gym-DSSAT: a crop model turned into a Reinforcement Learning environment

Romain Gautron, Emilio J. Padrón, Philippe Preux, Julien Bigot,
Odalric-Ambrym Maillard, David Emukpere

HAL Id: hal-03711132
https://inria.hal.science/hal-03711132v4

Submitted on 6 Sep 2022

Inria

gym-DSSAT: a crop model turned into a Reinforcement Learning environment

Romain Gautron, Emilio J. Padrón, Philippe Preux, Julien Bigot, Odalric-Ambrym Maillard, David Emukpere

gym-DSSAT: a crop model turned into a Reinforcement Learning environment

Romain Gautron[*], Emilio J. Padrón[†], Philippe Preux[‡], Julien Bigot[§], Odalric-Ambrym Maillard[¶], David Emukpere[‖]

Project-Team Scool

Abstract: Addressing a real world sequential decision problem with Reinforcement Learning (RL) usually starts with the use of a simulated environment that mimics real conditions. We present a novel open source RL environment for realistic crop management tasks. `gym-DSSAT` is a `gym` interface to the Decision Support System for Agrotechnology Transfer (`DSSAT`), a high fidelity crop simulator. `DSSAT` has been developed over the last 30 years and is widely recognized by agronomists. `gym-DSSAT` comes with predefined simulations based on real world maize experiments. The environment is as easy to use as any `gym` environment. We provide performance baselines using basic RL algorithms. We also briefly outline how the monolithic DSSAT simulator written in Fortran has been turned into a Python RL environment. Our methodology is generic and may be applied to similar simulators. We report on very preliminary experimental results which suggest that RL can help researchers to improve sustainability of fertilization and irrigation practices.

Key-words: crop management, crop model, agriculture, Reinforcement Learning, DSSAT, OpenAI gym, Python

[*] AIDA, Univ Montpellier, CIRAD, Montpellier, France & CGIAR Platform for Big Data in Agriculture, Alliance of Bioversity International and CIAT, Km 17, Recta Cali-Palmira 763537, Colombia
romain.gautron@cirad.fr
[†] UDC–Computer Architecture Group & CITIC (Center for ICT Research) & Edif. Área Científica, Campus Elviña S/N 15071, A Coruña, Spain
emilio.padron@udc.gal
[‡] Université de Lille, CNRS, Inria, F-59650 Villeneuve d'Ascq, France
philippe.preux@inria.fr
[§] Université Paris-Saclay, UVSQ, CNRS, CEA, Maison de la Simulation, 91191, Gif-sur-Yvette, France
julien.bigot@cea.fr
[¶] Université de Lille, Inria, CNRS, Centrale Lille UMR 9189 – CRIStAL, F-59000 Lille, France
odalric.maillard@inria.fr
[‖] Work done at Inria, F-59650 Villeneuve d'Ascq, France

gym-DSSAT : un modèle de cultures converti en un environnement d'apprentissage par renforcement

Résumé : La résolution d'un problème de décision séquentielle en conditions réelles s'appuie très souvent sur l'utilisation d'un simulateur qui reproduit ces conditions réelles. Nous introduisons un nouvel environnement pour l'apprentissage par renforcement (AR) qui propose des tâches d'apprentissage réalistes pour la conduite de cultures. `gym-DSSAT` est une interface `gym` avec le simulateur de cultures Decision Support System for Agrotechnology Transfer (`DSSAT`), un simulateur de haute fidélité. `DSSAT` a été développé durant les 30 dernières années et est largement reconnu par les agronomes. `gym-DSSAT` propose des simulations prédéfinies, basées sur des expérimentations au champ avec du maïs. L'environnement est aussi simple à utiliser que n'importe quel autre environnement `gym`. Nous proposons des performances de base dans l'environnement en utilisant des algorithmes d'AR conventionnels. Nous décrivons également brièvement comment le simulateur monolithique DSSAT, codé en Fortran, a été transformé en un environnement d'AR en Python. Notre approche est générique et peut être appliquée à des simulateurs similaires. Quoique très préliminaires, les premiers résultats expérimentaux indiquent que l'AR peut aider les chercheurs à rendre les pratiques de fertilisation et d'irrigation plus durables.

Mots-clés : itinéraire technique, conduite des cultures, modèle de culture, agriculture, Apprentissage par Renforcement, DSSAT, OpenAI gym, Python

Contents

Software availability

gym-DSSAT [https://gitlab.inria.fr/rgautron/gym_dssat_pdi] is an open source software, released under a 3-Clause BSD licence. A complete documentation is available [https://rgautron.gitlabpages.inria.fr/gym-dssat-docs/]. gym-DSSAT uses a modification of the Decision Support System for Agrotechnology Transfer (DSSAT) software (https://dssat.net/) and the PDI Data Interface (PDI) library (https://pdi.dev/master/). Both DSSAT and PDI are open source software, released under a 3-Clause BSD licence. In this work, we used gym-DSSAT 0.0.7.

1 Introduction

During a growing season, farmers perform series of crop operations in their fields in order to reach production objectives. They make these decisions under uncertainty, for instance weather uncertainty. We consistently use the adjective uncertain for events with unsure realizations. Reinforcement Learning (RL) addresses such problems where an agent learns to control the evolution of an unknown and uncertain dynamical system, in order to perform a given task. In RL, addressing a complex real-world problem usually starts with the use of a high-fidelity simulator which mimics real learning conditions. We present gym-DSSAT, an RL environment based on a celebrated crop model, the Decision Support System for Agrotechnology Transfer

(`DSSAT`, Hoogenboom et al., 2019) cropping system model. In this introduction, we define the concepts of crop management, mechanistic models and RL, and show how `gym-DSSAT` ties together these notions as an RL environment for crop management tasks.

Crop management is the series of crop operations a farmer performs in a field in order to reach production objectives (Sebillotte, 1974, 1978), such as reaching at least minimum yield and grain protein content. In a field, complex physical, chemical and biological dynamical processes interact (Husson et al., 2021). Uncertain factors, such as weather events, drive the evolution of this dynamical system. In rainfed cropping systems, i.e. non-irrigated cropping systems, rainfall is a major determinant of maize yield besides nutrient availability (Mueller et al., 2012; Kadam et al., 2014; Li et al., 2019). Water stress occurring during maize flowering period may greatly reduce final grain yield (Kamara et al., 2003). Weather forecasts remain highly uncertain beyond 1-month lead time (Hao et al., 2018). Consequently, at the beginning of the growing season, harvest is highly uncertain in rainfed cropping systems.

Learning sustainable crop management practices is not a trivial task. Nitrogen fertilization requires future minimum rainfall and temperature following the application for the fertilized nitrogen to become available to plants. For an efficient nitrogen fertilizer management, available nitrogen in soil must match plant uptake, both in time and quantity (Meisinger and Delgado, 2002). Indigenous soil nitrogen supply, i.e. nitrogen supply which does not come from fertilizer applications during the current growing season, is often the first crop nitrogen supplier (Cassman et al., 2002). If total nitrogen supply is greater than total plant uptake, the excess of nitrogen will be a source of water pollution, especially with excessive rainfall. If total nitrogen supply is less than total plant nitrogen uptake, then crops may suffer nitrogen deficiency. Maize nitrogen uptake depends on growth stage, and is greater during silking (Hanway, 1963). Early and severe maize nitrogen deficiencies require earlier nitrogen supply compared to situations without such early nitrogen deficiencies (Binder et al., 2000). Thereby, designing an optimal fertilization policy is a complex task. At the time a farmer makes a decision on fertilization, future plant nitrogen uptake, temperature, rainfall and other important factors that determine nitrogen plant nutrition are uncertain and so are the consequences of nitrogen applications (Morris et al., 2018).

In order to address complex crop management decisions, such as designing fertilization or irrigation policies, scientists have developed specialized simulators. Mechanistic models are based on the laws of nature and implemented with expert knowledge to simulate physical, chemical, and/or biological processes with high fidelity (Sokolowski and Banks, 2012). These models have often evolved into complex software over decades of research and collaborative development. Crop models, often called process-based crop models, are mechanistic models which a user uses to simulate crop growth, generally at the plot scale. They model interactions among crops, soil, atmosphere, and crop operations (e.g. planting, fertilizing: see Wallach et al., 2018). As an example, the Decision Support System for Agrotechnology Transfer[1] (`DSSAT`, Hoogenboom et al., 2019) software is a high-fidelity crop model developed over the past three decades. `DSSAT` is widely recognized by agronomists for crop simulations. It is based on the daily integration of a set of partial differential equations describing the various processes at stake. For instance, nitrogen dynamics partially depend on soil dynamics (e.g. mineralization processes or soil water flows) and plant uptake (itself partially determined by physiological processes such as carbohydrate allocation in plant, depending on growth stages). Crop models can be used as exploratory tools to find best management practices. For instance He et al. (2012) identified best sweetcorn irrigation and fertilization practices in Florida, USA, based on simulations.

Reinforcement Learning (RL, Sutton and Barto, 2018) is a domain of Machine Learning (ML) and more generally Artificial Intelligence (AI) that addresses sequential decision problems under uncertainty. A decision maker, called an agent, interacts with a dynamical system called the
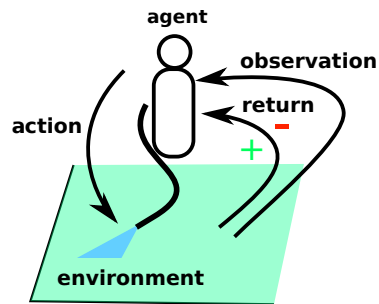
---

[1] https://dssat.net

Figure 1: The Reinforcement Learning loop. The goal of the decision maker, called the agent, is to control the evolution of a dynamical system called the environment, in order to perform a given task. Sequentially, the agent observes the environment and takes an action based on this observation. The action affects the environment, and the agent receives a return that indicates how it is performing regarding the task to perform. The process repeats until the decision series eventually ends.

environment which dynamics may be stochastic. The goal of the agent is to control the evolution of the environment in order to perform a given task. Along a series of decisions, named an episode, the agent sequentially interacts with its environment until the decision sequence eventually ends. At each time step of an episode, the agent observes its environment, decides on an action and performs it. After the agent has taken an action, the action impacts the environment, and the agent receives a return from the environment. In general, the return is a scalar value, which indicates how the agent is performing regarding his task. The agent task is to maximize, in expectation, the total reward it has collected during an episode. To do so, the agent learns from multiple episodes in a trial and error fashion. Figure 1 illustrates the interaction loop occurring during an episode. One can think of the "hot and cold" kid game where the hunter's goal is to find a hidden object in a room. Each time after the hunter has moved, if he gets closer to its target, the other kids indicate "hotter", else "colder". Based on trial and error, the hunter will try to refine its position to maximize the temperature. This process repeats until the hunter finally finds the object, and the episode ends. This simple example illustrates the concepts of RL, where the hunter is the agent, the environment is the room with the position of the hunter and the hidden object, and finally the temperature is the return. RL generalizes these concepts to the stochastic case where after each action, the environment evolution and returns are drawn from probability distributions. RL seems an relevant tool to solve crop management problems, and in particular, to address sustainable agriculture challenges (Binas et al., 2019; Gautron et al., 2022).

