EvASim: Evolutionary Agent-Based Aggression Simulation

Intelligent Software Engineering Project

Authors: Giovanni Antonioni, Luca Tassinari

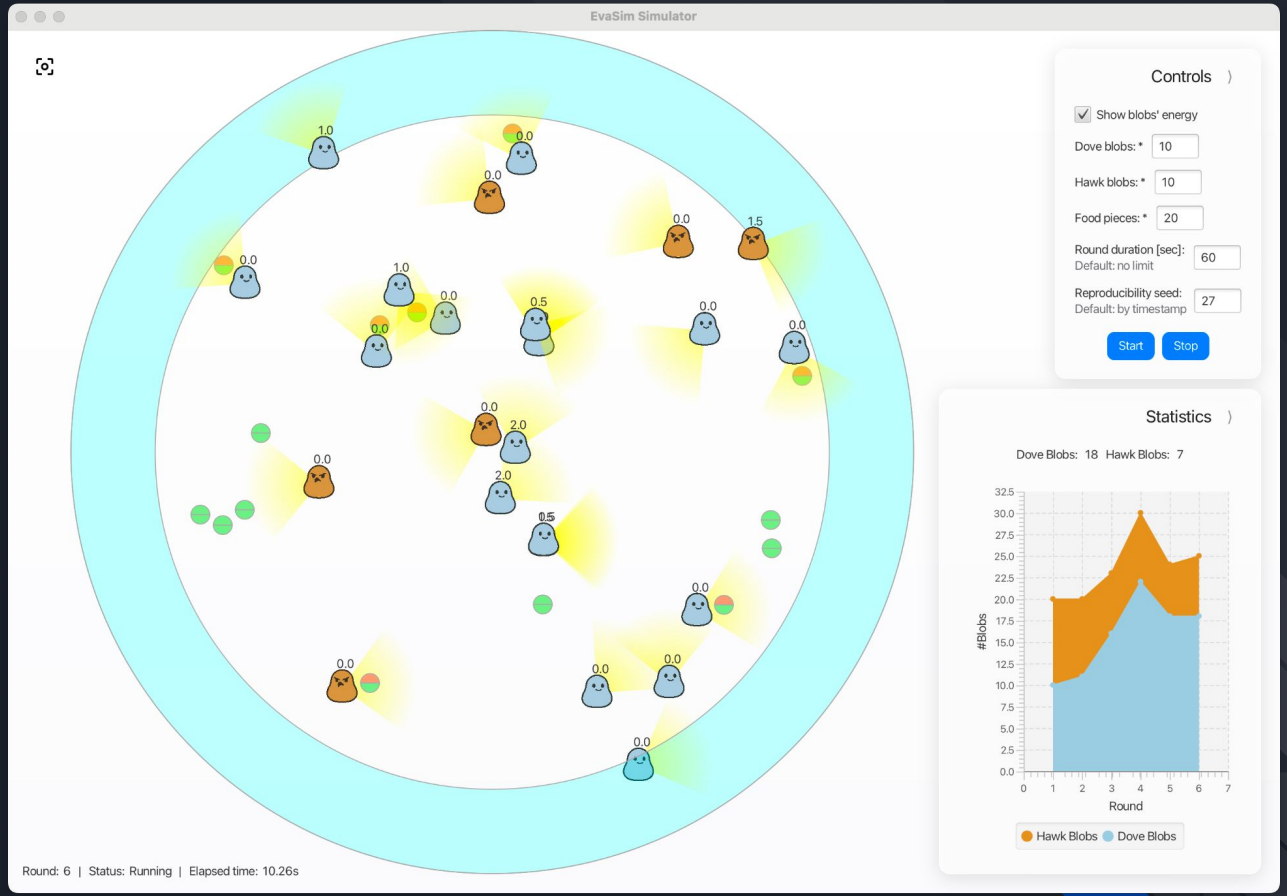
Project Goals

The goal of the project is to simulate an environment using a BDI Agent framework in which agents simulate two types of creatures, Doves and Hawks, that compete for survival based on their behaviors, observing how the evolution of the species unfolds [1] [2].

[1] The project has been taken inspiration from the Simulating the Evolution of Aggression, by Primer's video.

