MineRL

Release 0.3.0

William H. Guss, Brandon Houghton

Jul 04, 2022

TUTORIALS AND GUIDES

Welcome to documentation for the MineRL project and its related repositories and components!

CHAPTER

ONE

WHAT IS MINERL

MineRL is a research project started at Carnegie Mellon University aimed at developing various aspects of artificial intelligence within Minecraft. In short MineRL consists of several major components:

- MineRL-v0 Dataset – One of the largest imitation learning datasets with over **60 million frames** of recorded human player data. The dataset includes a set of environments which highlight many of the hardest problems in modern-day Reinforcement Learning: sparse rewards and hierarchical policies.
- `minerl` – A rich python3 package for doing artificial intelligence research in Minecraft. This includes two major submodules. We develop `minerl` in our spare time, please consider supporting us on Patreon
  - `minerl.env` – A growing set of OpenAI Gym environments in Minecraft. These environments leverage a **synchronous**, **stable**, and **fast** fork of Microsoft Malmo called *MineRLEnv*.
  - `minerl.data` – The main python module for ext with the *MineRL-v0* dataset

1.1 Installation

Welcome to MineRL! This guide will get you started.

To start using the data and set of reinforcement learning environments comprising MineRL, you'll need to install the main python package, `minerl`.

1. First **make sure you have JDK 1.8** installed on your system.
   a. Windows installer – On windows go this link and follow the instructions to install JDK 8.
   b. On Mac, you can install java8 using homebrew and AdoptOpenJDK (an open source mirror, used here to get around the fact that Java8 binaries are no longer available directly from Oracle):

```
brew tap AdoptOpenJDK/openjdk
brew cask install adoptopenjdk8
```

   c. On Debian based systems (Ubuntu!) you can run the following:

```
sudo add-apt-repository ppa:openjdk-r/ppa
sudo apt-get update
sudo apt-get install openjdk-8-jdk
```

2. Now install the `minerl` package!:

```
pip3 install --upgrade minerl
```

Note: depending on your system you may need the user flag: `pip3 install --upgrade minerl --user` to install property

3. (Optional) Download the dataset ~ 60 GB:

```
import minerl
minerl.data.download(directory="/your/local/path/")
```

Or simply download a single experiment

```
minerl.data.download('/your/local/path',experiment='MineRLObtainDiamondVectorObf-v0
↪')
```

For a complete list of published experiments, checkout the environment documentation. If you are here for the MineRL competition, checkout the competition environments.

That's it! Now you're good to go :)

Caution: Currently `minerl` only supports environment rendering in **headed environments** (machines with monitors attached).

In order to run `minerl` environments without a head use a software renderer such as `xvfb`:

```
xvfb-run python3 <your_script.py>
```

1.2 Hello World: Your First Agent

With the `minerl` package installed on your system you can now make your first agent in Minecraft!

To get started, let's first import the necessary packages

```
import gym
import minerl
```

1.2.1 Creating an environment

Now we can choose any one of the many environments included in the `minerl` package. To learn more about the environments checkout the environment documentation.

For this tutorial we'll choose the `MineRLNavigateDense-v0` environment. In this task, the agent is challenged with using a first-person perspective of a random Minecraft map and navigating to a target.

To create the environment, simply invoke `gym.make`

```
env = gym.make('MineRLNavigateDense-v0')
```

Caution: Currently `minerl` only supports environment rendering in **headed environments** (servers with monitors attached).

In order to run `minerl` environments without a head use a software renderer such as `xvfb`:

```
xvfb-run python3 <your_script.py>
```

Note: The first time you run this command to complete, it will take a while as it is recompiling Minecraft with the MineRL simulator mod!

If you're worried and want to make sure something is happening behind the scenes install a logger **before** you create the envrionment.

```
import logging
logging.basicConfig(level=logging.DEBUG)

env = gym.make('MineRLNavigateDense-v0')
```

1.2.2 Taking actions

As a warm up let's create a random agent.

Now we can reset this environment to its first position and get our first observation from the agent by resetting the environment.

```
obs = env.reset()
```

The `obs` variable will be a dictionary containing the following observations returned by the environment. In the case of the `MineRLNavigate-v0` environment, three observations are returned: `pov`, an RGB image of the agent's first person perspective; `compassAngle`, a float giving the angle of the agent to its (approximate) target; and `inventory`, a dictionary containing the amount of `'dirt'` blocks in the agent's inventory (this is useful for climbing steep inclines).

```
{
    'pov': array([[[ 63,  63,  68],
        [ 63,  63,  68],
        [ 63,  63,  68],
        ...,
        [ 92,  92, 100],
        [ 92,  92, 100],
        [ 92,  92, 100]],,


        ...,


        [[ 95, 118, 176],
        [ 95, 119, 177],
        [ 96, 119, 178],
        ...,
        [ 93, 116, 172],
        [ 93, 115, 171],
        [ 92, 115, 170]]], dtype=uint8),
    'compassAngle': -63.48639,
    'inventory': {'dirt': 0}
}
```

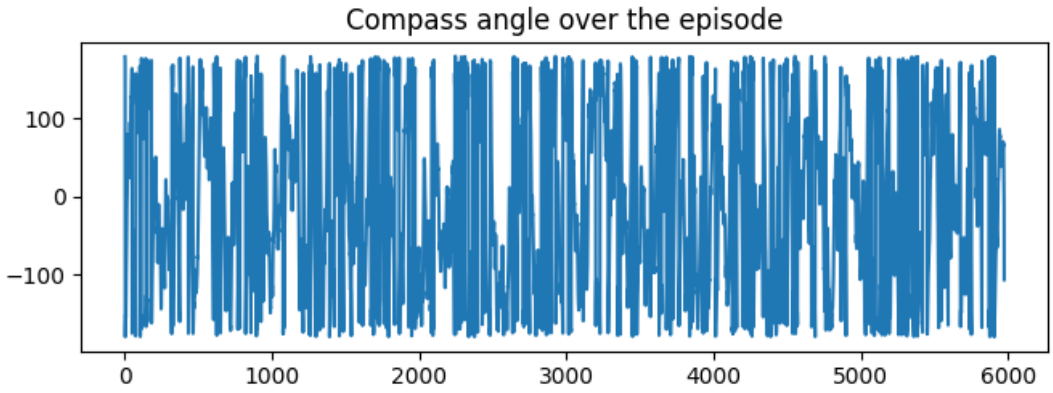
Note: To see the exact format of observations returned from and the exact action format expected by `env.step` for any environment refer to the environment reference documentation!

Now let's take actions through the environment until time runs out or the agent dies. To do this, we will use the normal OpenAI Gym `env.step` method.

```
done = False

while not done:
    action = env.action_space.sample()
    obs, reward, done, _ = env.step(action)
```

After running this code you should see your agent move sporadically until the `done` flag is set to true. To confirm that our agent is at least qualitatively acting randomly, on the right is a plot of the compass angle over the course of the experiment.



1.2.3 No-op actions and a better policy

Now let's make a hard-coded agent that actually runs towards the target.