In the vast majority of RL applications, researchers only experiment with simulated RL environments. Nonetheless, RL algorithms ultimately intend to directly learn from real-world interactions (Sutton and Barto, 2018, Chapter 17, Section 6). Still, real-world RL applications generally begin with the use of a simulator of the environment as testbed for candidate algorithms, and/or used to facilitate real-world learning with the help of prior knowledge learned from simulated interactions. In the latter case, such knowledge transfer from imperfect simulations to reality is still challenging in practice (e.g. Golemo et al., 2018). The simulation of real conditions require complex models that accurately mimic the evolution of the environment. These simulators embed state-of-the-art and continuously evolving knowledge. Crop models are consequently of great interest to address real world crop management problems with RL.

Crop modellers historically belong to scientific communities that are generally far from the ML/RL communities. Crop models were not designed to fit into an RL interaction loop. Most

of widely used crop models (see examples in Camargo and Kemanian, 2016) internally work on a daily state update during the growing season but do not allow daily interactions with the user (be it human or virtual). A user first parametrizes a simulation, which often requires substantial domain specific knowledge. Then the simulator runs until reaching a final state which is generally crop maturity. After completion of the simulation, the user accesses partial in-season intermediate and final field states that have been internally stored during the execution. Moreover, crop modelers usually have implemented these models using Fortran, C, or C++ programming languages whereas RL researchers tend to favor Python nowadays. It follows that turning a crop simulator into a proper RL environment –without the burden of simulation setting requiring advanced expert knowledge– is challenging. To turn the monolithic `DSSAT` Fortran crop model into an RL environment, we introduce the use of the PDI Data Interface[2] (`PDI`) which allows loose coupling between C/C++/Fortran code and Python code. Beyond `DSSAT`, this approach may be used to turn other C/C++/Fortran monolithic mechanistic models into RL environments. We think this approach could reveal the value of many existing simulators as RL environments.

Section 2 presents similar works which turned crop models into RL environments. Section 3 briefly introduces mathematical and practical formalization of RL problems. Section 4 describes `gym-DSSAT` features and decision problems. In Section 5, we show the internals of `gym-DSSAT` in a nutshell. Section 6 provides an example of how to address the problem of maize nitrogen fertilization in `gym-DSSAT` as a use case, and discusses execution time and reproduciblity of experiments using `gym-DSSAT`. Finally, in Section 7 we open on limits of our current crop management environment and discuss future improvements.

2   Related work

Early seminal works addressed agricultural decision-making under uncertainty at the farm scale (Tintner, 1955; Freund, 1956). The first case of an RL agent interacting with a crop simulator in order to learn crop management is found in Garcia (1999). The author used a modification of the `Déciblé` crop model (Chatelin et al., 2005). The RL agent learned wheat sowing and nitrogen fertilization under pollution constraints. During simulations, weather series were stochastically generated. The modified version of `Déciblé` is not available anymore. In Garcia (1999), the RL agent did not manage to outperform the crop management policy of an expert. Opportunities modern RL techniques bring for learning sustainable crop intensification have been prospected by Binas et al. (2019); Gautron et al. (2022). Recently, several works directly used crop models or surrogate models as RL environments (e.g. Sun et al., 2017; Wang et al., 2020; Chen et al., 2021). However, none of these works has provided an open source and standardized crop management RL environment.

Overweg et al. (2021) proposed `CropGym`, a `gym` interface to train an agent to perform wheat nitrogen fertilization. The environment uses the Python Crop Simulation Environment (PCSE) `LINTUL3` (Shibu et al., 2010) wheat crop model. Fertilization is treated as a weekly choice of a discrete amount of fertilizer to apply. The authors successfully trained an RL algorithm to address nitrogen fertilization in their RL environment. The agent performed better than the two expert fertilization policies they considered. In the aforementioned RL environment, there is no built-in stochastic weather generation. Overfitting describes the fact an algorithm, after being trained, performs poorly in unseen situations, despite having shown good performance in training situations. In `CropGym`, simulations use a limited set of historical weather records, which may favor overfitting due to limited randomness, especially for data intensive algorithms used in deep RL (see Section 3.1).

---

[2] https://pdi.dev

Contribution `gym-DSSAT` provides both maize fertilization and irrigation RL problems. Our RL environment features a built-in stochastic weather generator. We designed `gym-DSSAT` to allow researchers to easily customize realistic crop simulations of one of the most celebrated crop simulator, the `DSSAT` crop model. `DSSAT` datasets being abundant in the literature, `gym-DSSAT` allows to mimic a wide range of real-world growing conditions. Our Python RL crop management environment provides to the user a simple standardized interface, and still results in a lightweight software. Our technical approach is generalizable to any of the 41 other crops `DSSAT` simulates, and more broadly to other C/C++/Fortan mechanistic models.

3   Formalization of RL decision problems

Sections 3.1 and 3.2 present most common mathematical formalization of RL decision problems. Section 3.3 presents `gym`, a practical pythonic interface to RL environments.

3.1   From Markov decision processes to reinforcement learning

Though RL paradigm may address a wide range of sequential decision problems, RL is usually employed to solve Markov Decision Problems (MDP). We introduce minimal materials on MDPs, for the reader to get an appropriate understanding of this paper. For an in-depth presentation of MDPs, see Puterman (1994).

Markov decision process A Markov Decision (MD) process describes the evolution of a dynamical system over discrete time. The system evolution is impacted by the actions an agent can perform. An MD process $\mathcal{M}$ is defined by a tuple $\mathcal{M} = \langle \mathcal{S}, \mathcal{A}, \mathbf{p}, \mathbf{r} \rangle$. At each decision step $t \in \{1, 2, 3, \cdots\}$, an agent observes the state of the environment $s_t \in \mathcal{S}$ and takes an action $a_t \in \mathcal{A}$, where $\mathcal{S}$ is the state space, i.e. the set of all possible states and $\mathcal{A}$ is the action space, i.e. the set of all possible actions. Each action $a \in \mathcal{A}$ leads to a stochastic transition from current state s_t to next state s_{t+1}. $\mathbf{p}$, the transition function, defines the transition dynamics: $\mathbf{p}(s, a, s')$ is the probability the environment transits to state s' if action a has been performed in state s. After performing an action, the agent receives a return, or reward, from the environment. Returns are given by the real function $\mathbf{r}$, named return function. $\mathbf{r}(s, a, s')$ is the expected return when action a is performed in state s leading to next state s'. The interaction between an agent and an MD process generates a sequence $s_0, a_0, r_0, s_1, a_1, r_1, s_2, a_2, r_2, etc.$, called an episode, as Figure 2 illustrates. An MD process verifies the Markov property: the probability law of s_{t+1} is fully specified by the knowledge of the current state s_t and the action performed in this state a_t at time t (and $\mathcal{M}$). There may exist a subset of states $\mathcal{S}_{\text{final}} \subset \mathcal{S}$, called the set of final states, such that when the agent reaches a state $s \in \mathcal{S}_{\text{final}}$, the episode ends.

Markov decision problem A Markov decision problem (MDP) is a Markov decision process in which the agent has to optimize a given objective function. Let us consider an MDP in which the agents performs a given number T of interactions and let us define the objective function J as:

$$J(T) = \sum_{t=0}^{T-1} r_t, \tag{1}$$

where r_t is the return collected by the agent at time step t. The state reached at time T is a final state. The agent goal is to maximize $J(T)$. A policy $\pi : \mathcal{S} \to \mathcal{P}(\mathcal{A})$ maps each state to a distribution over the set of actions $\mathcal{P}(\mathcal{A})$. A policy specifies which action the agent performs in any state. The objective function $J(T)$ depends on the returns the agent has collected between $t = 0$
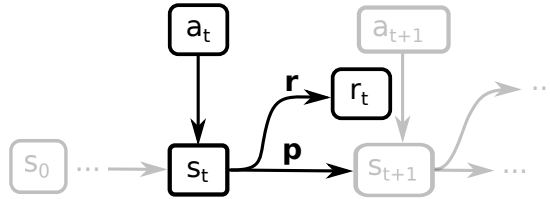
Figure 2: In reinforcement learning, a Markov decision process models the environment. At each time step t, the agent observes the environment current state s_t. Depending on s_t, the agent takes an action a_t according to its policy. As a consequence of taking action a_t, the environment transits to next state s_{t+1}, depending on the transition function $\mathbf{p}$, and the agent observes the return r_t which depends on the return function $\mathbf{r}$. This process repeats until the episode eventually ends.

and $t = T - 1$. Collected returns depend on the agent policy, consequently, $J(T)$ is a function of the agent policy. The more a policy maximizes $J(T)$, the better the policy is. Considering a policy π, we define the value of a state s as the expectation of the objective function when the agent follows policy π starting from state s:

$$V_\pi(s) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r_t \,\middle|\, s_0 = s \right], \forall s \in \mathcal{S}. \tag{2}$$

The Q-value of state s and action a is defined as the expectation of the objective function when the agent performs a in s and then follows π:

$$Q_\pi(s, a) = \mathbb{E}_\pi \left[\sum_{t=0}^{T-1} r_t \,\middle|\, s_0 = s, a_0 = a \right], \forall (s, a) \in \mathcal{S} \times \mathcal{A}. \tag{3}$$

The goal of the agent is to learn an optimal policy $\pi^\star$ that maximizes the value in all states. For the MDPs we consider in this paper, it can be proven that at least one optimal policy exists (Puterman, 1994). When an MDP is fully defined, i.e. $\mathcal{S}$, $\mathcal{A}$, $\mathbf{p}, \mathbf{r}$, and T are known to the agent, finding an optimal policy is an optimization problem where all necessary quantities to compute a solution are available. For instance, dynamic programming can be employed to approximate an optimal policy. When $\mathbf{p}, \mathbf{r}$ (and T) are unknown, then RL can be employed. In the latter case, in general $\mathbf{p}$ and $\mathbf{r}$ can only be sampled through interactions of the agent with the environment. Most of RL algorithms belong to one of the three following families (Sutton and Barto, 2018): (1) *critic* methods which are algorithms that learn a value function (e.g. Q-Learning, FQI, DQN) and then derive an optimal policy from it; (2) *actor* methods which are algorithms that directly learn an optimal policy (e.g. REINFORCE); (3) *actor-critic* methods which simultaneously combine actor and critic methods (e.g. A2C, PPO, SAC). In order to deal with potentially very large state and/or action spaces, RL algorithms generally use function approximators, to compactly represent value and/or policy functions. Deep RL is a special case of RL where function approximators are neural networks (Lapan, 2018).