[2] Full requirements can be found here, along with the full report

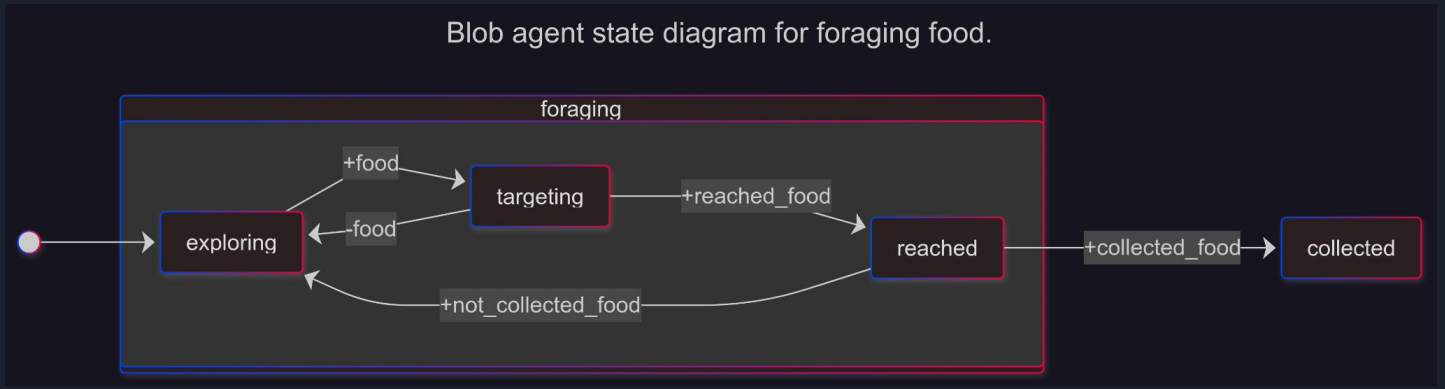
Demo



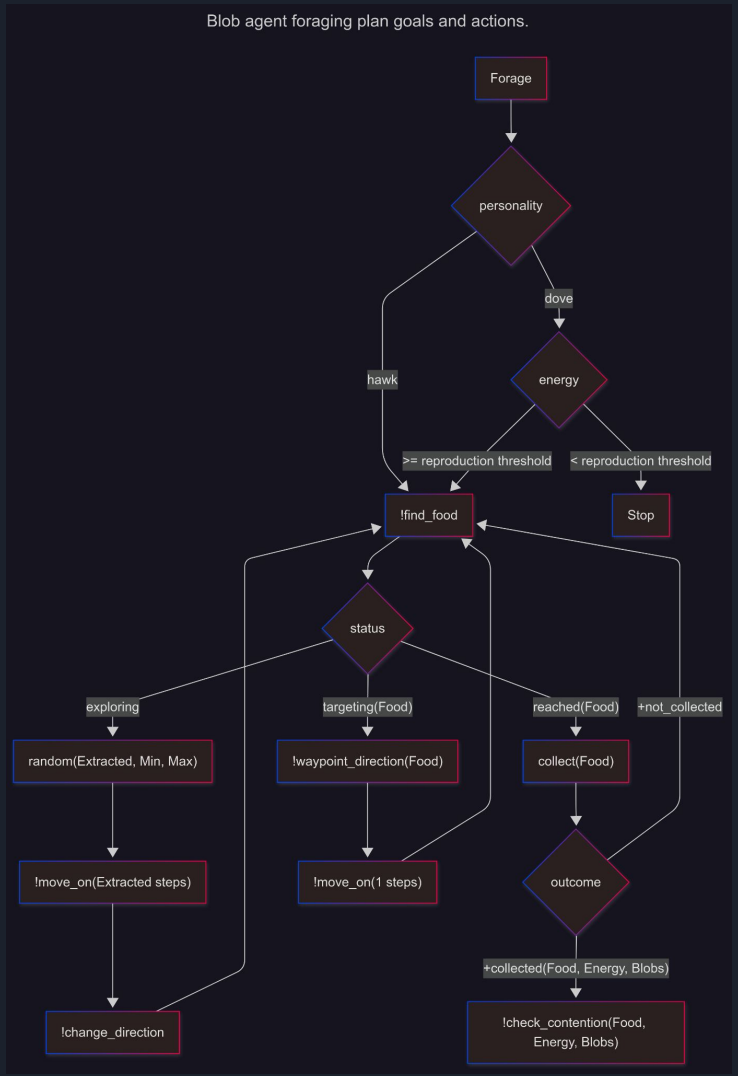
Code is open source and available at github.com/giovaz94/evasim/