To do this at every step of the environment we will take the *noop* action with a few modifications; in particular, we will only move forward, jump, attack, and change the agent's direction to minimize the angle between the agent's movement direction and it's target, `compassAngle`.

```
import minerl
import gym
env = gym.make('MineRLNavigateDense-v0')


obs  = env.reset()
done = False
net_reward = 0

while not done:
    action = env.action_space.noop()

    action['camera'] = [0, 0.03*obs["compassAngle"]]
    action['back'] = 0
```

(continues on next page)

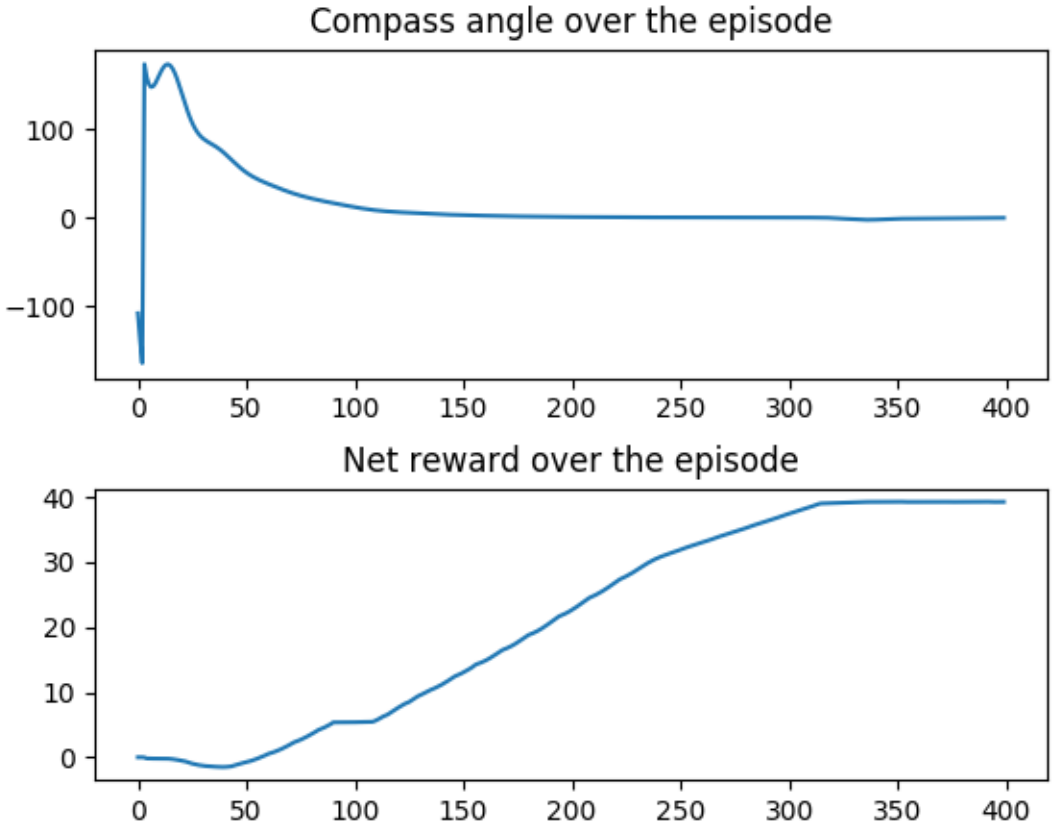
(continued from previous page)

```
    action['forward'] = 1
    action['jump'] = 1
    action['attack'] = 1

    obs, reward, done, info = env.step(
        action)

    net_reward += reward
    print("Total reward: ", net_reward)
```

After running this agent, you should notice marekedly less sporadic behaviour. Plotting both the `compassAngle` and the net reward over the episode confirm that this policy performs better than our random policy.



Congratulations! You've just made your first agent using the `minerl` framework!

1.3 Sampling The Dataset

Now that your agent can act in the environment, we should show it how to leverage human demonstrations.

To get started, let's ensure the data has been downloaded.

```
# Unix, Linux
$MINERL_DATA_ROOT="your/local/path" python3 -m minerl.data.download
```

(continues on next page)

(continued from previous page)

```
# Windows
$env:MINERL_DATA_ROOT="your/local/path"; python3 -m minerl.data.download
```

Or we can simply download a single experiment

```
# Unix, Linux
$MINERL_DATA_ROOT="your/local/path" python3 -m minerl.data.download "MineRLObtainDiamond-
↪v0"

# Windows
$env:MINERL_DATA_ROOT="your/local/path"; python3 -m minerl.data.download
↪"MineRLObtainDiamond-v0"
```

For a complete list of published experiments, checkout the environment documentation. You can also download the data in your python scripts

Now we can build the datast for `MineRLObtainDiamond-v0`

```
data = minerl.data.make(
    'MineRLObtainDiamond-v0')


for current_state, action, reward, next_state, done \
    in data.batch_iter(
        batch_size=1, num_epochs=1, seq_len=32):

        # Print the POV @ the first step of the sequence
        print(current_state['pov'][0])

        # Print the final reward pf the sequence!
        print(reward[-1])

        # Check if final (next_state) is terminal.
        print(done[-1])

        # ... do something with the data.
        print("At the end of trajectories the length"
              "can be < max_sequence_len", len(reward))
```

Warning: The `minerl` package uses environment variables to locate the data directory. For portability, plese define `MINERL_DATA_ROOT` as `/your/local/path/` in your system environment variables.

1.3.1 Moderate Human Demonstrations

MineRL-v0 uses community driven demonstrations to help researchers develop sample efficient techniques. Some of these demonstrations are less than optimal, however others could feacture bugs with the client, server errors, or adversarial behavior.

Using the MineRL viewer, you can help curate this dataset by viewing these demonstrations manually and reporting bad streams by submitting an issue to github with the following information:

1. The stream name of the stream in question
2. The reason the stream or segment needs to be modified
3. The sample / frame number(s) (shown at the bottom of the viewer)

1.4 K-means exploration

With the 2020 MineRL competition, we introduced vectorized obfuscated environments which abstract non-visual state information as well as the action space of the agent to be continuous vector spaces. See MineRLObtainDiamondVectorObf-v0 for documentation on the evaluation environment for that competition.

To use techniques in the MineRL competition that require discrete actions, we can use k-means to quantize the human demonstrations and give our agent n discrete actions representative of actions taken by humans when solving the environment.

To get started, let's download the MineRLTreechopVectorObf-v0 environment.

```
python -m minerl.data.download 'MineRLTreechopVectorObf-v0'
```

Note: If you are unable to download the data ensure you have the `MINERL_DATA_ROOT` env variable set as demonstrated in data sampling.

Now we load the dataset for MineRLTreechopVectorObf-v0 and find 32 clusters using sklearn learn

```
from sklearn.cluster import KMeans

dat = minerl.data.make('MineRLTreechopVectorObf-v0')

# Load the dataset storing 1000 batches of actions
act_vectors = []
for _, act, _, _,_ in tqdm.tqdm(dat.batch_iter(16, 32, 2, preload_buffer_size=20)):
    act_vectors.append(act['vector'])
    if len(act_vectors) > 1000:
        break

# Reshape these the action batches
acts = np.concatenate(act_vectors).reshape(-1, 64)
kmeans_acts = acts[:100000]

# Use sklearn to cluster the demonstrated actions
kmeans = KMeans(n_clusters=32, random_state=0).fit(kmeans_acts)
```

Now we have 32 actions that represent reasonable actions for our agent to take. Let's take these and improve our random hello world agent from before.

```
i, net_reward, done, env = 0, 0, False, gym.make('MineRLTreechopVectorObf-v0')
obs = env.reset()

while not done:
    # Let's use a frame skip of 4 (could you do better than a hard-coded frame skip?)
    if i % 4 == 0:
        action = {
            'vector': kmeans.cluster_centers_[np.random.choice(NUM_CLUSTERS)]
        }

    obs, reward, done, info = env.step(action)
    env.render()

    if reward > 0:
        print("+{} reward!".format(reward))
    net_reward += reward
    i += 1

print("Total reward: ", net_reward)
```

