Лекція 24

BDI: Переконання-Бажання-Наміри

Когнітивна архітектура раціональних агентів

Beliefs • Desires • Intentions • Plans • Deliberation

🤖 Інтелектуальні автономні агенти з цілеспрямованою поведінкою

Частина 1: Теоретичні основи

О. С. Бичков • 2025

План лекції (Частина 1)

1. Що таке BDI?
2. Історія та філософські основи
3. Beliefs (Переконання)
4. Desires (Бажання)
5. Intentions (Наміри)
6. Взаємозв'язок B-D-I
7. Цикл обробки BDI
8. Deliberation (Обдумування)
9. Means-End Reasoning
10. Плани та бібліотека планів
11. Порівняння з реактивними агентами
12. Порівняння з іншими архітектурами
13. Переваги BDI
14. Недоліки BDI
15. 🤖 BDI для БПЛА
16. Підсумок

Частина 2: Реалізація на Rust, повний BDI-агент БПЛА

Що таке BDI?

BDI (Beliefs-Desires-Intentions) — когнітивна архітектура
для моделювання раціональних агентів

Три ментальні стани:

- Beliefs (Переконання) — що агент знає/вважає про світ
- Desires (Бажання) — чого агент хоче досягти
- Intentions (Наміри) — до чого агент зобов'язався

Ключова ідея:
Агент приймає рішення подібно до людини —

"BDI — це спроба формалізувати людське практичне міркування"
— Michael Bratman, філософ

Історія BDI

Філософські корені:

1987 — Michael Bratman
"Intention, Plans, and Practical Reason"
Філософська теорія практичного міркування
Інтенції як "зобов'язання до дії"

Комп'ютерна реалізація:

1988 — PRS (Procedural Reasoning System)
Georgeff & Lansky, SRI International
Перша BDI-система для реального часу

1991 — dMARS
Australian AI Institute
Distributed Multi-Agent Reasoning System

1999 — AgentSpeak(L)
Формальна мова для BDI-агентів

2000s — Jason, JACK, Jadex
Сучасні BDI-фреймворки

Філософія Bratman

Bratman's теорія інтенцій:

1. Інтенції керують практичним міркуванням
   Наміри фільтрують допустимі варіанти дій

2. Інтенції мають інерцію (persistence)
   Ми не переглядаємо наміри щомиті
   Це економить когнітивні ресурси

3. Інтенції обмежують майбутні дії
   Прийняті наміри створюють зобов'язання

4. Інтенції ведуть до планування
   Намір досягти цілі викликає пошук способу

Приклад: намір "поїхати на конференцію" → плани "купити квиток, забронювати готель..."

5. Інтенції мають ієрархію
   Загальні → конкретні → дії

Beliefs (Переконання)

```
// Приклад Beliefs для БПЛА
struct Beliefs {
    // Власний стан
    my_position: Position,
    my_battery: u8,
    my_health: i32,

    // Знання про середовище
    known_obstacles: Vec<Obstacle>,
    known_targets: Vec<Target>,
    weather_conditions: Weather,

    // Знання про інших агентів
    friendly_agents: HashMap<AgentId, AgentInfo>,
    enemy_positions: Vec<Position>,

    // Стан місії
    mission_status: MissionStatus,
    base_location: Position,
}
```