Agent Design

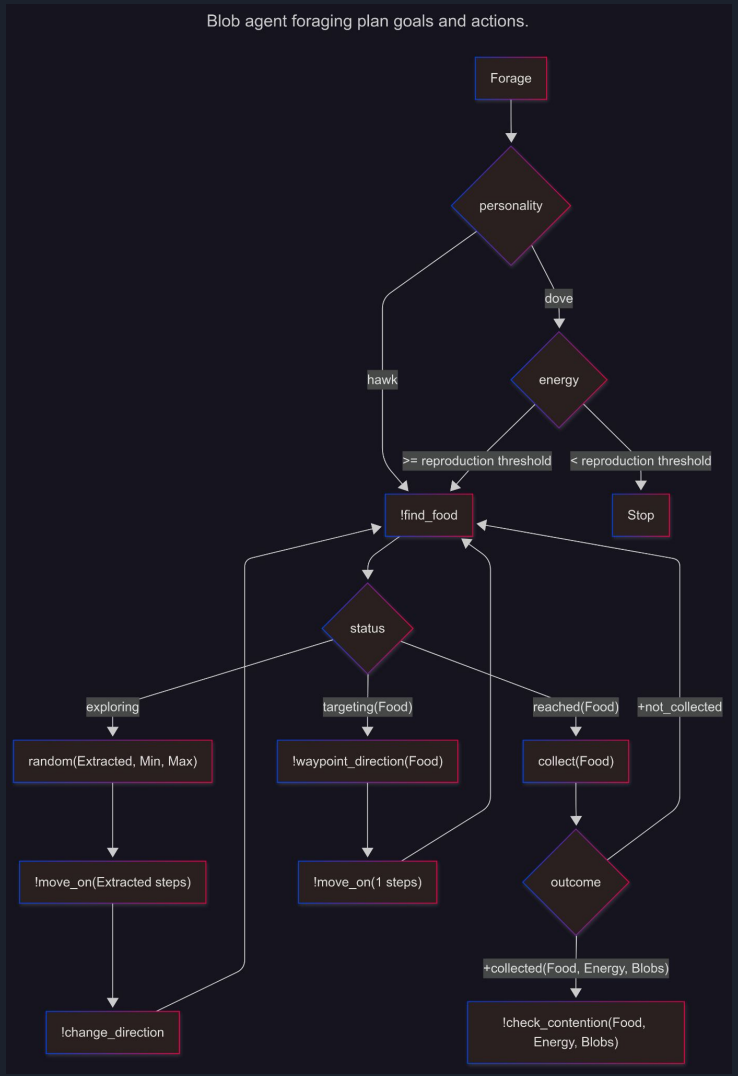
Food search



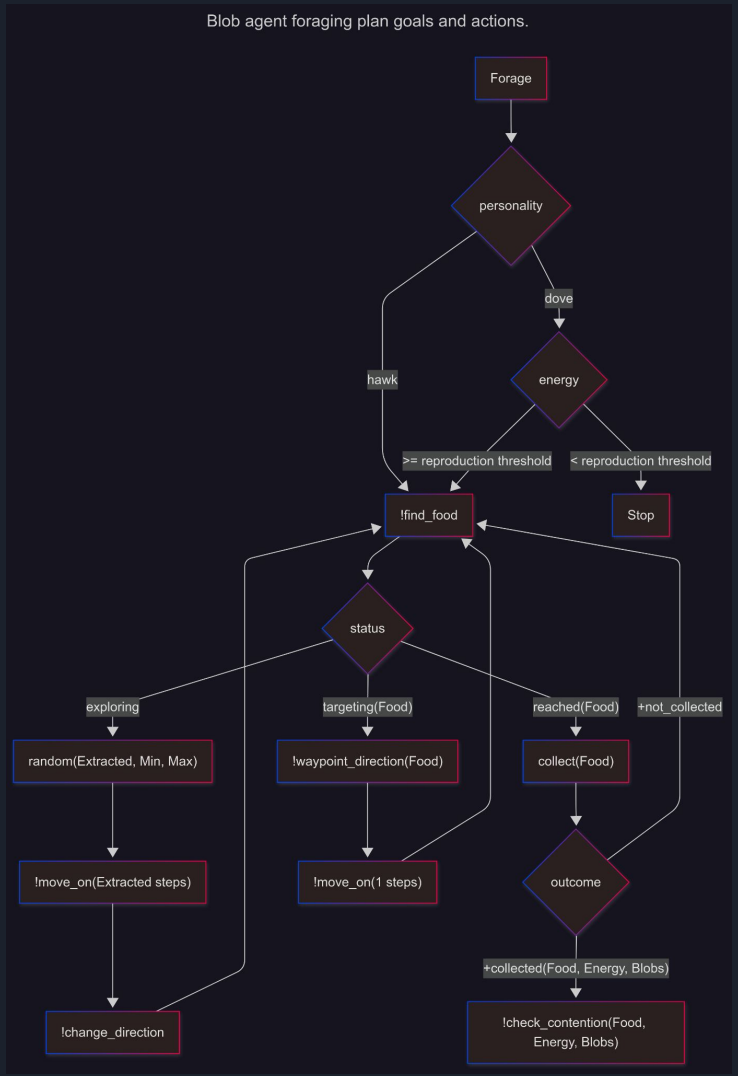
- Blob agents initial goal is to find and collect food.
  - Since they do not know their exact position and have limited sight, they explore the world randomly
- When an agent finds food, it moves toward it;
- Upon reaching the food, the agent tries to collect it;
  - Depending on the number of agents trying to collect the same food, an agent may either:
    - Successfully collect the food (collected state), or
    - Fail to collect it and return to the exploring state to continue searching.



Doves stop searching when they have enough energy to reproduce, while Hawks continue searching until the end of the round.



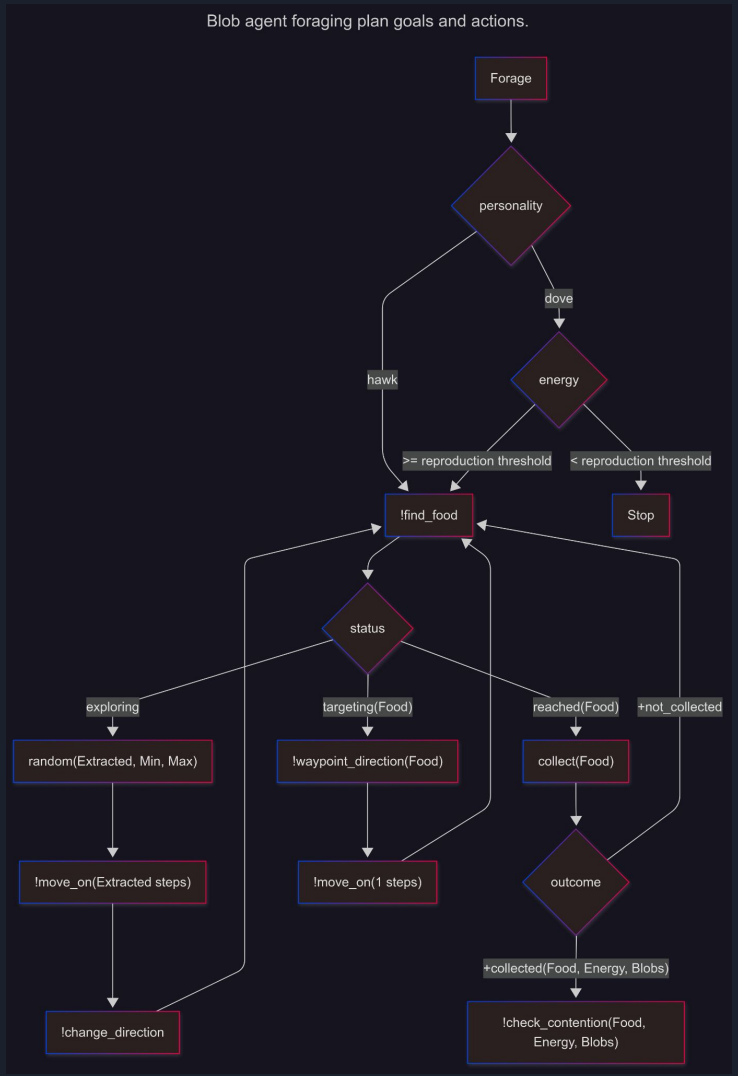
The exploration is performed by moving in a random direction for a certain number of steps (drawn randomly from a range of values), followed by a change of direction.



