MineRL

Release 0.4.0

William H. Guss, Brandon Houghton

Jun 22, 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 MineRL dataset and Gym 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 install --cask 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: You may need the user flag: `pip3 install --upgrade minerl --user` to install properly.

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>
```

Alternatively, you can use an environment variable which automatically adds xvfb when launching MineRL:

```
MINERL_HEADLESS=1 python3 <your_script.py>
```

Note: 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.

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 (can be as long as 15-30 minutes)!

```
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).

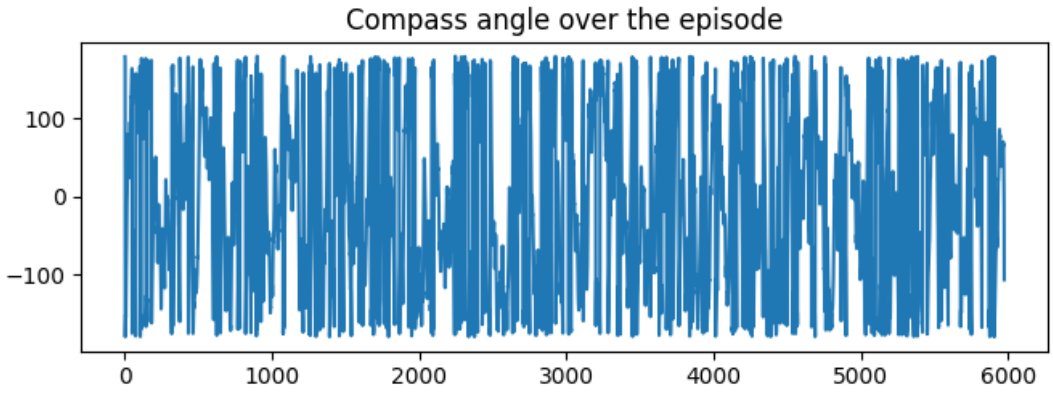
```
{
    'pov': array([[[ 63,  63,  68],
        [ 63,  63,  68],
        [ 63,  63,  68],
        ...,
        [ 92,  92, 100],
        [ 92,  92, 100],
        [ 92,  92, 100]],,

        ...,


        [[ 95, 118, 176],
        [ 95, 119, 177],
        [ 96, 119, 178],
        ...,
        [ 93, 116, 172],
        [ 93, 115, 171],
        [ 92, 115, 170]]], dtype=uint8),
    'compass': {'angle': array(-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 the agent should move sporadically until done flag is set to true. If you see a Minecraft window, it does not update while agent is playing, which is intended behaviour. 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.



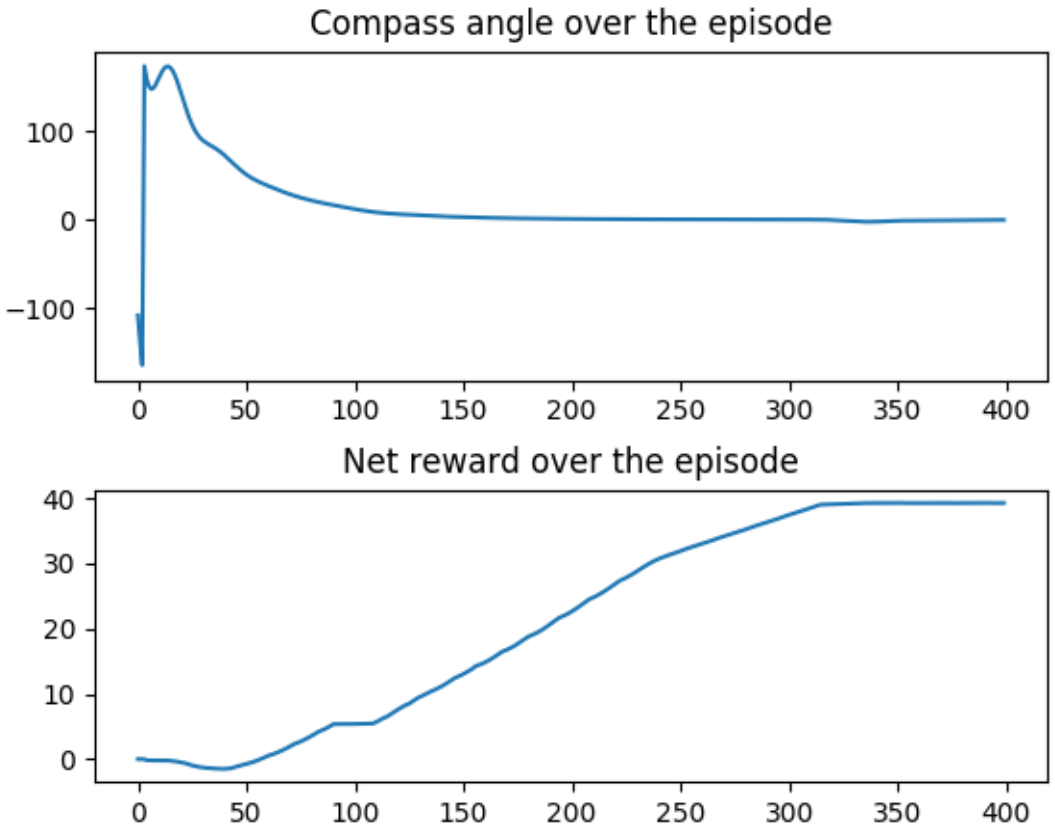
1.2.3 No-op actions and a better policy

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["compass"]["angle"]]
    action['back'] = 0
    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 Downloading and Sampling The Dataset

1.3.1 Introduction

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

To get started, let's download the minimal version of the dataset (two demonstrations from every environment). Since there are over 20 MineRL environments, this is still a sizeable download, at about 2 GB.

Then we will sample a few state-action-reward-done tuples from the `MineRLObtainDiamond-v0` dataset.

1.3.2 Setting up environment variables

The `minerl` package uses the `MINERL_DATA_ROOT` environment variable to locate the data directory. Please export `MINERL_DATA_ROOT=/your/local/path/`.

(Here are some tutorials on how to set environment variables on Linux/Mac and Windows computers.)

1.3.3 Downloading the MineRL Dataset with `minerl.data.download`

To download the minimal dataset into `MINERL_DATA_ROOT`, run the command:

```
python3 -m minerl.data.download
```

Note: The full dataset for a particular environment, or for a particular competition (Diamond or Basalt) can be downloaded using the `--environment ENV_NAME` and `--competition COMPETITION` flags.

`ENV_NAME` is any Gym environment name from the *documented environments*.

`COMPETITION` is `basalt` or `diamond`.

For more information, run `python3 -m minerl.data.download --help`.

As an example, to download the full dataset for "MineRLObtainDiamond-v0", you can run

```
python3 -m minerl.data.download --environment "MineRLObtainDiamond-v0"
```

1.3.4 Sampling the Dataset with `buffered_batch_iter`

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

There are two ways of sampling from the MineRL dataset: the deprecated but still supported *batch_iter*, and *buffered_batch_iter*. *batch_iter* is the legacy method, which we've kept in the code to avoid breaking changes, but we have recently realized that, when using *batch_size > 1*, *batch_iter* can fail to return a substantial portion of the data in the epoch.

If you are not already using `data_pipeline.batch_iter`, we recommend against it, because of these issues

The recommended way of sampling from the dataset is:

```
from minerl.data import BufferedBatchIter
data = minerl.data.make('MineRLObtainDiamond-v0')
iterator = BufferedBatchIter(data)
for current_state, action, reward, next_state, done \
    in iterator.buffered_batch_iter(batch_size=1, num_epochs=1):

