Env

Back to top

class gymnasium.**Env** [source]

The main Gymnasium class for implementing Reinforcement Learning Agents environments.

The class encapsulates an environment with arbitrary behind-the-scenes dynamics through the `step()` and `reset()` functions. An environment can be partially or fully observed by single agents. For multi-agent environments, see PettingZoo.

The main API methods that users of this class need to know are:

- `step()` - Updates an environment with actions returning the next agent observation, the reward for taking that actions, if the environment has terminated or truncated due to the latest action and information from the environment about the step, i.e. metrics, debug info.
- `reset()` - Resets the environment to an initial state, required before calling step. Returns the first agent observation for an episode and information, i.e. metrics, debug info.
- `render()` - Renders the environments to help visualise what the agent see, examples modes are “human”, “rgb_array”, “ansi” for text.
- `close()` - Closes the environment, important when external software is used, i.e. pygame for rendering, databases

Environments have additional attributes for users to understand the implementation

- `action_space` - The Space object corresponding to valid actions, all valid actions should be contained within the space.
- `observation_space` - The Space object corresponding to valid observations, all valid observations should be contained within the space.
- `spec` - An environment spec that contains the information used to initialize the environment from `gymnasium.make()`
- `metadata` - The metadata of the environment, e.g. *{“render_modes”: [“rgb_array”, “human”], “render_fps”: 30}*. For Jax or Torch, this can be indicated to users with *“jax”=True* or *“torch”=True*.
- `np_random` - The random number generator for the environment. This is automatically assigned during `super().reset(seed=seed)` and when assessing `np_random`.

See also

For modifying or extending environments use the `gymnasium.Wrapper` class

Note

To get reproducible sampling of actions, a seed can be set with `env.action_space.seed(123)`.

and `ActType`. The `ObsType` and `ActType` are [illegible] of the observations and actions used in `reset()` and `step()`. The environment's `observation_space` and `action_space` should have type `Space[ObsType]` and `Space[ActType]`, see a space's implementation to find its parameterized type.

Back to top

↑ Back to top

`Env.step(action: ActType) → tuple[ObsT Float, bool, bool, dict[str, Any]]` [source]

Run one timestep of the environment's dynamics using the agent actions.

When the end of an episode is reached (`terminated or truncated`), it is necessary to call `reset()` to reset this environment's state for the next episode.

Changed in version 0.26: The Step API was changed removing `done` in favor of `terminated` and `truncated` to make it clearer to users when the environment had terminated or truncated which is critical for reinforcement learning bootstrapping algorithms.

PARAMETERS:

action (*ActType*) – an action provided by the agent to update the environment state.

RETURNS:

- **observation** (*ObsType*) – An element of the environment's `observation_space` as the next observation due to the agent actions. An example is a numpy array containing the positions and velocities of the pole in CartPole.
- **reward** (*SupportsFloat*) – The reward as a result of taking the action.
- **terminated** (*bool*) – Whether the agent reaches the terminal state (as defined under the MDP of the task) which can be positive or negative. An example is reaching the goal state or moving into the lava from the Sutton and Barto Gridworld. If true, the user needs to call `reset()`.
- **truncated** (*bool*) – Whether the truncation condition outside the scope of the MDP is satisfied. Typically, this is a timelimit, but could also be used to indicate an agent physically going out of bounds. Can be used to end the episode prematurely before a terminal state is reached. If true, the user needs to call `reset()`.
- **info** (*dict*) – Contains auxiliary diagnostic information (helpful for debugging, learning, and logging). This might, for instance, contain: metrics that describe the agent's performance state, variables that are hidden from observations, or individual reward terms that are combined to produce the total reward. In OpenAI Gym <v26, it contains "TimeLimit.truncated" to distinguish truncation and termination, however this is deprecated in favour of returning terminated and truncated variables.
- **done** (*bool*) – (Deprecated) A boolean value for if the episode has ended, in which case further `step()` calls will return undefined results. This was removed in OpenAI Gym v26 in favor of terminated and truncated attributes. A done signal may be emitted for different reasons: Maybe the task underlying the environment was solved successfully, a certain timelimit was exceeded, or the physics simulation has entered an invalid state.

```
Env.reset(*, seed: int | None = None, options: dict[str, Any] | None = None) →
    tuple[ObsType, dict[str, Any]]
```
[source]

Resets the environment to an initial internal state, returning an initial observation and info.

Back to top

randomness can be controlled with th eter otherwise if the environment already has a random number generat and `reset()` is called with `seed=None`, the RNG is not reset.

Therefore, `reset()` should (in the typical use case) be called with a seed right after initialization and then never again.

For Custom environments, the first line of `reset()` should be `super().reset(seed=seed)` which implements the seeding correctly.

Changed in version v0.25: The `return_info` parameter was removed and now info is expected to be returned.

