**Universidade de Brasília**
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Integrating Deep Reinforcement Learning to GAMA Platform with a Multi-agent Model of Common-pool Resource Appropriation

Guilherme Mendel de Almeida Nascimento

Artigo apresentado como requisito parcial para conclusão do Bacharelado em Ciência da Computação

Orientadora
Prof. Dra. Célia Ghedini Ralha

Brasília
2023

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

# Integrating Deep Reinforcement Learning to GAMA Platform with a Multi-agent Model of Common-pool Resource Appropriation

Guilherme Mendel de Almeida Nascimento

Artigo apresentado como requisito parcial para conclusão do Bacharelado em Ciência da Computação

Prof. Dra. Célia Ghedini Ralha (Orientadora)
CIC/UnB

Prof. Dr. Li Weigang
CIC/UnB

MSc. Aurélio Ribeiro Costa
STI/STF

Prof. Dr. Marcelo Grandi Mandelli
Coordenador do Bacharelado em Ciência da Computação

Brasília, 16 de fevereiro de 2023

# Integrating Deep Reinforcement Learning to GAMA Platform with a Multi-agent Model of Common-pool Resource Appropriation

Guilherme Mendel de Almeida Nascimento

Department of Computer Science, University of Brasília
Campus Darcy Ribeiro, Brasília, Federal District, Brazil
guimendeln@gmail.com

**Abstract.** Exploratory agent-based simulations are a challenging investigative area for modeling societies. Natural resource systems used by multiple agents in a society can be classified as common-pool resources. Despite a broad agreement that multi-agent models of common-pool resource appropriation are accurate representations of aspects of human behavior, models of independent learning agents in complex real-time environments (e.g., game-like) are underrepresented within the repertoire of solutions available to agent-based simulation developers. To address this, we present the integration of deep reinforcement learning (DRL) algorithms to the well-known GAMA simulation platform, illustrated with a multi-agent model of common-pool resource appropriation called *The Commons Game*. This integration is implemented using an external WebSocket server to provide an interface for the execution of DRL algorithms. Our work aims to contribute to agent-based simulation developers that use the GAMA platform with models of learning agents.

## 1 Introduction

Agent-based modeling and simulation (ABMS) is a powerful approach to model systems comprised of interacting autonomous agents for real-world applications [3]. Applications examples range from modeling the stock market, supply chains, and consumer markets, to predicting the spread of epidemics, mitigating the threat of bio-warfare, and understanding the factors responsible for the fall of ancient civilizations. Some contend that ABMS "is a third way of doing science" and could augment traditional deductive and inductive reasoning as discovery methods [16].

ABMS with real-world applications that involve natural resource systems used by multiple individuals is classified as common-pool resources [19]. The common-pool resource appropriation (CPR) problem is well known within multi-agent social dilemmas, especially considering non-cooperative or self-interested agents where socially positive equilibria are hard to achieve. Although game theory research has, for decades, investigated aspects of human behavior that make multi-agent models possible, when it comes to complex real-time or video game-like environments, such models struggle to generate accurate predictions ( [24], [12], [25], [10]). This is especially true when considering the spatial and temporal resource dynamics of CPR models ( [4], [7], [6]). Thus, to model the behavior of groups of independent learning agents in a CPR game, deep reinforcement learning (DRL) can highlight the importance of learning through trial and error ( [18], [5]). DRL has significant applications in real-time systems such as self-driving cars, traffic light control, automated robots, healthcare, disease prediction, energy consumption reduction, resource management in computer clusters and cloud, online recommendation, and web system configuration ( [17], [2], [13]).

There is a wide range of available tools for ABMS. A comparative literature survey of the state-of-art is presented in [1]. Although the further investigation into this topic is stimulated, in this work we focus on the GAMA simulation platform [23].[1] GAMA is under a GNU General Public License, meant for building spatially explicit agent-based simulations. It relies on the notion of agent species with attributes and actions which interact to form experiments. GAMA is already an engine for several high-impact projects, such as [8] to simulate the impacts of social COVID-19 interventions and [9] to model agricultural territories to assess agricultural activity impacts.

The platform features its own modeling language— GAML (**GA**ma **M**odeling **L**anguage) is an agent-oriented language that extends object-oriented programming approach with powerful concepts like skills, declarative definitions, and agent migration, allowing for quick model expressivity. But GAML lacks support for the development of sophisticated learning agents that require machine learning. Although this may not be a problem for most ABMS developers, it imposes a limit on the experimental requirements of the platform considering autonomous agents. Thus, in this work, we define the research question: How would one design learning agents in GAMA?