When an agent "sees" food in its sight that still has uncollected pieces, it starts moving towards it, one step at a time, changing its direction to point towards the food.

If multiple foods are in the sight:

- nearest food “wins”
- food with yet uncollected pieces “wins”



When the agent reaches the food, it tries to collect it.

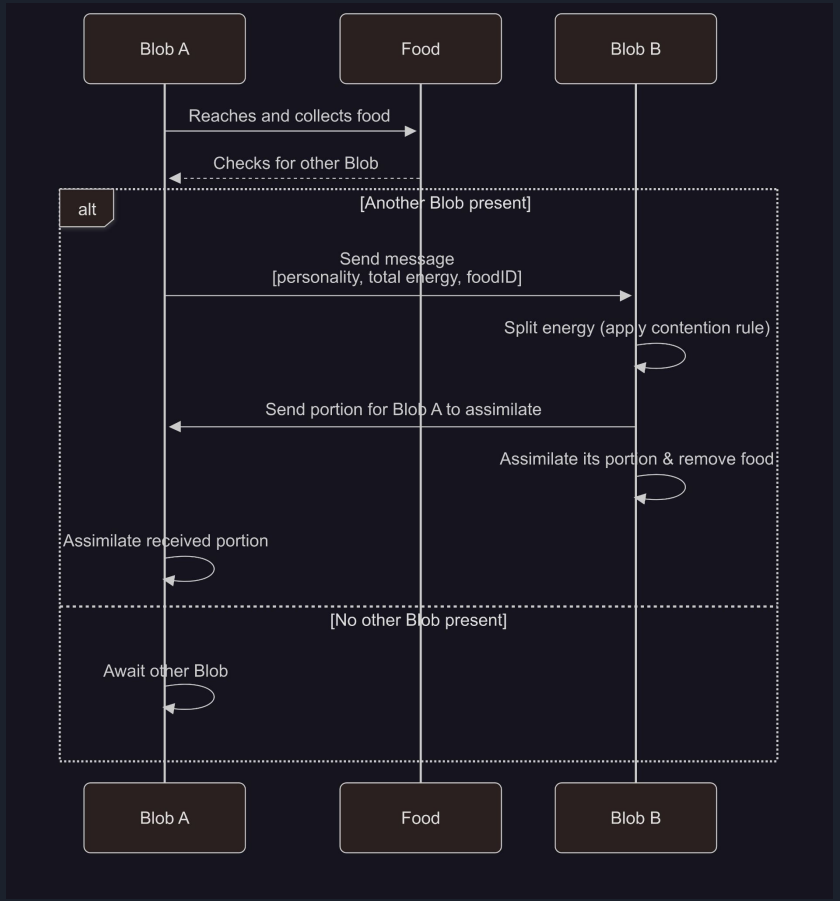
Depending on the outcome of the collection attempt, the agent may:

- proceed to the contention phase
- or return to exploring searching for food.

Food Contention

1. When a blob reaches a piece of food in the environment and successfully collects it, it will check whether any other Blob has collected the same food.
    a. If no other blob are present then it will await indefinitely, until someone shows up
    b. If another blob is present then It will start the *contention protocol*

2. On the contention protocol the last Blob that arrives to the food send the information to the first specifying:
    a. Its personality
    b. Total energy of the food
    c. The food identifier