Putting this all together we get:

Full snippet

```
import gym
import tqdm
import minerl
import numpy as np

from sklearn.cluster import KMeans

dat = minerl.data.make('MineRLTreechopVectorObf-v0')

act_vectors = []
NUM_CLUSTERS = 30

# Load the dataset storing 1000 batches of actions
for _, act, _, _, _ in tqdm.tqdm(dat.batch_iter(16, 32, 2, preload_buffer_size=20)):
    act_vectors.append(act['vector'])
    if len(act_vectors) > 1000:
        break

# Reshape these the action batches
acts = np.concatenate(act_vectors).reshape(-1, 64)
kmeans_acts = acts[:100000]

# Use sklearn to cluster the demonstrated actions
kmeans = KMeans(n_clusters=NUM_CLUSTERS, random_state=0).fit(kmeans_acts)

i, net_reward, done, env = 0, 0, False, gym.make('MineRLTreechopVectorObf-v0')
obs = env.reset()
```

(continues on next page)

(continued from previous page)

```
while not done:
    # Let's use a frame skip of 4 (could you do better than a hard-coded frame skip?)
    if i % 4 == 0:
        action = {
            'vector': kmeans.cluster_centers_[np.random.choice(NUM_CLUSTERS)]
        }

    obs, reward, done, info = env.step(action)
    env.render()

    if reward > 0:
        print("+{} reward!".format(reward))
    net_reward += reward
    i += 1

print("Total reward: ", net_reward)
```

Try comparing this k-means random agent with a random agent using `env.action_space.sample()`! You should see the human actions are a much more reasonable way to explore the environment!

1.5 Visualizing The Data `minerl.viewer`

To help you get familiar with the MineRL dataset, the `minerl` python package also provides a data trajectory viewer called `minerl.viewer`:

The `minerl.viewer` program lets you step through individual trajectories, showing the observation seen the player, the action they took (including camera, movement, and any action described by an MineRL environment's action space), and the reward they received.

```
usage: python3 -m minerl.viewer [-h] environment [stream_name]

positional arguments:
  environment  The MineRL environment to visualize. e.g.
               MineRLObtainDiamondDense-v0
  stream_name  (optional) The name of the trajectory to visualize. e.g.
               v3_absolute_zucchini_basilisk-13_36805-50154.

optional arguments:
  -h, --help   show this help message and exit
```

Try it out on a random trajectory by running:

```
# Make sure your MINERL_DATA_ROOT is set!
export MINERL_DATA_ROOT='/your/local/path'

# Visualizes a random trajectory of MineRLObtainDiamondDense-v0
python3 -m minerl.viewer MineRLObtainDiamondDense-v0
```

Try it out on a specific trajectory by running:

```
# Make sure your MINERL_DATA_ROOT is set!
export MINERL_DATA_ROOT='/your/local/path'
# Visualizes a specific trajectory. v3_absolute_zucch...
python3 -m minerl.viewer MineRLTreechop-v0 \
        v3_absolute_zucchini_basilisk-13_36805-50154
```

1.6 Interactive Mode `minerl.interactor`

Once you have started training agents, the next step is getting them to interact with human players. To help achieve this, the `minerl` python package provides a interactive Minecraft client called `minerl.interactor`:

The `minerl.interactor` allows you to connect a human-controlled Minecraft client to the Minecraft world that your agent(s) is using and interact with the agent in real time.

Note: For observation-only mode hit the `t` key and type `/gamemode sp` to enter spectator mode and become invisible to your agen``t(s).

Enables human interaction with the environment.

To interact with the environment add *make_interactive* to your agent's evaluation code and then run the *minerl.interactor.*

For example:

```
env = gym.make('MineRL...')

# set the environment to allow interactive connections on port 6666
# and slow the tick speed to 6666.
env.make_interactive(port=6666, realtime=True)

# reset the env
env.reset()

# interact as normal.
...
```

Then while the agent is running, you can start the interactor with the following command.

```
python3 -m minerl.interactor 6666 # replace with the port above.
```

The interactor will disconnect when the mission resets, but you can connect again with the same command. If an interactor is already started, it won't need to be relaunched when running the commnad a second time.

1.7 General Information

The `minerl` package includes several environments as follows. This page describes each of the included environments, provides usage samples, and describes the exact action and observation space provided by each environment!

Caution: In the MineRL Competition, many environments are provided for training, however competition agents will only be evaluated in `MineRLObtainDiamondVectorObf-v0` which has **sparse** rewards. See *MineRLObtainDiamondVectorObf-v0*.

Note: All environments offer a default no-op action via `env.action_space.no_op()` and a random action via `env.action_space.sample()`.

1.8 Environment Handlers

Minecraft is an extremely complex environment which provides players with visual, auditory, and informational observation of many complex data types. Furthermore, players interact with Minecraft using more than just embodied actions: players can craft, build, destroy, smelt, enchant, manage their inventory, and even communicate with other players via a text chat.

To provide a unified interface with which agents can obtain and perform similar observations and actions as players, we have provided first-class for support for this multi-modality in the environment: **the observation and action spaces of environments are `gym.spaces.Dict` spaces.** These observation and action dictionaries are comprised of individual fields we call *handlers*.

Note: In the documentation of every environment we provide a listing of the exact `gym.space` of the observations returned by and actions expected by the environment's step function. We are slowly building documentation for these handlers, and **you can click those highlighted with blue** for more information!

1.8.1 Environment Handlers

Minecraft is an extremely complex environment which provides players with visual, auditory, and informational observation of many complex data types. Furthermore, players interact with Minecraft using more than just embodied actions: players can craft, build, destroy, smelt, enchant, manage their inventory, and even communicate with other players via a text chat.

To provide a unified interface with which agents can obtain and perform similar observations and actions as players, we have provided first-class for support for this multi-modality in the environment: **the observation and action spaces of environments are `gym.spaces.Dict` spaces.** These observation and action dictionaries are comprised of individual fields we call *handlers*.

Note: In the documentation of every environment we provide a listing of the exact `gym.space` of the observations returned by and actions expected by the environment's step function. We are slowly building documentation for these handlers, and **you can click those highlighted with blue** for more information!

1.8.2 Observations

Visual Observations - `pov`, `third-persion`

`pov : Box(width, height, nchannels)`

An RGB image observation of the agent's first-person perspective.

Type
`np.uint8`

`third-person : Box(width, height, nchannels)`

An RGB image observation of the agent's third-person perspective.

> **Warning:** This observation is not yet supported by any environment.

Type
`np.uint8`

`compass-observation : Box(1)`

The current position of the *minecraft:compass* object from 0 (behind agent left) to 0.5 in front of agent to 1 (behind agent right)

Note: This observation uses the default Minecraft game logic which includes compass needle momentum. As such it may change even when the agent has stoped moving!

1.8.3 Actions

Camera Control - `camera`

`camera : Box(2) [delta_pitch, delta_yaw]`

This action changes the orientation of the agent's head by the corresponding number of degrees. When the `pov` observation is available, the camera changes its orientation pitch by the first component and its yaw by the second component. Both `delta_pitch` and `delta_yaw` are limited to [-180, 180] inclusive

Type
`np.float32`

Shape
[2]

`attack : Discrete(1) [attack]`

This action causes the agent to attack.