3.2   Partially Observable Markov Decision Process

An MDP is an idealized model of a real-world system because real systems are unlikely to verify the properties associated to MDPs, in particular the Markov property. A field plot involves many interleaved dynamical processes, and parameters which are still partially discovered/measurable

and the study of these dynamics are active areas of research (e.g. Husson et al., 2021). In an MDP, each state is supposed to contain all necessary information for the agent to be able to decide which action is the best to perform in order to optimize the objective function. Except from synthetic problems like games with complete information, such as the game of Go, for most systems, the exact environment state is unknown to the agent. In contrast, with real-world systems, the agent is likely to only access uncertain or incomplete observations of states. Such problems can be formalized as Partially Observable Markov Decision Problem (POMDP, Åström, 1965). POMDPs are a specific topic of study in the RL literature, and require *ad-hoc* algorithms to solve them (e.g. Spaan, 2012).

3.3 `gym` environments

OpenAI gym[3] is an open source toolkit initially developed by the Open AI company, that provides light RL environments with a standardized Application Programming Interface (API). `gym` API has become a reference in the RL community to create standardized RL environments in order to compare performances of RL algorithms. Many environments are available with `gym`, for instance with simulated games or physical dynamical systems, including robots. Typically, `gym` environments are straightforward to use: all simulated dynamics are pre-parametrized and hidden. The user instantiates an environment as simply as:

```python
import gym
env = gym.make("CartPole-v0") # create an instance of the environment CartPole-v0
```

As Figure 3 shows, `gym` is a wrapper that gives access to a more complex simulator. `gym` environments come with default attributes which specify action and observation spaces. For instance in the case of the `CartPole-v0` environment, the user gets the specifications of a four-dimensional state space and a set of two possible actions:

```python
>>> env.observation_space # outputs observation lower bound, upper bound, shape,
                                              data type
Box(-3.4028234663852886e+38, 3.4028234663852886e+38, (4,), float32)
>>> env.action_space # if Discrete class, outputs the number of possible values
Discrete(2)
```

The user interacts with the environment through standardized methods. `gym` is independent of the implementation of the agent policy. The agent interacts with the environment by calling the `step()` method with an argument a_t specifying the action to take, in order to receive the transition and reward generated by **p** and **r**. The objective function is neither part of `gym` implementation, hence to optimize Equation 2, one specifies γ as a parameter of the learning algorithm trying to build an optimal policy.

To illustrate the simplicity of interactions, we exhibit a basic RL loop:

```python
observation = env.reset() # reset the environment and get initial observation
# >>> observation
# array([-0.03325944, -0.02851367,  0.00086817, -0.00618905])

done = False # True when the episode is ended
while not done:
    action = policy(observation)  # get action depending on agent policy
    observation, reward, done, info = env.step(action) # perform the action
    # update the policy
env.render() # graphical representation of environment state
env.close()  # gracefully exits the environment
```
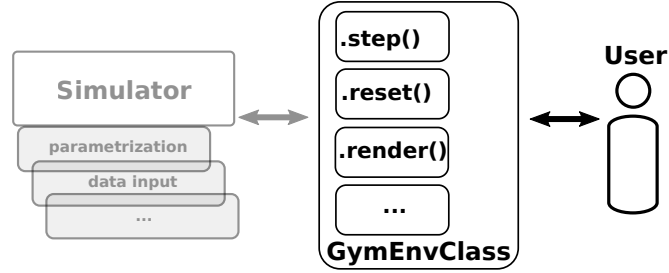
---

[3]https://gym.openai.com/

Figure 3: From a user's perspective, `gym` environments are simplified interfaces to simulators, through standardized methods.

`observation` corresponds to a possibly incomplete MDP state s_t, `reward` corresponds to r_t, `done` is True if the episode has ended, i.e. if the agent has reached a final state, and finally `info` provides optional extra information about the environment. We refer to the documentation available at https://gym.openai.com/ for further details.

4   Decisions problems in `gym-DSSAT`

In Section 4.1 we introduce the default crop management problems `gym-DSSAT` provides and Section 4.2 outlines how a user can create customized crop management problems.

4.1   Default crop management problems of `gym-DSSAT`

By default, `gym-DSSAT` sequential decision problems simulate a maize experiment which has been carried out in 1982 in the experimental farm of the University of Florida, Gainesville, USA (Hunt and Boote, 1998, `UFGA8201` experiment). An episode is a simulation of a growing season. A simulation starts prior to planting and ends at crop harvest which is automatically defined as the crop maturity date. Crop maturity, a final state in `gym-DSSAT`, depends on crop growth, which depends itself on crop management and weather events, and the time to reach it is stochastic. Note that other final states exist in `gym-DSSAT`. For instance, improper crop management or too stressing weather conditions may lead to early crop failure, which is also a final state. During the whole growing season (about 160 days on average), an RL agent daily decides on the crop management action(s) to perform: fertilize and/or irrigate. By default, for each episode, the weather is generated by the `WGEN` stochastic weather simulator (Richardson, 1985; Soltani and Hoogenboom, 2003). `WGEN` has been parametrized based on historical weather records of the location of the original experiment. The duration between the starting date of the simulation and the planting date, which lasts about one month, induces stochastic soil conditions at the time of planting (e.g. soil nitrate, or soil water content), as a result of stochastic weather events.

The number of measurable attributes in a field is extremely large. In contrast, farmers have been described to make crop management decisions based on a limited practicable number of field observations (Papy, 1998). For this reason, in `gym-DSSAT`, the RL agent only accesses a restricted subset of `DSSAT` state variables which constitutes the observation space of the environment. Based on agronomic knowledge, we selected this subset with the constraint of the variables to be realistically measurable/proxiable in real conditions. These observation variables are mixed, and take either continuous or discrete values. We documented all observation and action variables in

the `gym-DSSAT` YAML configuration file[4]. This file includes description of variables type, range, and agronomic meaning.

In `DSSAT`, the `WGEN` stochastic weather simulator is implemented as a first-order Markov chain, but all other processes are deterministic. Therefore, `gym-DSSAT` decision problems are Markovian. Because the agent only accesses a subset of all `DSSAT` internal variables, a `gym-DSSAT` problem is a POMDP, similarly to the real problems faced by farmers. From an RL perspective, one can rigorously address a `gym-DSSAT` decision problem as a POMDP, or follow the common pragmatic approach which treats a POMDP as an MDP. In contrast with many toy RL environments, the environment is autonomous: it evolves by itself and not only because an action has been performed by the agent. Indeed, if on a given day a farmer does not fertilize/irrigate, its field plot still evolves. A do-nothing action is always available at each time step, which corresponds to the spontaneous field evolution.

`DSSAT` simulates dynamics at the plot level; likewise, the agent performs actions on the whole field plot. Growing conditions such as soil characteristics and other crop operations such as soil tillage, cultivar choice are fixed. We defined default return functions based on agronomic knowledge following the reward shaping principle (Randløv and Alstrøm, 1998; Ng et al., 1999), such that rewards were as much frequent and as much informative as possible regarding the desired behaviour of the agent. Reward shaping aims both at facilitating an agent learning and to steer policies towards desirable trade-offs such as maximizing grain yield and minimizing induced pollution. We define return functions in a standalone Python file, and users can find admissible values of actions in the environment YAML configuration file, or in `gym-DSSAT` action space attribute.

By default, `gym-dssat` provides three RL problems:

1. A **fertilization problem** in which the agent can apply every day a real valued quantity of nitrogen, as indicated in Table 1. Crops are rainfed, and no irrigation is applied during the growing season, excepted a single one before planting. `DSSAT` automatically performs planting operation when soil temperature and humidity lie in favorable ranges. Denoting $\text{trnu}(t, t+1)$ the plant nitrogen uptake (kg/ha) from its environment between days t and $t+1$; and $\text{anfer}(t)$ the nitrogen fertilizer application (kg/ha) on day t, we crafted the default fertilization return function as:

$$r(t) = \underbrace{\text{trnu}(t, t+1)}_{\substack{\text{plant nitrogen} \\ \text{uptake}}} - \underbrace{0.5}_{\substack{\text{penalty} \\ \text{factor}}} \times \underbrace{\text{anfer}(t)}_{\substack{\text{fertilizer} \\ \text{quantity}}} \tag{4}$$

The return is the daily population nitrogen uptake (to be maximized) which we penalize if the agent has fertilized the previous day. We defined the penalty factor based on expert knowledge such that the return corresponds to a desirable trade-off between agronomic, economical and environmental potentially conflicting objectives. Table 2 details the observation space.

2. An **irrigation problem** in which the agent can provide every day a real valued quantity of water to irrigate, as indicated in Table 1. Independently of agent actions, this problem features at the same time a deterministic low input nitrogen fertilization (see Table 3). Planting date is fixed, about one month after the beginning of simulation. The daily-based return is the daily change in above the ground population biomass (to be maximized), which we penalize if the agent has irrigated the previous day, similarly to the fertilization problem. We provide default reward function in Appendix Figure 6 and observation space in Appendix Table 8.

---

[4]https://gitlab.inria.fr/rgautron/gym_dssat_pdi/-/blob/stable/gym-dssat-pdi/gym_dssat_pdi/envs/configs/env_config.yml

Table 1: `gym-DSSAT` available actions

action	description	range
fertilization	daily nitrogen fertilization amount (kg/ha)	[0,200]
irrigation	daily irrigation amount (L/m^2)	[0,50]

Table 2: Default observation space for the fertilization task.

	definition
istage	DSSAT maize growing stage
vstage	vegetative growth stage (number of leaves)
topwt	above the ground population biomass (kg/ha)
grnwt	grain weight dry matter (kg/ha)
swfac	index of plant water stress (unitless)
nstres	index of plant nitrogen stress (unitless)
xlai	plant population leaf area index (m^2 leaf/m^2 soil)
dtt	growing degree days for current day (°C/day)
dap	days after planting (day)
cumsumfert	cumulative nitrogen fertilizer applications (kg/ha)
rain	rainfall for the current day (L/m^2/day)
ep	actual plant transpiration rate (L/m^2/day)

3 A mixed **fertilization and irrigation problem** which combines both previous decision problems: every day, the agent can fertilize and/or irrigate. Planting date is fixed, about one month after the beginning of simulation. In this case, the return has two components, one for each sub-problem: this is a multi-objective problem (e.g. Hayes et al., 2021). The default observation space is the union of the observation spaces of the fertilization and irrigation problems.