- Початкові знання

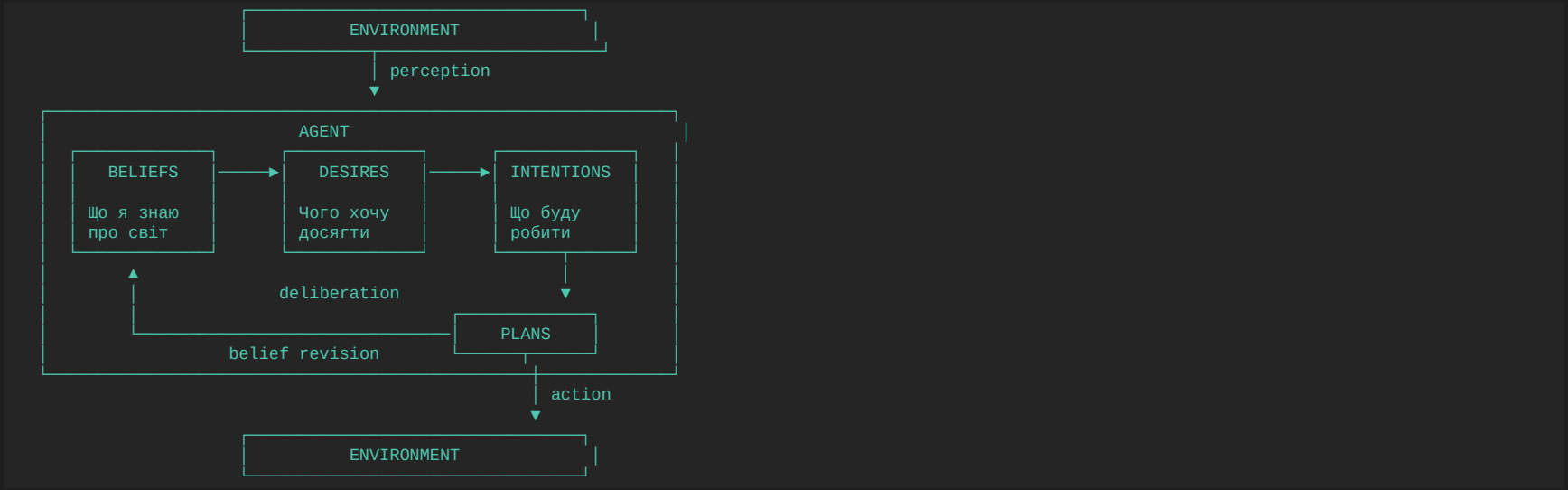
```
// Приклад Desires для БПЛА
#[derive(Clone, PartialEq)]
enum Desire {
    // Achieve goals
    ReachPosition(Position),
    DestroyTarget(TargetId),
    CompletePatrol(Area),

    // Maintain goals
    MaintainFormation(FormationId),
    KeepAltitude(f32),
    StayConnected,

    // Avoid goals
    AvoidCollision,
    AvoidNoFlyZone(Area),
    // Query goals
    LocateTarget(TargetType),
    ScanArea(Area),
}

struct DesireWithPriority {
    desire: Desire,
    priority: u8,  // 0-255
}
```

- Avoid goals — уникнути стану
- Query goals — дізнатись інформацію

```
// Intention = Desire + Commitment + Plan
struct Intention {
    // Що саме намір
    goal: Desire,

    // План досягнення
    plan: Plan,

    // Поточний стан виконання
    status: IntentionStatus,

    // Умови скасування
    drop_conditions: Vec<Condition>,

    // Час створення
    created_at: Instant,
}

enum IntentionStatus {
    Active,          // Виконується
    Suspended,       // Призупинено
    Achieved,        // Досягнуто
    Failed,          // Провалено
    Dropped,         // Скасовано
}
```

- Influence — активація практичне міркування
- Commitment — зобов'язання діяти

Взаємозв'язок Beliefs-Desires-Intentions



Цикл обробки BDI-агента

Основний цикл BDI-агента:

loop {
  1. Perception — отримати дані від сенсорів
  2. Belief Revision — оновити переконання
  3. Option Generation — генерувати можливі бажання
  4. Deliberation — обрати intentions з desires
  5. Means-End Reasoning — обрати план
  6. Execute — виконати наступну дію плану
}

```
Perception → BRF → Options → Deliberate → Plan → Execute → repeat
```

- Plan Selection — вибір плану для intention

Deliberation (Обдумування)

```
fn deliberate(beliefs: &Beliefs, desires: &[Desire], current_intentions: &[Intention]) ->
Vec<Intention> {
    let mut new_intentions = Vec::new();

    // Фільтруємо desires що можливо досягти
    let achievable = desires.iter()
        .filter(|d| is_achievable(d, beliefs))
        .collect::<Vec<_>>();

    // Сортуємо за пріоритетом
    let mut prioritized = achievable.clone();
    prioritized.sort_by_key(|d| d.priority);

    // Перевіряємо сумісність з поточними intentions
    for desire in prioritized {
        if is_compatible(desire, current_intentions) {
            if let Some(plan) = find_plan(desire, beliefs) {
                new_intentions.push(Intention::new(desire.clone(), plan));
            }
        }
    }

    new_intentions
}
```

- Constraint-based — задовольняє обмеження
- Deadline-based — затерміновісно

```
// Plan Library
struct PlanLibrary {
    plans: Vec<PlanTemplate>,
}

struct PlanTemplate {
    name: String,
    trigger: Desire,              // Для якого бажання
    context: Vec<Condition>,      // Коли застосовний
    body: Vec<PlanStep>,          // Кроки плану
}

enum PlanStep {
    Action(Action),               // Примітивна дія
    Subgoal(Desire),              // Підціль (рекурсія)
    Conditional(Condition, Box<PlanStep>),  // Умовний крок
    Loop(Condition, Box<PlanStep>),         // Цикл
}
```