With this question in consideration, our work aims to contribute to ABMS developers presenting the design and implementation of the integration of DRL to the GAMA platform by using an external WebSocket server as an interface to the GAMA application. We called the WebSocket server the *brain server*. The message protocol is presented, and the server's life cycle is depicted through diagrams. The integrated solution is illustrated with the game-like multi-agent model of CPR *The Commons Game* [20]. Though the model is designed for DRL, one could easily adapt it to other machine-learning algorithms.

The rest of the manuscript presents a short background overview in Section 2, the integrated DRL to GAMA platform presentation in Section 3, the experimental results in Section 4, and the conclusion and future work in Section 5.

[1] For a subset of the scientific papers that have been written either about GAMA or using the platform as an experimental/modeling support see `https://gama-platform.org/wiki/References`

## 2 Background

In this section, we give an overview of reinforcement learning (RL), DRL, and multi-agent reinforcement learning (Section 2.1), GAMA platform modeling functionalities (Section 2.2), and *The Commons Game* (Section 2.3). Please note the notation adopted in this paper is the same as in [22].

### 2.1 DRL overview

Machine learning comprises three fields: supervised, unsupervised, and semi-supervised learning. According to [22], RL is another category of machine learning, defined not by characterizing learning methods, but by characterizing a learning problem.

Whereas supervised learning requires data collections to generalize labels and find hidden patterns, RL needs no data and is concerned with learning from an agent's experience. It essentially maps situations to behavior with the aid of its knowledge of the environment's dynamics, which it acquires through the agent's exploration. This process can be formalized as a Markov Decision Process (MDP), in which an agent interacts with an environment in discrete time steps, giving rise to a new environment *states* and *rewards*.

In [22], a finite MDP is defined by a time step $t \in \mathbb{N}$, and the environment is modeled through a state $S_t \in \mathcal{S}$. Based on this state, the agent issues an action $A_t \in \mathcal{A}(s)$, which then results in a new state for time step $t+1$, as well as a reward $R_{t+1} \in \mathcal{R} \subset \mathbb{R}$. The probabilities of state $S_t$ and reward $R_t$ occurring at time step $t$ are given by the model as the stochastic functions $p(s'|s,a)$ and $r(s,a)$ of time step $t-1$'s state and action: $S_{t-1}$ and $A_{t-1}$, respectively. Figure 1 illustrates the agent-environment interaction cycle of traditional RL, where we replaced $S_t$ by the state observation $O_t$ of the agent as used in this work.

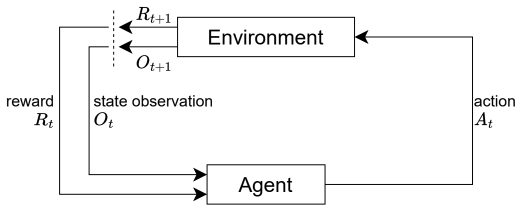

**Fig. 1.** Agent-environment interaction cycle of RL algorithm (adapted from [22]).

The mechanism responsible for mapping the observed state to an agent's action is called a *policy* $\pi(a|s)$ — the probability distribution of $\mathcal{A}$ for each state in $\mathcal{S}$. One may calculate the *expected return* $G$ that represents the expected total reward accrued by starting at some state $s$ and subsequently following a given policy $\pi$, with the *value function* $v_\pi(s)$. In RL, the agent's objective is to find a policy that best approximates the *optimal policy* $\pi*$, which, for any given state $s$, elects $a$ that maximizes the resultant $G$.

As the state space grows arbitrarily, tabularly computing values for each state becomes unfeasible, which calls for the adoption of approximation methods. One common way to achieve this is the combination of RL and Deep Learning (DL). DL is the most widely used computational approach in machine learning — it achieves outstanding results on complex cognitive tasks and even beats human performance when given access to massive amounts of data. DL employs transformations and graph technologies to build multi-layer learning models, and its most utilized network type is convolutional neural networks (CNNs) [2].

DL can be applied to RL by taking an episode's state observation and its action values as the input data and target output, respectively. DRL strives to approximate generated values to the ones encountered throughout the simulation, one such example being the classic Deep Q-learning (DQL) algorithm. In [20], authors formalize DQL using $O(s, i)$ as the observation of state $s$ for an agent $i$, and $Q^i(s, a)$ the action-value function of this state and action $a$. The policy is defined as $\pi^i(a|O(s, i)) = (1 - \epsilon)\mathbf{1}_{a=\arg_a \max Q^i(s,a)} + \frac{\epsilon}{|A^i|}$.

Applying an algorithm designed for single-agent environments, such as DQL, to multi-agent environments is known to break the MDP assumption [14]. But the work of [20] achieves good results with this application nevertheless, setting a reasonable precedent for this paper's choice of using DQL as well.