PARAMETERS:

- **seed** (*optional int*) – The seed that is used to initialize the environment's PRNG (*np_random*) and the read-only attribute *np_random_seed*. If the environment does not already have a PRNG and `seed=None` (the default option) is passed, a seed will be chosen from some source of entropy (e.g. timestamp or /dev/urandom). However, if the environment already has a PRNG and `seed=None` is passed, the PRNG will *not* be reset and the env's `np_random_seed` will *not* be altered. If you pass an integer, the PRNG will be reset even if it already exists. Usually, you want to pass an integer *right after the environment has been initialized and then never again*. Please refer to the minimal example above to see this paradigm in action.
- **options** (*optional dict*) – Additional information to specify how the environment is reset (optional, depending on the specific environment)

RETURNS:

- **observation** (*ObsType*) – Observation of the initial state. This will be an element of `observation_space` (typically a numpy array) and is analogous to the observation returned by `step()`.
- **info** (*dictionary*) – This dictionary contains auxiliary information complementing `observation`. It should be analogous to the `info` returned by `step()`.

`Env.render() → RenderFrame | list[RenderFrame] | None` [source]

Compute the render frames as specified by `render_mode` during the initialization of the environment.

The environment's `metadata` render modes (*env.metadata["render_modes"]*) should contain the possible ways to implement the render modes. In addition, list versions for most render modes is achieved through *gymnasium.make* which automatically applies a wrapper to collect rendered frames.

initialised in `__init__`.

Back to top

By convention, if the `render_mode` is:

- None (default): no render is computed.
- “human”: The environment is continuously rendered in the current display or terminal, usually for human consumption. This rendering should occur during `step()` and `render()` doesn’t need to be called. Returns `None`.
- “rgb_array”: Return a single frame representing the current state of the environment. A frame is a `np.ndarray` with shape `(x, y, 3)` representing RGB values for an x-by-y pixel image.
- “ansi”: Return a strings (`str`) or `StringIO.StringIO` containing a terminal-style text representation for each time step. The text can include newlines and ANSI escape sequences (e.g. for colors).
- “rgb_array_list” and “ansi_list”: List based version of render modes are possible (except Human) through the wrapper, `gymnasium.wrappers.RenderCollection` that is automatically applied during `gymnasium.make(..., render_mode="rgb_array_list")`. The frames collected are popped after `render()` is called or `reset()`.

Note

Make sure that your class’s `metadata` `"render_modes"` key includes the list of supported modes.

Changed in version 0.25.0: The render function was changed to no longer accept parameters, rather these parameters should be specified in the environment initialised, i.e., `gymnasium.make("CartPole-v1", render_mode="human")`

`Env.close()` [source]

After the user has finished using the environment, close contains the code necessary to “clean up” the environment.

This is critical for closing rendering windows, database or HTTP connections. Calling `close` on an already closed environment has no effect and won’t raise an error.

Back to top

Env.action_space: spaces.Space[ActType

The Space object corresponding to valid actions, all valid actions should be contained with the space. For example, if the action space is of type *Discrete* and gives the value *Discrete(2)*, this means there are two valid discrete actions: *0* & *1*.

```
>>> env.action_space
Discrete(2)
>>> env.observation_space
Box(-inf, inf, (4,), float32)
```

Env.observation_space: spaces.Space[ObsType]

The Space object corresponding to valid observations, all valid observations should be contained with the space. For example, if the observation space is of type `Box` and the shape of the object is `(4,)`, this denotes a valid observation will be an array of 4 numbers. We can check the box bounds as well with attributes.

```
>>> env.observation_space.high
array([4.8000002e+00, inf, 4.1887903e-01, inf], dtype=float32)
>>> env.observation_space.low
array([-4.8000002e+00, -inf, -4.1887903e-01, -inf], dtype=float32)
```

Env.metadata: dict[str, Any] = {'render_modes': []}

The metadata of the environment containing rendering modes, rendering fps, etc

Env.render_mode: str | None = None

The render mode of the environment determined at initialisation

Env.spec: EnvSpec | None = None

The `EnvSpec` of the environment normally set during `gymnasium.make()`

property Env.unwrapped: Env[ObsType, ActType]

Returns the base non-wrapped environment.

RETURNS:

Env – The base non-wrapped `gymnasium.Env` instance

property Env.np_random: Generator

RETURNS:

Instances of `np.random.Generator`

---

property Env.**np_random_seed**: **int**

Returns the environment's internal `_np_random_seed` that if not set will first initialise with a random int as seed.

If `np_random_seed` was set directly instead of through `reset()` or `set_np_random_through_seed()`, the seed will take the value -1.

RETURNS:

int – the seed of the current *np_random* or -1, if the seed of the rng is unknown

Implementing environments

When implementing an environment, the `Env.reset()` and `Env.step()` functions must be created to describe the dynamics of the environment. For more information, see the environment creation tutorial.

Creating environments

To create an environment, gymnasium provides `make()` to initialise the environment along with several important wrappers. Furthermore, gymnasium provides `make_vec()` for creating vector environments and to view all the environment that can be created use `pprint_registry()`.