Type
`np.float32`

Shape
[1]

Tool Control - `camera`

`equip : Enum('none', 'air', 'wooden_axe', 'wooden_pickaxe', 'stone_axe', 'stone_pickaxe', 'iron_axe', 'iron_pickaxe'))`

This action equips the first instance of the specified item from the agents inventory to the main hand if the specified item is present, otherwise does nothing. `none` is never a valid item and functions as a no-op action. `air` matches any empty slot in an agent's inventory and functions as an un-equip action.

Type
`np.int64`

Shape
[1]

Note: Agents may un-equip items by performing the `equip air` action.

1.9 Basic Environments

Warning: The following basic environments are NOT part of the 2020 MineRL Competition! Feel free to use them for exploration but agents may only be trained on MineRL competition environments!

1.9.1 MineRLTreechop-v0

In treechop, the agent must collect 64 *minercaft:log*. This replicates a common scenario in Minecraft, as logs are necessary to craft a large amount of items in the game, and are a key resource in Minecraft.

The agent begins in a forest biome (near many trees) with an iron axe for cutting trees. The agent is given +1 reward for obtaining each unit of wood, and the episode terminates once the agent obtains 64 units.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLTreechop-v0") # A MineRLTreechop-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLTreechop-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.2 MineRLNavigate-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is sparse.**

In this environment, the agent spawns on a random survival map.

Observation Space

```
Dict({
    "compassAngle": "Box(low=-180.0, high=180.0, shape=())",
    "inventory": {
            "dirt": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "place": "Enum(dirt,none)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigate-v0") # A MineRLNavigate-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigate-v0")
```

(continues on next page)

(continued from previous page)

```
# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.3 MineRLNavigateExtreme-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is sparse.**

In this environment, the agent spawns in an extreme hills biome.

Observation Space

```
Dict({
    "compassAngle": "Box(low=-180.0, high=180.0, shape=())",
    "inventory": {
            "dirt": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "place": "Enum(dirt,none)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateExtreme-v0") # A MineRLNavigateExtreme-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateExtreme-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.4 MineRLNavigateDense-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.**

In this environment, the agent spawns on a random survival map.

Observation Space

```
Dict({
    "compassAngle": "Box(low=-180.0, high=180.0, shape=())",
    "inventory": {
            "dirt": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "place": "Enum(dirt,none)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```python
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateDense-v0") # A MineRLNavigateDense-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateDense-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.5 MineRLNavigateExtremeDense-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.**

In this environment, the agent spawns in an extreme hills biome.

Observation Space

```
Dict({
    "compassAngle": "Box(low=-180.0, high=180.0, shape=())",
    "inventory": {
            "dirt": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "place": "Enum(dirt,none)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateExtremeDense-v0") # A MineRLNavigateExtremeDense-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateExtremeDense-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.6 MineRLObtainDiamond-v0

In this environment the agent is required to obtain a diamond. The agent begins in a random starting location on a random survival map without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected summary of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode the agent is rewarded **every** time it obtains an item in the requisite item hierarchy to obtaining a diamond. The rewards for each item are given here:

```
<Item reward="1" type="log" />
<Item reward="2" type="planks" />
<Item reward="4" type="stick" />
<Item reward="4" type="crafting_table" />
<Item reward="8" type="wooden_pickaxe" />
<Item reward="16" type="cobblestone" />
<Item reward="32" type="furnace" />
<Item reward="32" type="stone_pickaxe" />
<Item reward="64" type="iron_ore" />
<Item reward="128" type="iron_ingot" />
<Item reward="256" type="iron_pickaxe" />
<Item reward="1024" type="diamond" />
```

Observation Space

```
Dict({
    "equipped_items.mainhand.damage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.maxDamage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.type": "Enum(air,iron_axe,iron_pickaxe,none,other,stone_axe,
↪stone_pickaxe,wooden_axe,wooden_pickaxe)",
    "inventory": {
            "coal": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "crafting_table": "Box(low=0, high=2304, shape=())",
            "dirt": "Box(low=0, high=2304, shape=())",
            "furnace": "Box(low=0, high=2304, shape=())",
            "iron_axe": "Box(low=0, high=2304, shape=())",
            "iron_ingot": "Box(low=0, high=2304, shape=())",
            "iron_ore": "Box(low=0, high=2304, shape=())",
            "iron_pickaxe": "Box(low=0, high=2304, shape=())",
            "log": "Box(low=0, high=2304, shape=())",
            "planks": "Box(low=0, high=2304, shape=())",
            "stick": "Box(low=0, high=2304, shape=())",
            "stone": "Box(low=0, high=2304, shape=())",
            "stone_axe": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "torch": "Box(low=0, high=2304, shape=())",
            "wooden_axe": "Box(low=0, high=2304, shape=())",
            "wooden_pickaxe": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "craft": "Enum(crafting_table,none,planks,stick,torch)",
    "equip": "Enum(air,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,wooden_axe,
↪wooden_pickaxe)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "nearbyCraft": "Enum(furnace,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,
↪wooden_axe,wooden_pickaxe)",
    "nearbySmelt": "Enum(coal,iron_ingot,none)",
    "place": "Enum(cobblestone,crafting_table,dirt,furnace,none,stone,torch)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainDiamond-v0") # A MineRLObtainDiamond-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainDiamond-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.7 MineRLObtainDiamondDense-v0

In this environment the agent is required to obtain a diamond. The agent begins in a random starting location on a random survival map without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected summary of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode the agent is rewarded **every** time it obtains an item in the requisite item hierarchy to obtaining a diamond. The rewards for each item are given here:

```
<Item reward="1" type="log" />
<Item reward="2" type="planks" />
<Item reward="4" type="stick" />
<Item reward="4" type="crafting_table" />
<Item reward="8" type="wooden_pickaxe" />
<Item reward="16" type="cobblestone" />
<Item reward="32" type="furnace" />
<Item reward="32" type="stone_pickaxe" />
<Item reward="64" type="iron_ore" />
<Item reward="128" type="iron_ingot" />
<Item reward="256" type="iron_pickaxe" />
<Item reward="1024" type="diamond" />
```

Observation Space

```
Dict({
    "equipped_items.mainhand.damage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.maxDamage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.type": "Enum(air,iron_axe,iron_pickaxe,none,other,stone_axe,
↪stone_pickaxe,wooden_axe,wooden_pickaxe)",
    "inventory": {
            "coal": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "crafting_table": "Box(low=0, high=2304, shape=())",
            "dirt": "Box(low=0, high=2304, shape=())",
            "furnace": "Box(low=0, high=2304, shape=())",
            "iron_axe": "Box(low=0, high=2304, shape=())",
            "iron_ingot": "Box(low=0, high=2304, shape=())",
            "iron_ore": "Box(low=0, high=2304, shape=())",
            "iron_pickaxe": "Box(low=0, high=2304, shape=())",
            "log": "Box(low=0, high=2304, shape=())",
            "planks": "Box(low=0, high=2304, shape=())",
            "stick": "Box(low=0, high=2304, shape=())",
            "stone": "Box(low=0, high=2304, shape=())",
            "stone_axe": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "torch": "Box(low=0, high=2304, shape=())",
            "wooden_axe": "Box(low=0, high=2304, shape=())",
            "wooden_pickaxe": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "craft": "Enum(crafting_table,none,planks,stick,torch)",
    "equip": "Enum(air,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,wooden_axe,
↪wooden_pickaxe)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "nearbyCraft": "Enum(furnace,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,
↪wooden_axe,wooden_pickaxe)",
    "nearbySmelt": "Enum(coal,iron_ingot,none)",
    "place": "Enum(cobblestone,crafting_table,dirt,furnace,none,stone,torch)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainDiamondDense-v0") # A MineRLObtainDiamondDense-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainDiamondDense-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.8 MineRLObtainIronPickaxe-v0

