BDI-based RTS Simulation

Cooperative and Competitive Dynamics in Belief-Desire-Intention-based Multi-Agent Systems from a Real-Time Strategy Game Simulation

Giuseppe Speciale[1095306]

Multi Agent Systems, Università degli Studi di Bologna - 2024

Abstract. This project implements a Multi-Agent System (MAS) using the Belief-Desire-Intention (BDI) model within a real-time strategy simulation. It explores agent interactions in a dynamic environment, focusing on emergent cooperative and competitive behaviors. Agents gather resources to let their colony grow, while defending it from invaders and attacking others at the same time. The simulation features a fully functioning 2D environment and User Interface complete with plots and tunable parameters.

Keywords: Simulation · Emergence · BDI · RTS · MAS

1 Introduction

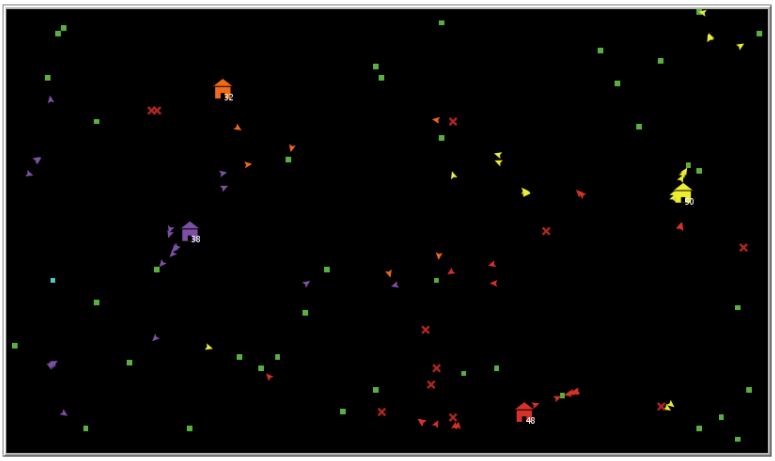

Fig. 1. Screenshot of a simulation. Four colonies have survived up until this moment, while two have already gone extinct. The red cross markers indicate where fights have taken place while the number near houses (bases) shows their current health.

1.1 Objective

The goal of this project is to design, implement, and analyze a fully functioning Multi-Agent System (MAS) modeled after the mechanics of a Real-Time Strategy (RTS) game. The system operates on a 2D grid where agents, representing individual units within a colony, engage in activities such as resource collection, territorial expansion, and defense. The overarching aim is to observe the emergence of complex behaviors and strategies from simple rules and agent interactions.

1.2 Development

The project has been built in NetLogo to exploit the rapid prototyping capabilities of the software and of its language, especially in a Multi Agent setting. The simulation presents a 2D screen in which the agents can be seen operating in real time, and an interactive interface that lets the user modify the parameters of the simulation without having to resort to the code itself. All agents and objects in the environment are modeled as special breeds of the turtles agent-set (like sub-classes in OOP), each of which has its own unique properties - the main one being the *soldier* breed, the main actors of the simulation. The Belief-Desire-Intentions paradigm has been built in as a custom library for the specific purpose of this project.

1.3 Installation Guide

To play the simulation, it's sufficient to have the latest NetLogo 6.4.0 version installed and a folder with the *main.nlogo* file inside of it. The simulation also presents itself with an Info page with easy explanations on how to operate the interface and correctly execute the simulation. The parameters are:

- **(Button)** *setup*: resets the simulation, creating new randomly positioned agents and resources.
- **(Button)** *go*: plays/pauses the simulation. The speed can be adjusted by the integrated bar on top of the interface.
- **(Button)** *clear*: removes all fight markers (red crosses) from the map.
- **(Switch)** *show-res*: Displays currently held resources by each agent on top of them.
- **(Switch)** *show-plan*: Displays the intention that each agent is currently executing. The game can't have both labels at the same time.
- **(Plots)** Represent, in time, how many resources each faction is in possess of (deposited, not held) and how many agents are alive for each faction.
- **(Outputs)** Give the number of resources in a readable way, along with the names of the colors the corresponding factions for ease of readability. Casualties are kept track of with a counter.

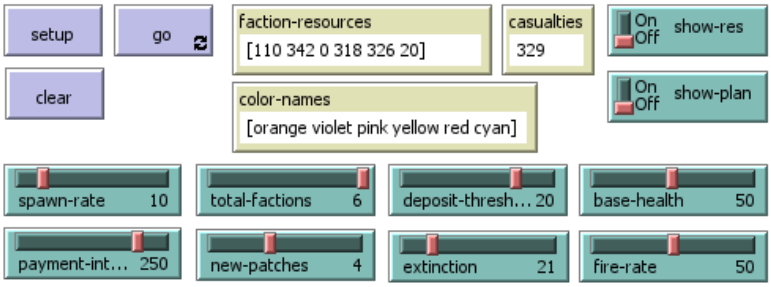

Fig. 2. Screenshot of the parameter settings, showing the slide-bars and buttons the user can interact with to change the simulation.

- *spawn-rate*: determines how fast agents are generated from their bases.
- *total-factions*: how many agents of different factions will spawn in the setup. Only changeable at the start. Does not always reflect the number of factions after a long simulation because some could go extinct.
- *deposit-threshold*: how many resources does an agent need to change its intention, go back home and securely store them all.
- *base-health*: life points of each base, only tunable at the start. When an agent is attacking, it decreases its life bit by bit, and when it reaches 0 the base dies and no longer spawns agents nor stores resources.
- *payment-interval*: determines the number of ticks after which, from each colony, resources gets subtracted in an equal number to their currently alive agents. If the colony fails to pay in that interval, no more allies get spawned for them.
- *new-patches*: how many new green patches (resources) spawn at each tick. It's a random number from 1 to it: the higher, the more plentiful the search for the agents.
- *extinction*: threshold value for the resources stored by colonies. While a base is under this value, no more allies get spawned from it until they can "pay the rent" again (this helps to prevent exponential growth).
- *fire-rate*: how fast the agents attack bases, once they get there. The lower, the faster (time intervals for the agents to shoot).

2 Game Mechanics

The game is played by independent and fully autonomous agents called *soldiers*, represented and visualised as colored arrows in a black 2D environment. Their role is mainly to gather resources (green squares) by moving towards the closest one, which gets stored in a local inventory. If they have enough, they spend them immediately to build a base, which in the simulation is a house in their color, that has the main function of storing resources for the colony. After having collected enough resources (*deposit-threshold*) they move towards their base

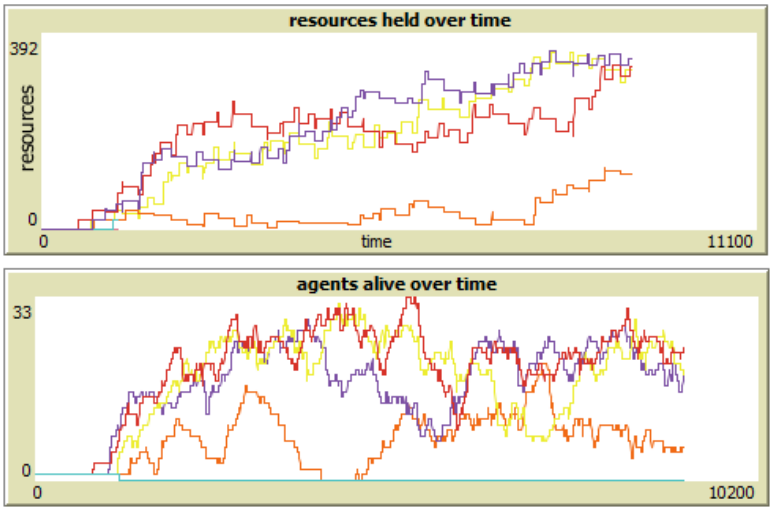

Fig. 3. Screenshot of two plots, showing color-coded information about living agents and resourced store by each colony.

to secure them. The base now spawn more soldiers in time, but draws from its resources (every *payment-interval* ticks) for each of the agents alive of their own colony. If they can't, then the base cannot spawn more allies.

Each agent gets generated with random starting attributes, such as its strength and speed, the faction to which it belongs (its color), additional data variables to store its targets and, most importantly, the beliefs, desires and intentions lists. They act as vectors of data through which the game updates in an iterative manner and executes the agents' needs and wants. Among the custom procedures written for the BDI architecture to work, *perceive-environment* simulates the agents' sight by giving them a small radius in which they can recognize bases or enemies close to them.

If an agents find another one close enough to be of a different color, so from another faction, they *fight*: the stronger agent survives, losing their difference in strength, while the other dies, losing all of its resources which will have been stolen by the winner. If an agent moves close to its base and has half of the resources needed for the *deposit-threshold* it will deposit them anyway out of fear of losing them. If instead it moves close to an enemy one and has enough resources to do so, it upgrades to an *archer* and starts shooting at the enemy base, detracting life from it. If a soldier sees a nearby archer it immediately fights it to fend off the menace: if a base were to be destroyed the agents wouldn't have a place to deposit their resources anymore.