We did not define returns of decision problems as economic returns to avoid issues due to cost variations over time (e.g. petrochemicals). Fossil fuel necessary to produce artificial nitrogen fertilizers (see the Haber process Modak, 2002) or to pump water are highly variable over time, making these decision problems non-stationary. Consequently, optimal solutions are likely to change through time. This is why we chose an arbitrary penalization of actions as a proxy of a notion of cost with sound agronomic trade-off, as shown in Equation 4. Despite their apparent simplicity, from an agronomic perspective, the three aforementioned decision problems are non-trivial. These problems can be made harder by providing a more restricted and/or noisy observation space to the agent, see the discussion of the fertilization use case (Section 6.1).

Table 3: Expert fertilization policy. 'DAP' stands for Day After Planting.

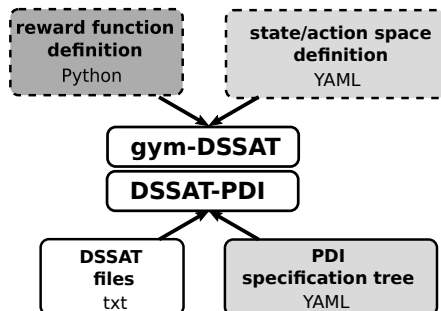| DAP	quantity (kg N/ha)
40	27
45	35
80	54

Figure 4: Configuration files used by the crop management reinforcement learning environment. At the top of the figure, files in dashed boxes define the reward function and state and action spaces of the Markov decision process. Dashed boxes indicate straightforward to customize configuration files. At the bottom of the figure, `DSSAT` files parametrize simulations, and the `PDI` specification tree is a technical file which manages the communication between `DSSAT-PDI` and `gym-DSSAT`.
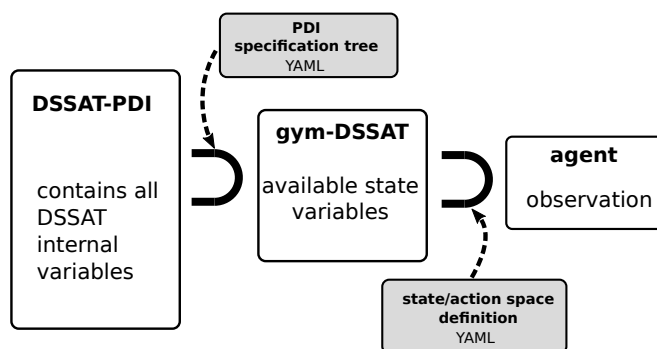

Figure 5: Successive subsets of `DSSAT` state variables until agent observations. Boxes filled with grey indicate files defining state variable subsets.

4.2 Custom scenario definition

A user can customize `gym-DSSAT` problems, with an ease that depends on the features to be modified, see Figure 4. An observation is a subset of `DSSAT` internal state variables. Figure 5 shows the technical files which define the subset of variables constituting an observation. A user can straightforwardly modify the observation space in the YAML configuration file. In the same way, the definition of the return functions can be easily modified by the user by editing a standalone Python file[5]. Built-in `DSSAT` features can be directly leveraged, such as environmental modifications with changes in atmospheric CO_2 concentration or meteorological features, to mimic the effects of climate change. Including other state variables, actions, crops, soil or weather generation parametrizations requires a deeper understanding of how `gym-DSSAT` works and some agronomic knowledge. This goes beyond the scope of this report; additional information is available in `gym-DSSAT` GitLab page.

---

[5]https://gitlab.inria.fr/rgautron/gym_dssat_pdi/-/blob/stable/gym-dssat-pdi/gym_dssat_pdi/envs/configs/rewards.py

5   Software architecture of the environment

In contrast with the simplicity of use of `gym-DSSAT`, we had to modify the original `DSSAT` simulator in a non-trivial manner to enable daily interactions with an agent and to interface the modified `DSSAT` Fortran program with Python. `DSSAT` was not designed to be used in an interaction loop. In this section, we detail how we have technically proceeded.

5.1   The PDI Data Interface

The PDI Data Interface (`PDI`, Roussel et al., 2017) was the key element in `gym-DSSAT` which turned the original monolithic `DSSAT` simulator implemented in Fortran into an interactive Python RL environment. `PDI` is a library designed to decouple C/C++/Fortran codes, typically high-performance numerical simulations, from Input/Output (I/O) concerns. It offers a declarative low-invasive API to instrument the simulation source code, enabling the exposition of selected memory buffers used in the simulation to be read/written from/to `PDI`, and the notification to `PDI` of significant steps of the simulation. By itself, `PDI` does not provide any tool for the manipulation of data, instead it offers an event-driven plugin system to ease interfacing external tools with the simulation.

`PDI` moves most of the logic for the I/O interface away from the code: specifically, a YAML file is used to describe data structures and to specify when and which actions (provided by the different `PDI` plugins) to trigger on the selected data. The exposed data is selected by adding a few `PDI` calls in the source code with a very simple syntax. Other I/O libraries in the High Performance Computing field follow a similar declarative approach, such as `ADIOS-II` (Godoy et al., 2020), `Damaris` (Dorier et al., 2016) or `XIOS` (Meurdesoif et al., 2013). However, most of these alternatives are mainly focused on providing high-level abstractions of high-performace I/O operations, working with some domain-specific assumptions and providing additional features on top of parallel I/O streams, such as burst buffering or compression. `PDI` design has a general and global approach, aiming at more versatile scenarios, with a plugin system that enables substantially different possibilities and I/O strategies, such as the interaction with external Python code. As a result, `PDI` makes possible the implementation of `gym-DSSAT`: an external software (`gym`), directly interacts with a modification of a stand-alone, monolithic simulator (`DSSAT`).

Figure 6 shows a simplified example of `PDI` use in `gym-DSSAT`, for the fertilization problem. Figure 6a lists chunks of the YAML file with declarations of exposed variables in the simulation code and definitions of events to be triggered. This YAML file corresponds to the `PDI` specification tree file in Figure 4. Figure 6b shows a snippet of the instrumented Fortran source code of `DSSAT`, with `PDI` initialization and three exposed simulation variables: two are read by `PDI` and will be available as observation variables, the third one is written by `PDI`, and corresponds to the action decided by the agent regarding crop fertilization for the current day. The whole instrumented code corresponds to `DSSAT-PDI`, see Section 5.2.

5.2   Internals of `gym-DSSAT`

We present a generic procedure which is an important methodological contribution of this work. `gym-DSSAT` is made of two communicating processes, as shows Figure 7a:

(i) `DSSAT-PDI` which is the compiled Fortran code of a modification of the original `DSSAT` crop model, using the `PDI` library.

(ii) `gym-DSSAT` which, from a user perspective, is the usual `gym` interface to the RL environment.

```
                                                PROGRAM CSM

                                                ! [...]
                                                CALL pc_parse_path("dssat-pdi.yml", conf)
                                                CALL pdi_init(conf)
                                                ! [...]

                                                ! read DSSAT internal state variables
data:                                           CALL pdi_expose("DAP", DAP, pdi_out)
### action                                      CALL pdi_expose("ISTAGE", ISTAGE, pdi_out)
  ANFER: float                                  ! [...]
#[...]
                                                IF (IFERI == 'L') THEN
### state                                           ! specify fertilization for today
  DAP: int                                          CALL pdi_expose("ANFER", ANFER(I), pdi_in)
  ISTAGE: int
#[...]                                               IF (ANFER(I) > 0) THEN
                                                        FERTILIZE_TODAY = .TRUE.
on_event:                                           ENDIF
                                                ENDIF
  fertilize:                                    !     [...]
    with: {anfer: $ANFER}
    exec:                                       CALL pdi_finalize()
      anfer.itemset(action['anfer'])            CALL PC_tree_destroy(conf)
#[...]                                           END PROGRAM CSM
```

(a) PDI YAML file.   (b) DSSAT code instrumented with PDI calls.

Figure 6: Simplified example of PDI use in gym-DSSAT for the fertilization decision problem. The left-hand side corresponds to the PDI specification tree (Figure 4), and the right-hand side to the Fortran code of DSSAT-PDI (Section 5.2).

DSSAT-PDI The modification of the original DSSAT software, named DSSAT-PDI, allows an agent to daily interact with the crop simulator during a growing season. This interaction loop consists in repeatedly pausing DSSAT, reading DSSAT internal variables, providing these internal variables to the agent, specifying the action(s) of the agent to DSSAT and finally resuming DSSAT execution. DSSAT being in continuous development, the goal was to modify as little as possible the original source code for easy updates. Minimal interventions on DSSAT code have been facilitated by the PDI library. PDI manages data communication with a Python process, through the PDI pycall plugin. Figure 7b illustrates how DSSAT-PDI works. During the execution of the internal daily loop of DSSAT, PDI code snippets allow data coupling: accessing, writing in memory variables and triggering events. DSSAT-PDI execution starts with an initialization event, which provides all necessary elements for PDI, DSSAT-PDI and gym-DSSAT to start. Then, DSSAT-PDI enters its daily loop which consists in all successive daily updates of the crop simulator state during a growing season. While the daily loop executes, when the get state event occurs, PDI stores the values of a subset of DSSAT internal state variables in the PDI Store. After then, the PDI pycall plugin accesses these values, executes a Python script corresponding to the interaction with the agent, and stores back in the PDI Store the action(s) taken by the agent. Then, when the set action event occurs, PDI writes the variables corresponding to the agent action(s) into DSSAT memory and releases DSSAT daily loop execution. Finally, at the end of the simulation, a finalization event occurs to gracefully terminate the whole process. For the same parametrization and input, DSSAT-PDI and the vanilla DSSAT both consistently provide the same output.

gym-DSSAT From a user's perspective, the gym-DSSAT environment is a simple interface to DSSAT-PDI, but from a technical point of view, gym-DSSAT handles all the execution of DSSAT-PDI.

`gym-DSSAT` provides the necessary data input to `DSSAT-PDI`, including parametrization and weather data; it manages data communication and translation between the RL agent and `DSSAT-PDI` without extra effort. Finally, `gym-DSSAT` is responsible for the graceful termination of `DSSAT-PDI`.

Messaging between `DSSAT-PDI` and `gym-DSSAT`. During the execution of a block of Python code, the `PDI pycall plugin` accesses `DSSAT-PDI` state variables, which have been previously stored in the `PDI Store`. Nevertheless, the data available in this Python process still requires to be communicated to `gym-DSSAT`, another Python process which is independent of the `DSSAT-PDI` process. As shown in Figure 7a, the communication between `gym-DSSAT` and `DSSAT-PDI` is powered by ZeroMQ (Hintjens, 2013) Python sockets, with the `PyZMQ` package. Python sockets exchange data as JSON files, encoded as strings. Every transaction is a blocking event such that `DSSAT-PDI` daily loop is resumed only after `DSSAT-PDI` has received agent action(s).

6 Experimenting with `gym-DSSAT`

In this section, we provide an RL use case for the maize fertilization problem using `gym-DSSAT`. We also discuss execution time and reproducibility issues using `gym-DSSAT`.

## 6.1 Use case: learning an efficient maize fertilization

As a simple use case, we present an example of how to address the fertilization task. We provide the irrigation use case in Appendix A. The source code of these experiments is available in `gym-DSSAT` GitLab page.

Methods We consider the nitrogen fertilization task, as introduced in Section 4.1. The decision problem being on a finite horizon, i.e. each episode lasted during a growing season, we defined the objective function of the agent as the undiscounted sum of returns, see Equation 1. Table 2 presents the subset of `DSSAT` internal variables we have selected to define the observation space provided to the agent. These observation variables were selected as they could be realistically measured on farm. As a common practice, we pragmatically addressed this decision problem as an MDP, even though it is a POMDP (Section 4). We used the Proximal Policy Optimization algorithm (PPO, Schulman et al., 2017), as implemented in `Stable-Baslines3 1.4.0` (Hill et al., 2018). PPO belongs to the family of deep RL actor-critic methods (see Section 3.1) and uses gradient descent to search for a good policy. PPO generally performs well on a wide range of problems and has been adopted as a standard baseline by the RL community. It is versatile as it can deal with both continuous and/or discrete actions and observation variables. In this experiment, we considered three policies:

- We first considered the most trivial fertilization policy: the "null" policy that never fertilized. As there still remains nitrogen in soil before cultivation (Morris et al., 2018), without mineral fertilization, the reference experiment, or *control*, is usually the null policy. Agronomists then measure the effect of a nitrogen fertilization policy as a performance gain compared to the null policy, in order to decouple the effect of nitrogen fertilizer from the effect of already available nitrogen in soil (Vanlauwe et al., 2011).

- The second baseline is the 'expert' policy, which is the fertilization policy of the original maize field experiment (Hunt and Boote, 1998, `UFGA8201` experiment number #1), see Section 4.1. As Table 3 shows, this policy consists in three deterministic nitrogen fertilizer applications, which only depend on the number of days after planting.
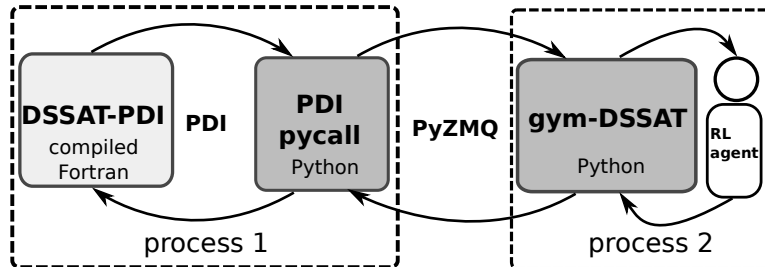
(a) The reinforcement learning environment consists of two interacting processes. (i) the core modification of the `DSSAT` simulator, `DSSAT-PDI`, with its `PDI` module to execute Python code (`pycall plugin`); (ii) the `gym` Python interface `gym-DSSAT`. `PyZMQ` handles messaging between (i) and (ii) through Python sockets.
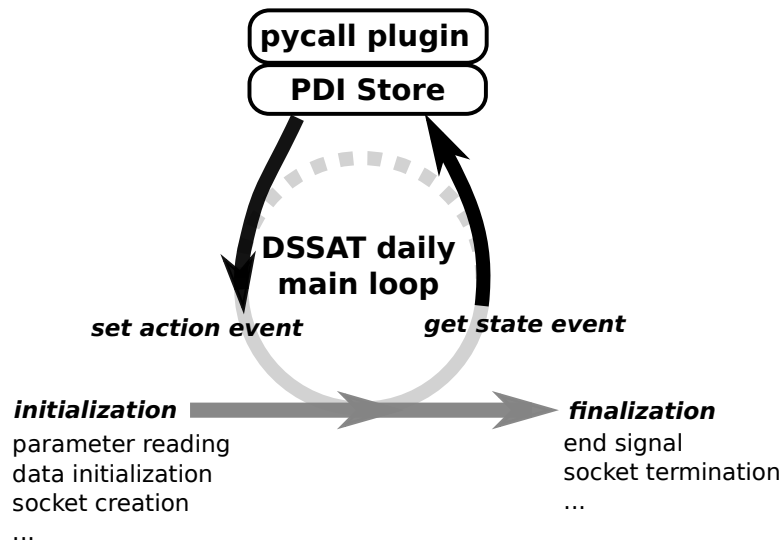
(b) Simplified `PDI` data coupling and program execution of `DSSAT-PDI` which is the instrumented code of `DSSAT`. `PDI` handles the oftware initialization, data exchange with `gym-DSSAT` during the whole simulated growing season through the `pycall plugin`, and finally software graceful termination. The execution of the Python code by `PDI pycall plugin` is a blocking transaction.

Figure 7: The elements of `gym-DSSAT`.

- Finally, the policy learned by PPO. As our goal was not to obtain the best performance with an RL algorithm, but to simply establish a baseline, we used PPO default hyper-parameters as set in `Stable-Baselines3 1.4.0`. It is most likely better PPO hyper-parameters may be found. We trained PPO during 10^6 episodes, with stochastic weather generation. The training procedure was light in terms of computation: it was possible to complete the 10^6 episodes in about 1.5 hour of computation with a standard 8 core laptop. During training, the performance of PPO was evaluated on a validation environment every 10^3 episodes. We seeded the validation environment with a different seed than for the training environment. Consequently, the validation environment generated a different sequence of weather series compared to the training environment. The model with the best validation performance was saved as the result of the training.

In order to compare fertilization policies, we measured their performance with 10^3 episodes in a test environment. Test environment also featured stochastic weather generation, but with isolated seeds i.e. different from the ones used in training and evaluation environments of PPO. This guaranteed that while testing policies, none of the stochastic weather series have been met by PPO during training or evaluation phases, in order to avoid over-optimistic performance measures (Stone, 1974). In the performance analysis of policies, the evolution of returns r_t provides information about the learning process from an RL perspective, but returns are still not directly interpretable from an agronomic perspective. Performance analysis of crop management options require multiple evaluation criteria (Doré et al., 2006; Duru et al., 2015). To remedy this problem, we used a subset of `DSSAT` internal state variables, provided in Table 4, as performance indicators. Note that these variables are not necessarily contained in the observation space of the fertilization problem (Table 2) because we used them for another purpose than algorithm training. Each of these performance criteria covariates with the other ones. For instance, increasing the total fertilizer amount is likely to increase the grain yield, but also likely to increase the pollution induced by nitrate leaching. The agronomic nitrogen use efficiency (ANE, Equation 1, Vanlauwe et al., 2011) is a common indicator of fertilization sustainability. For a fertilization policy π, denoting $\texttt{grnwt}^\pi$ the dry matter grain yield of the policy π (kg/ha), $\texttt{grnwt}^0$ the dry matter grain yield with no fertilization (kg/ha), and $\texttt{cumsumfert}^\pi$ the total fertilizer quantity applied with policy π (kg/ha), we have:

$$\text{ANE}^\pi(t) = \frac{\texttt{grnwt}^\pi(t) - \texttt{grnwt}^0(t)}{\texttt{cumsumfert}^\pi(t)} \tag{5}$$

The ANE indicates the grain yield response with respect to the null policy provided by each unit of nitrogen fertilizer. Maximizing the ANE relates to economic and environmental aspects, leading to an efficient use of fertilizer which limits risks of pollution. Performance indicators presented in Table 4 express a complex trade-off between conflicting objectives.

Results Figure 8a displays the evolution of undiscounted cumulated rewards (Equation 1) of policies, against the day of simulation. PPO ended with the highest mean cumulated return compared to the null and expert policies. PPO cumulated returns were less variable than with the expert policy, as can be seen from the reduced range of values between upper and lower quantiles. Figure 8b provides a 2D histogram of fertilizer applications, against the day of simulation. The darker a cell, the more frequent the fertilizer application. PPO fertilizer applications were more frequent at the beginning of the growing season and around day of simulation 60. The latter application date corresponds to the beginning of the floral initiation stage, see Table 11 in Appendix. Nevertheless, the variability of rates and application dates of PPO indicated that PPO policy did not only depend on days after planting as the expert policy did, but also depended

Table 4: Performance indicators for fertilization policies. An hyphen means `gym-DSSAT` does not directly provide the variable, but it can be easily derived.

variable	definition	comment
`grnwt`	grain yield (kg/ha)	quantitative objective to be maximized
`pcngrn`	massic fraction of nitrogen in grains	qualitative objective to be maximized
`cumsumfert`	total fertilization (kg/ha)	cost to be minimized
–	application number	cost to be minimized
–	nitrogen use efficiency (kg/kg)	agronomic criteria to be maximized
`cleach`	nitrate leaching (kg/ha)	loss/pollution to be minimized

(a) Mean cumulated return of each of the 3 policies against the day of the simulation. Shaded area displays the $[0.05, 0.95]$ quantile range for each policy.

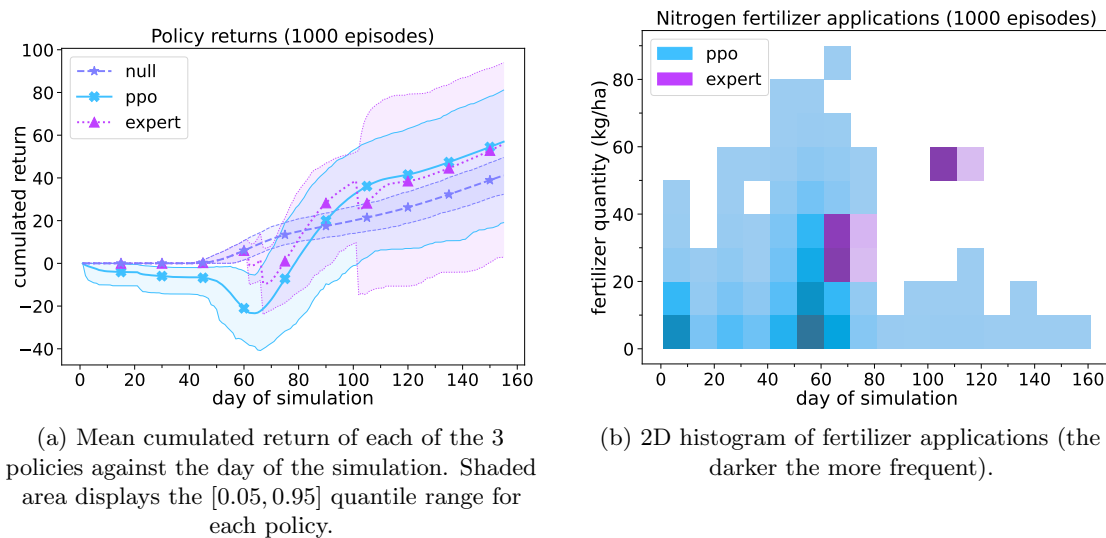(b) 2D histogram of fertilizer applications (the darker the more frequent).