- Reactive plans — прості ситуація→дія
- Procedural plans — готові процедури

```
// Приклад плану для БПЛА
let patrol_plan = PlanTemplate {
    name: "patrol_area".to_string(),
    trigger: Desire::CompletePatrol(Area::default()),
    context: vec![
        Condition::BatteryAbove(20),
        Condition::NoActiveThreats,
    ],
    body: vec![
        // 1. Піднятись на робочу висоту
        PlanStep::Action(Action::SetAltitude(100.0)),

        // 2. Для кожної waypoint
        PlanStep::Loop(
            Condition::WaypointsRemaining,
            Box::new(PlanStep::Sequence(vec![
                // Летіти до waypoint
                PlanStep::Subgoal(Desire::ReachPosition(Position::NEXT_WAYPOINT)),
                // Сканувати область
                PlanStep::Action(Action::ScanArea(50.0)),
                // Якщо виявлено ціль
                PlanStep::Condition(
                    Condition::TargetDetected,
                    Box::new(PlanStep::Action(Action::ReportTarget)),
                ),
            ]))
        ),

        // 3. Повернутись на базу
        PlanStep::Subgoal(Desire::ReachPosition(Position::BASE)),
```

Реактивні агенти vs BDI

Аспект	Реактивний	BDI
Архітектура	Stimulus-Response	Deliberative
Стан	Мінімальний/відсутній	Beliefs, Desires, Intentions
Планування	Відсутнє	Means-End Reasoning
Рішення	Правила if-then	Deliberation process
Адаптивність	Швидка	Повільніша, але гнучкіша
Складність	Низька	Висока
Пояснення	Важке	Можливе (intentions)
Use case	Прості задачі	Складні автономні задачі

Гібридний підхід: Layered Architecture

- Reactive layer — швидка реакція на загрози
- BDI layer — стратегічне планування

Порівняння когнітивних архітектур

Архітектура	Основа	Сила	Слабкість
BDI	Practical reasoning	Гнучкість, пояснення	Складність
SOAR	Production rules	Навчання	Overhead
ACT-R	Cognitive model	Реалістичність	Специфічність
Subsumption	Layers	Робастність	Негнучкість
FSM	States	Простота	Не масштабується
Utility AI	Scoring	Гнучкість	Налаштування

BDI найкраще підходить для:

- Автономних агентів з довгостроковими цілями
- Систем де потрібно пояснити поведінку
- Динамічних середовищ з невизначеністю

Переваги BDI

✅ Інтуїтивність
Модель близька до людського мислення
Легко проектувати та розуміти

✅ Гнучкість
Адаптація до змін середовища
Вибір альтернативних планів

✅ Пояснюваність
Можна пояснити "чому" агент діє
Intentions як причини дій

✅ Модульність
Beliefs, Desires, Plans — окремі модулі
Легко розширювати

✅ Теоретична основа
Формальна семантика
Доведені властивості

✅ Реалістична поведінка
Commitment до цілей
Не "смикається" між цілями

Недоліки BDI

❌ Складність реалізації
Багато компонентів для координації
Потребує ретельного проектування

❌ Обчислювальна вартість
Deliberation та planning — дорогі операції
Не підходить для hard real-time

❌ Проблема commitment
Занадто сильний commitment — ігнорує зміни
Занадто слабкий — "метушиться"

❌ Складність балансування
Коли переглядати intentions?
Як часто deliberate?

❌ Знання про плани
Потрібна бібліотека планів
Не генерує плани "з нуля"

❌ Масштабування
Багато beliefs/desires = повільна deliberation

MAC: BDI архітектура для БПЛА

Чому BDI ідеально підходить для БПЛА:

- Автономність — БПЛА повинен діяти самостійно
- Цілеспрямованість — є чіткі місії
- Динамічне середовище — ситуація змінюється
- Пояснюваність — командир повинен розуміти дії
- Пріоритезація — безпека > місія > економія

Typical Beliefs для БПЛА:
- Власна позиція, швидкість, батарея
- Позиції союзників та противників
- Стан місії, waypoints
- Погодні умови, no-fly zones

Typical Desires:
- Виконати місію (patrol, attack, escort)
- Уникнути колізій
- Зберегти енергію
- Повернутись на базу при потребі

🤖 MAC: Приклад BDI-рішення БПЛА

Сценарій: БПЛА на патрулюванні виявляє ворожу техніку

Beliefs:
• my_position = (100, 200, 50)
• battery = 45%
• detected_target = Tank at (150, 180)
• friendly_support = None
• distance_to_base = 500m

Desires (з пріоритетами):
1. SURVIVE (priority: 100)
2. COMPLETE_MISSION (priority: 80)
3. DESTROY_TARGET (priority: 70)
4. CONSERVE_ENERGY (priority: 30)

Deliberation:
• Battery 45% — достатньо для атаки та повернення
• Ціль в зоні досяжності
• Немає підтримки — ризик