The game simulation starts with a number of agents from 2 up to *total-factions*, and shows how many resources have been collected and how many agents are alive at each time instant by the colored plots in the UI. The simulation shows agents moving through space and shows fights, bases and bullets through the use of other custom breeds of the turtles agent-set. The simulation can show the agents' current intention or their number of currently held resources through the use of switches, and can have some of its parameters modified even at run-time by the user through slide-bars.

3 Behavioral Modeling

Agents have been designed using the Belief-Desire-Intention (BDI) model, which allows for the simulation of autonomous decision-making processes based on individual goals and environmental perceptions. The BDI model, as anticipated, is composed of 3 main procedures that update at each time instant the internal state of any agent's model, represented in the form of 3 ordered lists:

- *update-beliefs*
- *update-desires*
- *update-intentions*

The procedures will be explained in detail in the following subsections.

3.1 Beliefs

In the beliefs each agent stores what it thinks the world is like. That includes the position to the next resource to collect, how many enemies are close to it and if there are any bases nearby. The closeness is given by the *scan-radius* parameter, which scans everything inside it. The only exception to this rule is the closeness to the nearest resource, which is global (to solve an issue where agents would be stuck or crash the entire simulation with no resources nearby). The *perceive-environment* procedure is the only other function involved in the update of beliefs: it scans the radius for everything the agent needs to see and adds new beliefs also based on the internal state of the agent itself. For example, on collected resources, if they're above certain threshold values the agent can perform either of these activities:

- Greater than *deposit-threshold*: insert *can-deposit* among its beliefs. The agent goes for a long exploration path before coming back to its base. It's never believed if the base of its colony has been destroyed or not yet build (or else, it breaks the simulation)
- Greater than half the *deposit-threshold* value but close to its base: will believe it can deposit immediately, giving the opportunity.
- Greater than *archer-cost* and close to an enemy base: it believes it's possible to attack (*can-attack*) and weaken the enemy.

- Greater than *base-cost* and with no base for its colony yet: believes it can build one. Also works if the previous one gets attacked and destroyed: all agents share a boolean *has-base* value. No more than one base per colony prohibits exponential growth and explosion of population numbers.

The agent also believes it's possible to repel attackers (*can-defend*) when it finds one nearby. This is done with no regard to the faction of the attacker (it just has to be different than the one of the defender): justice is equal in this way and attacks are punished way more severely. This is also the most urgent desire so it's put at the end of the believing updates to better catch the opportunity to seize the invaders.

3.2 Desires

The agent desires something if it's in its best interest to act on that specific task. The main desire of the agent is to collect resources, so it's always put in the list if not already present:

```
if not member? "collect-resources" desires
[set desires lput "collect-resources" desires]
```

The presence of a desire determines if an intention will be made, and is in turn determined by the beliefs already present in the agent's architecture. Desires are in turn removed from the agent's list should the condition for its existence cease to be true.

- Desire to *defend* the base: the corresponding belief has to be present in its list, and the target has to be non-null. This is done to prevent defenders from staying in that state when the attackers have been dealt with and there aren't really any more threats.
- Desire to *deposit* resources: other than having the necessary conditions from the beliefs, an agent should have a positive amount of resources to be eligible to deposit them. This is done to avoid bugs in which agents get stuck inside their base trying to deposit null values of resources.
- Other desires (*attack* and *build-base*) also check if the desire isn't already present in the list before appending it, to avoid redundancy and infinite loops.

3.3 Intentions

Intentions represent what the agent will do next. After having scanned its environment, confronting his beliefs with his internal state and having expressed its desires, the agent now defines what is the action it will execute in the current time instant, just before repeating the same iter again. The process has been split in two for ease of deployment:

- *form-intentions*: updates the list based on the current desires and internal state of the agent. Elements can be put in a queue or substitute the list entirely. For example:

```
if member? "collect-resources" desires
[set intentions lput "collect" intentions]
```

This puts the intention at the end of the list, so that when no intention arises, this is the only one left to be executed;

```
if member? "deposit" desires and has-base = true
[set intentions (list "move-to-base" "deposit-resources")]
```

This instead overwrites the list entirely (if the conditions are met) and queues two actions in order: to move to the base, and to deposit when it's arrived (dealt with by the next procedure).

- *execute-intentions*: selects the first element of the list and executes the corresponding procedure in the code. Actions get popped and then removed from the list to avoid loops. Example:

```
if next-action = "attack-enemy-base" [
attack-base target-base
set intentions but-first intentions]
```

After the action gets selected, it gets executed by calling the *attack-base* procedure with the *target-base* as its parameter. It removes itself from the intentions list after it's done.

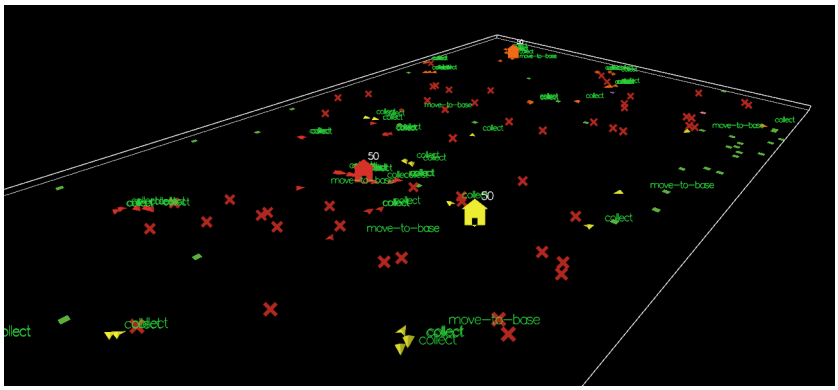

Fig. 4. 3D view of the simulation with each *current-plan* visible.

4 Observations and Results

A number of observations can be made from the repeated experiments with the differently tuned parameters. Being the simulation non-deterministic, although the logic of thought of a single agent is simple enough, each time the events and the settings change drastically. This gives us enough insight to take away interesting results on the behaviour of the soldiers:

- **Emergent Behaviour**: agents always follow the closest resource, which naturally brings them to flock together and walk almost always in big groups, especially in the later parts of the simulations. The fighting mechanics, combined with the randomly generated agent parameters, made for a balanced equilibrium (no agent dominated the scene) and more than often colonies struggled in a tug-of-war until one outlasted the other (not in all cases). More than often, the number of starting colonies decreased a lot in the first phase of each simulation: struggling to get to build a base often means immediate demise. Attack and defense of bases works fine, as colonies continuously spawn allies; it signifies defeat the instant the colony has difficulties with its resource management, since no more allies can get close and fend off invaders.
- **Decision Making**: the BDI architecture works consistently and reliably. Agents are able to correctly change their state (current plan) and then go back to their previous tasks when the occasion presents itself. They mainly prioritize defense over everything, then attacking, and ultimately allocating their resources. The sensory radius and speed of the agents has had an important impact on the social dynamics as one step too late could mean defeat for a base that is being attacked with no defenders. Agents are able to rapidly and efficiently switch between their multiple intentions with ease, with sporadic episodes of indeterminacy (being unable to choose which attacker to fend off when wanting to defend their base if being in the middle of them, resulting in being stuck going back and forth without ever reaching one).
- **Resource Management**: the ability of a colony to prosper and live a little longer (before the destruction of their base and the death of every living agent) depends on how many resources they can keep and the position of their base. One that is distant from all others has no problems getting resources stored up, while others constantly suffer from attacks and fights from being cluttered and close to each other. Sometimes even putting down the base too late can mean demise as these precious instant can signify great leverage for those that came earlier. Extinction and the rate of spawn of resources can be played with during any simulation to regulate the birth of new agents into the game: too much and they clutter the space completely, too little and no one plays anymore - or one alliance takes the monopoly by swarming everyone.
- **General Observations**: the system built is unpredictable and non-deterministic. The erratic nature of the spawning of the resources led to completely random initial positioning of the bases, which led to the ones more alienated to

thrive and prosper due to having less obstacles in the way, while the more cluttered ones suffered from the restraining given by the continuous fighting. The generally best strategy resulted is to avoid conflict.