In this environment the agent is required to obtain an iron pickaxe. The agent begins in a random starting location, on a random survival map, without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected view of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode **the agent is rewarded only once per item the first time it obtains that item in the requisite item hierarchy for obtaining an iron pickaxe.** The reward for each item is given here:

```
<Item amount="1" reward="1" type="log" />
<Item amount="1" reward="2" type="planks" />
<Item amount="1" reward="4" type="stick" />
<Item amount="1" reward="4" type="crafting_table" />
<Item amount="1" reward="8" type="wooden_pickaxe" />
<Item amount="1" reward="16" type="cobblestone" />
<Item amount="1" reward="32" type="furnace" />
<Item amount="1" reward="32" type="stone_pickaxe" />
<Item amount="1" reward="64" type="iron_ore" />
<Item amount="1" reward="128" type="iron_ingot" />
<Item amount="1" reward="256" type="iron_pickaxe" />
```

Observation Space

```
Dict({
    "equipped_items.mainhand.damage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.maxDamage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.type": "Enum(air,iron_axe,iron_pickaxe,none,other,stone_axe,
↪stone_pickaxe,wooden_axe,wooden_pickaxe)",
    "inventory": {
            "coal": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "crafting_table": "Box(low=0, high=2304, shape=())",
            "dirt": "Box(low=0, high=2304, shape=())",
            "furnace": "Box(low=0, high=2304, shape=())",
            "iron_axe": "Box(low=0, high=2304, shape=())",
            "iron_ingot": "Box(low=0, high=2304, shape=())",
            "iron_ore": "Box(low=0, high=2304, shape=())",
            "iron_pickaxe": "Box(low=0, high=2304, shape=())",
            "log": "Box(low=0, high=2304, shape=())",
            "planks": "Box(low=0, high=2304, shape=())",
            "stick": "Box(low=0, high=2304, shape=())",
            "stone": "Box(low=0, high=2304, shape=())",
            "stone_axe": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "torch": "Box(low=0, high=2304, shape=())",
            "wooden_axe": "Box(low=0, high=2304, shape=())",
            "wooden_pickaxe": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "craft": "Enum(crafting_table,none,planks,stick,torch)",
    "equip": "Enum(air,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,wooden_axe,
↪wooden_pickaxe)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "nearbyCraft": "Enum(furnace,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,
↪wooden_axe,wooden_pickaxe)",
    "nearbySmelt": "Enum(coal,iron_ingot,none)",
    "place": "Enum(cobblestone,crafting_table,dirt,furnace,none,stone,torch)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainIronPickaxe-v0") # A MineRLObtainIronPickaxe-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

##########################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainIronPickaxe-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.9.9 MineRLObtainIronPickaxeDense-v0

In this environment the agent is required to obtain an iron pickaxe. The agent begins in a random starting location, on a random survival map, without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected view of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode **the agent is rewarded only once per item the first time it obtains that item in the requisite item hierarchy for obtaining an iron pickaxe.** The reward for each item is given here:

```
<Item amount="1" reward="1" type="log" />
<Item amount="1" reward="2" type="planks" />
<Item amount="1" reward="4" type="stick" />
<Item amount="1" reward="4" type="crafting_table" />
<Item amount="1" reward="8" type="wooden_pickaxe" />
<Item amount="1" reward="16" type="cobblestone" />
<Item amount="1" reward="32" type="furnace" />
<Item amount="1" reward="32" type="stone_pickaxe" />
<Item amount="1" reward="64" type="iron_ore" />
<Item amount="1" reward="128" type="iron_ingot" />
<Item amount="1" reward="256" type="iron_pickaxe" />
```

Observation Space

```
Dict({
    "equipped_items.mainhand.damage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.maxDamage": "Box(low=-1, high=1562, shape=())",
    "equipped_items.mainhand.type": "Enum(air,iron_axe,iron_pickaxe,none,other,stone_axe,
↪stone_pickaxe,wooden_axe,wooden_pickaxe)",
    "inventory": {
            "coal": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "crafting_table": "Box(low=0, high=2304, shape=())",
            "dirt": "Box(low=0, high=2304, shape=())",
            "furnace": "Box(low=0, high=2304, shape=())",
            "iron_axe": "Box(low=0, high=2304, shape=())",
            "iron_ingot": "Box(low=0, high=2304, shape=())",
            "iron_ore": "Box(low=0, high=2304, shape=())",
            "iron_pickaxe": "Box(low=0, high=2304, shape=())",
            "log": "Box(low=0, high=2304, shape=())",
            "planks": "Box(low=0, high=2304, shape=())",
            "stick": "Box(low=0, high=2304, shape=())",
            "stone": "Box(low=0, high=2304, shape=())",
            "stone_axe": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "torch": "Box(low=0, high=2304, shape=())",
            "wooden_axe": "Box(low=0, high=2304, shape=())",
            "wooden_pickaxe": "Box(low=0, high=2304, shape=())"
    },
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))"
})
```

Action Space

```
Dict({
    "attack": "Discrete(2)",
    "back": "Discrete(2)",
    "camera": "Box(low=-180.0, high=180.0, shape=(2,))",
    "craft": "Enum(crafting_table,none,planks,stick,torch)",
    "equip": "Enum(air,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,wooden_axe,
↪wooden_pickaxe)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "nearbyCraft": "Enum(furnace,iron_axe,iron_pickaxe,none,stone_axe,stone_pickaxe,
↪wooden_axe,wooden_pickaxe)",
    "nearbySmelt": "Enum(coal,iron_ingot,none)",
    "place": "Enum(cobblestone,crafting_table,dirt,furnace,none,stone,torch)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainIronPickaxeDense-v0") # A MineRLObtainIronPickaxeDense-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

##########################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainIronPickaxeDense-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10 Competition Environments

1.10.1 MineRLTreechopVectorObf-v0

In treechop, the agent must collect 64 *minercaft:log*. This replicates a common scenario in Minecraft, as logs are necessary to craft a large amount of items in the game, and are a key resource in Minecraft.

The agent begins in a forest biome (near many trees) with an iron axe for cutting trees. The agent is given +1 reward for obtaining each unit of wood, and the episode terminates once the agent obtains 64 units.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLTreechopVectorObf-v0") # A MineRLTreechopVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLTreechopVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.2 MineRLNavigateVectorObf-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is sparse.**

In this environment, the agent spawns on a random survival map.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateVectorObf-v0") # A MineRLNavigateVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

########################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.3 MineRLNavigateExtremeVectorObf-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is sparse.**

In this environment, the agent spawns in an extreme hills biome.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateExtremeVectorObf-v0") # A MineRLNavigateExtremeVectorObf-
↪v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateExtremeVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.4 MineRLNavigateDenseVectorObf-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.**

In this environment, the agent spawns on a random survival map.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateDenseVectorObf-v0") # A MineRLNavigateDenseVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
```

(continues on next page)

(continued from previous page)

```
data = minerl.data.make("MineRLNavigateDenseVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.5 MineRLNavigateExtremeDenseVectorObf-v0

In this task, the agent must move to a goal location denoted by a diamond block. This represents a basic primitive used in many tasks throughout Minecraft. In addition to standard observations, the agent has access to a "compass" observation, which points near the goal location, 64 meters from the start location. The goal has a small random horizontal offset from the compass location and may be slightly below surface level. On the goal location is a unique block, so the agent must find the final goal by searching based on local visual features.