Intention: REPORT_AND_TRACK (не атакувати без підтримки)
Plan: fly_to_safe_distance → report_target → track_target → wait_for_support

```
// Ієрархія бажань за пріоритетом
enum DesireCategory {
    // Критичні (завжди виконуються)
    Safety {
        AvoidCollision,      // Уникати зіткнень
        EmergencyLanding,    // Аварійна посадка
        AvoidHostileFire,    // Уникати вогню
    },

    // Високий пріоритет
    Survival {
        ReturnToBase,        // При низькій батареї
        MaintainComms,       // Зв'язок з командуванням
        EvadeThreats,        // Ухилятись від загроз
    },

    // Середній пріоритет
    Mission {
        CompletePatrol,      // Патрулювання
        EngageTarget,        // Атака цілі
        ProvideSupport,      // Підтримка союзників
    },

    // Низький пріоритет
    Efficiency {
        OptimizePath,        // Оптимізувати маршрут
        ConserveEnergy,      // Економити енергію
        MaintainFormation,   // Тримати формацію
    },
}
```

🤖 MAC: Умови перегляду Intentions

Коли переглядати intentions?

✅ Intention succeeded — ціль досягнута
✅ Intention impossible — ціль недосяжна
✅ Higher priority desire — з'явилась важливіша ціль
✅ Context changed — умови плану більше не виконуються

❌ НЕ переглядати коли:
• Невеликі зміни середовища
• План все ще виконується нормально
• Немає кращої альтернативи

Рекомендація: Single-minded з periodic review (кожні N тіків)

• Single-minded commitment — до успіху/неможливості
• Open-minded commitment — часта переоцінка

Формальна семантика BDI

BDI Logic (Rao & Georgeff, 1991):

Модальні оператори:
- BEL(φ) — агент вірить що φ
- DES(φ) — агент бажає φ
- INT(φ) — агент має намір φ

Властивості:
- BEL(φ) → ¬DES(¬φ) — не бажаємо те, що вважаємо неможливим
- INT(φ) → DES(φ) — intention передбачає desire
- INT(φ) → BEL(можливо φ) — намір передбачає віру в можливість

Commitment strategies (формально):
- Blind: INT(φ) до виконання φ
- Single-minded: INT(φ) до BEL(φ) або BEL(¬можливо φ)
- Open-minded: INT(φ) доки DES(φ)

Формалізація дозволяє доводити властивості агента!

Підсумок: Частина 1

BDI — когнітивна архітектура раціональних агентів:

- Beliefs — що агент знає про світ
- Desires — чого агент хоче
- Intentions — до чого зобов'язався

Ключові процеси:
- Belief Revision — оновлення знань
- Deliberation — вибір intentions
- Means-End Reasoning — вибір плану

Переваги:
- Інтуїтивність, гнучкість, пояснюваність

Недоліки:
- Складність, обчислювальна вартість

→ Частина 2: Реалізація на Rust, повний BDI-агент

🤖 Ідеально для автономних БПЛА:
- Цілеспрямована поведінка
- Адаптація до змін
- Пріоритезація (Safety > Mission)

Лекція 24 (продовження)

BDI: Реалізація на Rust

Повний BDI-агент БПЛА

BeliefBase • PlanLibrary • Deliberation • Execution

🤖 Production-ready BDI-агент для симуляції

Частина 2: Практична реалізація

О. С. Бичков • 2025

План (Частина 2)

1. Структура проєкту
2. Core types
3. BeliefBase реалізація
4. Desire та Goal types
5. Intention та Plan
6. PlanLibrary
7. BDI Agent struct
8. Belief Revision
9. Deliberation Engine
10. Plan Execution
11. 🤖 BDI Drone Agent
12. 🤖 Patrol Plan
13. 🤖 Combat Plan
14. 🤖 Integration
15. Testing BDI
16. Підсумок

```
bdi_agent/
├── Cargo.toml
├── src/
│   ├── main.rs              # Entry point
│   ├── lib.rs               # Public API
│   │
│   ├── beliefs/             # Belief system
│   │   ├── mod.rs
│   │   ├── belief_base.rs   # BeliefBase struct
│   │   ├── perception.rs    # Sensor processing
│   │   └── revision.rs      # Belief revision function
│   │
│   ├── desires/             # Desire/Goal system
│   │   ├── mod.rs
│   │   ├── desire.rs        # Desire enum
│   │   └── priority.rs      # Priority calculation
│   │
│   ├── intentions/          # Intention system
│   │   ├── mod.rs
│   │   ├── intention.rs     # Intention struct
│   │   └── commitment.rs    # Commitment strategies
│   │
│   ├── plans/               # Planning system
│   │   ├── mod.rs
│   │   ├── plan.rs          # Plan struct
│   │   ├── library.rs       # Plan library
│   │   └── execution.rs     # Plan executor
│   │
│   ├── deliberation/        # Deliberation engine
│   │   ├── mod.rs
```

```rust
// src/lib.rs - Core types

use glam::Vec3;
use std::time::Instant;

/// Унікальний ідентифікатор агента
#[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
pub struct AgentId(pub u64);

/// Позиція в 3D просторі
#[derive(Debug, Clone, Copy, Default)]
pub struct Position(pub Vec3);

/// Ідентифікатор цілі
#[derive(Debug, Clone, Copy, PartialEq, Eq, Hash)]
pub struct TargetId(pub u64);

/// Рівень загрози
#[derive(Debug, Clone, Copy, PartialEq, Eq, PartialOrd, Ord)]
pub enum ThreatLevel {
    None,
    Low,
    Medium,
    High,
    Critical,
}

/// Результат виконання дії
#[derive(Debug, Clone)]
pub enum ActionResult {
```

```rust
// src/beliefs/belief_base.rs

use std::collections::HashMap;

/// База переконань агента
#[derive(Debug, Clone)]
pub struct BeliefBase {
    // === Власний стан ===
    pub self_position: Position,
    pub self_velocity: Vec3,
    pub battery_level: u8,
    pub health: i32,
    pub is_armed: bool,

    // === Середовище ===
    pub known_targets: HashMap<TargetId, TargetBelief>,
    pub known_obstacles: Vec<ObstacleBelief>,
    pub known_agents: HashMap<AgentId, AgentBelief>,
    pub no_fly_zones: Vec<Area>,

    // === Місія ===
    pub mission_area: Option<Area>,
    pub waypoints: Vec<Position>,
    pub current_waypoint_index: usize,
    pub base_position: Position,

    // === Метадані ===
    pub last_update: Instant,
    pub confidence: f32,  // 0.0 - 1.0
}
```

```
impl BeliefBase {
    pub fn new(start_position: Position, base: Position) -> Self {
        BeliefBase {
            self_position: start_position,
            self_velocity: Vec3::ZERO,
            battery_level: 100,
            health: 100,
            is_armed: true,
            known_targets: HashMap::new(),
            known_obstacles: Vec::new(),
            known_agents: HashMap::new(),
            no_fly_zones: Vec::new(),
            mission_area: None,
            waypoints: Vec::new(),
            current_waypoint_index: 0,
            base_position: base,
            last_update: Instant::now(),
            confidence: 1.0,
        }
    }

    /// Перевірка умови
    pub fn check(&self, condition: &Condition) -> bool {
        match condition {
            Condition::BatteryAbove(level) => self.battery_level > *level,
            Condition::BatteryBelow(level) => self.battery_level < *level,
            Condition::TargetInRange(range) => {
                self.known_targets.values()
                    .any(|t| self.self_position.distance_to(&t.position) < *range)
            }
```

```rust
// src/desires/desire.rs

/// Типи бажань агента
#[derive(Debug, Clone, PartialEq)]
pub enum Desire {
    // === Achieve Goals ===
    ReachPosition(Position),
    DestroyTarget(TargetId),
    CompletePatrol,
    ReturnToBase,

    // === Maintain Goals ===
    MaintainAltitude(f32),
    KeepDistance(AgentId, f32),
    StayInArea(Area),

    // === Avoid Goals ===
    AvoidPosition(Position, f32),  // position, radius
    AvoidCollision,
    EvadeThreats,

    // === Query Goals ===
    ScanArea(Area),
    LocateTarget(TargetId),
}

/// Бажання з пріоритетом
#[derive(Debug, Clone)]
pub struct PrioritizedDesire {
    pub desire: Desire,
```

```rust
// src/desires/priority.rs

impl PrioritizedDesire {
    pub fn new(desire: Desire, beliefs: &BeliefBase) -> Self {
        let (priority, category) = calculate_priority(&desire, beliefs);
        PrioritizedDesire {
            desire,
            priority,
            category,
            deadline: None,
        }
    }
}

fn calculate_priority(desire: &Desire, beliefs: &BeliefBase) -> (u8, DesireCategory) {
    match desire {
        // Safety - найвищий пріоритет
        Desire::AvoidCollision => (255, DesireCategory::Safety),
        Desire::EvadeThreats => {
            let threat_level = beliefs.known_targets.values()
                .map(|t| t.threat_level)
                .max()
                .unwrap_or(ThreatLevel::None);
            let priority = match threat_level {
                ThreatLevel::Critical => 250,
                ThreatLevel::High => 230,
                ThreatLevel::Medium => 200,
                _ => 150,
            };
            (priority, DesireCategory::Safety)
```

```rust
// src/intentions/intention.rs

use crate::plans::Plan;

/// Намір агента (Desire + Plan + Status)
#[derive(Debug, Clone)]
pub struct Intention {
    /// Унікальний ID
    pub id: IntentionId,

    /// Бажання яке реалізуємо
    pub goal: Desire,

    /// План досягнення
    pub plan: Plan,

    /// Поточний крок плану
    pub current_step: usize,

    /// Статус виконання
    pub status: IntentionStatus,

    /// Умови при яких скидаємо intention
    pub drop_conditions: Vec<Condition>,

    /// Час створення
    pub created_at: Instant,

    /// Пріоритет (наслідується від desire)
    pub priority: u8,
```

```
impl Intention {
    pub fn new(goal: Desire, plan: Plan, priority: u8) -> Self {
