

DAIIA Assignment Report

Course: Distributed Artificial Intelligence and Intelligent Agents

Assignment: Final Project - Mars Colony: The Evolution of Trust

Lorenzo Deflorian, Riccardo Fragale, Juozas Skarbalius

KTH Royal Institute of Technology

January 5, 2026

Contents

1	Running Instructions	2
2	General Overview	2
3	Agent Types and Interactions	3
4	Environment and Places	5
5	Continuous Running System	9
6	FIPA Communication	12
7	Charts and Monitoring	14
8	Challenge 1: BDI Agents	17
9	Challenge 2: Reinforcement Learning	22
10	Final Remarks	28

1. Running Instructions

- **How to run the program:**
 - Import the project files into GAMA platform
 - Navigate to the folder `src/project/models`
 - Open the file `MarsColony.gaml`
 - From the GAMA interface, click the play button to start the simulation
 - Use the simulation controls to adjust speed, pause, or reset as needed
- **Expected inputs:** No manual inputs are required. The simulation runs automatically with predefined parameters that can be modified in the global section if desired (e.g., desired number of agents per type, map dimensions, learning parameters).

2. General Overview

Solution Summary

This project presents a Mars Colony simulation with five agent roles (Engineer, Medic, Scavenger, Parasite, and Commander). Agents must survive in a resource-constrained environment by managing biological needs (oxygen, energy, health) and interacting with other agents.

The main focus is whether decentralized learning can reduce harmful interactions over time. Agents use a BDI (Belief-Desire-Intention) controller for survival behaviors and Q-Learning to guide social decisions (e.g., whether to trade with a specific partner).

Key Features:

- Five distinct agent types with unique roles and behaviors
- Continuous simulation with supply shuttle system maintaining population
- BDI architecture for survival-oriented decision making
- Q-Learning with adaptive curiosity for trust-based parasite detection
- Facility replenishment system (greenhouse, oxygen generator) for survival alternatives
- FIPA communication for long-distance messaging (storm warnings)
- Real-time learning metrics visualization (trust, precision, recall)
- Behavioral metrics visualization (sociability, happiness, generosity)
- Survival metrics visualization (agent lifespan tracking)

Current Implementation State:

- Death system is enabled with resource-based pressure enabling agent selection
- Oxygen and energy refill plans are enabled with slow facility replenishment (backup survival path)
- Trading remains the faster/better survival strategy than facilities alone
- Maximum colony size is capped at 60 agents
- Agents spawn at the habitat dome location
- Q-Learning uses exponential moving average (sociability = 0.35) for faster convergence
- Adaptive curiosity scales from 5% to 50% based on average trust
- Adaptive sociability recovery prevents learning stagnation

3. Agent Types and Interactions

3.1 Explanation. The model includes five agent types with different roles, capabilities, and interaction rules. All agents share three core biological traits: `oxygen_level`, `energy_level`, and `health_level`, which decrease over time and constrain survival. Each agent type follows specific rules that determine how it trades resources with other agents.

Survival Mechanisms: Agents can sustain themselves through two paths: (1) Trading with other agents for immediate resource boosts, or (2) Using facility replenishment systems (Greenhouse and OxygenGenerator) for slower but reliable resource restoration. Facilities provide +0.5 energy/cycle and +0.3 oxygen/cycle within their proximity radius, creating a survival baseline while trading provides faster growth opportunities.

Agent Types:

1. **Engineer:** Repairs broken oxygen generators via BDI plan. Provides oxygen (+10) during trades.
2. **Medic:** Heals other agents at the med-bay via a queue system. (Healing is performed at the Med-Bay, not through trading.)
3. **Scavenger:** Ventures into wasteland to mine resources. Provides raw materials (converted to +20 energy) during trades.
4. **Parasite:** Antagonist agent that steals resources without reciprocating. Uses `presented_role` to disguise as other types.
5. **Commander:** Broadcasts storm warnings via FIPA. Provides both oxygen (+10) and energy (+10) during trades.

Each agent type has its own interaction logic. For example, Engineers repair generators, Medics heal at the med-bay, Scavengers mine resources, Parasites steal, and Commanders broadcast alerts.

3.2 Code. The agent types are defined as species inheriting from the `Human` base species. Each agent initializes with its role and presented role:

```

species Engineer parent: Human {
    init {
        do add_desire(wander_desire);
        role <- "Engineer";
        presented_role <- "Engineer";
    }
    // ... Engineer-specific behaviors
}

species Parasite parent: Human {
    init {
        do add_desire(wander_desire);
        role <- "Parasite";
        presented_role <- one_of(["Engineer", "Medic", "Scavenger", "Commander"]);
    }
}

```

The trading mechanism is implemented in the `attempt_trade` action, which handles different behaviors based on agent types:

```

action attempt_trade(Human partner) {
    bool i_gave <- false;
    bool partner_gave <- false;

    if (!(self is Parasite)) {
        if (self is Scavenger and raw_amount > 0) {
            raw_amount <- raw_amount - 1;
            i_gave <- true;
            ask partner { energy_level <- min(max_energy_level, energy_level + 20.0);
        }
        // ... other agent type behaviors
    }

    if (partner is Parasite) {
        partner_gave <- false; // Parasites never give back
    }

    if (self is Parasite) {
        // Parasite performs harmful, one-sided interaction
        i_gave <- true;
        partner_gave <- false;
        // ... stealing logic
    }
}

```

```

if (i_gave and partner_gave) { return 100.0; } // Strong reward for mutual cooperation
if (i_gave and !partner_gave) {
    if (partner is Parasite) { return -80.0; } // Victim strongly penalized for being exploited
    if (self is Parasite) { return -80.0; } // Parasite strongly penalized for being ungenerous
    return 20.0; // Non-parasite helpfully gave, good reward for altruism
}
return 0.0; // Neutral outcome for non-interaction
}

```

3.3 Demonstration.

1. **Input:** Simulation starts with all five agent types present in the habitat dome.

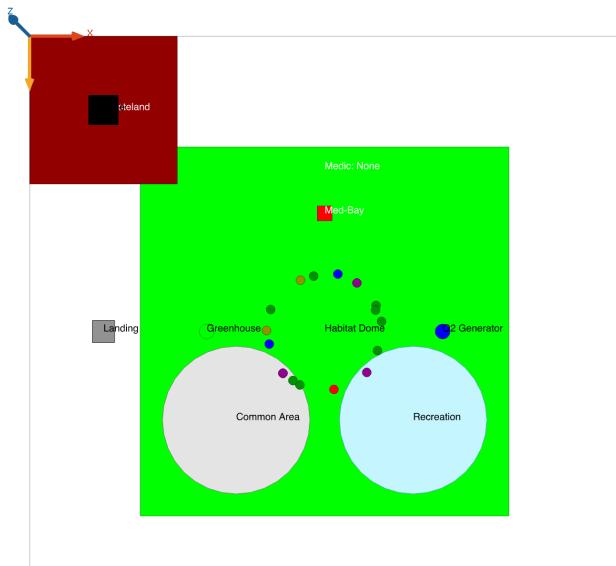


Figure 1: Agents inside the map

2. **Interpretation:** The simulation starts with all five agent types present in the habitat dome. Agents are rendered with distinct visuals and begin moving immediately.
3. **Input:** Agents engage in trading interactions within the habitat dome.
4. **Screenshot:** 2
5. **Interpretation:** As can be seen from 2 trades follow the intended type-specific rules: Scavengers provide energy via raw resources, Engineers provide oxygen/repairs, Commanders provide oxygen+energy, and Parasites attempt one-sided interactions. Medic contributions occur through the Med-Bay queue (not via trade).