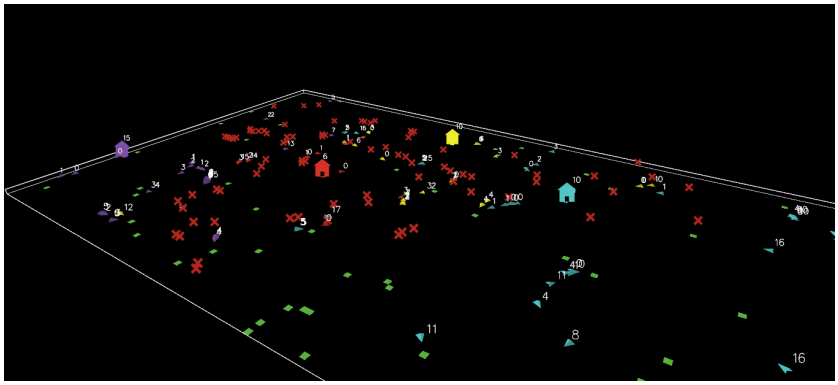

Fig. 5. 3D view of the simulation with each agent showing their currently held resources.

4.1 Flaws and Weaknesses

The system built is not perfect; some procedures are simplified to the point of being elementary - for the sake of being fast enough for the simulation to run smoothly - and even through relentless polishing, the simulation sometimes shows errors of some sort (hopefully it's Netlogo itself and not the code). Among the weaker points and the sporadically encountered bugs that have been found, these are the ones that stand the most:

- The movement is very basic and straightforward, with no searching, obstacle-avoidance or path-finding algorithms. A better and more fine-tuned version could include building/destroying walls, optimizing which path to follow to gather more resources in less time or to avoid enemies, and coalitions.
- It has happened on rare occasions that the faction resources have dropped under zero. This is physically impossible and should not be permitted by the code, so it's probably a memory allocation issue in the archer code, but still breaks the plot by dropping steeply and randomly - should be mainly fixed but still possible.
- Agents sometimes get stuck when defending from an attacking archer by going back and forth on the same adjacent patches without ever starting the fight. This has been mostly fixed by making the fighting radius larger and iteratively changing the speed of the defender, so to avoid missing the discrete steps jump.

- Limitations of the software used only permit simple data manipulation. With more powerful engines, agents could store past data and build Markov Chains or have a history of relationships with other factions (it will be discussed in the next segment).

4.2 Future Improvements

Numerous additional elements can be introduced to the simulation to add more levels of complexity. Each would bring something new to the overall social dynamics of the game but would also require additional computational power, a clear understanding of the underlying mechanics to be able to extrapolate useful data from the experiments, and most importantly a lot more time to dedicate to the matter.

- Communication has been simplified and reduced to the simple crossing of two agent's path or the inclusion in one's visual area, while usually in most Multi Agent and Distributed Complex Systems simulations the communication graph is not fully connected, and in various BDI experiments messages are exchanged with privacy protocols and confirmations of receipt in a P2P manner. Although complex and heavy on computation, it gives an ulterior element to play with: data is not absolute but can travel in different ways and agents of the same colony can have different beliefs until matched by full connection.
- The factions could keep track of alliances in a matrix data type. Factions can become friendly if some parameters are met and hostile if they aren't (for example: helping out invaders is a friendly act while killing a soldier is hostile). In this case, the distributed communication setting from earlier seems convenient to give way to interesting scenarios (example of a betrayal: an agent gets killed but no other of its faction sees it so the relationships between the murderous faction and the victim's one remain amicable).
- Giving the agents the chance to record the history between the factions and then re-use it to validate alliances and quarrels could make for very interesting simulations in the long term: being able to observe the behaviour of colonies that have helped each other out or suddenly break out of their alliance and go to war can branch into game theory and machine learning as well. Measurement could go through space, other than time, by tracking out heatmap of movements (something similar is done with fight markers in this code).
- More aspects of the agent's behaviour can come alive, or become tunable as well: the intent to attack, defend, fight and return to the base could also depend on internal mechanisms, such as the total number of resources held during its life, the enemies killed, the allies around one's self. Example: a *greed* parameter that sets the bar on when an agent is going to want to fight another to steal its resources (apart from all other conditions), or a *leader* token signalling which agent is stronger/has killed more enemies, so that others can group under him with ease (flocking dynamics).

References

1. Jonathan Wiens, *Using BDI-extended NetLogo Agents in Undergraduate CS Research and Teaching*, 2013.
2. Ilias Sakellariou, *Enhancing NetLogo to Simulate BDI Communicating Agents*, 2008.
3. Evaggelos Raptis, *BDI Agents in Games*, 2014.
4. Andrea Traldi, Francesco Bruschetti, *Real-Time BDI Agents: a model and its implementation*, 2022.
5. Andrea Omicini, *Multi-Agent Systems Lecture Slides*, 2024.