The agent is given a sparse reward (+100 upon reaching the goal, at which point the episode terminates). **This variant of the environment is dense reward-shaped where the agent is given a reward every tick for how much closer (or negative reward for farther) the agent gets to the target.**

In this environment, the agent spawns in an extreme hills biome.

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLNavigateExtremeDenseVectorObf-v0") # A␣
↪MineRLNavigateExtremeDenseVectorObf-v0 env

obs = env.reset()
```

(continues on next page)

(continued from previous page)

```
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLNavigateExtremeDenseVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.6 MineRLObtainDiamondVectorObf-v0

In this environment the agent is required to obtain a diamond. The agent begins in a random starting location on a random survival map without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected summary of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode the agent is rewarded **every** time it obtains an item in the requisite item hierarchy to obtaining a diamond. The rewards for each item are given here:

```
<Item reward="1" type="log" />
<Item reward="2" type="planks" />
<Item reward="4" type="stick" />
<Item reward="4" type="crafting_table" />
<Item reward="8" type="wooden_pickaxe" />
<Item reward="16" type="cobblestone" />
<Item reward="32" type="furnace" />
<Item reward="32" type="stone_pickaxe" />
<Item reward="64" type="iron_ore" />
<Item reward="128" type="iron_ingot" />
<Item reward="256" type="iron_pickaxe" />
<Item reward="1024" type="diamond" />
```

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainDiamondVectorObf-v0") # A MineRLObtainDiamondVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

########################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainDiamondVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.7 MineRLObtainDiamondDenseVectorObf-v0

In this environment the agent is required to obtain a diamond. The agent begins in a random starting location on a random survival map without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected summary of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode the agent is rewarded **every** time it obtains an item in the requisite item hierarchy to obtaining a diamond. The rewards for each item are given here:

```
<Item reward="1" type="log" />
<Item reward="2" type="planks" />
<Item reward="4" type="stick" />
<Item reward="4" type="crafting_table" />
<Item reward="8" type="wooden_pickaxe" />
<Item reward="16" type="cobblestone" />
<Item reward="32" type="furnace" />
<Item reward="32" type="stone_pickaxe" />
<Item reward="64" type="iron_ore" />
<Item reward="128" type="iron_ingot" />
<Item reward="256" type="iron_pickaxe" />
<Item reward="1024" type="diamond" />
```

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainDiamondDenseVectorObf-v0") # A␣
↪MineRLObtainDiamondDenseVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainDiamondDenseVectorObf-v0")
```

(continues on next page)

(continued from previous page)

```
# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.8 MineRLObtainIronPickaxeVectorObf-v0

In this environment the agent is required to obtain an iron pickaxe. The agent begins in a random starting location, on a random survival map, without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected view of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode **the agent is rewarded only once per item the first time it obtains that item in the requisite item hierarchy for obtaining an iron pickaxe.** The reward for each item is given here:

```
<Item amount="1" reward="1" type="log" />
<Item amount="1" reward="2" type="planks" />
<Item amount="1" reward="4" type="stick" />
<Item amount="1" reward="4" type="crafting_table" />
<Item amount="1" reward="8" type="wooden_pickaxe" />
<Item amount="1" reward="16" type="cobblestone" />
<Item amount="1" reward="32" type="furnace" />
<Item amount="1" reward="32" type="stone_pickaxe" />
<Item amount="1" reward="64" type="iron_ore" />
<Item amount="1" reward="128" type="iron_ingot" />
<Item amount="1" reward="256" type="iron_pickaxe" />
```

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainIronPickaxeVectorObf-v0") # A␣
↪MineRLObtainIronPickaxeVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#####################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainIronPickaxeVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.10.9 MineRLObtainIronPickaxeDenseVectorObf-v0

In this environment the agent is required to obtain an iron pickaxe. The agent begins in a random starting location, on a random survival map, without any items, matching the normal starting conditions for human players in Minecraft. The agent is given access to a selected view of its inventory and GUI free crafting, smelting, and inventory management actions.

During an episode **the agent is rewarded only once per item the first time it obtains that item in the requisite item hierarchy for obtaining an iron pickaxe.** The reward for each item is given here:

```
<Item amount="1" reward="1" type="log" />
<Item amount="1" reward="2" type="planks" />
<Item amount="1" reward="4" type="stick" />
<Item amount="1" reward="4" type="crafting_table" />
<Item amount="1" reward="8" type="wooden_pickaxe" />
<Item amount="1" reward="16" type="cobblestone" />
<Item amount="1" reward="32" type="furnace" />
<Item amount="1" reward="32" type="stone_pickaxe" />
<Item amount="1" reward="64" type="iron_ore" />
<Item amount="1" reward="128" type="iron_ingot" />
<Item amount="1" reward="256" type="iron_pickaxe" />
```

Observation Space

```
Dict({
    "pov": "Box(low=0, high=255, shape=(64, 64, 3))",
    "vector": "Box(low=-1.2000000476837158, high=1.2000000476837158, shape=(64,))"
})
```

Action Space

```
Dict({
    "vector": "Box(low=-1.0499999523162842, high=1.0499999523162842, shape=(64,))"
})
```

Usage

```
import gym
import minerl

# Run a random agent through the environment
env = gym.make("MineRLObtainIronPickaxeDenseVectorObf-v0") # A␣
↪MineRLObtainIronPickaxeDenseVectorObf-v0 env

obs = env.reset()
done = False

while not done:
    # Take a no-op through the environment.
    obs, rew, done, _ = env.step(env.action_space.noop())
    # Do something

#######################################

# Sample some data from the dataset!
data = minerl.data.make("MineRLObtainIronPickaxeDenseVectorObf-v0")

# Iterate through a single epoch using sequences of at most 32 steps
for obs, rew, done, act in data.seq_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11 Windows FAQ

This note serves as a collection of fixes for errors which may occur on the Windows platform.

1.11.1 The `freeze_support` error (multiprocessing)

```
RuntimeError:
       An attempt has been made to start a new process before the
       current process has finished its bootstrapping phase.

   This probably means that you are not using fork to start your
   child processes and you have forgotten to use the proper idiom
   in the main module:

       if __name__ == '__main__':
           freeze_support()
           ...

   The "freeze_support()" line can be omitted if the program
   is not going to be frozen to produce an executable.
```

The implementation of `multiprocessing` is different on Windows, which uses `spawn` instead of `fork`. So we have to wrap the code with an if-clause to protect the code from executing multiple times. Refactor your code into the following structure.