4. Environment and Places

4.1 Explanation. The environment is organized into distinct locations where agents move, interact, and perform activities. It includes a safe zone (habitat dome) and a hazard zone (wasteland), each with different effects on survival.

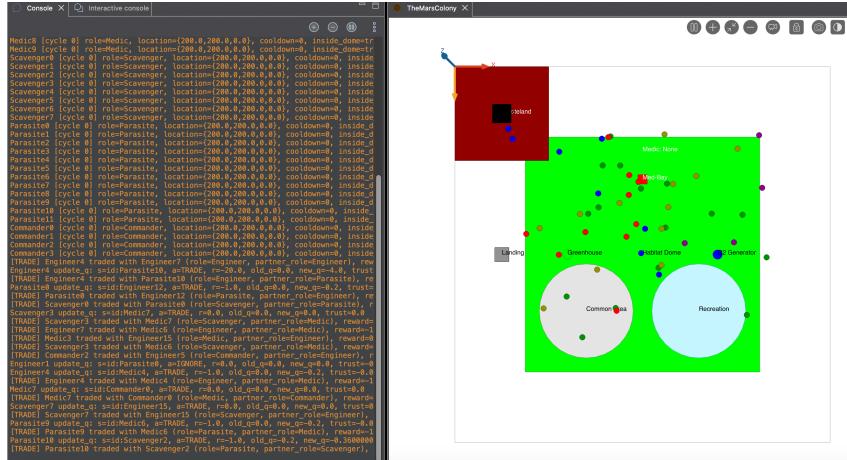


Figure 2: Trading between agents

Places:

1. **Habitat Dome:** Large safe zone (250x250 units) containing Greenhouse, Oxygen Generator, Med-Bay, and the two meeting places used for social interaction.
2. **Common Area:** Meeting/trading area (circle) inside the dome.
3. **Recreation Area:** Meeting/trading area (circle) inside the dome.
4. **Wasteland:** Danger zone (100x100 units) where Scavengers mine. Oxygen depletes 1.2x faster, and random dust storms increase risk.
5. **Med-Bay:** Facility within the dome where injured agents queue for treatment. The queue status is shown visually (orange when patients are waiting).
6. **Landing Pad:** Location where agents retire and despawn.
7. **Rock Mine:** Location in wasteland where Scavengers collect raw materials.

4.2 Code. The places are defined as species with specific geometries and behaviors:

```

species HabitatDome {
    geometry shape <- rectangle(250, 250);
    Greenhouse greenhouse;
    OxygenGenerator oxygen_generator;
    MedBay med_bay;
    CommonArea common_area;
    RecreationArea recreation_area;

    init {
        location <- point(200, 200);
        shape <- shape at_location location;
        // Create and position facilities
        create Greenhouse number: 1 returns: greenhouses;
    }
}
```

```

greenhouse <- greenhouses[0];
// ... initialize other facilities
}

}

species Wasteland {
    geometry shape <- rectangle(100, 100);
    bool dust_storm <- false;
    int storm_timer <- 0;

    reflex manage_storm {
        if (dust_storm) {
            storm_timer <- storm_timer - 1;
            if (storm_timer <= 0) {
                dust_storm <- false;
            }
        } else {
            if (flip(0.01)) {
                dust_storm <- true;
                storm_timer <- 15;
            }
        }
    }
}

species Greenhouse {
    point location;
    float replenish_rate <- 0.5; // Energy restored per cycle

    reflex replenish_energy {
        list<Human> nearby_agents <- Human where
            (each.location distance_to location <= facility_proximity);
        loop h over: nearby_agents {
            ask h {
                if (energy_level < max_energy_level) {
                    energy_level <- min(max_energy_level,
                                         energy_level + myself.replenish_rate);
                }
            }
        }
    }
}

species OxygenGenerator {
    point location;
    bool is_broken <- false;
    float replenish_rate <- 0.3; // Oxygen restored per cycle
}

```

```

reflex replenish_oxygen when: not is_broken {
    list<Human> nearby_agents <- Human where
        (each.location distance_to location <= facility_proximity);
    loop h over: nearby_agents {
        ask h {
            if (oxygen_level < max_oxygen_level) {
                oxygen_level <- min(max_oxygen_level,
                    oxygen_level + myself.replenish_rate);
            }
        }
    }
}

```

Oxygen depletion rates differ based on location:

```

reflex update_oxygen{
    if (habitat_dome.shape covers location) {
        oxygen_level <- max(0, oxygen_level - oxygen_decrease_rate);
    } else {
        float decrease <- oxygen_decrease_rate * oxygen_decrease_factor_in_wasteland;
        if (wasteland.dust_storm and (wasteland.shape covers location)) {
            decrease <- decrease * 2.0;
        }
        oxygen_level <- max(0, oxygen_level - decrease);
    }
}

```

4.3 Demonstration.

1. **Input:** Simulation displays the full map with all places visible.
2. **Screenshot:**

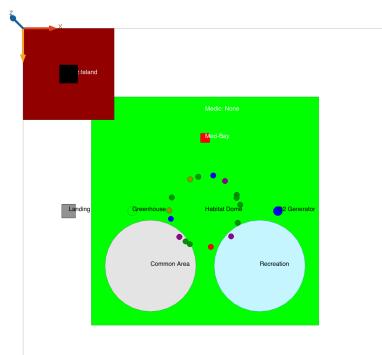


Figure 3: Mars colony map

3. **Interpretation:** All places are initialized and positioned as expected. The habitat dome functions as the primary safe zone, while the wasteland introduces harsher environmental conditions.

4. **Input:** A dust storm occurs in the wasteland.

5. **Screenshot:**

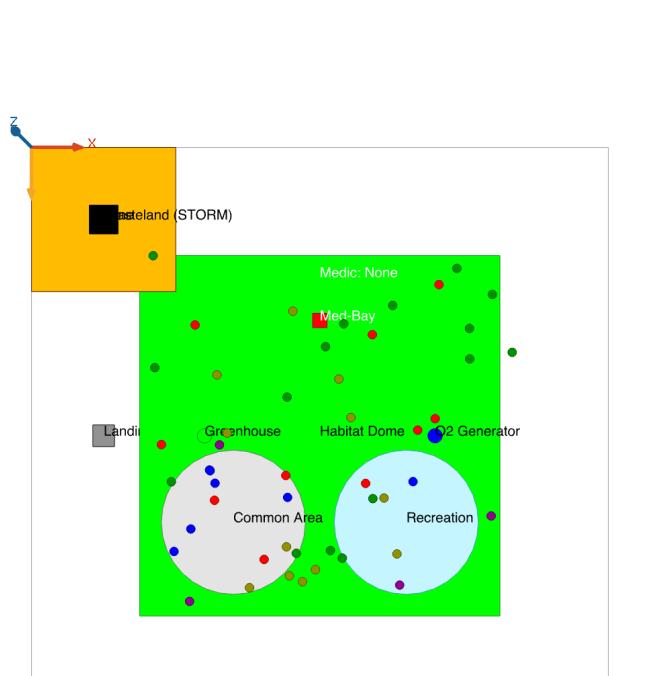


Figure 4: Storm in the Wasteland

6. **Interpretation:** During a dust storm, the wasteland changes color to provide immediate visual feedback.