## 2.2 GAMA Platform

Agent-based simulation models in the GAMA platform are specified in the GAML language.[2] One must first define the species that will interact in the simulation by giving them attributes, actions, and reflexes. Attributes are the data each agent owns, such as name or size. Actions are essentially procedures (or functions) that can be called, and reflexes are procedures that execute each cycle with a given probability (by default 100%).

After specifying its species, one has to indicate which species comprise each experiment, set up parameters, such as species count, and specify how to display the experiment to be ready to run it. Species can also optionally be assigned *skills*, built-in models that group related variables and actions. One of particular interest to this paper's proposal is the **network** skill, which provides agents the ability to send and receive messages through various protocols, including WebSocket.

The *global* species is special: it defines the attributes and actions that describe the simulation's environment and automatically inherits from several built-in variables and actions. All other species are first instantiated by its *init* procedure, and also have direct access to its attributes and actions.

[2] There is a tutorial designed to learn GAML available in `https://gama-platform.org/wiki/LearnGAMLStepByStep`.

## 2.3 The Commons Game

Although it is a well-studied issue, humanity still faces the problem of CPR. Its simulation attempts have so far been limited by the complexity of the environments, which renders its analysis far too complex to fathom. In [20], the authors propose a model which places agent learning in the center. Agents learn through trial and error how to respond to the environment in a process that, collectively, produces a population ever so better suited to avoid resource depletion. They illustrate this with *The Commons Game*, wherein a group of agents competes in a 2-dimensional grid world to collect apples, which will only respawn as long as there are other apples in the vicinity.

In this work, we reproduced *The Commons Game* using their CPR model to provide a hands-on example of functional integration of DRL algorithms to the GAMA platform. The game arises in a 2-dimensional grid world with a partially observable, stochastic, episodic, dynamic, discrete, and multi-agent environment. It has a single type of agent referred to as the *scavenger*.

*Scavengers* can move through the grid, change their orientation, and emit a *tag* beam, which will temporarily time-out any other *scavengers* caught in its path. They get rewards for the collection of apples, and albeit they can tag other agents, tagging yields no particular reward. A *scavenger*'s orientation defines which of the four cardinal directions it is currently facing and determines what portion of the current scenario will be visible to it and which direction to shoot the tag beam to. Note that an agent's observation is an RGB slice of the scenario, and is rotated so that the *scavenger* is, in this observation window, always facing north.

## 2.4 Social Outcome Metrics

In [20], the authors propose four social outcome metrics to summarize group behavior, calculated for each episode of the game's simulation. We briefly summarized them, as we use them for illustrative rather than evaluative purposes.

For $N$ independent agents and an episode of duration $T$, the $i$-th agent's reward sequence may be defined as $\{r_t^i | t = 1, ..., T\}$, the observation sequence as $\{o_t^i | t = 1, ..., T\}$, and the return as $R = \sum_{t=1}^{T} r_t^i$.

The *Utilitarian Metric (U)* is the sum of all agents' rewards scaled by the episode length (i.e., efficiency). The Gini coefficient [11] is used for the *Equality Metric (E)*. The *Sustainability Metric (S)* measures how late in the episode rewards were collected, on average. The *Peace Metric (P)* is the average number of untagged agent steps. Note that for a given $i$-th episode, expectations are taken over every $i$-th episode of all training suites to minimize noise.

$$U = \mathbb{E}\left[\frac{\sum_{i=1}^{N} R^i}{T}\right],\ E = \mathbb{E}\left[1 - \frac{\sum_{i=1}^{N}\sum_{j=1}^{N} |R^i - R^j|}{2N\sum_{i=1}^{N} R^i}\right],$$

$$S = \mathbb{E}\left[\frac{1}{N}\sum_{i=1}^{N} t^i\right],\ \text{where } t^i = \mathbb{E}[t|r_t^i > 0],$$

$$P = \frac{\mathbb{E}\left[NT - \sum_{i=1}^{N}\sum_{t=1}^{T} I(o_t^i)\right]}{T},\ \text{where } I(o) = \begin{cases} 1 & \text{if } o = \text{time-out observation.} \\ 0 & \text{otherwise.} \end{cases}$$

## 3 The Integration Approach

The high-level cycle of RL (Figure 1) was used as inspiration to build a similar interaction cycle between the GAMA platform and the *brain server* (respective counterparts of environment and agent). Figure 2 illustrates this correspondence. Note that the *brain server* processes a state observation $O_t$ and reward $R_t$ message from a *scavenger* agent in GAMA and returns an action label $A_t$.

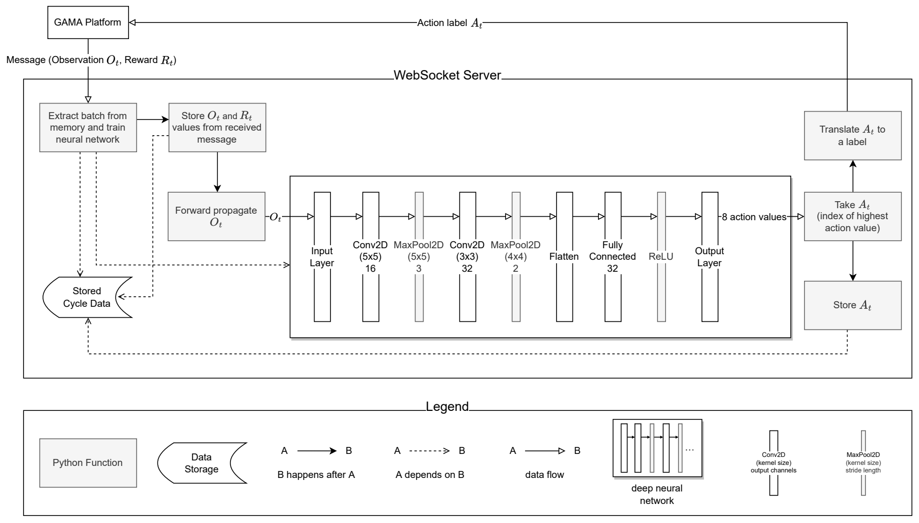

**Fig. 2.** DRL (*brain server*) and GAMA platform integration with WebSocket.

Thus, the integration consists of two parts: the GAMA platform, which stores the environment data and models its dynamics, and the *brain server* that is responsible for implementing the RL algorithm, storing the index of the highest action value of *scavengers*, as well as providing an interface for mapping rewards and state observations to actions. Communication between those parts is possible through the WebSocket protocol, and the text messages are in JavaScript Object Notation (JSON) format [15].

The GAMA platform is responsible for starting the communication session. It must first inform the current simulation *context*, and each agent informs their *id* (identification token). Such a message triggers the *brain server* to load the neural network parameters previously stored for this context for each connected agent. This marks the only need for context identification: to allow the scoping of different simulation contexts such that agents never share learned parameters through different contexts.

After all identification messages are sent the main RL loop takes place for each agent. The GAMA platform sends a state observation with a reward for the previous cycle, and the *brain server* responds with action. The reward sent in the first cycle is discarded by the *brain server*. Once the simulation is over, the GAMA platform shuts down the WebSocket connection, at which point the *brain server* collects its instantiated agents and stores the new neural network parameters. Figure 3 illustrates these interactions with a UML sequence diagram. Please note that the scavengers inside the GAMA platform received their observations from the environment, and should be interpreted as forwarding these observations in the diagram, instead of generating them.

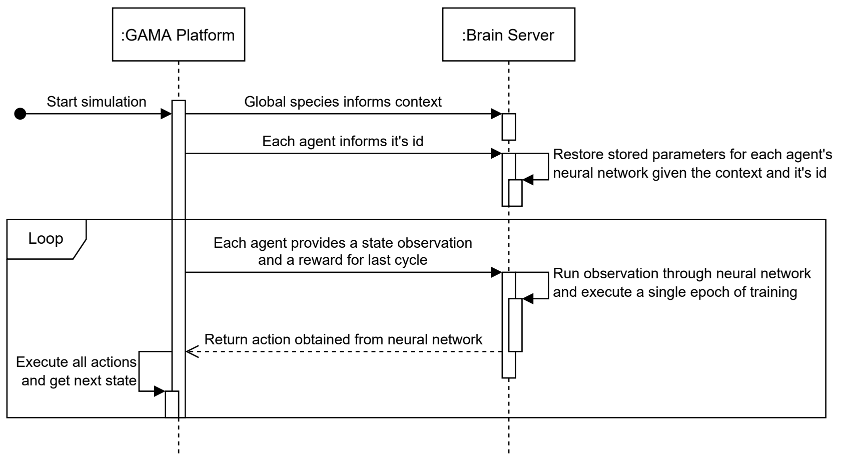
:GAMA Platform
:Brain Server
Start simulation
Global species informs context
Each agent informs it's id
Restore stored parameters for each agent's neural network given the context and it's id
Loop
Each agent provides a state observation and a reward for last cycle
Run observation through neural network and execute a single epoch of training
Return action obtained from neural network
Execute all actions and get next state

**Fig. 3.** UML sequence diagram describing the event sequence that defines the GAMA platform and *brain server* interactions.

## 3.1 Message Syntax

*Brain server* messages are simple strings that label the *scavenger* agent's next action using JSON format. The GAMA platform messages are of three types: context identification, agent identification, and action request. The context identification message syntax (line 1) is sent by the *global* species when the GAMA simulation starts. The agent identification message is the first message every agent sends (lines 2 and 3), an object with two keys: *id*, pointing to its unique identifier token string, and *connect*, which should always hold true.

```
{
1. "context": <string>
}
```

```
{
 2. "id": <string>,
 3. "connect": true
}
```

The action request message must provide a *request* key, an object with two keys: *state*, holding a description of this agent's current observation, and *reward*, holding the numeric reward of the previous cycle.

```
{
  "id": <string>,
  "request": {
      "state": <any>,
      "reward": <number>
  }
}
```

## 3.2 The Commons Game Implementation

In *The Commons Game* [20], the authors propose six scenarios. In this work, we focus on the second scenario, which features an open area map with resources scattered arbitrarily. The GAMA contexts in the implementation are the game *scenarios*, which define the starting state of the environment. The *scavenger* spawn points are focused on the lower right section of the map, as illustrated in Figure 4. Notice how the *scavengers* start all facing north. In this scenario, it would be impossible for an agent to single-handled exhaust the resource pool, but when all agents harvest simultaneously, depletion becomes much more likely.

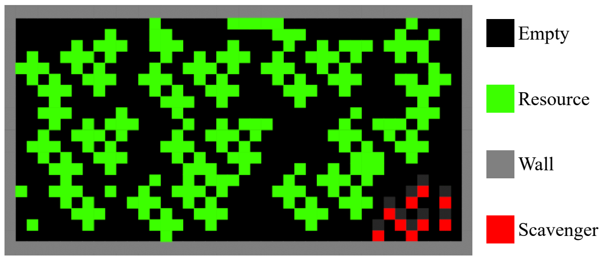

**Fig. 4.** Starting state of this paper's unique scenario.

In GAML, there are no classes but *species* for which all properties are public. However, one can make an analogy between a species and a class in object-oriented programming (OOP). Figure 5 presents the UML class diagram of the GAMA implementation with seven species: *scavenger, resource, laser tag, wall, gridcell, json*, and *global*.

The *scavenger* orientation is indicated visually by an adjacent darkened cell to the same direction it is facing, and it moves and tags relative to it. There are four types of actions: moving in cardinal directions, where it can move one

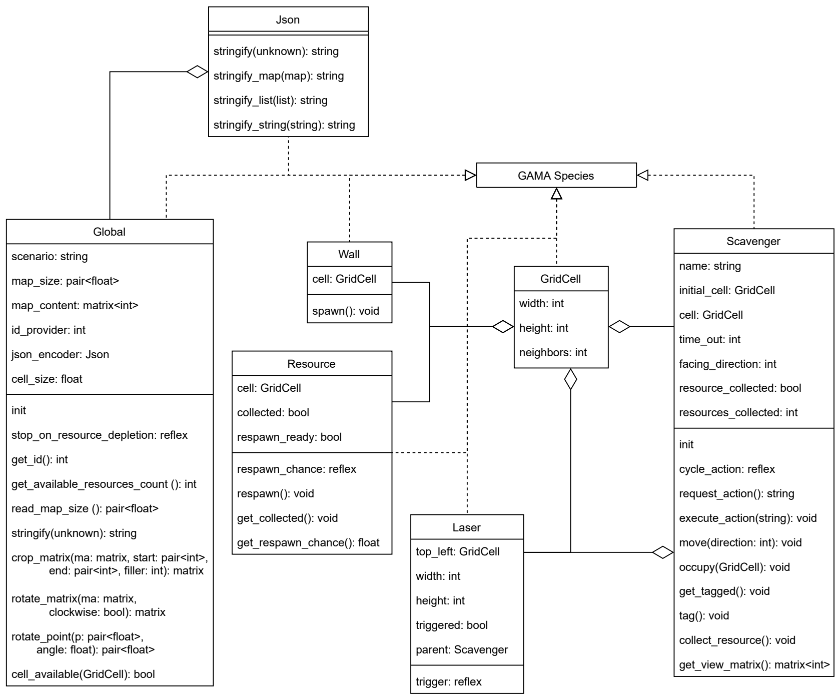

**Fig. 5.** UML-like description of the GAMA model.

cell ahead, behind, or sideways (left and right); changing its orientation by $90^o$, where it can turn to one of its sides; tagging, where it emits a time-out beam ahead; and idle, where it does nothing.

The *scavenger* has a reflex for messaging the *b*rain server with its state observation window and reward for the previous cycle and receiving the action to execute at that cycle. Tagged *scavengers* are removed from the scenario for 25 cycles and reinserted at their initial positions. Each member of the resource species sits idle until it is collected, at which point it's removed from the scenario. They have a chance to reappear every cycle. This probability is as defined in [20], and is proportional to the number of other resources within 2 cells: one or two apples give 1%, three to four apples give 5%, and over four apples 10%.

The *wall* species has no behavior but serves as an impassable cell to *scavengers*. The *tag* species is instantiated by *scavengers*: it gets destroyed on its second cycle of life lasting a single iteration. In this iteration, it finds all other *scavengers* inside its perimeter and times them out.

The *json* species is never instantiated but provides methods to convert data into the JSON format and is used by other species that need to send messages. The *gridcell* species provides other species access to cells. A cell is a simple tuple of integers representing the discrete coordinates of the grid world.

The *global* species has two functions: initializing the world from a map description file and storing the *map*, a matrix the size of the scenario's grid. Each cell stores 0 if it's empty, 1 for a resource, 2 for a *scavenger*, and 3 for a wall. With the aid of this map, *scavengers* know when a move would be illegal (e.g., walking into a wall), know when they walk into a resource and trigger a collection, and derive their observation windows.

The *brain server* is implemented with the PyTorch framework. There are two classes, one for a neural network, which implements DL, and one for a *scavenger*, which implements RL and uses the other class. For each connected *scavenger*, the server instantiates the *scavenger* class, loads the neural network parameters for the *scavenger*'s id and current scenario, and interfaces all of its subsequent requests to it, returning the resulting action. Before sending back the action, the neural network randomly samples 32 entries with which to execute a single epoch of training.

When a *scavenger* sends its state observation $O_t$ and last cycle's reward $R_t$, the neural network obtains the resulting action $A_t$ and stores these three pieces of data, constructing a memory of initial observation, action, resulting reward, and resulting observation. Notice how one row's resulting observation is the next row's initial observation, as in the form:

$$\begin{array}{l} \cdots \\ O_{t-1}, A_{t-1}, R_t, O_t \\ O_t, A_t, R_{t+1}, O_{t+1} \\ \cdots \end{array}$$

The work presented intends to cooperate with open science as it is available in the GitHub repository. We invite the curious reader to fork our repositories and try out different configurations of the Python server.[3]

## 4 Experimental Results

We simulated each episode with 1000 time-steps to demonstrate the integration of DRL with the GAMA platform. GAMA simulations were run in the default GUI mode from which the screenshots were taken, and the *batch* mode to yield faster executions. The majority of the simulation time was spent on neural network training calls, instead of WebSocket message exchanges. Table 1 presents the experimental setup parameters.

The right image of Figure 6 presents a screenshot taken at cycle 105 of the simulation. The left image illustrates the observation window of the leftmost *scavenger* of the screenshot. A *scavenger* $i$ perceives the world through a slice of the grid-world $O(i) \in \mathbb{R}^{3\times21\times20}$ (i.e., RGB matrix). Each world object is expressed through a color, where black means an empty cell, green is a resource (i.e., apple), red another *scavenger*, gray a wall, and blue the observing *scavenger*. Out-of-bounds cells are black, and a *scavenger* does not see emitted tags.

The screenshot of Figura 6 presents the moment of the first resource collection in the first episode. The *scavenger* in blue performed the collection and its next

[3] GitHub for the GAMA platform `https://github.com/guiMendel/scavengers-simulation` and the *brain server* implementation: `https://github.com/guiMendel/scavengers-backend`.

**Table 1.** Parameters used in the experiments. The justification is empty when extracted as-is from [20]. The *Scavenger* Vision Range is equal to the vision rectangle Width minus one.

| Parameter | Value | Justification |
|---|---|---|
| *Scavenger* Vision Rectangle Width | 21 | Extracted from observation space type dimensions |
| *Scavenger* Vision Range | 20 | - |
| Resource Collection Reward | 1 | - |
| Tag Width | 5 | - |
| Tag Range | 20 | - |
| Time-out Duration | 25 | - |
| Max Cycles per Episode | 1000 | - |
| Activation Function | ReLU | - |
| Conv2D Padding | Same | Conserves more of the little initial resolution |
| Loss Function | Cross Entropy | Default for multi-class classification |
| Optimizer | Adam | Performed best |
| Agent Policy | $\epsilon - greedy$ | - |
| Maximum and Minimum $\epsilon$ | 1.0 and 0.1 | - |
| Discount Rate $\gamma$ | 0.99 | - |

action was moving ahead. One might take notice of the big yellow rectangle on the left side of the scenario. It is a tag being emitted by the *scavenger* on its right, the same one whose observation window is displayed on the left side of the image, at the exact time of the screenshot, though it entirely missed any other *scavengers*. It's also evident that there are only four *scavengers*, less than the initial 12, due to the other eight being in time-out for having been tagged in previous cycles.

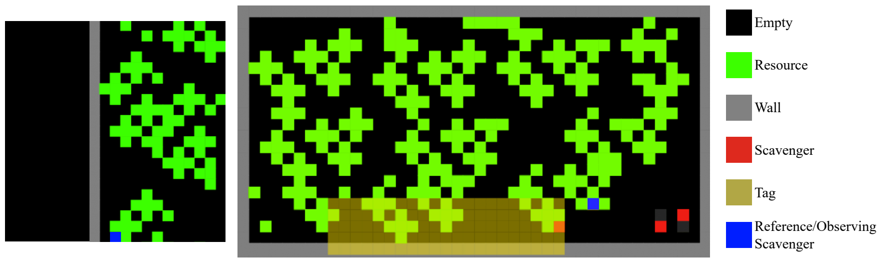

**Fig. 6.** Screenshot of the scenario at the first resource collection in the first episode. The observing *scavenger*'s figure is on the left, and the reference scavenger's figure is on the right.

Table 2 presents the social metrics for this episode where the minimum value is zero for all metrics. The low *Utilitarian Metric (U)* indicates a poor resource collection rate, while *Equality Metric (E)* indicates most collections were done by a small group of *scavengers*. Few collections resulted in a high *Sustainability Metric (S)* result, while a low *Peace Metric (P)* indicates a reason for such low equality outcome, as *scavengers* tagged each other too much, leaving only a few to harvest.

## 5 Conclusion

In this work, we investigated how one would design learning agents in GAMA. As a result, the design and implementation of DRL integration to the GAMA plat-

**Table 2.** Social metric values observed in the first episode.

| Social Metric | Maximum Value | Observed Value |
|---|---|---|
| Utilitarian (U) | 12.0 | 0.012 |
| Equality (E) | 1.0 | 0.111 |
| Sustainability (S) | 1000.0 | 905.1 |
| Peace (P) | 12.0 | 4.812 |

form were presented. Such integration was illustrated with a CPR model called *The Commons Game* presented by [20]. The main contribution is to present a hands-on example of the functional integration of machine learning algorithms to the GAMA platform, of special interest to ABMS developers. We hope this work might serve as inspiration for other projects in GAMA involving RL, DRL, neural networks, and other machine learning techniques.

In future work, there are opportunities to replicate *The Commons Game* with different RL algorithms such as Proximal Policy Optimization [21]. Also, the development of a plugin to fit the particular needs of DRL integrated into GAMA would be interesting.

## References

1. Sameera Abar, Georgios K. Theodoropoulos, Pierre Lemarinier, and Gregory M.P. O'Hare. Agent based modelling and simulation tools: A review of the state-of-art software. *Computer Science Review*, 24:13–33, 2017.
2. Laith Alzubaidi, Jinglan Zhang, Amjad J. Humaidi, Ayad Al-Dujaili, Ye Duan, Omran Al-Shamma, J. Santamaría, Mohammed A. Fadhel, Muthana Al-Amidie, and Laith Farhan. Review of deep learning: concepts, cnn architectures, challenges, applications, future directions. *Journal of Big Data*, 8(1):53, 2021.
3. E. Bonabeau. Agent-based modeling: methods and techniques for simulating human systems. volume 99 of *No. Suppl. 3*, pages 7280–7287. The National Academy of Sciences, 2002.
4. F. Bousquet, O. Barreteau, P. d'Aquino, M. Etienne, S. Boissau, S. Aubert, C. Le Page, D. Babin, and J.C. Castella. Multi-agent systems and role games: collective learning processes for ecosystem management. In *Complexity and ecosystem management. The theory and practice of multi-agent systems*. Edward Elgar, 2002.
5. Lorenzo Canese, Gian Carlo Cardarilli, Luca Di Nunzio, Rocco Fazzolari, Daniele Giardino, Marco Re, and Sergio Spanò. Multi-agent reinforcement learning: A review of challenges and applications. *Applied Sciences*, 11(11), 2021.
6. Cássio Giorgio Couto Coelho and Célia Ghedini Ralha. MASE-EGTI: an agent-based simulator for environmental land change. *Environ. Model. Softw.*, 147:105252, 2022.
7. Aisha D. Farooqui and Muaz A. Niazi. Game theory models for communication between agents: a review. *Complex Adaptive Systems Modeling*, 4(1):13, 2016.
8. Benoit Gaudou, Nghi Quang Huynh, Damien Philippon, Arthur Brugière, Kevin Chapuis, Patrick Taillandier, Pierre Larmande, and Alexis Drogoul. Comokit: A modeling kit to understand, analyze, and compare the impacts of mitigation policies against the covid-19 epidemic at the scale of a city. *Frontiers in Public Health*, 8, 2020.

9. Benoit Gaudou, Christophe Sibertin-Blanc, Olivier Therond, Frédéric Amblard, Yves Auda, Jean-Paul Arcangeli, Maud Balestrat, Marie-Hélène Charron, Etienne Gondet, Yi Hong, Romain Lardy, Thomas Louail, Eunate Mayor, David Panzoli, Sabine Sauvage, José Sánchez Pérez, Patrick Taillandier, Van Nguyen, Maroussia Vavasseur, and Pierre Mazzega. The maelia multi-agent platform for integrated analysis of interactions between agricultural land-use and low-water management strategies. In $14^{th}$ *International Workshop on Multi-Agent-Based Simulation (MABS'13)*, 2013.
10. Philipp Geiger and Christoph-Nikolas Straehle. Learning game-theoretic models of multiagent trajectories using implicit layers. In $35^{th}$ *AAAI Conference on Artificial Intelligence, AAAI 2021,* $35^{rd}$ *Conference on Innovative Applications of Artificial Intelligence, IAAI 2021, The* $11^{th}$ *Symposium on Educational Advances in Artificial Intelligence, EAAI 2021, Virtual Event, February 2-9, 2021*, pages 4950–4958. AAAI Press, 2021.
11. C. Gini. *Variabilità e mutabilità: contributo allo studio delle distribuzioni e delle relazioni statistiche. [Fasc. I.].* Studi economico-giuridici pubblicati per cura della facoltà di Giurisprudenza della R. Università di Cagliari. Tipogr. di P. Cuppini.
12. Robert P. Goldman, David J. Musliner, Mark S. Boddy, Edmund H. Durfee, and Jianhui Wu. "unrolling" complex task models into mdps. In *Game Theoretic and Decision Theoretic Agents, Papers from the 2007 AAAI Spring Symposium, Technical Report SS-07-02, Stanford, California, USA, March 26-28, 2007*, pages 23–30. AAAI, 2007.
13. Tanmoy Hazra and Kushal Anjaria. Applications of game theory in deep learning: a survey. *Multimedia Tools and Applications*, 81(6):8963–8994, 2022.
14. Guillaume J. Laurent, Laëtitia Matignon, and N. Le Fort-Piat. The world of independent learners is not markovian. 15(1):55–64, jan 2011.
15. Teng Lv, Ping Yan, and Weimin He. Survey on json data modelling. *Journal of Physics: Conference Series*, 1069(1):012101, aug 2018.
16. Charles M. Macal and Michael J. North. Agent-based modeling and simulation. In *Winter Simulation Conference*, WSC '09, page 86–98. Winter Simulation Conference, 2009.
17. Amirhosein Mosavi, Yaser Faghan, Pedram Ghamisi, Puhong Duan, Sina Faizollahzadeh Ardabili, Ely Salwana, and Shahab S. Band. Comprehensive review of deep reinforcement learning methods and applications in economics. *Mathematics*, 8(10), 2020.
18. Thanh Thi Nguyen, Ngoc Duy Nguyen, and Saeid Nahavandi. Deep reinforcement learning for multiagent systems: A review of challenges, solutions, and applications. *IEEE Transactions on Cybernetics*, 50(9):3826–3839, 2020.
19. Elinor Ostrom. Reformulating the commons. *Ambiente & Sociedade*, (Ambient. soc., 2002 (10)), Jan 2002.
20. Julien Pérolat, Joel Z Leibo, Vinicius Zambaldi, Charles Beattie, Karl Tuyls, and Thore Graepel. A multi-agent reinforcement learning model of common-pool resource appropriation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
21. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. 2017.
22. Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction.* The MIT Press, $2^{nd}$ edition, 2018.

23. Patrick Taillandier, Benoit Gaudou, Arnaud Grignard, Quang-Nghi Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon, and Alexis Drogoul. Building, composing and experimenting complex spatial models with the gama platform. *GeoInformatica*, 23(2):299–322, 2019.
24. José M. Vidal and Edmund H. Durfee. Predicting the expected behavior of agents that learn about agents: The clri framework. *Autonomous Agents and Multi-Agent Systems*, 6:77–107, 2000.
25. James R. Wright and Kevin Leyton-Brown. Level-0 meta-models for predicting human behavior in games. In *Proceedings of the Fifteenth ACM Conference on Economics and Computation*, EC '14, page 857–874, New York, NY, USA, 2014. Association for Computing Machinery.