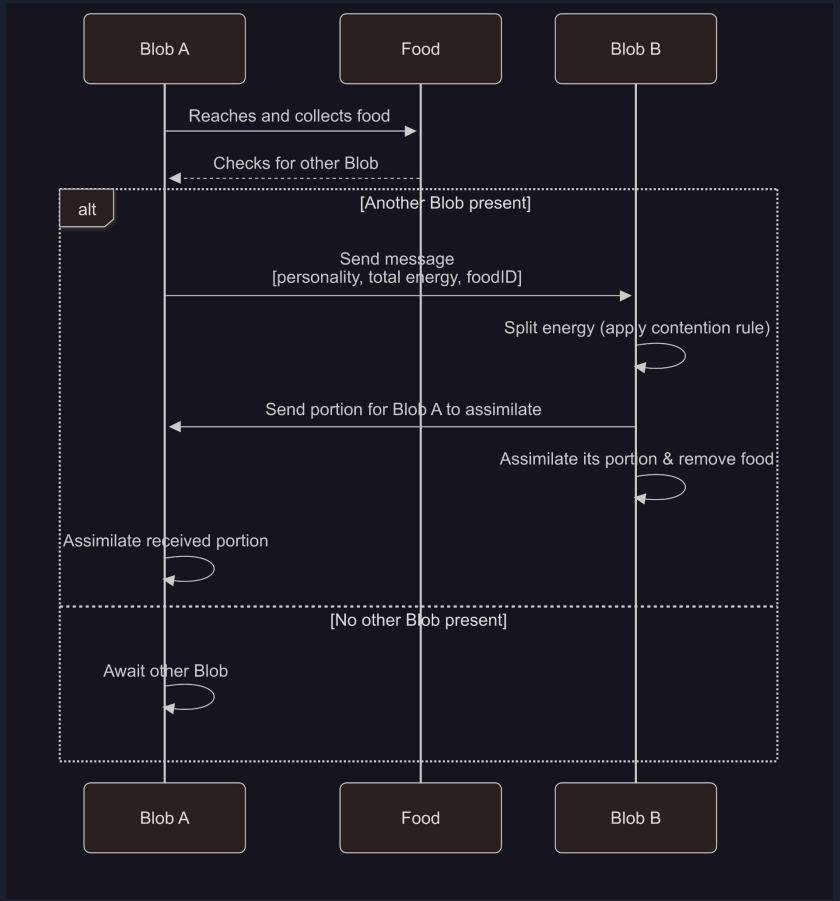
Food Contention

3. The blob that receives message will split the food energy according to a contention rule.

   A *contention rules* determines how the energy should be splitted based on both blob personalities

4. The blob that executes the split will now assimilate the energy and send the portion of the split to the other blob and, as final step will remove food from the environment and resume the normal execution (foraging goal).

5. The blob receives the energy to assimilate and resume the normal execution (foraging goal).



Implementation Details

Techs used for the project

Kotlin + *JaKtA*

```
/**
 * Blob agent factory.
 */
fun MasScope.blobAgent(blob: Blob) = agent( name = blob.id.value) {  2 Usages   Luca Tassinari +1
    beliefs {
        fact { personality( term = if (blob.personality is Hawk) "hawk" else "dove") }
        fact { energy( term = 0.0) }
        fact { direction( term = tupleOf( ...terms = 0.0, 0.0)) }
        fact { speed(term = 20.0) }
        fact { status( term = exploring) }
    }
    actions {
        action( action = Random)
        action( action = WaypointDirection)
        action( action = InverseDirection)
        action( action = EndRound)
    }
    goals {
        achieve( goal = change_direction)
        achieve( goal = forage)
    }
    plans {
        forage()
        findFood()
        movement()
        collectFood(blob)
        contention(blob)
        endedRound()
    }
    timeDistribution { Time.real(value = 50) }
}
```

Analysis

Objective of the Analysis

Differences between our experiment with the original one:

Feature	Original experiment	Our simulation
Food Assignation	Pre-assigned pairs of blobs at the starting of round.	Ability to search for food in the environment.
Hunger Management	Limit of 1 food per round.	Dove will continue to search for food until they're able to reproduce themselves while Hawk will continue until the end of round. Round can have a finite duration.
Hawk Contention	The fight between two Hawk sets the total energy of contendent blobs to 0	The energy gained by the contention between two Hawk is 0 but total energy of respective blobs remains unaltered

Addressing reproducibility

Reproducibility is a crucial aspect of simulations—and of science in general— and is what distinguishes between scientific experiments from anecdotal evidence.

The main two problems that can affect reproducibility are:

1. the concurrency platform used by the underlying BDI framework used to carry out the agent reasoning cycle. JaKtA provides one thread per agent or one thread per MAS configurations.

   ► using one thread per agent was unsuitable, as the goal was to simulate the evolution of many discrete agents efficiently → one thread per MAS is used.

Addressing reproducibility

Reproducibility is a crucial aspect of simulations—and of science in general— and is what distinguishes between scientific experiments from anecdotal evidence.

The main two problems that can affect reproducibility are:

1. the concurrency platform used by the underlying BDI framework used to carry out the agent reasoning cycle. JaKtA provides one thread per agent or one thread per MAS configurations.

   ► using one thread per agent was unsuitable, as the goal was to simulate the evolution of many discrete agents efficiently → one thread per MAS is used.

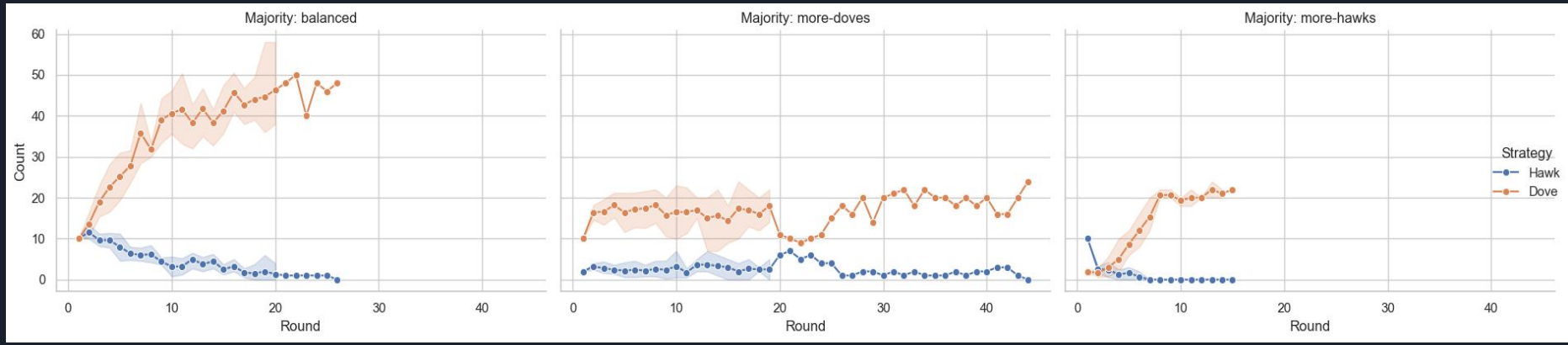
2. random number generation that guides the simulation, e.g., the random placement of the food in the world, the random movements of the agents, and the random decisions made by the agents;

   ► single unified random number generator provider which can be seeded by the user to ensure reproducibility

What we've simulated

Three different simulations, for each of these we've registered 5 runs.

1. Same number of Dove and Hawk (10 Hawk vs 10 Doves)
2. Population Majority for Dove (10 Doves vs 2 Hawk)
3. Population Majority for Hawk (10 Hawk vs 2 Doves)



Conclusions

What we learned:

- Gain a hands-on understanding of the agent programming paradigm through coding;
- using JaKtA allowed us to analyze certain aspects of its functionality in more detail, particularly during the debugging phase, as there were some minor shortcomings.
    - The exploration highlighted possible enhancements to the JaKtA framework, which have since been proposed.
- attempting to build even a simple simulation exposed us to several important challenges, such as performance issues when handling a large number of agents and difficulties in ensuring the reproducibility of experiments.

Possible extensions:

1. Food discovery by “tips” from other agents
    a. ***Doves*** provide correct tips about where other agents can find available foods, while **Hawks** provide wrong tips;
    b. only ***Doves*** listen to the correct tips.
2. ***Doves*** make network to defend against ***Hawks*** by "shouting" when they have been hawked, increasing the belief base of nearby ***Dove*** agents about the fact a precise blob is acting as a *hawk*. This can be leveraged in future contentions by ***Doves*** to be "smarter" than the adversary.

Thanks for your attention

Authors: Giovanni Antonioni, Luca Tassinari