5. Continuous Running System

5.1 Explanation. The simulation runs continuously by maintaining a target population through a supply shuttle system. When agents retire (after 2000 cycles) or die (when health reaches 0), new agents are spawned to maintain desired population levels. The system maintains at least 50 agents total across all types, with a maximum cap of 60 agents. Death events are tracked for survival metrics, including average lifespan of deceased agents.

The supply shuttle operates in "deficit mode," spawning new agents when any agent type falls below its desired count:

- Engineers: 16 desired
- Medics: 10 desired
- Scavengers: 8 desired

- Parasites: 12 desired
- Commanders: 4 desired

Additionally, the supply shuttle aggregates Q-tables and trust memory from all survivors to calculate averages (preparing for potential knowledge inheritance by new agents).

5.2 Code. The supply shuttle reflex checks population counts and spawns new agents:

```
reflex supply_shuttle when: enable_supply_shuttle {
    int total_colonists <- length(list(Engineer) + list(Medic) +
                                list(Scavenger) + list(Parasite) +
                                list(Commander));

    bool deficit_mode <- (current_number_of_engineers < desired_number_of_engineers) ||
                           (current_number_of_medics < desired_number_of_medics) or
                           (current_number_of_scavengers < desired_number_of_scavengers) or
                           (current_number_of_parasites < desired_number_of_parasites) or
                           (current_number_of_commanders < desired_number_of_commanders);

    if (deficit_mode) {
        int max_to_spawn <- max_colony_size - total_colonists;

        // Spawn engineers if needed
        if (delta_engineers > 0 and max_to_spawn > 0) {
            int to_spawn <- min(delta_engineers, max_to_spawn);
            create Engineer number: to_spawn returns: new_engineers;
            ask new_engineers {
                location <- habitat_dome.location;
                oxygen_level <- max_oxygen_level;
                energy_level <- max_energy_level;
            }
            engineers <- engineers + new_engineers;
            current_number_of_engineers <- current_number_of_engineers + to_spawn;
            max_to_spawn <- max_to_spawn - to_spawn;
        }
        // ... similar logic for other agent types
    }
}
```

Retirement is handled through BDI:

```
reflex update_eta {
    if (not enable_retirement) { return; }
    eta <- eta + eta_increment;
    if (eta >= retirement_age) {
        do add_belief(should_retire_belief);
```

```

        }
    }

plan do_retire intention: retire_desire {
    state <- "retiring";
    do goto target: landing_pad.location speed: movement_speed;

    if ((location distance_to landing_pad.location) <= facility_proximity) {
        do die_and_update_counter;
    }
}

```

5.3 Demonstration.

1. **Input:** Simulation runs for an extended period, allowing agents to reach retirement age (2000 cycles).
2. **Screenshot:**

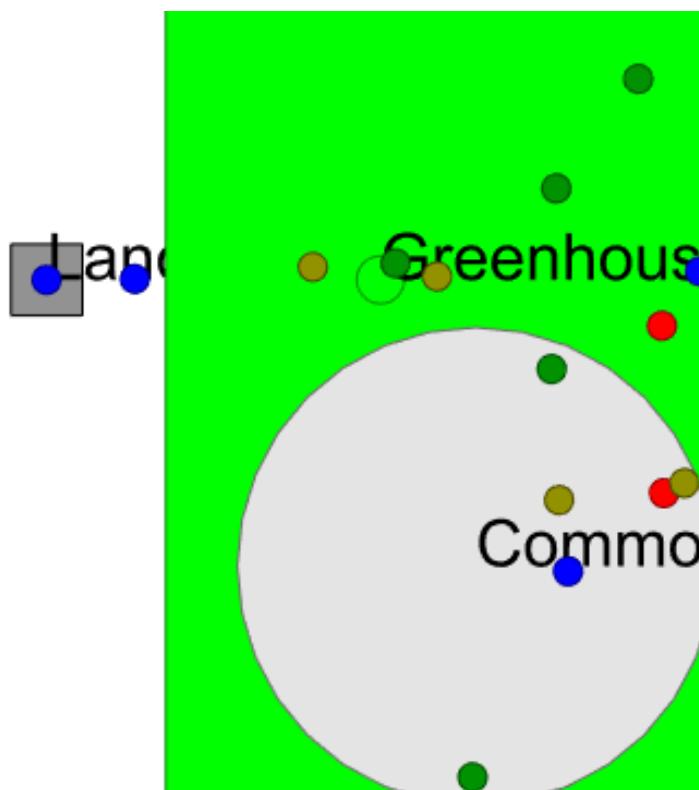


Figure 5: Agents going to the landing pad after retirement

3. **Interpretation:** The retirement system tracks agent age and triggers retirement when the threshold is reached. Retiring agents move to the landing pad and despawn.
4. **Input:** Agents retire, causing population counts to drop below desired levels.
5. **Screenshot:**



Figure 6: Supply shuttle arriving with new agents

6. **Interpretation:** When population counts fall below the targets, the supply shuttle spawns new agents to restore the desired levels. This keeps the simulation running without manual intervention.

6. FIPA Communication

6.1 Explanation. FIPA (Foundation for Intelligent Physical Agents) messaging is used for long-distance coordination. In this simulation, Commanders send storm warnings to agents in the wasteland when a dust storm is detected. The goal is to model asynchronous communication that can influence behavior at a distance.

When a dust storm occurs in the wasteland, Commanders detect it and broadcast a "Return to Base" message using the FIPA propose protocol. Agents receiving this message add a `storm_warning_belief` to their BDI system, which triggers the highest-priority desire (`escape_storm`) to return to the safe habitat dome.