```
import minerl
import gym

def main()
    # do your main minerl code
    env = gym.make('MineRLTreechop-v0')

if __name__ == '__main__':
    main()
```

1.12 `minerl.env`

The `minerl.env` package provides an optimized python software layer over *MineRLEnv*, a fork of the popular Minecraft simulator Malmo which enables **synchronous**, **stable**, and **fast** samples from the Minecraft environment.

1.12.1 MineRLEnv

class minerl.env.core.**MineRLEnv**(*xml*, *observation_space*, *action_space*, *env_spec*, *port=None*, *noop_action=None*, *docstr=None*)

Bases: Env

The MineRLEnv class.

Example

To actually create a MineRLEnv. Use any one of the package MineRL environments (Todo: Link.) literal blocks:

```
import minerl
import gym

env = gym.make('MineRLTreechop-v0') # Makes a minerl environment.

# Use env like any other OpenAI gym environment.
# ...
```

Parameters

- **xml** (*str*) – The path to the MissionXML file for this environment.
- **observation_space** (*gym.Space*) – The observation for the environment.
- **action_space** (*gym.Space*) – The action space for the environment.
- **port** (*int, optional*) – The port of an exisitng Malmo environment. Defaults to None.
- **noop_action** (*Any, optional*) – The no-op action for the environment. This must be in the action_space. Defaults to None.

STEP_OPTIONS = 0

close()

gym api close

init()

Initializes the MineRL Environment.

Note: This is called automatically when the environment is made.

Parameters

- **observation_space** (*gym.Space*) – The observation for the environment.
- **action_space** (*gym.Space*) – The action space for the environment.
- **port** (*int, optional*) – The port of an exisitng Malmo environment. Defaults to None.

Raises

- **EnvException** – If the Mission XML is malformed this is thrown.
- **ValueError** – The space specified for this environment does not have a default action.
- **NotImplementedError** – When multiagent environments are attempted to be used.

is_closed()

make_interactive(*port*, *max_players=10*, *realtime=True*)

Enables human interaction with the environment.

To interact with the environment add *make_interactive* to your agent's evaluation code and then run the *minerl.interactor*.

For example:

```
env = gym.make('MineRL...')

# set the environment to allow interactive connections on port 6666
# and slow the tick speed to 6666.
env.make_interactive(port=6666, realtime=True)