        # 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))
```

1.3.5 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 feature 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 --environment '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`:

> **Warning:** BASALT: `minerl.viewer` can load BASALT competition data, but is not yet updated to display the `use` or `equip` actions yet.

The `minerl.viewer` program lets you step through individual trajectories, showing the observation seen by 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.
               v4_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'
```

(continues on next page)

(continued from previous page)

```
# 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. v4_absolute_zucch...
python3 -m minerl.viewer MineRLTreechop-v0 \
        v4_absolute_zucchini_basilisk-13_36805-50154
```

1.6 Interactive Mode `minerl.interactor`

Warning: Interactor works in MineRL versions 0.3.7 and 0.4.4 (or above). Install 0.3.7 with `pip install minerl==0.3.7`, or the newest MineRL with `pip install git+https://github.com/minerllabs/minerl.git@dev`.

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 agent(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 Creating A Custom Environment

1.7.1 Introduction

MineRL supports many ways to customize environments, including modifying the Minecraft world, adding more observation data, and changing the rewards agents receive.

MineRL provides support for these modifications using a variety of handlers.

In this tutorial, we will introduce how these handlers work by building a simple parkour environment where an agent will perform an "MLG water bucket jump" onto a block of gold.

"An MLG water is when a player is falling out of the air, or when a player jumps off of something, and they throw down water before they hit the ground to break the fall, and prevent themselves from dying by fall damage." –Sportskeeda

The agent will then mine this gold block to terminate the episode.

See the complete code here.

1.7.2 Setup

Create a Python file named `mlg_wb_specs.py`

To start building our environment, let's import the necessary modules

```
from minerl.herobraine.env_specs.simple_embodiment import SimpleEmbodimentEnvSpec
from minerl.herobraine.hero.handler import Handler
import minerl.herobraine.hero.handlers as handlers
from typing import List
```

Next, we will add the following variables:

```
MLGWB_DOC = """
In MLG Water Bucket, an agent must perform an "MLG Water Bucket" jump
"""

MLGWB_LENGTH = 8000
```

`MLGWB_LENGTH` specifies how many time steps the environment can last until termination.

1.7.3 Contruct the Environment Class

In order to create our MineRL Gym environment, we need to inherit from `SimpleEmbodimentEnvSpec`. This parent class provides default settings for the environment.

```
class MLGWB(SimpleEmbodimentEnvSpec):
    def __init__(self, *args, **kwargs):
        if 'name' not in kwargs:
            kwargs['name'] = 'MLGWB-v0'

        super().__init__(*args,
                         max_episode_steps=MLGWB_LENGTH,
                         reward_threshold=100.0,
                         **kwargs)
```

`reward_threshold` is a number specifying how much reward the agent must get for the episode to be successful.

Now we will implement a number of methods which `SimpleEmbodimentEnvSpec` requires.

1.7.4 Modify the World

Lets build a custom Minecraft world.

We'll use the `FlatWorldGenerator` handler to make a super flat world and pass it a `generatorString` value to specify how we want the world layers to be created. "1;7,2x3,2;1" represents 1 layer of grass blocks above 2 layers of dirt above 1 layer of bedrock. You can use websites like "Minecraft Tools" to easily customize superflat world layers.

We also pass a `DrawingDecorator` to "draw" blocks into the world.

```
def create_server_world_generators(self) -> List[Handler]:
    return [
        handlers.FlatWorldGenerator(generatorString="1;7,2x3,2;1"),
        # generate a 3x3 square of obsidian high in the air and a gold block
        # somewhere below it on the ground
        handlers.DrawingDecorator("""
            <DrawCuboid x1="0" y1="5" z1="-6" x2="0" y2="5" z2="-6" type="gold_block"/>
            <DrawCuboid x1="-2" y1="88" z1="-2" x2="2" y2="88" z2="2" type="obsidian"/>
        """)
    ]
```

Note: Make sure `create_server_world_generators` and the following functions are indented under the `MLGWB` class.

1.7.5 Set the Initial Agent Inventory

Lets now lets use the `SimpleInventoryAgentStart` handler to give the agent a water bucket and a diamond pickaxe.

Lets also make the agent spawn high in the air (on the obsidian platform) with the `AgentStartPlacement` handler.

```
def create_agent_start(self) -> List[Handler]:
    return [
        # make the agent start with these items
        handlers.SimpleInventoryAgentStart([
            dict(type="water_bucket", quantity=1),
            dict(type="diamond_pickaxe", quantity=1)
        ]),
        # make the agent start 90 blocks high in the air
        handlers.AgentStartPlacement(0, 90, 0, 0, 0)
    ]
```

1.7.6 Create Reward Functionality

Lets use the `RewardForTouchingBlockType` handler so that the agent receives reward for getting to a gold block.

```
def create_rewardables(self) -> List[Handler]:
    return [
        # reward the agent for touching a gold block (but only once)
        handlers.RewardForTouchingBlockType([
            {'type':'gold_block', 'behaviour':'onceOnly', 'reward':'50'},
        ]),
        # also reward on mission end
        handlers.RewardForMissionEnd(50)
    ]
```

1.7.7 Construct a Quit Handler

We want the episode to terminate when the agent obtains a gold block.

```
def create_agent_handlers(self) -> List[Handler]:
    return [
        # make the agent quit when it gets a gold block in its inventory
        handlers.AgentQuitFromPossessingItem([
            dict(type="gold_block", amount=1)
        ])
    ]
```

1.7.8 Allow the Agent to Place Water

We want the agent to be able to place the water bucket, but `SimpleEmbodimentEnvSpec` does not provide this ability by default. Note that we call `super().create_actionables()` so that we keep the actions which `SimpleEmbodimentEnvSpec` does provide by default (like movement, jumping)

```
def create_actionables(self) -> List[Handler]:
    return super().create_actionables() + [
        # allow agent to place water
        handlers.KeybasedCommandAction("use"),
        # also allow it to equip the pickaxe
        handlers.EquipAction(["diamond_pickaxe"])
    ]
```

1.7.9 Give Extra Observations

In addition to the POV image data the agent receives as an observation, lets provide it with compass and lifestats data. We override `create_observables` just like the previous step.

```
def create_observables(self) -> List[Handler]:
    return super().create_observables() + [
        # current location and lifestats are returned as additional
        # observations
        handlers.ObservationFromCurrentLocation(),
        handlers.ObservationFromLifeStats()
    ]
```

1.7.10 Set the Time

Lets set the time to morning.

```
def create_server_initial_conditions(self) -> List[Handler]:
    return [
        # Sets time to morning and stops passing of time
        handlers.TimeInitialCondition(False, 23000)
    ]
```

1.7.11 Other Functions to Implement

`SimpleEmbodimentEnvSpec` requires that we implement these methods.

```
# see API reference for use cases of these first two functions

def create_server_quit_producers(self):
    return []

def create_server_decorators(self) -> List[Handler]:
    return []

# the episode can terminate when this is True
```

(continues on next page)

(continued from previous page)

```
def determine_success_from_rewards(self, rewards: list) -> bool:
    return sum(rewards) >= self.reward_threshold

def is_from_folder(self, folder: str) -> bool:
    return folder == 'mlgwb'

def get_docstring(self):
    return MLGWB_DOC
```

Congrats! You just made your first MineRL environment. Checkout the herobraine API reference to see many other ways to modify the world and agent.

See complete environment code here.

1.7.12 Using the Environment

Now you need to solve it

Create a new Python file in the same folder.

Here is some code to get you started:

You should see a Minecaft instance open then minimize. Then, you should see a window that shows the agent's POV.

```
import gym
from mlg_wb_specs import MLGWB

# In order to use the environment as a gym you need to register it with gym
abs_MLG = MLGWB()
abs_MLG.register()
env = gym.make("MLGWB-v0")

# this line might take a couple minutes to run
obs  = env.reset()

# