MLPro Documentations

Release 1.0.0

Detlef Arend, Steve Yuwono, Laxmikant Shrikant Baheti et al

Apr 30, 2024

HOME

Welcome to MLPro-Int-PettingZoo, an extension to MLPro to integrate the PettingZoo package. MLPro is a middleware framework for standardized machine learning in Python. It is developed by the South Westphalia University of Applied Sciences, Germany, and provides standards, templates, and processes for hybrid machine learning applications. PettingZoo, in turn, provides a diverse suite of environments for multi-agent reinforcement learning.

MLPro-Int-PettingZoo offers wrapper classes that allow the reuse of environments from PettingZoo in MLPro, or the other way around.

Preparation

Before running the examples, please install the latest versions of MLPro, PettingZoo, and MLPro-Int-PettingZoo as follows:

```
pip install mlpro-int-pettingzoo[full] --upgrade
```

See also

- MLPro - Machine Learning Professional
- MLPro-RL - Sub-framework for reinforcement learning
- PettingZoo - An API standard for multi-agent reinforcement learning
- Further MLPro extensions
- MLPro-Int-PettingZoo on GitHub

CHAPTER

ONE

REINFORCEMENT LEARNING

1.1 Howto RL-WP-001: MLPro Environment to PettingZoo Environment

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - The integrative middleware framework for standardized machine learning
## -- Package : mlpro_int_pettingzoo
## -- Module  : howto_rl_wp_001_mlpro_environment_to_petting_zoo_environment.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-10-02  0.0.0     SY       Creation
## -- 2021-10-02  1.0.0     SY       Released first version
## -- 2021-10-04  1.0.1     DA       Minor fix
## -- 2021-11-15  1.0.2     DA       Refactoring
## -- 2021-12-03  1.0.3     DA       Refactoring
## -- 2022-10-14  1.0.4     SY       Refactoring
## -- 2022-11-02  1.0.5     SY       Unable logging in unit test model
## -- 2023-03-02  1.0.6     LSB      Refactoring
## -- 2024-02-16  1.0.7     SY       Wrapper Relocation from MLPro to MLPro-Int-PettingZoo
## -------------------------------------------------------------------------------------------

"""
Ver. 1.0.7 (2024-02-16)

This module shows how to wrap mlpro's Environment class to petting zoo compatible.

1. How to setup an MLPro environment.

2. How to wrap MLPro's native envrionment to a Petting Zoo environment.
"""
```

(continues on next page)

(continued from previous page)

```
from mlpro.bf.various import Log
from mlpro_int_pettingzoo import WrEnvMLPro2PZoo
from mlpro.rl.pool.envs.bglp import BGLP
from pettingzoo.test import api_test


if __name__ == "__main__":
    logging = Log.C_LOG_ALL
else:
    logging = Log.C_LOG_NOTHING

# 1. Set up MLPro native environment
mlpro_env = BGLP(p_logging=logging)

# 2. Wrap the MLPro environment to PettingZoo compatible environment
env = WrEnvMLPro2PZoo(mlpro_env,
                      p_num_agents=5,
                      p_state_space=None,
                      p_action_space=None,
                      p_logging=logging).pzoo_env

# 3. Check whether the environment is valid
try:
    api_test(env, num_cycles=10, verbose_progress=False)
    print("test completed")
    assert True
except:
    print("test failed")
    assert False
```

Results

The Bulk Good Laboratory Plant (BGLP) environment will be wrapped to a PettingZoo compliant environment.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Environment BGLP: Reset
Starting API test
...
Passed API test
test completed
```

There are several lines of action processing logs due to the API tests. When there is no detected failure, the environment is successfully wrapped.