Figure 8: Undiscounted cumulated returns and applications for the fertilization problem.

on more factors. Note that while the expert policy was deterministic, the day of simulation of applications showed slight variations. This was because in simulations, the planting date was automatically determined within a time window, depending on soil conditions, depending itself on (stochastic) weather events. Because the expert policy specified fertilizer application dates in days after planting, and not in days of simulation, a shift in planting dates consistently induced a shift in the corresponding day of simulation of fertilizer applications.

Table 5 shows statistics of the performance indicators detailed in Table 4. As expected, there was no policy that was optimal for all performance criteria. PPO policy exhibited performance trade-offs between the expert and the null policies we deemed satisfying. Grain yield and nitrogen content in grains (a nutritional criteria) were close to the ones of expert policy. On average, PPO policy consumed about 28% less nitrogen than the expert policy. Consistently, PPO ANE (Equation 5) –a key metric of sustainable fertilization– was about 29% greater than for the expert policy. From a practical perspective, a good fertilization policy consists in a limited number of applications during an episode, as the expert policy suggests. Indeed, each nitrogen application costs both in terms of fertilizer (as a product of natural gas) and application costs (e.g. mechanized nitrogen broadcasting). The mean number of applications of PPO (about 6) was higher than for the expert policy (3), but still practicable. Finally, PPO policy showed a

Table 5: Mean (st. dev.) fertilization baselines performances computed using 1000 episodes. For each criterion, bold numbers indicate the best performing policy.

	null	**expert**	**PPO**
grain yield (kg/ha)	1141.1 (344.0)	**3686.5** (1841.0)	3463.1 (1628.4)
massic nitrogen in grains (%)	1.1 (0.1)	**1.7** (0.2)	1.5 (0.3)
total fertilization (kg/ha)	**0** (0)	115.8 (5.2)	82.8 (15.2)
application number	**0** (0)	3.0 (0.1)	5.7 (1.6)
nitrogen use efficiency (kg/kg)	n.a.	22.0 (14.1)	**28.3** (16.7)
nitrate leaching (kg/ha)	**15.9** (7.7)	18.0 (12.0)	18.3 (11.6)

slighlty lower nitrate leaching than the expert policy, which means less nitrate pollution induced by nitrogen fertilization.

Discussion   We have shown that with an off-the-shelf `Stable-Baselines3` PPO implementation, we have been able to learn a relevant fertilization policy that slightly outperforms the expert fertilization policy regarding the objective function. From an agronomic perspective, PPO policy reached superior nitrogen use efficiency, with a substantially reduced nitrogen fertilizer consumption compared to the expert policy, while still yielding maize grain with satisfying quantity and quality. PPO focused nitrogen fertilizer applications at the beginning of the floral initiation stage, where maize nitrogen needs are the greatest and most crucial (Hanway, 1963). The performance of PPO is likely to increase with a proper tuning. Nevertheless:

(1) The fertilization policy an agent has learned still requires *explainability*. For instance, discovering which are the most important observation variables that determine a fertilizer application, how their values impact fertilization, and if these results are consistent with the agronomic knowledge is a requirement. For crop management decision support systems, user trust is essential (Rose et al., 2016; Evans et al., 2017). As an example, Garcia (1999) translated an RL agent policy into a set of simple decision rules (e.g. "if condition 1 or condition 2, then do ...") which were easily interpretable and usable by farmers and/or agronomists.

(2) In real conditions, each field observation costs. As an example, the growth stage (`istage`, Table 2), which is an observation variable, would only require a periodic visual inspection of the field. Growth stage is consequently a realistic and inexpensive observation. In contrast, the measure of the daily population nitrogen uptake, necessary to compute the return (`trnu`, Equation 4), would require destructive plant sampling and extensive laboratory analysis. In case the agent is trained with real field trials, then computing rewards becomes necessary, and might be problematic. Consequently, in latter case, an alternative reward function could be employed. The cost of measuring each observation variable –related to the precision of measurement– and the frequency of these observations should be minimized for practical applications.

(3) Learning a relevant fertilization policy from scratch required 10^6 episodes. The stochastic weather time series `gym-DSSAT` used being sampled from independent and identically distributions, 10^6 episodes means 10^6 cultivation cycles under different weather conditions. If the objective of the experiment is to design *in silico* fertilization policies, then learning efficiency and field measure costs (2) are not problematic, but remark (1) still applies. If `gym-DSSAT` is used to mimic real-world conditions and the objective is to design an RL algorithm able

to learn/improve from real interactions, then the learning efficiency of the off-the-shelf PPO clearly precludes any straight application in real conditions. Thereby, researchers must reduce the sample complexity of the decision problem, i.e. simplify the problem to reduce the number of samples required to solve this problem, and/or researchers must use/design other RL algorithms with improved learning efficiency (e.g. using demonstration learning Taylor and Stone, 2009, to leverage existing expert policies).

6.2 Execution time and reproducibility

In this section, we now briefly highlight that `gym-DSSAT` is a lightweight RL environment and discuss reproducibility issues.

Execution time We performed all measures of `gym-DSSAT` execution time for the fertilization task. The mean duration of an episode was 156 ± 7 days (1 time step was 1 simulated day), averaged over 1000 episodes. We measured the following time executions averaged over 1000 episodes, each episode lasted until 100 time steps. In practice, we insured that all episodes did not end between step 1 and step 100, so environments had to update their state for the 100 time steps. During an episode, actions were randomly sampled from the action space. On a standard 8 core laptop, the mean running time to simulate one day in `gym-DSSAT` i.e. taking a single step in the environment was 2.56 ± 0.22 ms. In comparison, the mean running time of taking a step in `gym` default `MuJoCo` (version 2.1.0) environment `HumanoidStandup-v2` was 0.61 ± 0.21 ms. While `gym-DSSAT` is more expensive in time than typical `gym` environments, the simulation is still responsive enough for typical usage in RL experiments.

Reproducibility According to the Association For Computing Machinery (ACM), a computational experiment is said *reproducible* if an "[. . .] independent group can obtain the same result using the author's own artifacts"[6], summarized as "different team, same experimental setup". Based on our tests, we successfully reproduced the results of the present study on the same platform i.e. on the same hardware and software layers. This means that both results of `gym-DSSAT` and `Stable-Baselines3` PPO were reproducible on the same platform. Nevertheless, as a more general reproducibility issue, we cannot guarantee the cross-platform reproducibility. Reaching cross-platform reproducibility is a generally hard issue, even for deterministic software, due to the multiple factors at stake. As an example, compiling `DSSAT` Fortran code with two different compilers may not result in the same exact `DSSAT` outputs. This is because the order of multiple arithmetic operations, despite being mathematically commutative, may not follow the same order in practice and the final result might be different because of floating point number rounding effects. To enhance reproducibility, we provide Docker containers for various Linux distributions for `gym-DSSAT` (see installation instructions[7]).

7 Concluding remarks

In this paper, we briefly presented `gym-DSSAT`, a Reinforcement Learning (RL) environment for crop management, and exposed uses cases for fertilization and irrigation decision problems. `gym-DSSAT` is based on `DSSAT`, a celebrated crop simulator used by worldwide agronomists. To turn the original Fortran `DSSAT` software into a Python `gym` environment, we used a recently

---

[6]Artifact Review and Badging Version 1.1 - August 24, 2020, `https://www.acm.org/publications/policies/artifact-review-and-badging-current`

[7]`https://rgautron.gitlabpages.inria.fr/gym-dssat-docs/Installation/index.html`

introduced library, named `PDI`. Currently, only maize fertilization and irrigation problems are available. `gym-DSSAT` can be extended to any of the 41 other crops `DSSAT` currently simulates, such as wheat or sorghum and/or to other crop operations. Further predefined scenarios will be defined to reflect a diversity of soil and climate combinations. Weather forecasts being of major interest for crop management (Hoogenboom, 2000), short time weather predictions of stochastically generated weather will be provided in the state space. `gym-DSSAT` will be connected to `Ray rllib` (Liang et al., 2017) to enhance environment scalability. For both irrigation and fertilization use cases, we showed that an untuned RL algorithm was able to learn more sustainable practices than the expert policies we considered. Beyond the use cases we have provided, further work is still required to tailor RL algorithms to the idiosyncracies of crop management problems. The performance baselines of each decision problem can be iteratively refined, for instance using the expert policy with Transfer Learning (Taylor and Stone, 2009) or extra exploration such as with Random Network Distillation (Burda et al., 2018). With a limited software development effort, `PDI` can be used to turn other existing mechanistic models into `gym` environments, hence opening the doors of a potentially large number of mechanistic models to the RL community. We hope the whole approach we used to create `gym-DSSAT` will be replicated to other complementary C, C++ or Fortran based crop models, such as STICS (Brisson et al., 2003) and other mechanistic models.

Acknowledgments

References

Åström, K. J. (1965). Optimal control of markov processes with incomplete state information. *Journal of Mathematical Analysis and Applications*, 10(1):174–205.

Binas, J., Luginbuehl, L., and Bengio, Y. (2019). Reinforcement learning for sustainable agriculture. In *ICML Workshop Climate Change: How Can AI Help?*

Binder, D. L., Sander, D. H., and Walters, D. T. (2000). Maize response to time of nitrogen application as affected by level of nitrogen deficiency. *Agronomy Journal*, 92(6):1228–1236.

Brisson, N., Gary, C., Justes, E., Roche, R., Mary, B., Ripoche, D., Zimmer, D., Sierra, J., Bertuzzi, P., Burger, P., et al. (2003). An overview of the crop model stics. *European Journal of agronomy*, 18(3-4):309–332.

Burda, Y., Edwards, H., Storkey, A., and Klimov, O. (2018). Exploration by random network distillation. *arXiv preprint arXiv:1810.12894*.

Camargo, G. G. and Kemanian, A. R. (2016). Six crop models differ in their simulation of water uptake. *Agricultural and forest meteorology*, 220:116–129.

Cassman, K. G., Dobermann, A., and Walters, D. T. (2002). Agroecosystems, nitrogen-use efficiency, and nitrogen management. *AMBIO: A Journal of the Human Environment*, 31(2):132–140.

Chatelin, M.-H., Aubry, C., Poussin, J.-C., Meynard, J.-M., Massé, J., Verjux, N., Gate, P., and Le Bris, X. (2005). Déciblé, a software package for wheat crop management simulation. *Agricultural Systems*, 83(1):77–99.

Chen, M., Cui, Y., Wang, X., Xie, H., Liu, F., Luo, T., Zheng, S., and Luo, Y. (2021). A reinforcement learning approach to irrigation decision-making for rice using weather forecasts. *Agricultural Water Management*, 250:106838.

Doré, T., Martin, P., Le Bail, M., Ney, B., and Roger-Estrade, J. (2006). *L'agronomie aujourd'hui*. Editions Quae.