# reset the env
env.reset()

# interact as normal.
...
```

Then while the agent is running, you can start the interactor with the following command.

```
python3 -m minerl.interactor 6666 # replace with the port above.
```

The interactor will disconnect when the mission resets, but you can connect again with the same command. If an interactor is already started, it won't need to be relaunched when running the commnad a second time.

Parameters

- **port** (*int*) – The interaction port
- **realtime** (*bool, optional*) – If we should slow ticking to real time.. Defaults to True.
- **max_players** (*int*) – The maximum number of players

metadata = {'render.modes': ['rgb_array', 'human']}

noop_action()

Gets the no-op action for the environment.

In addition one can get the no-op/default action directly from the action space.

env.action_space.noop()

Returns
A no-op action in the env's space.

Return type
Any

reinit()

Use carefully to reset the episode count to 0.

render(*mode='human'*)

Renders the environment.

The set of supported modes varies per environment. (And some environments do not support rendering at all.) By convention, if mode is:

- human: render to the current display or terminal and return nothing. Usually for human consumption.
- rgb_array: Return an numpy.ndarray with shape (x, y, 3), representing RGB values for an x-by-y pixel image, suitable for turning into a video.
- ansi: Return a string (str) or StringIO.StringIO containing a terminal-style text representation. The text can include newlines and ANSI escape sequences (e.g. for colors).

Note:

Make sure that your class's metadata 'render.modes' key includes
the list of supported modes. It's recommended to call super() in implementations to use the functionality of this method.

Parameters
mode (*str*) – the mode to render with

Example:

class MyEnv(Env):
metadata = {'render.modes': ['human', 'rgb_array']}

def render(self, mode='human'):

if mode == 'rgb_array':
return np.array(...) # return RGB frame suitable for video

elif mode == 'human':
... # pop up a window and render

else:
super(MyEnv, self).render(mode=mode) # just raise an exception

reset()

Resets the environment to an initial state and returns an initial observation.

Note that this function should not reset the environment's random number generator(s); random variables in the environment's state should be sampled independently between multiple calls to *reset()*. In other words, each call of *reset()* should yield an environment suitable for a new episode, independent of previous episodes.

Returns
the initial observation.

Return type
observation (object)

seed(*seed=42*, *seed_spaces=True*)

Seeds the environment!

This also seeds the aciton_space and observation_space sampling.

Note: THIS MUST BE CALLED BEFORE `env.reset()`

Parameters

- **seed** (*long, optional*) – Defaults to 42.
- **seed_spaces** (*bool, option*) – If the observation space and action space shoud be seeded. Defaults to True.

status()

Get status from server. head - Ping the the head node if True.

step(*action*)

Run one timestep of the environment's dynamics. When end of episode is reached, you are responsible for calling *reset()* to reset this environment's state.

Accepts an action and returns a tuple (observation, reward, done, info).

Parameters
action (*object*) – an action provided by the agent

Returns
agent's observation of the current environment reward (float) : amount of reward returned after previous action done (bool): whether the episode has ended, in which case further step() calls will return undefined results info (dict): contains auxiliary diagnostic information (helpful for debugging, and sometimes learning)

Return type
observation (object)

1.12.2 InstanceManager

class minerl.env.malmo.**CustomAsyncRemoteMethod**(*proxy*, *name*, *max_retries*)

Bases: _AsyncRemoteMethod

class minerl.env.malmo.**InstanceManager**

Bases: object

The Minecraft instance manager library. The instance manager can be used to allocate and safely terminate existing Malmo instances for training agents.

Note: This object never needs to be explicitly invoked by the user of the MineRL library as the creation of one of the several MineRL environments will automatically query the InstanceManager to create a new instance.

Note: In future versions of MineRL the instance manager will become its own daemon process which provides instance allocation capability using remote procedure calls.

DEFAULT_IP = 'localhost'

class Instance(*port=None*, *existing=False*, *status_dir=None*, *seed=None*, *instance_id=None*)

Bases: object

A subprocess wrapper which maintains a reference to a minecraft subprocess and also allows for stable closing and launching of such subprocesses across different platforms.

The Minecraft instance class works by launching two subprocesses: the Malmo subprocess, and a watcher subprocess with access to the process IDs of both the parent process and the Malmo subprocess. If the parent process dies, it will kill the subprocess, and then itself.

This scheme has a single failure point of the process dying before the watcher process is launched.

MAX_PIPE_LENGTH = 500

close()

Closes the object.

get_output()

property host

kill()

Kills the process (if it has been launched.)

launch(*daemonize=False*)

property port

release_lock()

property status_dir

KEEP_ALIVE_PYRO_FREQUENCY = 5

MAXINSTANCES = None

MINECRAFT_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.3-readthedocs/lib/python3.7/site-packages/minerl/env/Malmo/Minecraft'

REMOTE = False

SCHEMAS_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.3-readthedocs/lib/python3.7/site-packages/minerl/env/Malmo/Schemas'

STATUS_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.3-readthedocs/lib/python3.7/site-packages/sphinx/performance'

X11_DIR = '/tmp/.X11-unix'

classmethod add_existing_instance(*port*)

classmethod add_keep_alive(*_pid*, *_callback*)

classmethod allocate_pool(*num*)

classmethod configure_malmo_base_port(*malmo_base_port*)

Configure the lowest or base port for Malmo

classmethod get_instance(*pid*, *instance_id=None*)

Gets an instance from the instance manager. This method is a context manager and therefore when the context is entered the method yields a InstanceManager.Instance object which contains the allocated port and host for the given instance that was created.

Yields

The allocated InstanceManager.Instance object.

Raises

- **RuntimeError** – No available instances or the maximum number of allocated instances reached.
- **RuntimeError** – No available instances and automatic allocation of instances is off.

headless = False

classmethod is_remote()

managed = True

ninstances = 0

classmethod shutdown()

class minerl.env.malmo.SeedType(*value*)

Bases: IntEnum

The seed type for an instance manager.

Values:

0 - NONE: No seeding whatsoever. 1 - CONSTANT: All envrionments have the same seed (the one specified

to the instance manager) (or alist of seeds , separated)

2 - GENERATED: All environments have different seeds generated from a single random generator with the seed specified to the InstanceManager.

3 - SPECIFIED: Each instance is given a list of seeds. Specify this like 1,2,3,4;848,432,643;888,888,888 Each instance's seed list is separated by ; and each seed is separated by ,

CONSTANT = 1

GENERATED = 2

NONE = 0

SPECIFIED = 3

classmethod get_index(*type*)

minerl.env.malmo.**launch_instance_manager**()

Defines the entry point for the remote procedure call server.

minerl.env.malmo.**launch_queue_logger_thread**(*output_producer*, *should_end*)

1.13 minerl.data

The `minerl.data` package provides a unified interface for sampling data from the *MineRL-v0 Dataset*. Data is accessed by making a dataset from one of the minerl environments and iterating over it using one of the iterators provided by the `minerl.data.DataPipeline`

The following is a description of the various methods included within the package as well as some basic usage examples. To see more detailed descriptions and tutorials on how to use the data API, please take a look at our numerous getting started manuals.

1.13.1 MineRLv0

class minerl.data.**DataPipeline**(*data_directory: <module 'posixpath' from '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.3-readthedocs/lib/python3.7/posixpath.py'>*, *environment: str*, *num_workers: int*, *worker_batch_size: int*, *min_size_to_dequeue: int*, *random_seed=42*)

Bases: `object`

Creates a data pipeline object used to itterate through the MineRL-v0 dataset

property action_space

action space of current MineRL environment

Type
Returns

batch_iter(*batch_size: int*, *seq_len: int*, *num_epochs: int = - 1*, *preload_buffer_size: int = 2*, *seed: Optional[int] = None*, *include_metadata: bool = False*)

Returns batches of sequences length SEQ_LEN of the data of size BATCH_SIZE. The iterator produces batches sequentially. If an element of a batch reaches the end of its

Parameters

- **batch_size** (*int*) – The batch size.
- **seq_len** (*int*) – The size of sequences to produce.
- **num_epochs** (*int, optional*) – The number of epochs to iterate over the data. Defaults to -1.
- **preload_buffer_size** (*int, optional*) – Increase to IMPROVE PERFORMANCE. The data iterator uses a queue to prevent blocking, the queue size is the number of trajectories to load into the buffer. Adjust based on memory constraints. Defaults to 32.
- **seed** (*int, optional*) – [int]. NOT IMPLEMENTED Defaults to None.
- **include_metadata** (*bool, optional*) – Include metadata on the source trajectory. Defaults to False.

Returns
A generator that yields (sarsd) batches

Return type
Generator

get_trajectory_names()

Gets all the trajectory names

Returns
[description]

Return type
A list of experiment names

load_data(*stream_name: str*, *skip_interval=0*, *include_metadata=False*)

Iterates over an individual trajectory named stream_name.

Parameters

- **stream_name** (*str*) – The stream name desired to be iterated through.
- **skip_interval** (*int, optional*) – How many sices should be skipped.. Defaults to 0.
- **include_metadata** (*bool, optional*) – Whether or not meta data about the loaded trajectory should be included.. Defaults to False.

Yields
A tuple of (state, player_action, reward_from_action, next_state, is_next_state_terminal). These are tuples are yielded in order of the episode.

property observation_space

action space of current MineRL environment

Type
Returns

static read_frame(*cap*)

sarsd_iter(*num_epochs=- 1*, *max_sequence_len=32*, *queue_size=None*, *seed=None*, *include_metadata=False*)

Returns a generator for iterating through (state, action, reward, next_state, is_terminal) tuples in the dataset. Loads num_workers files at once as defined in minerl.data.make() and return up to max_sequence_len consecutive samples wrapped in a dict observation space

Parameters

- **num_epochs** (*int, optional*) – number of epochs to iterate over or -1 to loop forever. Defaults to -1
- **max_sequence_len** (*int, optional*) – maximum number of consecutive samples - may be less. Defaults to 32
- **seed** (*int, optional*) – seed for random directory walk - note, specifying seed as well as a finite num_epochs will cause the ordering of examples to be the same after every call to seq_iter
- **queue_size** (*int, optional*) – maximum number of elements to buffer at a time, each worker may hold an additional item while waiting to enqueue. Defaults to 16*self.number_of_workers or 2* self.number_of_workers if max_sequence_len == -1
- **include_metadata** (*bool, optional*) – adds an additional member to the tuple containing metadata about the stream the data was loaded from. Defaults to False

Yields

A tuple of (state, player_action, reward_from_action, next_state, is_next_state_terminal, (metadata)). Each element is in the format of the environment action/state/reward space and contains as many samples are requested.

seq_iter(*num_epochs=- 1*, *max_sequence_len=32*, *queue_size=None*, *seed=None*, *include_metadata=False*)

DEPRECATED METHOD FOR SAMPLING DATA FROM THE MINERL DATASET.

This function is now `DataPipeline.batch_iter()`

property spec: EnvSpec

minerl.data.**download**(*directory=None*, *resolution='low'*, *texture_pack=0*, *update_environment_variables=True*, *disable_cache=False*, *experiment=None*, *minimal=False*)

Downloads MineRLv0 to specified directory. If directory is None, attempts to download to $MINERL_DATA_ROOT. Raises ValueError if both are undefined.

Parameters

- **directory** (*os.path*) – destination root for downloading MineRLv0 datasets
- **resolution** (*str, optional*) – one of ['low', 'high'] corresponding to video resolutions of [64x64, 256,128] respectively (note: high resolution is not currently supported). Defaults to 'low'.
- **texture_pack** (*int, optional*) – 0: default Minecraft texture pack, 1: flat semi-realistic texture pack. Defaults to 0.
- **update_environment_variables** (*bool, optional*) – enables / disables exporting of MINERL_DATA_ROOT environment variable (note: for some os this is only for the current shell) Defaults to True.
- **disable_cache** (*bool, optional*) – downloads temporary files to local directory. Defaults to False
- **experiment** (*str, optional*) – specify the desired experiment to download. Will only download data for this experiment. Note there is no hash verification for individual experiments
- **minimal** (*bool, optional*) – download a minimal version of the dataset

minerl.data.**make**(*environment=None*, *data_dir=None*, *num_workers=4*, *worker_batch_size=32*, *minimum_size_to_dequeue=32*, *force_download=False*)

Initalizes the data loader with the chosen environment

Parameters

- **environment** (*string*) – desired MineRL environment
- **data_dir** (*string, optional*) – specify alternative dataset location. Defaults to None.
- **num_workers** (*int, optional*) – number of files to load at once. Defaults to 4.
- **force_download** (*bool, optional*) – specifies whether or not the data should be downloaded if missing. Defaults to False.

Returns
initalized data pipeline

Return type
DataPipeline

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

N

O

P

R

S

X