        static COUNTER: std::sync::atomic::AtomicU64 = std::sync::atomic::AtomicU64::new(0);

        Intention {
            id: IntentionId(COUNTER.fetch_add(1, std::sync::atomic::Ordering::Relaxed)),
            goal,
            plan,
            current_step: 0,
            status: IntentionStatus::Active,
            drop_conditions: Vec::new(),
            created_at: Instant::now(),
            priority,
        }
    }

    /// Чи має бути скинутий намір
    pub fn should_drop(&self, beliefs: &BeliefBase) -> bool {
        self.drop_conditions.iter().any(|c| beliefs.check(c))
    }

    /// Чи завершено виконання
    pub fn is_finished(&self) -> bool {
        matches!(self.status,
            IntentionStatus::Succeeded |
            IntentionStatus::Failed |
            IntentionStatus::Dropped
        )
    }
```

```rust
// src/plans/plan.rs

/// План досягнення цілі
#[derive(Debug, Clone)]
pub struct Plan {
    pub name: String,
    pub steps: Vec<PlanStep>,
}

/// Крок плану
#[derive(Debug, Clone)]
pub enum PlanStep {
    /// Примітивна дія
    Action(Action),

    /// Підціль (рекурсивне планування)
    Subgoal(Desire),

    /// Умовне виконання
    IfThen {
        condition: Condition,
        then_step: Box<PlanStep>,
        else_step: Option<Box<PlanStep>>,
    },

    /// Цикл
    While {
        condition: Condition,
        body: Box<PlanStep>,
    },
```

```rust
// src/plans/library.rs

/// Шаблон плану
#[derive(Debug, Clone)]
pub struct PlanTemplate {
    pub name: String,
    pub trigger: Desire,
    pub context: Vec<Condition>,
    pub body: Vec<PlanStep>,
}

/// Бібліотека планів
pub struct PlanLibrary {
    templates: Vec<PlanTemplate>,
}

impl PlanLibrary {
    pub fn new() -> Self {
        PlanLibrary { templates: Vec::new() }
    }

    pub fn add_template(&mut self, template: PlanTemplate) {
        self.templates.push(template);
    }

    /// Знайти план для бажання
    pub fn find_plan(&self, desire: &Desire, beliefs: &BeliefBase) -> Option<Plan> {
        self.templates.iter()
            .filter(|t| t.matches_trigger(desire))
            .filter(|t| t.context_satisfied(beliefs))
```

```rust
// src/deliberation/engine.rs

pub struct DeliberationEngine {
    plan_library: PlanLibrary,
}

impl DeliberationEngine {
    pub fn new(plan_library: PlanLibrary) -> Self {
        DeliberationEngine { plan_library }
    }

    /// Генерація можливих бажань
    pub fn generate_options(&self, beliefs: &BeliefBase) -> Vec<PrioritizedDesire> {
        let mut desires = Vec::new();

        // Завжди: avoid collision
        desires.push(PrioritizedDesire::new(Desire::AvoidCollision, beliefs));

        // Low battery -> return to base
        if beliefs.battery_level < 20 {
            desires.push(PrioritizedDesire::new(Desire::ReturnToBase, beliefs));
        }

        // Threats detected -> evade
        if beliefs.known_targets.values().any(|t| t.threat_level >= ThreatLevel::High) {
            desires.push(PrioritizedDesire::new(Desire::EvadeThreats, beliefs));
        }

        // Mission waypoints -> patrol
        if !beliefs.waypoints.is_empty() {
```

```rust
// src/agent/mod.rs

pub struct BDIAgent {
    pub id: AgentId,
    pub beliefs: BeliefBase,
    pub desires: Vec<PrioritizedDesire>,
    pub intentions: Vec<Intention>,

    deliberation: DeliberationEngine,
    commitment_strategy: CommitmentStrategy,

    // Метрики
    tick_count: u64,
    last_deliberation: Instant,
}

#[derive(Debug, Clone, Copy)]
pub enum CommitmentStrategy {
    Blind,              // Ніколи не переглядає
    SingleMinded,       // До успіху або неможливості
    OpenMinded,         // Часто переглядає
}

impl BDIAgent {
    pub fn new(
        id: AgentId,
        start_pos: Position,
        base: Position,
        plan_library: PlanLibrary,
    ) -> Self {
```

```rust
impl BDIAgent {
    /// Головний цикл BDI агента
    pub fn tick(&mut self, perception: Perception) -> Vec<Action> {
        self.tick_count += 1;

        // 1. Belief Revision
        self.update_beliefs(perception);

        // 2. Option Generation
        self.desires = self.deliberation.generate_options(&self.beliefs);

        // 3. Review current intentions
        self.review_intentions();

        // 4. Deliberate (if needed)
        if self.should_deliberate() {
            if let Some(new_intention) = self.deliberation.deliberate(
                &self.desires,
                &self.intentions,
                &self.beliefs,
            ) {
                self.intentions.push(new_intention);
            }
            self.last_deliberation = Instant::now();
        }

        // 5. Execute intentions
        self.execute_intentions()
    }
```

```
impl BDIAgent {
    fn update_beliefs(&mut self, perception: Perception) {
        // Оновлюємо власний стан
        self.beliefs.self_position = perception.position;
        self.beliefs.self_velocity = perception.velocity;
        self.beliefs.battery_level = perception.battery;
        self.beliefs.health = perception.health;

        // Оновлюємо інформацію про цілі
        for detected_target in perception.detected_targets {
            self.beliefs.known_targets.insert(
                detected_target.id,
                TargetBelief {
                    id: detected_target.id,
                    position: detected_target.position,
                    velocity: detected_target.velocity,
                    threat_level: detected_target.threat_level,
                    last_seen: Instant::now(),
                    confidence: 1.0,
                },
            );
        }

        // Зменшуємо confidence для старих даних
        let now = Instant::now();
        self.beliefs.known_targets.retain(|_, target| {
            let age = now.duration_since(target.last_seen).as_secs_f32();
            target.confidence = (1.0 - age / 30.0).max(0.0);
            target.confidence > 0.1
        });
```

```
impl BDIAgent {
    fn review_intentions(&mut self) {
        for intention in &mut self.intentions {
            if intention.is_finished() {
                continue;
            }

            // Перевіряємо drop conditions