Cross Reference

- *API Reference*

1.2 Howto RL-WP-002: Run Multi-Agent on PettingZoo Environment

Executable code

```
## -------------------------------------------------------------------------------------------
## -- Project : MLPro - The integrative middleware framework for standardized machine learning
## -- Package : mlpro_int_pettingzoo
## -- Module  : howto_rl_wp_002_run_multiagent_with_own_policy_on_petting_zoo_environment.py
## -------------------------------------------------------------------------------------------
## -- History :
## -- yyyy-mm-dd  Ver.      Auth.    Description
## -- 2021-08-26  0.0.0     SY       Creation
## -- 2021-08-27  1.0.0     SY       Released first version
## -- 2021-09-11  1.0.0     MRD      Change Header information to match our new library name
## -- 2021-09-23  1.1.0     SY       Updated wrapper WrEnvPZoo class, provides two different envs
## -- 2021-09-29  1.1.1     SY       Change name: WrEnvPZoo to WrEnvPZOO2MLPro
## -- 2021-10-06  1.1.2     DA       Refactoring
## -- 2021-10-18  1.1.3     DA       Refactoring
## -- 2021-11-15  1.1.4     DA       Refactoring
## -- 2021-11-15  1.1.4     DA       Refactoring
## -- 2021-11-16  1.1.5     DA       Added explicit scenario reset with constant seeding
## -- 2021-12-03  1.1.6     DA       Refactoring
## -- 2022-02-25  1.1.7     SY       Refactoring due to auto generated ID in class Dimension
## -- 2022-05-19  1.1.8     SY       Utilize RandomGenerator
## -- 2022-05-30  1.1.9     DA       Cleaned up/rearranged a bit
## -- 2022-05-30  1.1.8     SY       Update pistonball_v5 to pistonball_v6
## -- 2022-10-08  1.2.0     SY       Turn off render: causing error due to pzoo ver 1.22.0
## -- 2022-10-14  1.2.1     SY       Refactoring
## -- 2022-11-01  1.2.2     DA       Refactoring
## -- 2022-11-02  1.2.3     SY       Unable logging in unit test model and bug fixing
## -- 2022-11-07  1.3.0     DA       Refactoring
## -- 2023-02-21  1.4.0     DA       Added save + reload + rerun steps to demonstrate/validate
## --                                 persistence of pettingzoo scenarios
## -- 2024-01-23  1.4.1     SY       Debug due to introduction of PettingZoo version 1.24.3
## -- 2024-02-16  1.4.2     SY       Wrapper Relocation from MLPro to MLPro-Int-PettingZoo
## -------------------------------------------------------------------------------------------

"""
Ver. 1.4.2 (2024-02-16)

This module shows how to run an own policy inside the MLPro standard agent model with a
```

(continues on next page)

(continued from previous page)

```
↪wrapped
Petting Zoo environment.

You will learn:

1) How to set up a scenario for a Petting Zoo environment in MLPro

2) How to run the scenario

3) How to save, reload and rerun a scenario

"""


from pathlib import Path
from pettingzoo.butterfly import pistonball_v6
from pettingzoo.classic import connect_four_v3
from mlpro.bf.math import *
from mlpro.rl import *
from mlpro_int_pettingzoo import WrEnvPZOO2MLPro
from mlpro.rl.pool.policies.randomgenerator import RandomGenerator




# 1 RL Scenario based on PettingZoo Pistonball environment
class PBScenario (RLScenario):
    """
    Reference : https://www.pettingzoo.ml/butterfly/pistonball
    """

    C_NAME      = 'Pistonball V6'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        if p_visualize:
            zoo_env         = pistonball_v6.env(render_mode="human")
        else:
            zoo_env         = pistonball_v6.env(render_mode="ansi")
        self._env           = WrEnvPZOO2MLPro(zoo_env, p_visualize=p_visualize, p_
↪logging=p_logging)

        multi_agent         = MultiAgent(p_name='Pistonball_agents', p_ada=1, p_
↪visualize=p_visualize, p_logging=p_logging)
        agent_idx           = 0
        for k in self._env._zoo_env.agents:
            agent_name      = "Agent_"+str(agent_idx)
            as_ids          = self._env.get_action_space().get_dim_ids()
            agent_ospace    = self._env.get_state_space()
            agent_asspace   = self._env.get_action_space().spawn([as_ids[agent_idx]])
            agent           = Agent(p_policy=RandomGenerator(p_observation_space=agent_
↪ospace,
```

(continues on next page)

(continued from previous page)

```
                                                                p_action_space=agent_
↪asspace,
                                                                p_buffer_size=10,
                                                                p_ada=p_ada,
                                                                p_visualize=p_visualize,
                                                                p_logging=p_logging
                                                                ),
                                        p_envmodel=None,
                                        p_id=agent_idx,
                                        p_name=agent_name,
                                        p_ada=p_ada,
                                        p_visualize=p_visualize,
                                        p_logging=p_logging
                                        )
            multi_agent.add_agent(p_agent=agent)
            agent_idx += 1

        return multi_agent



# 2 Alternative RL Scenario based on PettingZoo Connect Four environment
class C4Scenario (RLScenario):
    """
    https://www.pettingzoo.ml/classic/connect_four
    """

    C_NAME      = 'Connect Four V3'

    def _setup(self, p_mode, p_ada: bool, p_visualize: bool, p_logging) -> Model:
        if p_visualize:
            zoo_env         = connect_four_v3.env(render_mode="human")
        else:
            zoo_env         = connect_four_v3.env(render_mode="ansi")
        self._env           = WrEnvPZOO2MLPro(zoo_env, p_visualize=p_visualize, p_
↪logging=p_logging)

        multi_agent         = MultiAgent(p_name='Connect4_Agents', p_ada=1, p_
↪visualize=p_visualize, p_logging=p_logging)
        agent_idx           = 0
        for k in self._env._zoo_env.action_spaces:
            agent_name      = "Agent_"+str(agent_idx)
            as_ids          = self._env.get_action_space().get_dim_ids()
            agent_sspace    = self._env.get_state_space()
            agent_asspace   = self._env.get_action_space().spawn([as_ids[agent_idx]])
            agent           = Agent(p_policy=RandomGenerator(p_observation_space=agent_
↪sspace,
                                                                p_action_space=agent_
↪asspace,
                                                                p_buffer_size=10,
                                                                p_ada=p_ada,
                                                                p_visualize=p_visualize,
```

(continues on next page)

(continued from previous page)

```
                                                                p_logging=p_logging
                                                                ),
                                        p_envmodel=None,
                                        p_id=agent_idx,
                                        p_name=agent_name,
                                        p_ada=p_ada,
                                        p_visualize=p_visualize,
                                        p_logging=p_logging
                                        )
                multi_agent.add_agent(p_agent=agent)
                agent_idx += 1

        return multi_agent



# 3 Create scenario and run some cycles
if __name__ == "__main__":
    # 3.1 Parameters for demo mode
    logging     = Log.C_LOG_ALL
    visualize   = True
    now         = datetime.now()
    path        = str(Path.home()) + os.sep + '%04d-%02d-%02d  %02d.%02d.%02d Howto RL␣
↪WP 003' % (now.year, now.month, now.day, now.hour, now.minute, now.second)
else:
    # 3.2 Parameters for internal unit test
    logging     = Log.C_LOG_NOTHING
    visualize   = False
    path        = None


# 3.3 Instantiate one of two prepared demo scenarios
myscenario  = PBScenario(
        p_mode=Mode.C_MODE_SIM,
        p_ada=True,
        p_cycle_limit=100,
        p_visualize=visualize,
        p_logging=logging
)

# myscenario  = C4Scenario(
#         p_mode=Mode.C_MODE_SIM,
#         p_ada=True,
#         p_cycle_limit=100,
#         p_visualize=visualize,
#         p_logging=logging
# )


# 3.4 Reset and run the scenario
myscenario.reset(1)
myscenario.run()
```

(continues on next page)

(continued from previous page)

```
if __name__ == '__main__':
    # 3.5 In demo mode we save, reload and rerun the entire scenario to demonstrate␣
↪persistence
    myscenario.save(path, 'dummy')
    input('\nPress ENTER to reload and run again...\n')
    myscenario = PBScenario.load(path, 'dummy')
    myscenario.reset(1)
    myscenario.run()
```

We use the Petting Zoo environment Pistonball as default testing environment in this example. However, in step 3.3 you can also change the environment into Connect Four.

Results

By running the example code, the environment window appears and the runtime log is dumped to the terminal.

```
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Petting Zoo Env Env "connect_four_v3": Operation mode␣
↪set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_0: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Agent 0 Agent_0 added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Agent Agent_1: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Policy DiscRandPolicy: Adaptivity switched on
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Agent 1 Agent_1 added.
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Instantiated
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Operation mode set to 0
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Process time 0:00:00 :␣
↪Scenario reset with seed 1
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Init vizualization for all␣
↪agents...
....
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  Multi-Agent Connect4_Agents: Start vizualization for all␣
↪agents...
YYYY-MM-DD  HH:MM:SS.SSSSSS  I  RL-Scenario Connect Four V3: Process time 0:00:12 End of␣
↪processing
```

Cross Reference

- *API Reference*

CHAPTER

TWO

WRAPPER ROOT CLASS

Ver. 2.4.0 (2024-04-19)

This module provides wrapper classes for PettingZoo multi-agent environments.