Dorier, M., Antoniu, G., Cappello, F., Snir, M., Sisneros, R., Yildiz, O., Ibrahim, S., Peterka, T., and Orf, L. (2016). Damaris: Addressing performance variability in data management for post-petascale simulations. *ACM Transactions on Parallel Computing (TOPC)*, 3(3):1–43.

Duru, M., Therond, O., Martin, G., Martin-Clouaire, R., Magne, M.-A., Justes, E., Journet, E.-P., Aubertot, J.-N., Savary, S., Bergez, J.-E., et al. (2015). How to implement biodiversity-based agriculture to enhance ecosystem services: a review. *Agronomy for sustainable development*, 35(4):1259–1281.

Evans, K. J., Terhorst, A., and Kang, B. H. (2017). From data to decisions: helping crop producers build their actionable knowledge. *Critical reviews in plant sciences*, 36(2):71–88.

Freund, R. J. (1956). The introduction of risk into a programming model. *Econometrica: Journal of the econometric society*, pages 253–263.

Garcia, F. (1999). Use of reinforcement learning and simulation to optimize wheat crop technical management. In *Proceedings of the International Congress on Modelling and Simulation (MODSIM'99) Hamilton, New-Zealand*, pages 801–806.

Gautron, R., Maillard, O.-A., Preux, P., Corbeels, M., and Sabbadin, R. (2022). Reinforcement learning for crop management support: Review, prospects and challenges. *Computers and Electronics in Agriculture*, 200:107182.

Godoy, W. F., Podhorszki, N., Wang, R., Atkins, C., Eisenhauer, G., Gu, J., Davis, P., Choi, J., Germaschewski, K., Huck, K., et al. (2020). Adios 2: The adaptable input output system. a framework for high-performance data management. *SoftwareX*, 12:100561.

Golemo, F., Taiga, A. A., Courville, A., and Oudeyer, P.-Y. (2018). Sim-to-real transfer with neural-augmented robot simulation. In *Conference on Robot Learning*, pages 817–828. PMLR.

Hanway, J. (1963). Growth stages of corn (zea mays, l.) 1. *Agronomy Journal*, 55(5):487–492.

Hao, Z., Singh, V. P., and Xia, Y. (2018). Seasonal drought prediction: advances, challenges, and future prospects. *Reviews of Geophysics*, 56(1):108–141.

Hayes, C. F., Rădulescu, R., Bargiacchi, E., Källström, J., Macfarlane, M., Reymond, M., Verstraeten, T., Zintgraf, L. M., Dazeley, R., Heintz, F., et al. (2021). A practical guide to multi-objective reinforcement learning and planning. *arXiv preprint arXiv:2103.09568*.

He, J., Dukes, M. D., Hochmuth, G. J., Jones, J. W., and Graham, W. D. (2012). Identifying irrigation and nitrogen best management practices for sweet corn production on sandy soils using ceres-maize model. *Agricultural Water Management*, 109:61–70.

Hill, A., Raffin, A., Ernestus, M., Gleave, A., Kanervisto, A., Traore, R., Dhariwal, P., Hesse, C., Klimov, O., Nichol, A., Plappert, M., Radford, A., Schulman, J., Sidor, S., and Wu, Y. (2018). Stable baselines. https://github.com/hill-a/stable-baselines.

Hintjens, P. (2013). *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc.

Hoogenboom, G. (2000). Contribution of agrometeorology to the simulation of crop production and its applications. *Agricultural and forest meteorology*, 103(1-2):137–157.

Hoogenboom, G., Porter, C., Boote, K., Shelia, V., Wilkens, P., Singh, U., White, J., Asseng, S., Lizaso, J., Moreno, L., et al. (2019). The dssat crop modeling ecosystem. *Advances in crop modelling for a sustainable agriculture*, pages 173–216.

Howell, T. A. (2003). Irrigation efficiency. *Encyclopedia of water science*, 467:500.

Hunt, L. A. and Boote, K. J. (1998). Data for model operation, calibration, and evaluation. In *Understanding options for agricultural production*, pages 9–39. Springer.

Husson, O., Sarthou, J.-P., Bousset, L., Ratnadass, A., Schmidt, H.-P., Kempf, J., Husson, B., Tingry, S., Aubertot, J.-N., Deguine, J.-P., Goebel, F.-R., and Lamichhane, J. R. (2021). Soil and plant health in relation to dynamic sustainment of eh and ph homeostasis: A review. *Plant and Soil*.

Kadam, N. N., Xiao, G., Melgar, R. J., Bahuguna, R. N., Quinones, C., Tamilselvan, A., Prasad, P. V. V., and Jagadish, K. S. (2014). Agronomic and physiological responses to high temperature, drought, and elevated co2 interactions in cereals. *Advances in agronomy*, 127:111–156.

Kamara, A., Menkir, A., Badu-Apraku, B., and Ibikunle, O. (2003). The influence of drought stress on growth, yield and yield components of selected maize genotypes. *The journal of agricultural science*, 141(1):43–50.

Lapan, M. (2018). *Deep Reinforcement Learning Hands-On: Apply modern RL methods, with deep Q-networks, value iteration, policy gradients, TRPO, AlphaGo Zero and more*. Packt Publishing Ltd.

Li, Y., Guan, K., Schnitkey, G. D., DeLucia, E., and Peng, B. (2019). Excessive rainfall leads to maize yield loss of a comparable magnitude to extreme drought in the united states. *Global change biology*, 25(7):2325–2337.

Liang, E., Liaw, R., Nishihara, R., Moritz, P., Fox, R., Gonzalez, J., Goldberg, K., and Stoica, I. (2017). Ray rllib: A composable and scalable reinforcement learning library. *arXiv preprint arXiv:1712.09381*, page 85.

Meisinger, J. J. and Delgado, J. A. (2002). Principles for managing nitrogen leaching. *Journal of soil and water conservation*, 57(6):485–498.

Meurdesoif, Y., Ozdoba, H., Caubel, A., and Marti, O. (2013). Xios. In *Second Workshop on Coupling Technologies for Earth System Models (CW2013), NCAR, Boulder, CO, USA, available at: http://forge.ipsl.jussieu.fr/ioserver/raw-attachment/wiki/WikiStart/XIOS-BOULDER.pdf (last access: 5 August 2021)*.

Modak, J. M. (2002). Haber process for ammonia synthesis. *Resonance*, 7(9):69–77.

Morris, T. F., Murrell, T. S., Beegle, D. B., Camberato, J. J., Ferguson, R. B., Grove, J., Ketterings, Q., Kyveryga, P. M., Laboski, C. A., McGrath, J. M., et al. (2018). Strengths and limitations of nitrogen rate recommendations for corn and opportunities for improvement. *Agronomy Journal*, 110(1):1.

Mueller, N. D., Gerber, J. S., Johnston, M., Ray, D. K., Ramankutty, N., and Foley, J. A. (2012). Closing yield gaps through nutrient and water management. *Nature*, 490(7419):254–257.

NeSmith, D. and Ritchie, J. (1992). Short-and long-term responses of corn to a pre-anthesis soil water deficit. *Agronomy journal*, 84(1):107–113.

Ng, A. Y., Harada, D., and Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, volume 99, pages 278–287.

Overweg, H., Berghuijs, H. N., and Athanasiadis, I. N. (2021). Cropgym: a reinforcement learning environment for crop management. *arXiv preprint arXiv:2104.04326*.

Papy, F. (1998). Savoir pratique sur les systèmes techniques et aide à la décision. *La conduite du champ cultivé. Points de vue d'agronomes. IRD*, pages 245–259.

Puterman, M. L. (1994). *Markov decision processes: discrete stochastic dynamic programming*. John Wiley & Sons.

Randløv, J. and Alstrøm, P. (1998). Learning to drive a bicycle using reinforcement learning and shaping. In *ICML*, volume 98, pages 463–471. Citeseer.

Richardson, C. (1985). Weather simulation for crop management models. *Transactions of the ASAE*, 28(5):1602–1606.

Rose, D. C., Sutherland, W. J., Parker, C., Lobley, M., Winter, M., Morris, C., Twining, S., Ffoulkes, C., Amano, T., and Dicks, L. V. (2016). Decision support tools for agriculture: Towards effective design and delivery. *Agricultural systems*, 149:165–174.

Roussel, C., Keller, K., Gaalich, M., Gomez, L. B., and Bigot, J. (2017). PDI, an approach to decouple I/O concerns from high-performance simulation codes. Working paper.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Sebillotte, M. (1974). Agronomie et agriculture. essai d'analyse des tâches de l'agronome. *Cahiers Orstom, série biologie*, 24:3–25.

Sebillotte, M. (1978). Itinéraires techniques et évolution de la pensée agronomique. *CR Acad. Agric. Fr*, 64(11):906–914.

Shibu, M. E., Leffelaar, P. A., Van Keulen, H., and Aggarwal, P. K. (2010). Lintul3, a simulation model for nitrogen-limited situations: Application to rice. *European Journal of Agronomy*, 32(4):255–271.

Sokolowski, J. A. and Banks, C. M. (2012). *Handbook of real-world applications in modeling and simulation*, volume 2. John Wiley & Sons.

Soltani, A. and Hoogenboom, G. (2003). A statistical comparison of the stochastic weather generators wgen and simmeteo. *Climate Research*, 24(3):215–230.

Spaan, M. T. (2012). Partially observable markov decision processes. In *Reinforcement Learning*, pages 387–414. Springer.

Stone, M. (1974). Cross-validatory choice and assessment of statistical predictions. *Journal of the Royal Statistical Society: Series B (Methodological)*, 36(2):111–133.

Sun, L., Yang, Y., Hu, J., Porter, D., Marek, T., and Hillyer, C. (2017). Reinforcement learning control for water-efficient agricultural irrigation. In *2017 IEEE International Symposium on Parallel and Distributed Processing with Applications and 2017 IEEE International Conference on Ubiquitous Computing and Communications (ISPA/IUCC)*, pages 1334–1341. IEEE.

Sutton, R. S. and Barto, A. G. (2018). *Reinforcement learning: An introduction.* MIT press.

Taylor, M. E. and Stone, P. (2009). Transfer learning for reinforcement learning domains: A survey. *Journal of Machine Learning Research*, 10(7).

Tintner, G. (1955). Stochastic linear programming with applications to agricultural economics. In *Proceedings of the Second Symposium in Linear Programming*, volume 1, pages 197–228. National Bureau of Standards Washington, DC.

Vanlauwe, B., Kihara, J., Chivenge, P., Pypers, P., Coe, R., and Six, J. (2011). Agronomic use efficiency of n fertilizer in maize-based systems in sub-saharan africa within the context of integrated soil fertility management. *Plant and soil*, 339(1):35–50.

Wallach, D., Makowski, D., Jones, J. W., and Brun, F. (2018). *Working with dynamic crop models: methods, tools and examples for agriculture and environment.* Academic Press.