            if intention.should_drop(&self.beliefs) {
                intention.status = IntentionStatus::Dropped;
                tracing::debug!(
                    intention = ?intention.id,
                    "Intention dropped due to conditions"
                );
                continue;
            }

            // Перевіряємо чи ціль досягнута
            if self.goal_achieved(&intention.goal) {
                intention.status = IntentionStatus::Succeeded;
                tracing::info!(
                    intention = ?intention.id,
                    goal = ?intention.goal,
                    "Intention succeeded"
                );
                continue;
            }

            // Перевіряємо чи ціль все ще можлива
            if !self.goal_possible(&intention.goal) {
```

```rust
impl BDIAgent {
    fn execute_intentions(&mut self) -> Vec<Action> {
        let mut actions = Vec::new();

        // Сортуємо intentions за пріоритетом
        self.intentions.sort_by(|a, b| b.priority.cmp(&a.priority));

        // Виконуємо найпріоритетніший активний intention
        if let Some(intention) = self.intentions.iter_mut()
            .find(|i| i.status == IntentionStatus::Active)
        {
            if let Some(step) = intention.plan.steps.get(intention.current_step) {
                match self.execute_step(step) {
                    StepResult::Action(action) => {
                        actions.push(action);
                    }
                    StepResult::Complete => {
                        intention.advance();
                    }
                    StepResult::InProgress => {
                        // Продовжуємо виконання
                    }
                    StepResult::Failed(reason) => {
                        intention.status = IntentionStatus::Failed;
                        tracing::error!(reason, "Step failed");
                    }
                }
            }
        }
```

```
// Створення плану патрулювання
pub fn create_patrol_plan() -> PlanTemplate {
    PlanTemplate {
        name: "patrol_waypoints".to_string(),
        trigger: Desire::CompletePatrol,
        context: vec![
            Condition::BatteryAbove(20),
            Condition::HasWaypoints,
        ],
        body: vec![
            // Піднятись на робочу висоту
            PlanStep::Action(Action::SetAltitude(100.0)),

            // Цикл по waypoints
            PlanStep::While {
                condition: Condition::HasWaypoints,
                body: Box::new(PlanStep::Sequence(vec![
                    // Летіти до поточного waypoint
                    PlanStep::Action(Action::MoveToWaypoint),

                    // Сканувати область
                    PlanStep::Action(Action::Scan(100.0)),

                    // Якщо виявлено ціль - повідомити
                    PlanStep::IfThen {
                        condition: Condition::TargetDetected,
                        then_step: Box::new(PlanStep::Action(
                            Action::Report("Target detected".to_string())
                        )),
                        else_step: None,
```

```
pub fn create_attack_plan() -> PlanTemplate {
    PlanTemplate {
        name: "attack_target".to_string(),
        trigger: Desire::DestroyTarget(TargetId(0)),  // Wildcard
        context: vec![
            Condition::BatteryAbove(30),
            Condition::IsArmed,
            Condition::TargetInRange(500.0),
        ],
        body: vec![
            // 1. Наблизитись до цілі
            PlanStep::Action(Action::ApproachTarget(200.0)),

            // 2. Цикл атаки
            PlanStep::While {
                condition: Condition::TargetAlive,
                body: Box::new(PlanStep::Sequence(vec![
                    // Прицілитись
                    PlanStep::Action(Action::AimAtTarget),

                    // Перевірка загроз
                    PlanStep::IfThen {
                        condition: Condition::UnderFire,
                        then_step: Box::new(PlanStep::Sequence(vec![
                            PlanStep::Action(Action::Evade),
                            PlanStep::Action(Action::Wait(2.0)),
                        ])),
                        else_step: Some(Box::new(
                            PlanStep::Action(Action::FireAtTarget)
                        )),
```

```rust
fn main() {
    tracing_subscriber::fmt::init();

    // Створюємо бібліотеку планів
    let mut plan_library = PlanLibrary::new();
    plan_library.add_template(create_patrol_plan());
    plan_library.add_template(create_attack_plan());
    plan_library.add_template(create_retreat_plan());
    plan_library.add_template(create_evade_plan());

    // Створюємо агента
    let mut agent = BDIAgent::new(
        AgentId(1),
        Position(Vec3::new(0.0, 0.0, 50.0)),
        Position(Vec3::ZERO),
        plan_library,
    );

    // Встановлюємо waypoints для патрулювання
    agent.beliefs.waypoints = vec![
        Position(Vec3::new(100.0, 0.0, 50.0)),
        Position(Vec3::new(100.0, 100.0, 50.0)),
        Position(Vec3::new(0.0, 100.0, 50.0)),
        Position(Vec3::new(0.0, 0.0, 50.0)),
    ];

    // Симуляція
    for tick in 0..1000 {
        let perception = get_sensor_data(&agent);
        let actions = agent.tick(perception);
```

```rust
#[cfg(test)]
mod tests {
    use super::*;

    #[test]
    fn test_deliberation_low_battery() {
        let mut agent = create_test_agent();
        agent.beliefs.battery_level = 15;

        let desires = agent.deliberation.generate_options(&agent.beliefs);

        // ReturnToBase має бути високопріоритетним
        let return_desire = desires.iter()
            .find(|d| matches!(d.desire, Desire::ReturnToBase));
        assert!(return_desire.is_some());
        assert!(return_desire.unwrap().priority > 200);
    }

    #[test]
    fn test_intention_creation() {
        let mut agent = create_test_agent();
        agent.beliefs.waypoints = vec![Position(Vec3::new(100.0, 0.0, 0.0))];

        let perception = Perception::default();
        agent.tick(perception);

        assert!(!agent.intentions.is_empty());
        assert!(matches!(agent.intentions[0].goal, Desire::CompletePatrol));
    }
```

