# Benchmarks

*Release 1.0.2.dev23+g97dc540*

**Mathïs Fédérico**

**Mar 02, 2023**

# API

Benchmarks is a tool to monitor and log reinforcement learning experiments. You build/find any compatible agent (only need an act method), you build/find a gym environment, and benchmarks will make them interact together ! Benchmarks also contains both tensorboard and weights&biases integrations for a beautiful and sharable experiment tracking ! Also, Benchmarks is cross platform compatible ! That's why no agents are built-in benchmarks itself.

You can build and run your own Agent in a clear and sharable manner !

```
import benchmarks as rl
import gym

class MyAgent(rl.Agent):

   def act(self, observation, greedy=False):
      """ How the Agent act given an observation """
      ...
      return action

   def learn(self):
      """ How the Agent learns from his experiences """
      ...
      return logs

   def remember(self, observation, action, reward, done, next_observation=None, info={},␣
↪**param):
      """ How the Agent will remember experiences """
      ...

env = gym.make('FrozenLake-v0', is_slippery=True) # This could be any gym-like␣
↪Environment !
agent = MyAgent(env.observation_space, env.action_space)

pg = rl.Playground(env, agent)
pg.fit(2000, verbose=1)
```

Note that 'learn' and 'remember' are optional, so this framework can also be used for baselines !

You can logs any custom metrics that your Agent/Env gives you and even chose how to aggregate them through different timescales. See the metric codes for more details.

```
metrics=[
     ('reward~env-rwd', {'steps': 'sum', 'episode': 'sum'}),
     ('handled_reward~reward', {'steps': 'sum', 'episode': 'sum'}),
     'value_loss~vloss',
     'actor_loss~aloss',
     'exploration~exp'
 ]
```

(continues on next page)

(continued from previous page)

```
pg.fit(2000, verbose=1, metrics=metrics)
```

The Playground will allow you to have clean logs adapted to your will with the verbose parameter:

- **Verbose 1**

  [episodes cycles - If your environment makes a lot of quick episodes.]

```
                 |  Env-rwd  |  Reward  |   Vloss   |   Aloss   |    Exp    |
Episode   10/200 | -0.484    | -0.0242  | 0.136     | -0.463    | 0.499     |
Episode   20/200 | -0.5      | -0.025   | 0.173     | -0.666    | 0.498     |
Episode   30/200 | -0.5      | -0.025   | 0.0968    | -0.404    | 0.497     |
Episode   40/200 | -0.5      | -0.025   | 0.0403    | -0.0717   | 0.497     |
Episode   50/200 | -0.5      | -0.025   | 0.0244    | -0.0925   | 0.496     |
Episode   60/200 | -0.499    | -0.025   | 0.0142    | -0.179    | 0.496     |
Episode   70/200 | -0.492    | -0.0246  | 0.00915   | -0.149    | 0.494     |
Episode   80/200 | -0.494    | -0.0247  | 0.0117    | -0.0301   | 0.491     |
Episode   90/200 | -0.493    | -0.0246  | 0.00764   | -0.00761  | 0.489     |
Episode  100/200 | -0.498    | -0.0249  | 0.00408   | -0.0187   | 0.487     |
Episode  110/200 | -0.498    | -0.0249  | 0.00293   | -0.0223   | 0.486     |
```

- **Verbose 2**

  [episode - To log each individual episode.]

```
                 |  Env-rwd  |  Reward  |   Vloss   |   Aloss   |    Ex
Episode    1/200 | -0.417    | -0.0208  | 0.313     | -0.127    | 0.5
Episode    2/200 | -0.405    | -0.0202  | 0.211     | 0.5       | 0.499
Episode    3/200 | -0.385    | -0.0193  | 0.239     | 0.586     | 0.499
Episode    4/200 | -0.445    | -0.0223  | 0.436     | 0.72      | 0.499
Episode    5/200 | -0.434    | -0.0217  | 0.525     | 0.708     | 0.499
Episode    6/200 | -0.434    | -0.0217  | 0.391     | 0.612     | 0.498
Episode    7/200 | -0.471    | -0.0236  | 0.269     | 0.442     | 0.498
Episode    8/200 | -0.46     | -0.023   | 0.188     | 0.297     | 0.498
Episode    9/200 | -0.459    | -0.0229  | 0.135     | 0.101     | 0.497
Episode   10/200 | -0.473    | -0.0236  | 0.134     | -0.0496   | 0.497

                 |  Env-rwd  |  Reward  |   Vloss   |   Aloss   |    Ex
Episode   11/200 | -0.47     | -0.0235  | 0.0982    | -0.171    | 0.497
Episode   12/200 | -0.472    | -0.0236  | 0.0899    | -0.285    | 0.497
Episode   13/200 | -0.472    | -0.0236  | 0.0722    | -0.362    | 0.497
```

- **Verbose 3**

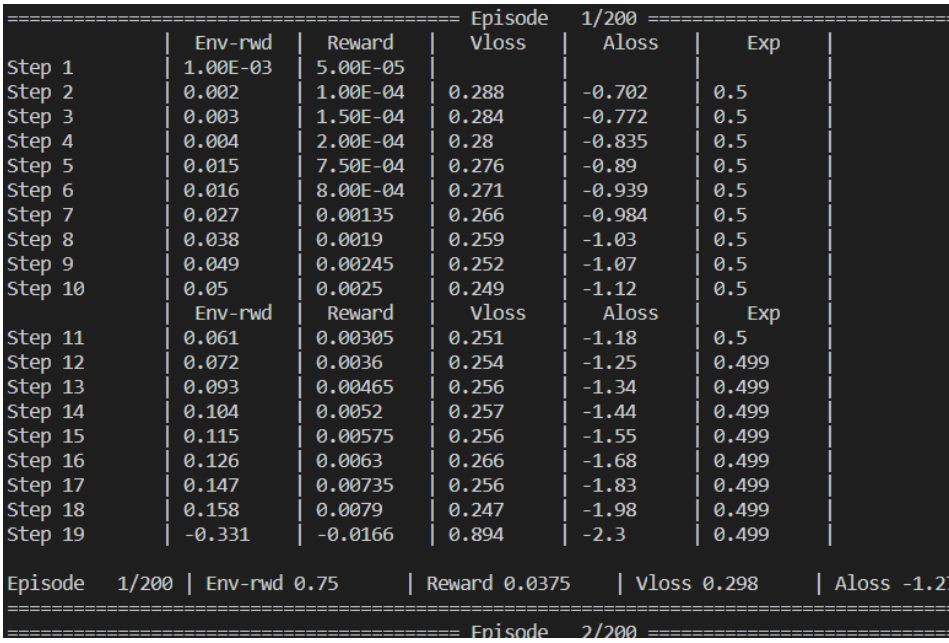
  [steps cycles - If your environment makes a lot of quick steps but has long episodes.]

```
================================== Episode    1/200 ==================================
             |  Env-rwd  |  Reward  |   Vloss   |   Aloss   |    Exp    |
Step 1-10    | 0.185     | 0.00925  | 5.08E-04  | -0.631    | 0.5       |
Step 11-20   | 1.07      | 0.0537   | 0.0048    | -1.63     | 0.499     |
Step 11-21   | 0.755     | 0.0377   | 0.0629    | -1.71     | 0.499     |

Episode    1/200 | Env-rwd 1         | Reward 0.05       | Vloss 0.0334    | Aloss -1.22     | Exp 0.499
================================================================================================
================================== Episode    2/200 ==================================
             |  Env-rwd  |  Reward  |   Vloss   |   Aloss   |    Exp    |
Step 1-10    | 0.185     | 0.00925  | 0.452     | -2.26     | 0.499     |
Step 11-20   | 1.07      | 0.0537   | 0.154     | -1.92     | 0.498     |
Step 11-21   | 0.755     | 0.0377   | 0.157     | -1.89     | 0.498     |

Episode    2/200 | Env-rwd 1         | Reward 0.05       | Vloss 0.296     | Aloss -2.07     | Exp 0.499
================================================================================================
================================== Episode    3/200 ==================================
```

- **Verbose 4**



```
========================================= Episode   1/200 ==========================
            |  Env-rwd  |  Reward   |   Vloss   |   Aloss   |    Exp    |
Step 1      | 1.00E-03  | 5.00E-05  |           |           |           |
Step 2      | 0.002     | 1.00E-04  | 0.288     | -0.702    | 0.5       |
Step 3      | 0.003     | 1.50E-04  | 0.284     | -0.772    | 0.5       |
Step 4      | 0.004     | 2.00E-04  | 0.28      | -0.835    | 0.5       |
Step 5      | 0.015     | 7.50E-04  | 0.276     | -0.89     | 0.5       |
Step 6      | 0.016     | 8.00E-04  | 0.271     | -0.939    | 0.5       |
Step 7      | 0.027     | 0.00135   | 0.266     | -0.984    | 0.5       |
Step 8      | 0.038     | 0.0019    | 0.259     | -1.03     | 0.5       |
Step 9      | 0.049     | 0.00245   | 0.252     | -1.07     | 0.5       |
Step 10     | 0.05      | 0.0025    | 0.249     | -1.12     | 0.5       |
            |  Env-rwd  |  Reward   |   Vloss   |   Aloss   |    Exp    |
Step 11     | 0.061     | 0.00305   | 0.251     | -1.18     | 0.5       |
Step 12     | 0.072     | 0.0036    | 0.254     | -1.25     | 0.499     |
Step 13     | 0.093     | 0.00465   | 0.256     | -1.34     | 0.499     |
Step 14     | 0.104     | 0.0052    | 0.257     | -1.44     | 0.499     |
Step 15     | 0.115     | 0.00575   | 0.256     | -1.55     | 0.499     |
Step 16     | 0.126     | 0.0063    | 0.266     | -1.68     | 0.499     |
Step 17     | 0.147     | 0.00735   | 0.256     | -1.83     | 0.499     |
Step 18     | 0.158     | 0.0079    | 0.247     | -1.98     | 0.499     |
Step 19     | -0.331    | -0.0166   | 0.894     | -2.3      | 0.499     |

Episode   1/200 | Env-rwd 0.75      | Reward 0.0375    | Vloss 0.298     | Aloss -1.2
==========================================================================================
========================================= Episode   2/200 ==========================
```

[step - To log each individual step.]

- **Verbose 5**

  [detailled step - To debug each individual step (with observations, actions, ...).]

```
========================================= Episode   1/200 ==========================
---------------------------------------------- Step 1 ----------------------------------------------
Env-rwd 1.00E-03
Reward 5.00E-05
Observation
[0.89496386 0.44759372 3.1709957  0.28344336 0.82520324 0.        ]
Action tf.Tensor([1. 0.], shape=(2,), dtype=float32)
Next_observation
[0.87711823 0.43898314 3.1709957  0.26964086 0.8058141  0.09999999]
------------------------------------------------------------------------------------------------------
---------------------------------------------- Step 2 ----------------------------------------------
Env-rwd 0.012
Reward 6.00E-04
Vloss 0.12
Aloss -0.00705
Exp 0.5
Observation
[0.87711823 0.43898314 3.1709957  0.26964086 0.8058141  0.09999999]
Action tf.Tensor([1. 0.], shape=(2,), dtype=float32)
Next_observation
[0.8417175  0.42407346 3.1709957  0.2446187  0.76712996 0.19999999]
------------------------------------------------------------------------------------------------------
---------------------------------------------- Step 3 ----------------------------------------------
```

The Playground also allows you to add Callbacks with ease, for example the WandbCallback to have a nice experiment tracking dashboard using weights&biases!

CHAPTER

ONE

# INSTALLATION

Install Benchmarks by running:

```
pip install benchmarks
```

CHAPTER

TWO

# DOCUMENTATION

See the latest complete documentation for more details.
See the development documentation to see what's coming !

CHAPTER

THREE

# CONTRIBUTE

- Issue Tracker.
- Projects.

CHAPTER

FOUR

# SUPPORT

If you are having issues, please contact us on Discord.

CHAPTER

FIVE

# LICENSE

The project is licensed under the GNU LGPLv3 license.
See LICENCE, COPYING and COPYING.LESSER for more details.

CHAPTER

SIX

# TABLE OF CONTENT

## 6.1 Benchmarks's Core

Benchmarks is based on those core objects: *Playground*, *Agent*, *TurnEnv*.

They are all linked by the *Playground*, as showned by this:

### 6.1.1 Playground

**class Playground**(*environement*, *agents*, *agents_order=None*)

A playground is used to run interactions between an environement and agent(s)

**env**

Environement in which the agent(s) will play.

**Type**
gym.Env

**agents**

List of agents to play.

**Type**
list of benchmarks.Agent

A playground is used to run agent(s) on an environement

**Parameters**

- **env** – Environement in which the agent(s) will play.
- **agents** (Union[*Agent*, List[*Agent*]]) – List of agents to play (can be only one agent).

**run**(*episodes*, *render=True*, *render_mode='human'*, *learn=True*, *steps_cycle_len=10*, *episodes_cycle_len=0.05*, *verbose=0*, *callbacks=None*, *logger=None*, *reward_handler=None*, *done_handler=None*, ***kwargs*)

Let the agent(s) play on the environement for a number of episodes.

Additional arguments will be passed to the default logger.

**Parameters**

- **episodes** (int) – Number of episodes to run.
- **render** (bool) – If True, call **|gym.render|** every step.
- **render_mode** (str) – Rendering mode. One of {'human', 'rgb_array', 'ansi'} (see **|gym.render|**).

- **learn** (bool) – If True, call `Agent.learn()` every step.
- **steps_cycle_len** (int) – Number of steps that compose a cycle.
- **episode_cycle_len** – Number of episodes that compose a cycle. If between 0 and 1, this in understood as a proportion.
- **verbose** (int) – The verbosity level: 0 (silent), 1 (cycle), 2 (episode), 3 (step_cycle), 4 (step), 5 (detailed step).
- **callbacks** (Optional[List[*Callback*]]) – List of *Callback* to use in runs.
- **reward_handler** (Union[Callable, *RewardHandler*, None]) – A callable to redifine rewards of the environement.
- **done_handler** (Union[Callable, *DoneHandler*, None]) – A callable to redifine the environement end.
- **logger** (Optional[*Callback*]) – Logging callback to use. If None use the default *Logger*.

**fit**(*episodes*, ***kwargs*)

Train the agent(s) on the environement for a number of episodes.

**test**(*episodes*, ***kwargs*)

Test the agent(s) on the environement for a number of episodes.

**set_agents_order**(*agents_order*)

Change the agents_order.

This will update the agents order.

**Parameters**
**agents_order** (list) – New agents indices order. Default is range(n_agents).

**Return type**
list

**Returns**
The updated agents ordered indices list.

### 6.1.2 Agent

**class Agent**

A general structure for any learning agent.

**abstract act**(*observation*, *greedy=False*)

How the *Agent* act given an observation.

**Parameters**

- **observation** – The observation given by the environment.
- **greedy** (bool) – If True, act greedely (without exploration).

**Return type**
Union[int, float, ndarray]

**learn**()

How the *Agent* learns from his experiences.

**Returns**
The agent learning logs (Has to be numpy or python).

**Return type**
logs

**remember**(*observation*, *action*, *reward*, *done*, *next_observation=None*, *info=None*, ***param*)

How the *Agent* will remember experiences.

Often, the agent will use a perfect hash functions to store observations efficiently.

#### Example

```
>>> self.memory.remember(self.observation_encoder(observation),
...                      self.action_encoder(action),
...                      reward, done,
...                      self.observation_encoder(next_observation),
...                      info, **param)
```

### 6.1.3 TurnEnv

**class TurnEnv**

Turn based multi-agents gym environment.

A layer over the gym environment class able to handle turn based environments with multiple agents.

---

**Note:** A *TurnEnv* must be in a *Playground* in order to work !

---

The only add in TurnEnv is the method "turn", On top of the main API basic methodes (see environment): * step: take a step of the environment given the action of the active player * reset: reset the environment and returns the first observation * render * close * seed

**action_space**

The Space object corresponding to actions.

**Type**
space

**observation_space**

The Space object corresponding to observations.

**Type**
space

**abstract step**(*action*)

Perform a step of the environement.

**Parameters**
**action** – The action taken by the agent who's turn was given by *turn()*.

**Returns**
The observation to give to the *Agent*. reward (float): The reward given to the *Agent* for this step. done (bool): True if the environement is done after this step. info (dict): Additional informations given by the environment.

**Return type**
observation

`abstract turn`(*state*)

Give the turn to the next agent to play.

Assuming that agents are represented by a list like range(n_player) where n_player is the number of players in the game.

**Parameters**
**state** – The state of the environement. Should be enough to determine which is the next agent to play.

**Returns**
The next player id

**Return type**
agent_id (int)

`abstract reset`()

Reset the environement and returns the initial state.

**Returns**
The observation for the first `Agent` to play

**Return type**
observation

### 6.1.4 Handlers

#### RewardHandler

`class RewardHandler`

Helper to modify the rewards given by the environment.

**You need to specify the method:**

- *reward(self, observation, action, reward, done, info, next_observation) -> float*

You can also define __init__ and reset() if you want to store anything.

`abstract reward`(*observation*, *action*, *reward*, *done*, *info*, *next_observation*, *logs*)

Replace the environment reward.

Often used to scale rewards or to do reward shaping.

**Parameters**

- **observation** – Current observation.
- **action** – Current action.
- **reward** – Current reward.
- **done** – done given by the environment.
- **info** – Addition informations given by the environment.
- **next_observation** – Next observation.

**Return type**
`float`

**reset**()

Reset the RewardHandler

Called automaticaly in *Playground.run()*. Useful only if variables are stored by the RewardHandler.

### DoneHandler

**class DoneHandler**

Helper to modify the done given by the environment.

**You need to specify the method:**

- *done(self, observation, action, reward, done, info, next_observation) -> bool*

You can also define __init__ and reset() if you want to store anything.

**abstract done**(*observation*, *action*, *reward*, *done*, *info*, *next_observation*, *logs*)

Replace the environment done.

Often used to make episodes shorter when the agent is stuck for example.

**Parameters**

- **observation** – Current observation.
- **action** – Current action.
- **reward** – Current reward.
- **done** – done given by the environment.
- **info** – Addition informations given by the environment.
- **next_observation** – Next observation.

**Return type**
bool

**reset**()

Reset the DoneHandler

Called automaticaly in *Playground.run()*. Used only if variables are stored by the DoneHandler.

## 6.2 Callbacks

### 6.2.1 Callback API

**class Callback**

An object to call functions while the *Playground* is running. You can define the custom functions *on_{position}* where position can be :

```
>>> run_begin
...     episodes_cycle_begin
...         episode_begin
...             steps_cycle_begin
...                 step_begin
...                 # env.step()
```

(continues on next page)

(continued from previous page)

```
...                 step_end
...             steps_cycle_end
...         # done==True
...         episode_end
...     episodes_cycle_end
... run_end
```

**set_params**(*params*)

Sets run parameters

**set_playground**(*playground*)

Sets reference to the used playground

**on_step_begin**(*step*, *logs=None*)

Triggers on each step beginning

**Parameters**

- **step** (int) – current step.
- **logs** (Optional[dict]) – current logs.

**on_step_end**(*step*, *logs=None*)

Triggers on each step end

**Parameters**

- **step** (int) – current step.
- **logs** (Optional[dict]) – current logs.

**on_steps_cycle_begin**(*step*, *logs=None*)

Triggers on each step cycle beginning

**Parameters**

- **step** (int) – current step.
- **logs** (Optional[dict]) – current logs.

**on_steps_cycle_end**(*step*, *logs=None*)

Triggers on each step cycle end

**Parameters**

- **step** (int) – current step.
- **logs** (Optional[dict]) – current logs.

**on_episode_begin**(*episode*, *logs=None*)

Triggers on each episode beginning

**Parameters**

- **episode** (int) – current episode.
- **logs** (Optional[dict]) – current logs.

**on_episode_end**(*episode*, *logs=None*)

Triggers on each episode end

**Parameters**

- **episode** (int) – current episode.
- **logs** (Optional[dict]) – current logs.

**on_episodes_cycle_begin**(*episode*, *logs=None*)

Triggers on each episode cycle beginning

**Parameters**

- **episode** (int) – current episode.
- **logs** (Optional[dict]) – current logs.

**on_episodes_cycle_end**(*episode*, *logs=None*)

Triggers on each episode cycle end

**Parameters**

- **episode** (int) – current episode.
- **logs** (Optional[dict]) – current logs.

**on_run_begin**(*logs=None*)

Triggers on each run beginning

**Parameters**
**logs** (Optional[dict]) – current logs.

**on_run_end**(*logs=None*)

Triggers on run end

**Parameters**
**logs** (Optional[dict]) – current logs.

### 6.2.2 Logger

**class Logger**(*metrics=None*, *detailed_step_metrics=None*, *episode_only_metrics=None*, *titles_on_top=True*)

Default logger in every *Playground* run.

This will print relevant informations in console.

You can regulate the flow of informations with the argument *verbose* in *run()* directly :

- 0 is silent (nothing will be printed)
- 1 is cycles of episodes (aggregated metrics over multiple episodes)
- 2 is every episode (aggregated metrics over all steps)
- 3 is cycles of steps (aggregated metrics over some steps)
- 4 is every step
- 5 is every step detailed (all metrics of all steps)

You can also replace it with you own *Logger*, with the argument *logger* in *run()*.

To build you own logger, you have to chose what metrics will be displayed and how will metrics be aggregated over steps/episodes/cycles. To do that, see the *Metric codes* format.

Default logger in every *Playground* run.

**Parameters**

- **metrics** (Optional[List[Union[str, tuple]]]) – Metrics to display and how to aggregate them.
- **detailed_step_metrics** (Optional[List[str]]) – Metrics to display only on detailed steps.
- **episode_only_metrics** (Optional[List[str]]) – Metrics to display only on episodes.
- **titles_on_top** (bool) – If true, titles will be displayed on top and not at every line.

**on_step_begin**(*step*, *logs=None*)

Triggers on each step beginning

**Parameters**

- **step** – current step.
- **logs** – current logs.

**on_step_end**(*step*, *logs=None*)

Triggers on each step end

**Parameters**

- **step** – current step.
- **logs** – current logs.

**on_steps_cycle_begin**(*step*, *logs=None*)

Triggers on each step cycle beginning

**Parameters**

- **step** – current step.
- **logs** – current logs.

**on_steps_cycle_end**(*step*, *logs=None*)

Triggers on each step cycle end

**Parameters**

- **step** – current step.
- **logs** – current logs.

**on_episode_begin**(*episode*, *logs=None*)

Triggers on each episode beginning

**Parameters**

- **episode** – current episode.
- **logs** – current logs.

**on_episode_end**(*episode*, *logs=None*)

Triggers on each episode end

**Parameters**

- **episode** – current episode.
- **logs** – current logs.

**on_episodes_cycle_begin**(*episode*, *logs=None*)

Triggers on each episode cycle beginning

**Parameters**

- **episode** – current episode.
- **logs** – current logs.

**on_episodes_cycle_end**(*episode*, *logs=None*)

Triggers on each episode cycle end

**Parameters**

- **episode** – current episode.
- **logs** – current logs.

**on_run_begin**(*logs=None*)

Triggers on each run beginning

**Parameters**

**logs** – current logs.

### 6.2.3 Metric codes

To fully represent a metric and how to aggregate it, we use metric codes as such:

<logs_key>~<display_name>.<aggregator_function>

Where logs_key is the metric key in logs

Display_name is the optional name that will be displayed in console.
If not specified, the logs_key will be displayed.

Finaly aggregator_function is one of { `avg`, `sum`, `last` }:

- `avg` computes the average of the metric while aggregating. (default)
- `sum` computes the sum of the metric while aggregating.
- `last` only shows the last value of the metric.

#### Examples

- `reward~rwd.sum` will aggregate the sum of rewards and display Rwd
- `loss` will show the average loss as Loss (no surname)
- `dt_step~` will show the average step_time with no name (surname is '')
- `exploration~exp.last` will show the last exploration value as Exp

# INDEX