.0, 0.7 + intensity * 0.3)
        elif phase == 'JOIE':
            if valence > 0.6:
                return min(1.0, 0.6 + valence * 0.4)
            return 0.3
        elif phase == 'ANXIETE':
            if valence < 0.3:
                return min(1.0, 0.5 + (1 - valence) * 0.5)
            return 0.2
        else:
            return 0.5
```

Listing 9 – MemoryManager avec phases

8.2 Création de traumas potentiels

```
def create_potential_trauma(self, emotion_state):
    """
    Cree un trauma potentiel suite a une phase PEUR.
    """
    if not emotion_state:
        return

    query = """
    CREATE (t:Trauma {
        name: $name,
        date: date($date),
        emotion_signature: $emotions,
        intensity: $intensity,
        trauma: true,
```

```
        forget_rate: 0.005,
        state: 'TraumaConsolider',
        immediate_transfer: true,
        phase_trigger: 'PEUR'
    })
    RETURN t
    """

    intensity = max(emotion_state['emotions'].values())

    self.graph.run(query, ...)
    print(f"Trauma potentiel cree (intensite={intensity:.3f})")
```

Listing 10 – Création de trauma en phase PEUR

9 Statistiques et Monitoring avec Phases

9.1 Enregistrement des phases dans Neo4j

```
def log_phase_transition(from_phase, to_phase, emotions, duration):
    """
    Enregistre une transition de phase dans Neo4j.
    """
    query = """
    CREATE (t:PhaseTransition {
        from_phase: $from_phase,
        to_phase: $to_phase,
        timestamp: datetime(),
        duration_previous: $duration,
        trigger_emotions: $emotions,
        emotion_max: $emotion_max
    })
    RETURN t
    """

    emotion_max = max(emotions.items(), key=lambda x: x[1])

    graph.run(query,
        from_phase=from_phase,
        to_phase=to_phase,
        duration=duration,
        emotions=list(emotions.values()),
        emotion_max=f"{emotion_max[0]}={emotion_max[1]:.3f}"
    )
```

Listing 11 – Logging des transitions de phase

9.2 Analyse des patterns de phases

```
// Transitions les plus frequentes
MATCH (t:PhaseTransition)
RETURN t.from_phase, t.to_phase, count(*) as count
ORDER BY count DESC
LIMIT 10

// Duree moyenne par phase
MATCH (t:PhaseTransition)
RETURN t.from_phase, avg(t.duration_previous) as avg_duration
ORDER BY avg_duration DESC

// Phases chroniques (>1h sans changement)
MATCH (t:PhaseTransition)
WHERE t.duration_previous > 3600
RETURN t.from_phase, t.timestamp, t.duration_previous
ORDER BY t.timestamp DESC

// Correlation phases / souvenirs crees
MATCH (s:Souvenir)
RETURN s.phase_at_creation, count(*) as count, avg(s.intensity) as
    avg_intensity
ORDER BY count DESC
```

Listing 12 – Requêtes d'analyse des phases

10 Roadmap de Développement

10.1 Phase 1 : Base avec Phases (2-3 semaines)

- ✓ Détecteur de phases (Python ou C++)
- ☐ Implémenter `EmotionUpdater` avec coefficients dynamiques
- ☐ Créer `MemoryManager` avec requêtes adaptées par phase
- ☐ Intégration RabbitMQ + Phase Detector
- ☐ Tests unitaires

10.2 Phase 2 : Mémoire Modulée (2-3 semaines)

- ☐ Calcul d'activation avec coefficients de phase
- ☐ Gestion MCT/MLT avec poids selon phase
- ☐ Enregistrement avec `phase_at_creation`
- ☐ Consolidation adaptée par phase
- ☐ Tests d'intégration Neo4j

10.3 Phase 3 : Amyghaleon Adaptatif (2-3 semaines)

- ☐ Seuils dynamiques selon phase

- ☐ Création traumas en phase PEUR
- ☐ Court-circuit avec contexte de phase
- ☐ Module Rêve avec analyse des phases
- ☐ Interface de monitoring

10.4 Phase 4 : Optimisation (1-2 semaines)

- ☐ Calibration coefficients par phase
- ☐ Performance Neo4j (index sur phases)
- ☐ Tests de charge avec changements de phase
- ☐ Documentation complète avec phases

11 Points Critiques avec Phases

11.1 Éviter les boucles infinies

Problème : Phase PEUR → active traumas → renforce PEUR → boucle

Solution : Durée minimale forcée + décrémentation graduelle

```
if phase == 'PEUR' and time_in_phase > 60:
    # Forcer decroissance des emotions critiques
    emotions['Peur'] *= 0.95
    emotions['Horreur'] *= 0.95
```

11.2 Transitions trop rapides

Problème : Oscillations ANXIÉTÉ ↔ PEUR

Solution : Hystérésis + durée minimale

```
HYSTERESIS_MARGIN = 0.15  # Marge pour rester dans phase actuelle
MIN_PHASE_DURATION = 30.0  # Secondes minimum
```

11.3 Phase PEUR chronique

Problème : Reste bloqué en phase PEUR

Solution : Seuil de sortie différent + timeout

```
if phase == 'PEUR' and time_in_phase > 300:  # 5 minutes
    # Forcer transition vers ANXIETE si emotions < 0.6
    if max(emotions['Peur'], emotions['Horreur']) < 0.6:
        force_transition('ANXIETE')
```

12 Configuration Recommandée

12.1 Fichier de configuration JSON