6.2 Code. Commanders monitor for storms and send FIPA messages:

```
species Commander parent: Human {
    reflex check_storm {
        if (wasteland.dust_storm) {
            list<Human> agents_in_wasteland <- Human where
                (wasteland.shape covers each.location);
            if (!empty(agents_in_wasteland)) {
                do start_conversation to: agents_in_wasteland
                    protocol: "fipa-propose"
                    performative: "propose"
                    contents: ["Return to Base"];
            }
        }
    }
}
```

Agents receive and process FIPA messages:

```
reflex receive_message when: !empty(mailbox) {
    message msg <- first(mailbox);
```

```

mailbox <- mailbox - msg;
string msg_contents <- string(msg.contents);
if (msg_contents contains "Return to Base") {
    if (not has_belief(storm_warning_belief)) {
        do add_belief(storm_warning_belief);
    }
}
}

```

The BDI system processes the belief:

```

rule belief: storm_warning_belief new_desire: escape_storm_desire strength: 200.0;

plan escape_storm intention: escape_storm_desire {
    state <- "escaping_storm";
    do goto target: habitat_dome.location speed: movement_speed;

    if (habitat_dome.shape covers location) {
        do remove_belief(storm_warning_belief);
        do remove_intention(escape_storm_desire, true);
        state <- "idle";
    }
}

```

6.3 Demonstration.

1. **Input:** A dust storm occurs in the wasteland while agents (particularly Scavengers) are present there.
2. **Screenshot:**

```

[COMMANDER] Commander0 detected dust storm and broadcasting 'Return to Base' to 2 agent(s) in wasteland
[COMMANDER] Commander1 detected dust storm and broadcasting 'Return to Base' to 2 agent(s) in wasteland
[COMMANDER] Commander2 detected dust storm and broadcasting 'Return to Base' to 2 agent(s) in wasteland
[COMMANDER] Commander3 detected dust storm and broadcasting 'Return to Base' to 2 agent(s) in wasteland
[FIPA] Scavenger1 (Scavenger) received 'Return to Base' message from Commander(0)
[FIPA] Scavenger2 (Scavenger) received 'Return to Base' message from Commander(0)
[OXYGEN GENERATOR] Generator broke at cycle 84
[FIPA] Scavenger1 (Scavenger) received 'Return to Base' message from Commander(1)
[FIPA] Scavenger4 (Scavenger) received 'Return to Base' message from Commander(1)
[FIPA] Scavenger1 (Scavenger) received 'Return to Base' message from Commander(2)
[FIPA] Scavenger4 (Scavenger) received 'Return to Base' message from Commander(2)
[FIPA] Scavenger1 (Scavenger) received 'Return to Base' message from Commander(3)
[FIPA] Scavenger4 (Scavenger) received 'Return to Base' message from Commander(3)

```

Figure 7: Agents receiving FIPA "Return to Base" messages during storm

3. **Interpretation:** Commanders detect the storm and broadcast FIPA messages to agents in the wasteland. Receiving agents process the messages and add the storm warning belief.
4. **Input:** Agents receive storm warning messages and begin moving toward the habitat dome.
5. **Screenshot:**

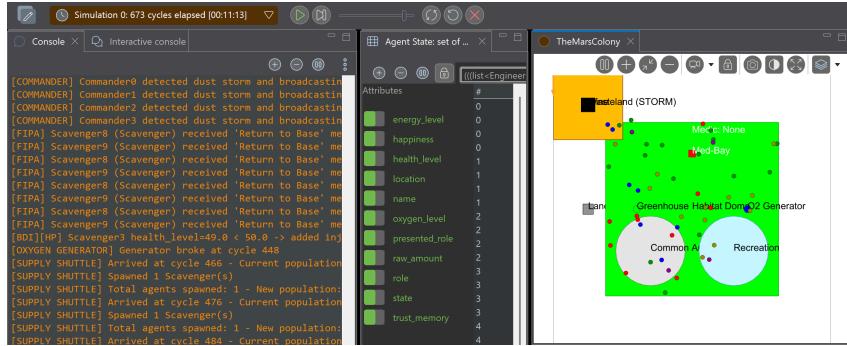


Figure 8: Agents returning to base after receiving storm warning

6. **Interpretation:** The warning message changes agent priorities: agents with a storm warning prioritize escape (strength 200.0) and move back to the habitat dome.

7. Charts and Monitoring

7.1 Explanation. The simulation provides real-time monitoring of learning metrics through charts and global values. The main chart tracks trust and parasite-detection metrics over time.

Global Values and Metrics:

- `avg_trust_to_parasites`: Average trust value agents have toward parasites
- `avg_trust_to_non_parasites`: Average trust value agents have toward cooperative agents
- `precision`: True positives / (True positives + False positives) for parasite detection
- `recall`: True positives / (True positives + False negatives) for parasite detection
- `total_trades`: Counter of total trade interactions
- `avg_sociability`: Average learning-rate parameter across agents (scaled to 0-100 range)
- `avg_happiness`: Average accumulated reward from social interactions
- `avg_generosity`: Average giving/receiving balance tracked during trades
- `avg_living_agent_age`: Average ETA (age) of currently living agents
- `avg_dead_agent_lifespan`: Average lifespan of deceased agents
- `total_deaths`: Total number of agents that have died

The expectation is that trust toward parasites decreases over time while trust toward non-parasites remains positive, and that precision/recall improve as agents accumulate interactions.

Interesting Conclusion: In repeated runs, negative outcomes with parasites reduce trust and increase avoidance. This functions like a decentralized reputation effect: agents do not need a centralized authority to change behavior, but instead adapt based on their own interaction history.

7.2 Code. Learning metrics are updated each cycle:

```

reflex update_learning_metrics {
    int total_agents <- length(list(Engineer) + list(Medic) +
                               list(Scavenger) + list(Parasite) +
                               list(Commander));

    map<string, bool> is_parasite_by_id <- map([]);

    loop h over: (list(Engineer) + list(Medic) + list(Scavenger) +
                  list(Parasite) + list(Commander)) {
        is_parasite_by_id["id:" + h.name] <- (h is Parasite);
    }

    float sum_par <- 0.0; int cnt_par <- 0;
    float sum_non <- 0.0; int cnt_non <- 0;
    int tp0 <- 0; int fp0 <- 0; int fn0 <- 0; int tn0 <- 0;

    float sum_soc <- 0.0;
    float sum_hap <- 0.0;
    float sum_gen <- 0.0;

    loop h over: (list(Engineer) + list(Medic) + list(Scavenger) +
                  list(Parasite) + list(Commander)) {
        sum_soc <- sum_soc + h.sociability;
        sum_hap <- sum_hap + h.happiness;
        sum_gen <- sum_gen + h.generosity;

        loop k over: keys(h.trust_memory) {
            float v <- h.trust_memory[k];
            bool gt_par <- ((is_parasite_by_id contains_key k) ?
                             is_parasite_by_id[k] : false);

            if (gt_par) {
                sum_par <- sum_par + v;
                cnt_par <- cnt_par + 1;
            } else {
                sum_non <- sum_non + v;
                cnt_non <- cnt_non + 1;
            }
        }

        bool pred_par <- v < 0.0;
        if (pred_par and gt_par) { tp0 <- tp0 + 1; }
        else if (pred_par and not gt_par) { fp0 <- fp0 + 1; }
    }
}

```

```

        else if ((not pred_par) and gt_par) { fn0 <- fn0 + 1; }
        else { tn0 <- tn0 + 1; }
    }
}

avg_trust_to_parasites <- (cnt_par > 0 ? sum_par / float(cnt_par) : 0.0);
avg_trust_to_non_parasites <- (cnt_non > 0 ? sum_non / float(cnt_non) : 0.0);
precision <- ((tp0 + fp0) > 0 ? float(tp0) / float(tp0 + fp0) : 0.0);
recall <- ((tp0 + fn0) > 0 ? float(tp0) / float(tp0 + fn0) : 0.0);

avg_sociability <- (total_agents > 0 ? sum_soc / float(total_agents) : 0.0);
avg_happiness <- (total_agents > 0 ? sum_hap / float(total_agents) : 0.0);
avg_generosity <- (total_agents > 0 ? sum_gen / float(total_agents) : 0.0);
}