See also: https://pypi.org/project/PettingZoo/

class mlpro_int_pettingzoo.wrappers.basics.**WrEnvPZOO2MLPro**(*p_zoo_env*, *p_state_space: MSpace = None*, *p_action_space: MSpace = None*, *p_visualize: bool = True*, *p_logging=True*)

Bases: Wrapper, Environment

This class is a ready to use wrapper class for Petting Zoo environments. Objects of this type can be treated as an environment object. Encapsulated petting zoo environment must be compatible to class pettingzoo.env.

Parameters

- **p_pzoo_env** – Petting Zoo environment object
- **p_state_space** (*MSpace*) – Optional external state space object that meets the state space of the gym environment
- **p_action_space** (*MSpace*) – Optional external action space object that meets the action space of the gym environment
- **p_visualize** (*bool*) – Boolean switch for env/agent visualisation. Default = True.
- **p_logging** – Log level (see constants of class Log). Default = Log.C_LOG_ALL.

C_TYPE = 'Wrapper PettingZoo2MLPro'

C_WRAPPED_PACKAGE = 'pettingzoo'

C_MINIMUM_VERSION = '1.20.0'

C_PLOT_ACTIVE: bool = True

C_SUPPORTED_MODULES = ['pettingzoo.classic', 'pettingzoo.butterfly', 'pettingzoo.atari', 'pettingzoo.mpe', 'pettingzoo.sisl']

_reduce_state(*p_state: dict*, *p_path: str*, *p_os_sep: str*, *p_filename_stub: str*)

The embedded PettingZoo env itself can't be pickled due to it's dependencies on Pygame. That's why the current env instance needs to be removed before pickling the object.

See also: https://stackoverflow.com/questions/52336196/how-to-save-object-using-pygame-surfaces-to-file-using-pickle

_complete_state(*p_path: str*, *p_os_sep: str*, *p_filename_stub: str*)

Custom method to complete the object state (=self) from external data sources. This method is called by standard method __setstate__() during unpickling the object from an external file.

Parameters

- **p_path** (*str*) – Path of the object pickle file (and further optional related files)
- **p_os_sep** (*str*) – OS-specific path separator.
- **p_filename_stub** (*str*) – Filename stub to be used for further optional custom data files

_recognize_space(*p_zoo_space*, *dict_name*) → ESpace

static setup_spaces()

Static template method to set up and return state and action space of environment.

Returns

- **state_space** (*MSpace*) – State space object
- **action_space** (*MSpace*) – Action space object

_reset(*p_seed=None*)

Custom method to reset the system to an initial/defined state. Use method _set_status() to set the state.

Parameters
p_seed (*int*) – Seed parameter for an internal random generator

simulate_reaction(*p_state: State*, *p_action: Action*) → State

Simulates a state transition based on a state and an action. The simulation step itself is carried out either by an internal custom implementation in method _simulate_reaction() or by an embedded external function.

Parameters

- **p_state** (*State*) – Current state.
- **p_action** (*Action*) – Action.

Returns
Subsequent state after transition

Return type
State

compute_reward(*p_state_old: State = None*, *p_state_new: State = None*) → Reward

Computes a reward for the state transition, given by two successive states. The reward computation itself is carried out either by a custom implementation in method _compute_reward() or by an embedded adaptive function.

Parameters

- **p_state_old** (*State*) – Optional state before transition. If None the internal previous state of the environment is used.
- **p_state_new** (*State*) – Optional tate after transition. If None the internal current state of the environment is used.

Returns
Reward object.

Return type
Reward

compute_success(*p_state: State*) → bool

Assesses the given state whether it is a 'success' state. Assessment is carried out either by a custom implementation in method _compute_success() or by an embedded external function.

Parameters
p_state (*State*) – State to be assessed.

Returns
success – True, if the given state is a 'success' state. False otherwise.

Return type
bool

compute_broken(*p_state: State*) → bool

Assesses the given state whether it is a 'broken' state. Assessment is carried out either by a custom implementation in method _compute_broken() or by an embedded external function.

Parameters
p_state (*State*) – State to be assessed.