```json
{
  "phases": {
    "serenite": {
      "alpha": 0.25,
      "beta": 0.15,
      "gamma": 0.12,
      "delta": 0.30,
      "theta": 0.10,
      "amyghaleon_threshold": 0.85,
      "memory_consolidation": 0.4,
      "learning_rate": 1.0
    },
    "peur": {
      "alpha": 0.60,
      "beta": 0.45,
      "gamma": 0.02,
      "delta": 0.70,
      "theta": 0.02,
      "amyghaleon_threshold": 0.50,
      "memory_consolidation": 0.8,
      "learning_rate": 0.3
    }
  },
  "phase_detector": {
    "hysteresis_margin": 0.15,
    "min_phase_duration": 30.0,
    "emergency_threshold_peur": 0.85,
    "emergency_threshold_horreur": 0.80
  }
}
```

Listing 13 – phase_config.json

12.2 Chargement dynamique

```python
import json

def load_phase_config(config_file='phase_config.json'):
    with open(config_file, 'r') as f:
        config = json.load(f)
    return config

# Utilisation
config = load_phase_config()
phase_detector = PhaseDetector(
    hysteresis_margin=config['phase_detector']['hysteresis_margin'],
    min_phase_duration=config['phase_detector']['min_phase_duration']
```

```
)
```

13 Résumé des Changements par rapport à la Version Originale

13.1 Ce qui reste identique

- Structure Neo4j des nœuds et relations
- Formule d'activation des souvenirs : $A(S_i) = \text{forget} \times (1 + R) \times \Sigma[C \times Me \times U]$
- Formule de fusion : $E_{\text{global}}(t + 1) = \tanh(\ldots)$
- Module Rêve pour consolidation
- Amyghaleon (principe général)

13.2 Ce qui change

Aspect	**Avant**	**Maintenant**
Coefficients $\alpha, \beta, \gamma, \delta, \theta$	Fixes (0.3, 0.2, 0.1, 0.4, 0.1)	Dynamiques selon phase
Seuil Amyghaleon	Fixe (0.85)	Variable (0.50 à 0.95)
Requêtes Neo4j	Génériques	Adaptées par phase
Consolidation	Uniforme	Selon phase de création
Pipeline	C++ → MCEE → Neo4j	C++ → **Phase** → MCEE → Neo4j
Souvenirs	État simple	+ `phase_at_creation`
Architecture	4 composants	**5 composants** (+ Phase Detector)

Table 2 – Comparaison des versions

14 Dépendances Mises à Jour

```
# requirements_mcee.txt
pika>=1.3.0
py2neo>=2021.2.3
numpy>=1.21.0
scikit-learn>=1.0.0
python-dateutil>=2.8.0
# NOUVEAU: Pour le detecteur de phases (si version Python)
typing-extensions>=4.0.0
```

Listing 14 – requirements_mcee.txt

15 Formules Résumées (Version avec Phases)

Pipeline MCEE avec Phases

1. **Phase Active** $\rightarrow (\alpha_{\text{phase}}, \beta_{\text{phase}}, \gamma_{\text{phase}}, \delta_{\text{phase}}, \theta_{\text{phase}}, \text{seuil}_{\text{phase}})$
2. **Mise à jour émotions** :

$$E_i(t+1) = E_i(t) + \alpha \cdot Fb_{\text{ext}} + \beta \cdot Fb_{\text{int}} - \gamma \cdot \Delta t + \delta \cdot I_S + \theta \cdot W_t$$

3. **Variance** (interprétation selon phase) :

$$Var_i(t) = \frac{1}{m} \sum_j [E_i(t) - S_{i,j}]^2$$

4. **Fusion** :

$$E_{\text{global}}(t+1) = \tanh\left(E_{\text{global}}(t) + \sum_i [E_i(t+1) \cdot (1 - Var_{\text{global}})] \right)$$

5. **Activation souvenirs** (filtrés par phase) :

$$A(S_i) = \text{forget} \times (1+R) \times \sum_k [C \times Me \times U]$$

6. **Amyghaleon** vérifie :

$$\max(\text{Peur}, \text{Horreur}, \text{Anxiété}) > \text{seuil}_{\text{phase}}$$

A Tableau Complet des Phases

Phase	α	β	γ	δ	θ	**Seuil A.**	**Cons.**	**Learn.**	**Focus**	**Comportement**
SÉRÉNITÉ	0.25	0.15	0.12	0.30	0.10	0.85	0.4	1.0	0.5	Équilibre, apprentissage optimal
JOIE	0.40	0.25	0.08	0.35	0.05	0.95	0.5	1.3	0.3	Renforcement positif, risque sous-estimé
EXPLORATION	0.35	0.10	0.10	0.25	0.15	0.80	0.6	1.5	0.8	Apprentissage MAXIMAL, attention focalisée
ANXIÉTÉ	0.40	0.30	0.06	0.45	0.08	0.70	0.4	0.8	0.6	Hypervigilance, biais négatif
PEUR	0.60	0.45	0.02	0.70	0.02	0.50	0.8	0.3	0.95	⚠ URGENCE - Traumas dominants

Phase	α	β	γ	δ	θ	**Seuil A.**	**Cons.**	**Learn.**	**Focus**	**Comportement**
TRISTESSE	0.20	0.40	0.05	0.55	0.12	0.90	0.5	0.6	0.4	Rumination, introspection
DÉGOÛT	0.50	0.25	0.08	0.40	0.08	0.75	0.6	0.9	0.7	Évitement, associations négatives
CONFUSION	0.35	0.30	0.15	0.50	0.15	0.80	0.3	0.7	0.5	Recherche info, incertitude

Fin du résumé technique (version avec phases) — MCEE v2.0