```

The chart is defined in the experiment:

```

display LearningMetrics {
    chart "Avg Trust & Detection" type: series {
        data 'Avg trust (parasites)' value: avg_trust_to_parasites;
        data 'Avg trust (non-parasites)' value: avg_trust_to_non_parasites;
        data 'Precision' value: precision;
        data 'Recall' value: recall;
    }
}

display BehavioralMetrics {
    chart "Sociability, Happiness & Generosity" type: series {
        data 'Avg Sociability' value: avg_sociability color: \#blue;
        data 'Avg Happiness' value: avg_happiness color: \#green;
        data 'Avg Generosity' value: avg_generosity color: \#orange;
    }
}

display SurvivalMetrics {
    chart "Agent Lifespan & Survival" type: series {
        data 'Avg Living Agent Age' value: avg_living_agent_age color: rgb(0, 102, 204);
        data 'Avg Dead Agent Lifespan' value: avg_dead_agent_lifespan color: rgb(204, 0, 0);
    }
}

```

7.3 Demonstration.

1. **Input:** Simulation runs for an extended period to allow learning to occur.
2. **Interpretation of the figure below:** The learning metrics chart shows how trust and detection accuracy change over time. Precision and recall follow the standard machine learning definitions.

3. Screenshot:

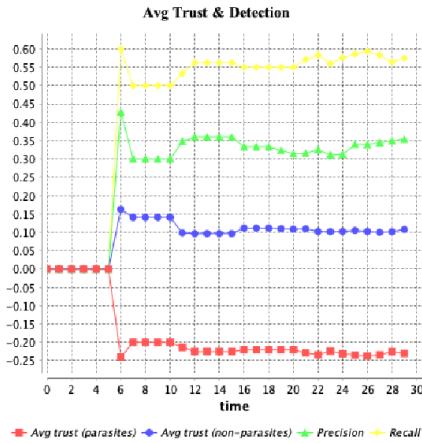


Figure 9: Learning metrics

4. **Input:** Agent state inspector shows trust memory values for different agents.

5. Screenshot:

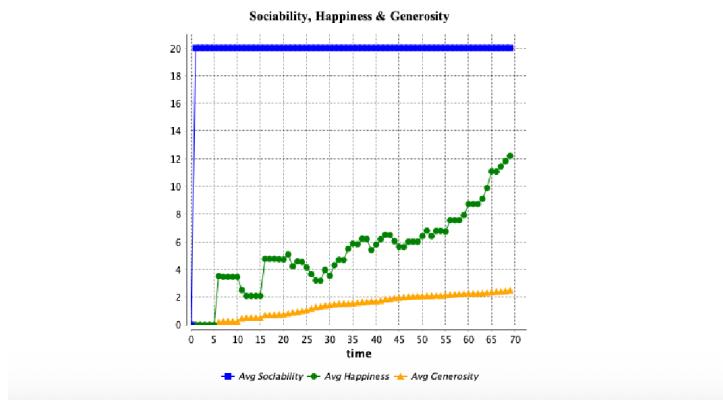


Figure 10: Behavioural metrics map

6. **Interpretation:** Trust memory maps separate parasites (negative trust) from cooperative agents (positive trust), reflecting the cumulative outcomes of trade interactions.

8. Challenge 1: BDI Agents

8.1 Explanation. The simulation implements a full BDI (Belief-Desire-Intention) architecture using GAMA's Simple BDI control. This architecture allows agents to make decisions based on their beliefs about the world, desires (goals) they want to achieve, and intentions (plans) to fulfill those desires.

Beliefs (Sensors): Agents maintain beliefs about their internal state and external environment:

- `suffocating_belief`: Triggered when oxygen level < 20%
- `starving_belief`: Triggered when energy level < 20%
- `injured_belief`: Triggered when health level < 50%
- `should_retire_belief`: Triggered when ETA \geq 2000 cycles
- `storm_warning_belief`: Triggered by FIPA messages from Commander
- `generator_broken_belief`: Engineer-specific, triggered when oxygen generator is broken
- `patients_waiting_belief`: Medic-specific, triggered when med-bay queue has patients
- `mission_time_belief`: Scavenger-specific, triggered with 20% probability per cycle
- `want_to_trade_belief`: Triggered when agent assesses trading motivation (probability: 0.5 + curiosity)

Desires (Goals): Desires are ranked by rule strengths, determining priority:

- `retire` (500.0) - Highest priority
- `escape_storm` (200.0) - High priority
- `has_oxygen` (100.0) - High priority
- `has_energy` (25.0) - Medium priority
- `be_healthy` (12.0) - Medium-low priority
- `seek_trading_area` (8.0) - Medium-low priority
- `fix_generator` (9.0) - Engineer only
- `heal_patients` (7.0) - Medic only
- `mine_resources` (5.0) - Scavenger only
- `wander` (default) - Lowest priority

Intentions (Plans): Plans define the actions agents take to fulfill desires. Each plan is associated with a specific desire and executes until the desire is satisfied or the plan is completed.

8.2 Code. Beliefs are managed through perception reflexes:

```
reflex perception {
    if (oxygen_level < oxygen_level_threshold) {
        if (not has_belief(suffocating_belief)) {
            do add_belief(suffocating_belief);
        }
    } else {
        if (has_belief(suffocating_belief)) {
            do remove_belief(suffocating_belief);
        }
    }
    // Similar logic for starving_belief and injured_belief
}
```

Desires are created through rules that link beliefs to desires:

```
rule belief: should_retire_belief new_desire: retire_desire strength: 500.0;
rule belief: storm_warning_belief new_desire: escape_storm_desire strength: 200.0;
rule belief: suffocating_belief new_desire: has_oxygen_desire strength: 100.0;
rule belief: starving_belief new_desire: has_energy_desire strength: 25.0;
rule belief: injured_belief new_desire: be_healthy_desire strength: 12.0;
rule belief: want_to_trade_belief new_desire: seek_trading_area_desire strength: 8.0;
```

Plans execute intentions:

```
plan get_health intention: be_healthy_desire
    finished_when: health_level >= max_health_level {
        if ((location distance_to habitat_dome.med_bay.location) <= facility_proximity) {
            state <- "waiting_at_med_bay";
            ask habitat_dome.med_bay { do add_to_queue(myself); }
        } else {
            state <- "going_to_med_bay";
            do goto target: habitat_dome.med_bay.location speed: movement_speed;
        }

        if (health_level >= max_health_level) {
            ask habitat_dome.med_bay { do remove_from_queue(myself); }
            do remove_belief(injured_belief);
        }
    }

plan get_oxygen intention: has_oxygen_desire
    finished_when: oxygen_level >= max_oxygen_level * 0.8 {
        if ((location distance_to habitat_dome.oxygen_generator.location) <= facility_proximity) {
            state <- "at_oxygen_generator";
        }
    }
```

```

        // Facility reflex handles replenishment automatically
    } else {
        state <- "going_to_oxygen";
        do goto target: habitat_dome.oxygen_generator.location speed: movement_speed;
    }
}

plan get_energy intention: has_energy_desire
    finished_when: energy_level >= max_energy_level * 0.8 {
        if ((location distance_to habitat_dome.greenhouse.location) <= facility_proximity)
            state <- "at_greenhouse";
        // Facility reflex handles replenishment automatically
    } else {
        state <- "going_to_greenhouse";
        do goto target: habitat_dome.greenhouse.location speed: movement_speed;
    }
}

```

Engineer-specific BDI behavior:

```

species Engineer parent: Human {
    predicate generator_broken_belief <- new_predicate("generator_broken");
    predicate fix_generator_desire <- new_predicate("fix_generator");

    reflex check_generator {
        if (habitat_dome.oxygen_generator.is_broken) {
            if (not has_belief(generator_broken_belief)) {
                do add_belief(generator_broken_belief);
            }
        }
    }
}

rule belief: generator_broken_belief new_desire: fix_generator_desire strength: 9
    plan fix_generator intention: fix_generator_desire {
        if ((location distance_to habitat_dome.oxygen_generator.location) <= facility_proximity)
            habitat_dome.oxygen_generator.is_broken <- false;
        do remove_belief(generator_broken_belief);
        do remove_intention(fix_generator_desire, true);
        state <- "idle";
    } else {
        state <- "going_to_oxygen_generator";
        do goto target: habitat_dome.oxygen_generator.location speed: movement_speed;
    }
}

```

8.3 Demonstration.

1. **Input:** An agent's oxygen level drops below 20% threshold.

2. Screenshot:

Figure 11: Oxygen levels down

3. **Interpretation:** When oxygen falls below the threshold, the perception reflex adds `suffocating_belief`. The BDI system creates `has_oxygen_desire` with high priority (100.0), and the agent executes `get_oxygen` to reach the oxygen generator. Once within range, the generator replenishes oxygen at 0.3 per cycle.

4. **Input:** An agent's health drops below 50% threshold.

5. Screenshot:

Figure 12: Health level below threshold

6. **Interpretation:** When health drops below the threshold, the agent adds the `injured_belief`, creates the `be_healthy_desire`, and executes the `get_health` plan. The agent navigates to the med-bay and queues for treatment.

7. **Input:** The oxygen generator breaks (5% probability per cycle).

8. Screenshot:

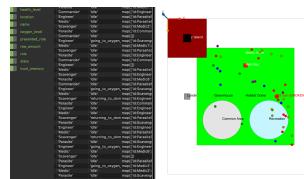


Figure 13: Oxygen generator broken

9. **Interpretation:** Engineers detect the broken generator via `check_generator`, add `generator_broken_belief`, and execute the repair plan. Only Engineers respond to generator failures.
10. **Input:** Multiple agents have different priority desires simultaneously (e.g., one suffocating, one injured, one with storm warning).
11. **Screenshot:**

attributes	th_level	location	name	oxygen_level	presented_role	raw_amount	role	state	trust_memory
energy_level	0	(224.1644836276)	'Engineer0'	99.20000000000000	'Engineer'	0	'Engineer'	'going_to_oxygen'	map(['id:Commander'])
happiness	0	(124.8890636482)	'Medic0'	98.20000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Parasite0'])
health_level	0	(102.9320536482)	'Scavenger0'	99.20000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Engineer1'])
location	0	(172.64597183700)	'Parasite0'	98.70000000000000	'Parasite'	0	'Parasite'	'going_to_oxygen'	map(['id:Engineer1'])
name	0	(224.2304192269)	'Commander0'	99.08000000000000	'Commander'	0	'Commander'	'idle'	map([])
oxygen_level	0	(161.47426183700)	'Medic1'	98.18000000000000	'Medic'	0	'Medic'	'going_to_oxygen'	map(['id:Scavenger1'])
presented_role	0	(102.9320536482)	'Scavenger1'	99.20000000000000	'Scavenger'	0	'Scavenger'	'idle'	map(['id:Parasite1'])
raw_amount	0	(302.2282946416)	'Parasite1'	98.18000000000000	'Parasite'	0	'Parasite'	'idle'	map([])
role	0	(94.3044720390)	'Commander1'	99.20000000000000	'Commander'	0	'Commander'	'idle'	map(['id:Medic2'])
state	0	(238.4898383044)	'Engineer2'	99.20000000000000	'Engineer'	0	'Engineer'	'idle'	map(['id:Engineer2'])
trust_memory	0	(208.038303044)	'Medic2'	98.80000000000000	'Medic'	0	'Medic'	'idle'	map([])
role	0	(107.85357800717)	'Scavenger2'	99.10000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Engineer1'])
state	0	(99.98452786664)	'Engineer3'	99.20000000000000	'Engineer'	0	'Parasite'	'idle'	map(['id:Engineer2'])
trust_memory	0	(74.4527733333)	'Commander2'	99.10000000000000	'Commander'	0	'Commander'	'idle'	map(['id:Commander'])
role	0	(204.2643733774)	'Engineer4'	99.20000000000000	'Engineer'	0	'Engineer'	'going_to_oxygen'	map(['id:Medic3'])
state	0	(83.5688381908)	'Medic3'	98.20000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Parasite4'])
trust_memory	0	(105.2005295403)	'Scavenger3'	99.20000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Engineer2'])
role	0	(76.943147514605)	'Parasite3'	99.20000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Engineer2'])
state	0	(69.039147514605)	'Commander3'	99.20000000000000	'Commander'	0	'Commander'	'idle'	map(['id:Engineer2'])
trust_memory	0	(105.727286770705)	'Engineer4'	99.08000000000000	'Engineer'	0	'Engineer'	'idle'	map(['id:Engineer2'])
role	0	(93.70499761905)	'Medic4'	99.70000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Engineer2'])
state	0	(72.7207738842)	'Scavenger4'	98.20000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map([])
trust_memory	0	(86.09051965834)	'Parasite4'	99.20000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Medic5'])
role	0	(224.1534793747)	'Commander4'	99.20000000000000	'Commander'	0	'Commander'	'going_to_oxygen'	map(['id:Scavenger4'])
state	0	(100.19051965834)	'Medic5'	98.18000000000000	'Medic'	0	'Medic'	'idle'	map([])
trust_memory	0	(108.4050801044)	'Scavenger5'	99.20000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Medic4'])
role	0	(73.1000000000)	'Parasite5'	99.20000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Engineer1'])
state	0	(277.6757544646)	'Commander5'	100.0	'Commander'	0	'Commander'	'idle'	map([])
trust_memory	0	(125.8619409292)	'Medic6'	99.18000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Engineer1'])
role	0	(107.9799355515)	'Scavenger6'	98.60000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Parasite6'])
state	0	(105.6683835327)	'Parasite6'	99.70000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Parasite6'])
trust_memory	0	(224.149147514605)	'Commander6'	99.20000000000000	'Commander'	0	'Commander'	'going_to_oxygen'	map(['id:Medic6'])
role	0	(66.7770598334)	'Medic7'	99.70000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Parasite2'])
state	0	(107.7608238826)	'Scavenger7'	98.60000000000000	'Scavenger'	0	'Scavenger'	'going_to_mine'	map(['id:Parasite7'])
trust_memory	0	(81.16296985584)	'Parasite7'	98.68000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Engineer1'])
role	0	(224.149147514605)	'Commander7'	99.20000000000000	'Commander'	0	'Commander'	'going_to_oxygen'	map(['id:Parasite7'])
state	0	(74.56026188494)	'Medic8'	99.70000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Parasite7'])
trust_memory	0	(76.27648779084)	'Parasite8'	98.68000000000000	'Parasite'	0	'Parasite'	'idle'	map(['id:Medic8'])
role	0	(224.2080598443)	'Engineer8'	99.70000000000000	'Engineer'	0	'Engineer'	'going_to_oxygen'	map(['id:Scavenger8'])
state	0	(107.9406503537)	'Medic9'	99.70000000000000	'Medic'	0	'Medic'	'idle'	map(['id:Engineer1'])
trust_memory	0	(21.9267053777)	'Parasite9'	99.68	'Parasite'	0	'Parasite'	'idle'	map(['id:Engineer1'])
role	0	(224.1825391075)	'Engineer9'	99.70000000000000	'Engineer'	0	'Engineer'	'going_to_oxygen'	map(['id:Commander'])
state	0	(188.5078860952)	'Scavenger9'	98.18000000000000	'Scavenger'	0	'Parasite'	'idle'	map([])
trust_memory	0	(314.2493143886)	'Engineer10'	99.20000000000000	'Engineer'	0	'Engineer'	'idle'	map([])

Figure 14: Different states

12. **Interpretation:** Desire selection follows rule strengths. For example, agents with a storm warning (200.0) prioritize escape over other tasks, while suffocating agents (100.0) prioritize oxygen above low-priority goals such as healing (12.0).

9. Challenge 2: Reinforcement Learning

9.1 Explanation. The simulation implements Q-Learning, a reinforcement learning algorithm, to enable agents to learn which other agents are trustworthy through experience. Agents learn to identify parasite agents by tracking the outcomes of trade interactions (e.g., trades of energy, health, etc.) and updating Q-values accordingly.

There are two fixed-coordinate areas where agents can trade. When agents sense a need for a resource replenishment (e.g., low energy, low health), they move to a trading area and randomly select another agent present to attempt a trade. The trade interaction results in rewards based on the actions of both agents (mutual cooperation, altruism, theft, etc.). Agents update their Q-values based on these rewards, allowing them to learn trustworthiness over time. **Q-Learning Implementation:**

- **State:** Agent ID ("id:agent_name") - Each agent tracks trust per individual, not per type
- **Actions:** TRADE or IGNORE (selected via ϵ -greedy: 20% random exploration, 80% exploitation)

- **Rewards:**

- Mutual cooperation (both give): +100.0
- Altruism (non-parasite gives goods without receiving anything back): +20.0
- Victim trusts parasite: -80.0
- Parasite steals: -80.0
- IGNORE (avoid risky interaction): +2.0
- Failed/no-interaction: 0.0

- **Q-Value Update:** $Q(s, a) = Q(s, a) + \alpha(reward - Q(s, a))$ where $\alpha = 0.35$ (learning rate/sociability)
- **Trust Calculation:** $trust = \frac{Q[TRADE] - Q[IGNORE]}{100.0}$, clamped to $[-1.0, 1.0]$
- **Detection:** Trust < 0 indicates predicted parasite
- **Adaptive Learning:** Sociability gradually recovers if it drops below 0.15 to prevent learning stagnation

Learning Evolution: Initially, agents interact randomly (curiosity = 20%). Parasites attempt to exploit others, gaining -80 rewards when caught stealing. Over time, Q-values for parasite interactions decrease due to accumulated negative rewards. Trust values drop toward -0.8 to -1.0 for parasites. Agents learn to prefer IGNORE actions (reward +2) over trading with distrusted partners. Adaptive curiosity increases exploration when agents are struggling (low average trust), forcing them out of local optima. The learning rate (sociability = 0.35) enables fast convergence. The learning metrics track average trust to parasites vs non-parasites, plus precision/recall for parasite detection.

9.2 Code. Q-Learning state and action selection:

```

map<string, list<float>> Q <- map();
map<string, float> trust_memory <- map();

float sociability <- 0.35; // Learning rate (increased for faster convergence)
float curiosity <- 0.20; // Exploration rate

action ensure_state_exists(string s) {
    if (!(Q contains_key s)) {
        Q[s] <- [0.0, 0.0]; // [Q(TRADE), Q(IGNORE)]
    }
    if (!(trust_memory contains_key s)) {
        trust_memory[s] <- 0.0;
    }
}

action choose_action(string s) {
    // Epsilon-greedy: explore or exploit
}

```

```

if (flip(curiosity)) {
    return (flip(0.5) ? "TRADE" : "IGNORE");
}
list<float> qs <- Q[s];
return (qs[0] >= qs[1] ? "TRADE" : "IGNORE");
}

```

Q-value update with proper exponential moving average:

```

action update_q(string s, string a, float r) {
    do ensure_state_exists(s);
    int idx <- (a = "TRADE" ? 0 : 1);

    list<float> qs <- Q[s];
    float old <- qs[idx];
    // Exponential moving average: converges to average reward
    float updated <- old + sociability * (r - old);
    qs[idx] <- updated;
    Q[s] <- qs;

    // Calculate trust and downscale trust (from -100 and +100 to -1.0 to +1.0)
    float pref <- Q[s][0] - Q[s][1];
    trust_memory[s] <- max(-1.0, min(1.0, pref / 100.0)); // Normalize by max expected
}

reflex adapt_learning_parameters when: cycle mod 100 = 0 and length(keys(trust_memory))
// Calculate average trust from recent interactions
float total_trust <- 0.0;
int count <- 0;
loop t over: values(trust_memory) {
    total_trust <- total_trust + t;
    count <- count + 1;
}
float avg_trust_value <- total_trust / float(count);

// Adapt curiosity: increase when learning is poor, decrease when good
if (avg_trust_value < -0.2) {
    curiosity <- min(0.50, base_curiosity + 0.15);
} else if (avg_trust_value < 0.0) {
    curiosity <- min(0.45, base_curiosity + 0.10);
} else if (avg_trust_value < 0.3) {
    curiosity <- base_curiosity;
} else if (avg_trust_value < 0.6) {
    curiosity <- max(0.10, base_curiosity - 0.05);
} else {
    curiosity <- max(0.05, base_curiosity - 0.10);
}

```

```

// Gradually recover sociability if it has dropped too low
if (sociability < 0.15) {
    sociability <- sociability + 0.01;
}
trust_memory[s] <- max(-1.0, min(1.0, pref / 100.0));
}

```

Adaptive learning parameters:

```

reflex adapt_learning_parameters when: cycle mod 100 = 0 and
length(keys(trust_memory)) > 0 {
// Calculate average trust
float total_trust <- 0.0;
loop t over: values(trust_memory) {
    total_trust <- total_trust + t;
}
float avg_trust <- total_trust / float(length(values(trust_memory)));

// Adapt curiosity: explore more when struggling, exploit when successful
if (avg_trust < -0.2) {
    curiosity <- min(0.50, base_curiosity + 0.15);
} else if (avg_trust < 0.0) {
    curiosity <- min(0.45, base_curiosity + 0.10);
} else if (avg_trust < 0.3) {
    curiosity <- base_curiosity;
} else if (avg_trust < 0.6) {
    curiosity <- max(0.10, base_curiosity - 0.05);
} else {
    curiosity <- max(0.05, base_curiosity - 0.10);
}

// Recover sociability if stuck in local optima
if (sociability < 0.15) {
    sociability <- sociability + 0.01;
}
}

```

Trade interaction with learning:

```

reflex assess_trading_motivation when:
trade_cooldown = 0
and (habitat_dome.shape covers location)
and not has_belief(storm_warning_belief)
and not has_belief(want_to_trade_belief)
{
if (flip(0.5 + curiosity)) {
    do add_belief(want_to_trade_belief);
}

```

```

    }
}

reflex learn_and_trade when:
    trade_cooldown = 0
    and (habitat_dome.common_area.shape covers location or habitat_dome.recreation_are
    and not has_belief(storm_warning_belief)
{
    list<Human> all_humans <- list(Engineer) + list(Medic) +
                                list(Scavenger) + list(Parasite) +
                                list(Commander);
    list<Human> nearby <- [];

    loop h over: all_humans {
        if (h != self) {
            float dist <- h.location distance_to location;
            if (dist <= meet_distance) {
                nearby <- nearby + [h];
            }
        }
    }

    if (empty(nearby)) { return; }

    Human partner <- one_of(nearby);
    string s <- "id:" + partner.name;
    do ensure_state_exists(s);

    string a <- choose_action(s);
    if (a = "IGNORE") {
        do update_q(s, a, 2.0);
        trade_cooldown <- trade_cooldown_max;
        return;
    }

    float reward <- attempt_trade(partner);
    do update_q(s, a, reward);
    happiness <- happiness + reward;
    total_trades <- total_trades + 1;
    trade_cooldown <- trade_cooldown_max; // Cooldown is 5 cycles
}

```

9.3 Demonstration.

1. **Input:** Simulation starts with agents having empty Q-tables and trust memory (Ref. 15). Agents begin trading with each other.
2. **Screenshot:**



Figure 15: Initial Q values

3. **Interpretation:** Initially, Q-values are zero, so agents explore trade decisions frequently under the ϵ -greedy policy.
4. **Input:** Simulation runs for several hundred cycles, allowing multiple trade interactions with the same agents.
5. **Screenshot:**

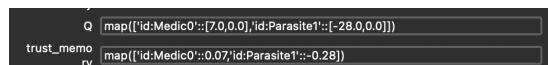


Figure 16: Q values after 10 cycles

6. **Interpretation:** The Q-table accumulates positive and negative values based on trade outcomes (Fig. 16). After repeated interactions, agents start separating parasites from cooperative partners and increasingly exploit learned preferences (80% greedy).
7. **Input:** Simulation runs for an extended period (1000+ cycles) with learning metrics being tracked.
8. **Screenshot:**

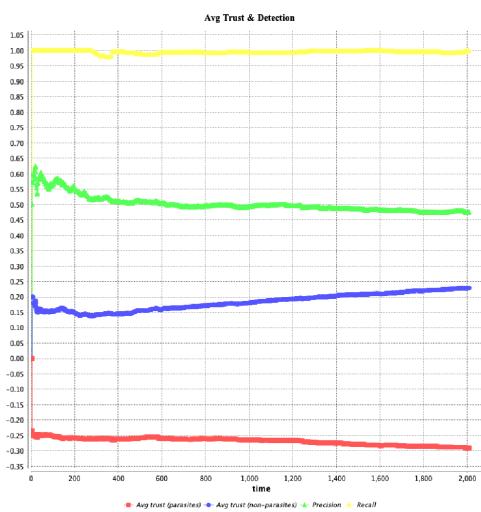


Figure 17: Avg. trust after extended learning

9. **Interpretation:** After extended learning, average trust becomes strongly negative for parasites and remains positive for cooperative agents (Fig. 17). Adaptive curiosity settles around 10–15% exploration once average trust is high.

10. **Input:** Simulation runs for an extended period (1000+ cycles) with average agent lifespan being tracked.

11. Screenshot:

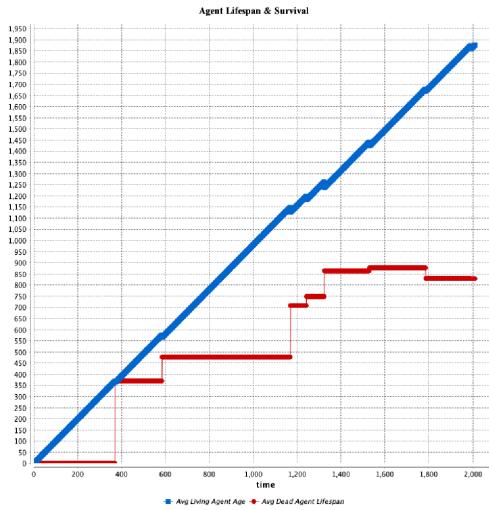


Figure 18: Avg. lifespan of dead and alive agents

12. **Interpretation:** As shown in 18, the average lifespan of living agents increases over time, while the average lifespan of deceased agents stabilizes. This is consistent with fewer damaging interactions as agents learn to avoid risky partners.

10. Final Remarks

This project shows how individual learning can shape system-level behavior in a distributed multi-agent setting. The Mars Colony simulation combines BDI architecture for survival behaviors with Q-Learning for social decisions, allowing agents to adapt their interactions based on experience.

Key Achievements:

- Implemented a full BDI architecture with multiple belief types, prioritized desires, and executable plans
- Implemented Q-Learning with state-action-reward framework for trust-based decision making
- Created a continuous simulation system that maintains population through supply shuttles
- Integrated FIPA communication for long-distance messaging
- Demonstrated learning evolution through metrics showing improved parasite detection over time

Limitations:

- Learning is individual - agents don't directly share knowledge (though they could observe patterns)
- Parasites use simple disguise mechanism - could be enhanced with more sophisticated deception
- The `patience` parameter (discount factor) is defined but not used in the learning algorithm

Potential Improvements:

- Apply Q-table inheritance so new agents start with averaged knowledge from survivors
- Implement knowledge sharing mechanisms (e.g., agents can share trust information)
- Implement the `patience` parameter (discount factor) for temporal discounting of future rewards
- Add more sophisticated parasite strategies (e.g., temporary cooperation, selective stealing)
- Implement reputation systems where agents can observe others' interactions
- Add more complex reward structures (e.g., reputation-based rewards, long-term relationship benefits)
- Enhance visualization with additional charts (population over time, trade success rates, death rates, etc.)

Conclusion: Over time, agents learn to avoid parasites through reinforcement learning. Without centralized control, behavior shifts as agents adapt to negative experiences, which provides an intuition for how trust and avoidance can emerge in multi-agent systems.