Returns
broken – True, if the given state is a 'broken' state. False otherwise.

Return type
bool

init_plot(*p_figure: Figure = None*, *p_plot_settings: list = Ellipsis*, *p_plot_depth: int = 0*, *p_detail_level: int = 0*, *p_step_rate: int = 0*, ***p_kwargs*)

Initializes the plot functionalities of the class.

Parameters

- **p_figure** (*Matplotlib.figure.Figure, optional*) – Optional MatPlotLib host figure, where the plot shall be embedded. The default is None.
- **p_plot_settings** (*PlotSettings*) – Optional plot settings. If None, the default view is plotted (see attribute C_PLOT_DEFAULT_VIEW).

update_plot(***p_kwargs*)

Updates the plot.

Parameters
****p_kwargs** – Implementation-specific plot data and/or parameters.

get_cycle_limit()

Returns limit of cycles per training episode.

class mlpro_int_pettingzoo.wrappers.basics.**WrEnvMLPro2PZoo**(*p_mlpro_env: Environment*, *p_num_agents*, *p_state_space: MSpace = None*, *p_action_space: MSpace = None*, *p_logging=True*)

Bases: Wrapper

This class is a ready to use wrapper class for MLPro to PettingZoo environments. Objects of this type can be treated as an AECEnv object. Encapsulated MLPro environment must be compatible to class Environment. To be noted, this wrapper is not capable for parallel environment yet.

Parameters

- **p_mlpro_env** (*Environment*) – MLPro's Environment object
- **p_num_agents** (*int*) – Number of Agents

- **p_state_space** (*MSpace*) – Optional external state space object that meets the state space of the MLPro environment
- **p_action_space** (*MSpace*) – Optional external action space object that meets the action space of the MLPro environment

C_TYPE = 'Wrapper MLPro2PettingZoo'

C_WRAPPED_PACKAGE = 'pettingzoo'

C_MINIMUM_VERSION = '1.20.0'

class raw_env(*p_mlpro_env*, *p_num_agents*, *p_state_space: MSpace = None*, *p_action_space: MSpace = None*, *p_render_mode='human'*)

Bases: AECEnv

metadata: dict[str, Any] = {'name': 'pzoo_custom', 'render_modes': ['human', 'ansi']}

_recognize_space(*p_mlpro_space*)

step(*action*)

Accepts and executes the action of the current agent_selection in the environment.

Automatically switches control to the next agent.

observe(*agent_id*)

Returns the observation an agent currently can make.

last() calls this function.

reset(*seed*, *options*)

Resets the environment to a starting state.

render(*mode='human'*)

Renders the environment as specified by self.render_mode.

Render mode can be *human* to display a window. Other render modes in the default environments are *'rgb_array'* which returns a numpy array and is supported by all environments outside of classic, and *'ansi'* which returns the strings printed (specific to classic environments).

close()

Closes any resources that should be released.

Closes the rendering window, subprocesses, network connections, or any other resources that should be released.

CHAPTER

THREE

RELEASE NOTES

Release Notes on GitHub:

- Latest release
- Release history

Upcoming:

- Upcoming release

CHAPTER

FOUR

DISCLAIMER

If you require any more information or have any questions about our project's disclaimer, please feel free to contact us by email at mlpro@listen.fh-swf.de.

4.1 Disclaimers for MLPro Usage

All the information on this website is published in good faith and for general information purpose only. We do not make any warranties about the completeness, reliability and accuracy of this information. Any action you take upon the information you find on this project is strictly at your own risk. The South Westphalia University of Applied Sciences, Germany will not be liable for any losses and/or damages in connection with the use of MLPro.

From our website, you can visit other websites by following hyperlinks to such external sites. While we strive to provide only quality links to useful and ethical websites, we have no control over the content and nature of these sites. These links to other websites do not imply a recommendation for all the content found on these sites. Site owners and content may change without notice and may occur before we have the opportunity to remove a link which may have gone 'bad'.

Please be also aware that when you leave our website, other sites may have different privacy policies and terms which are beyond our control. Please be sure to check the Privacy Policies of these sites as well as their "Terms of Service" before engaging in any business or uploading any information.

4.2 Consent

By using our website, you hereby consent to our disclaimer and agree to its terms.

PYTHON MODULE INDEX

m

INDEX

Symbols

C

G

I

M

O

R

S

U

W