Wang, L., He, X., and Luo, D. (2020). Deep reinforcement learning for greenhouse climate control. In *2020 IEEE International Conference on Knowledge Graph (ICKG)*, pages 474–480. IEEE.

A Irrigation use case

We provide a simple baseline for the irrigation problem, as introduced in Section 4.1.

Methods Overall, the irrigation use case follows the same methods than the fertilization use case (Section 6.1). It only differs from the fertilization use case in the observation space and return function. Table 8 details the default observation space for the irrigation problem. Denoting $\texttt{topwt}(t, t+1)$ the above the ground population biomass change between t and $t+1$ (kg/ha); and $\texttt{amir}(t)$ the irrigated water on day t (L/m^2), the default irrigation return function was defined as:

$$r(t) = \underbrace{\texttt{topwt}(t, t+1)}_{\substack{\text{change in above} \\ \text{the ground biomass}}} - \underbrace{15}_{\substack{\text{penalty} \\ \text{factor}}} \times \underbrace{\texttt{amir}(t)}_{\substack{\text{irrigated water} \\ \text{quantity}}} \qquad (6)$$

We considered 3 different policies:

- The 'null' policy that never irrigated, which corresponded to rainfed crops. Agronomists may measure the effect of an irrigation policy as a performance gain compared to the null policy, in order to decouple the effect of irrigation from the effect of rainfall (Howell, 2003).

- The second baseline was the "expert" policy, which was an approximation of the irrigation policy of the original maize field experiment (Hunt and Boote, 1998, UFGA8201 experiment number #3), see Section 4.1. As Table 6 shows, this policy consisted in sixteen deterministic water applications, which only depended on the number of days after planting. In contrast with the fertilization expert policy (Table 3), this irrigation expert policy was a simplistic approximation of the true expert policy of the original field experiment. Indeed, the true expert policy, unavailable, was likely to depend on more factors (e.g. soil moisture, or days without effective rainfall in a given growth stage) rather than only on days after planting. Nevertheless, the irrigation policy in Table 6 was still a convenient baseline for this experiment.

- The policy learned by PPO.

For an irrigation policy π, denoting $\texttt{grnwt}^\pi$ the dry matter grain yield of the policy π (kg/ha) ; $\texttt{grnwt}^0$ the dry matter grain yield with no fertilization (kg/ha); and $\texttt{totir}^\pi$ the total irrigated

Table 6: Expert irrigation policy. 'DAP' stands for Day After Planting.

DAP	quantity (L/m^2)
6	13
20	10
37	10
50	13
54	18
65	25
69	25
72	13
75	15
77	19
80	20
84	20
91	15
101	19
104	4
105	25

Table 7: Performance indicators for irrigation policies. An hyphen means `gym-DSSAT` does not directly provide the variable, but it can be easily derived.

variable	definition	comment
grnwt	grain yield (kg/ha)	quantitative objective to be maximized
totir	total irrigation (L/m^2)	cost to be minimized
–	application number	cost to be minimized
–	water use efficiency (kg/m^3)	agronomic criteria to be maximized
runoff	running-off water (L/m^2)	loss to be minimized
cleach	nitrate leaching (kg/ha)	loss/pollution to be minimized

water with policy π (L/m^2), we define the water use efficiency (WUE, Equation 15, Howell, 2003) as:

$$\text{WUE}^\pi(t) = 10 \times \frac{\texttt{grnwt}^\pi(t) - \texttt{grnwt}^0(t)}{\texttt{totir}^\pi(t)} \tag{7}$$

Similarly to the fertilization use case, Table 7 shows the performance indicators we considered for the irrigation problem. In particular, for excessive irrigation, nitrate leaching may increase (Meisinger and Delgado, 2002). Thus, nitrate leaching is a pollution performance indicator of irrigation.

Results Regarding the maximization of the undiscounted objective function, PPO showed the best mean performance and slightly outperformed the expert policy, but had an increase variance than the latter, see the wider range of values between upper and lower quantiles in Figure 9a. PPO water applications were more frequently found between days 80 and 120 of the simulation, which mostly corresponds to the grain filling stage, see Table 10 in Appendix. During this period, in most cases, PPO irrigated less water than the expert policy, see Figure 9b. As indicated in Table 9, PPO irrigation policy consumed in average about 49% less water than the expert policy,

Table 8: Default observation space for the irrigation task.

	definition
istage	DSSAT maize growing stage
vstage	vegetative growth stage (number of leaves)
grnwt	grain weight dry matter (kg/ha)
topwt	above the ground population biomass (kg/ha)
xlai	plant population leaf area index (m^2 leaf/m^2 soil)
tmax	maximum temperature for current day °C
srad	solar radiation during the current day (MJ/m^2/day)
dtt	growing degree days for current day (°C/day)
dap	duration after planting (day)
sw	volumetric soil water content in soil layers (cm^3 [water] / cm^3 [soil])
ep	actual plant transpiration rate (L/m^2/day)
wtdep	depth to water table (cm)
rtdep	root depth (cm)
totir	total irrigated water (L/m^2)

Table 9: Mean (st. dev.) irrigation baselines performances computed using 1000 episodes. For each criterion, bold numbers indicate the best performing policy.

	null	expert	PPO
grain yield (kg/ha)	3734.8 (1852.2)	**8306.6** (562.0)	7082.2 (1455.7)
total irrigation (l/m^2)	**0** (0)	264.0 (0)	133.8 (40.3)
application number	**0** (0)	16.0 (0.0)	16.2 (3.7)
water use efficiency (kg/m^3)	n.a.	17.3 (7.1)	**26.3** (13.6)
runoff (L/m^2)	**0.4** (3.5)	**0.4** (3.5)	**0.4** (3.5)
nitrate leaching (kg/ha)	**18.5** (12.6)	24.6 (9.0)	18.7 (9.6)

while maintaining a maize grain yield close to the one of the expert policy. Consistently, the water use efficiency (Equation 7) of PPO policy was 54% higher than for the expert policy. Total nitrate leaching for PPO policy was very close to the null policy, and was about 24% less than for the expert policy. The number of water applications were similar for both expert and PPO policies. Null, expert, and PPO policies had similar water runoff, indicating no water loss due to excessive irrigation of expert or PPO policies.

Discussion PPO showed a great efficiency advantage over the expert policy, while maintaining a comparable average grain yield. Water applications of PPO were most frequently focused during maize anthesis period, where maize water needs are the greatest and most crucial with respect to grain yield (NeSmith and Ritchie, 1992). Because the expert irrigation policy was likely to be a poor simplification of the real expert irrigation strategy, the advantage PPO irrigation strategy showed might be overly optimistic. However, because PPO has shown largely reduced irrigated water and nitrate leaching, we still deem these results interesting. An alternative baseline could be to reproduce the built-in automatic irrigation policy implemented in DSSAT (Hoogenboom et al., 2019), and compare its performance to the irrigation policy of PPO.

(a) Mean cumulated return of each of the 3 policies against the day of the simulation. Shaded area displays the $[0.05, 0.95]$ quantile range for each policy.

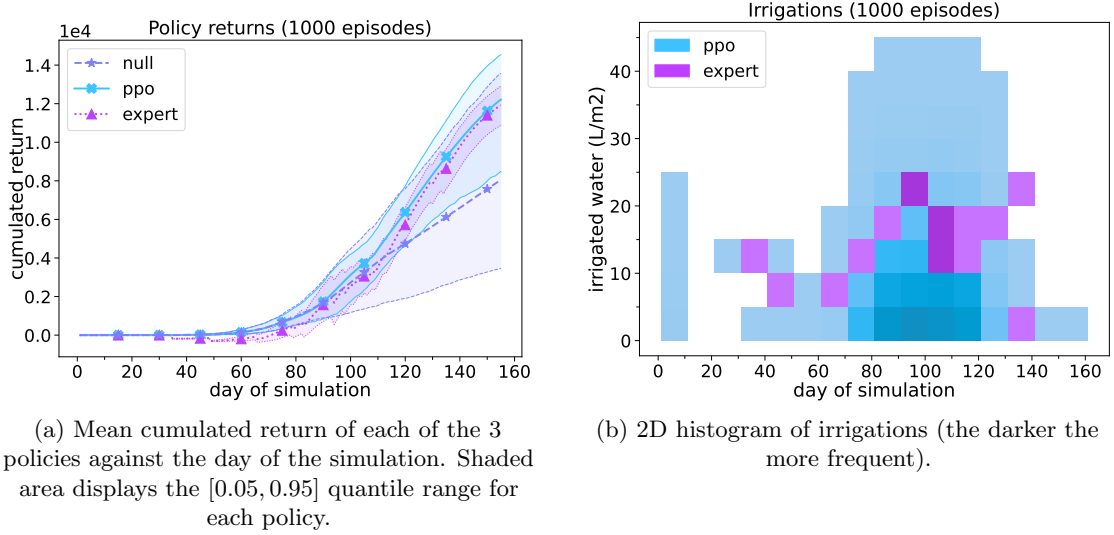(b) 2D histogram of irrigations (the darker the more frequent).

Figure 9: Undiscounted cumulated returns and applications for the irrigation problem.

Table 10: Mean (st. dev.) days of simulation to reach growth stages for the irrigation problem (1000 episodes).

istage	meaning	null	expert	ppo
8	50% of plants germinated	28 (0)	28 (0)	28 (0)
9	50% of plants with some part visible at soil surface	29 (0)	29 (0)	29 (0)
1	end of juvenile stage	40 (3)	40 (3)	40 (3)
2	50% of plants completed floral initiation	64 (4)	64 (4)	64 (4)
3	50% of plants with some silks visible outside husks	69 (4)	69 (4)	69 (4)
4	beginning of grain filling	110 (4)	110 (4)	110 (4)
5	end of grain filling	120 (4)	120 (4)	120 (4)
6	50% of plants at harvest maturity	158 (4)	158 (4)	158 (4)

Table 11: Mean (st. dev.) days of simulation to reach growth stages for the fertilization problem (1000 episodes).

istage	meaning	null	expert	ppo
8	50% of plants germinated	22 (1)	22 (1)	22 (1)
9	50% of plants with some part visible at soil surface	23 (1)	23 (1)	23 (1)
1	end of juvenile stage	34 (3)	34 (3)	34 (3)
2	50% of plants completed floral initiation	60 (5)	60 (5)	60 (5)
3	50% of plants with some silks visible outside husks	65 (5)	65 (5)	65 (5)
4	beginning of grain filling	107 (4)	107 (4)	107 (4)
5	end of grain filling	117 (4)	117 (4)	117 (4)
6	50% of plants at harvest maturity	155 (5)	155 (5)	155 (5)

B   Fertilization use case complement

Table 11 provides statistics about the growth stages for the three policies of the fertilization use case.