Performance Tips

Оптимізація BDI агента:

✅ Deliberation frequency
Не deliberate кожен tick
Використовуйте Single-minded commitment

✅ Plan caching
Кешуйте знайдені плани
Інвалідуйте при зміні beliefs

✅ Lazy evaluation
Обчислюйте пріоритети тільки для топ-N desires

✅ Belief indexing
HashMap для швидкого пошуку targets
Spatial index для позицій

❌ Уникайте:
- Глибоких рекурсій в планах
- Занадто частого перегляду intentions
- Великої кількості drop conditions

Підсумок лекції

BDI реалізація на Rust:

- BeliefBase — структура з усіма знаннями
- PrioritizedDesire — бажання з пріоритетом
- Intention — desire + plan + status
- PlanLibrary — бібліотека шаблонів
- DeliberationEngine — вибір intentions

Головний цикл:
1. Perception → Belief Revision
2. Generate Options (desires)
3. Review Intentions
4. Deliberate (if needed)
5. Execute

🤖 Плани для БПЛА:
- Patrol — патрулювання waypoints
- Attack — атака цілі
- Retreat — повернення на базу

Завдання

1. Базове (2 години):
   Реалізувати BeliefBase та Desire enum
   Тести для priority calculation

2. Середнє (4 години):
   Повний BDI цикл
   3 плани: patrol, return, hover
   Логування рішень

3. Складне (8 годин):
   Multi-agent BDI система
   Комунікація між агентами
   Координація через beliefs

4. Challenge (16 годин):
   Гібридна архітектура:
   • Reactive layer для уникнення
   • BDI layer для планування
   Візуалізація decision tree

BDI архітектуру опановано!

Beliefs • Desires • Intentions • Plans

Питання?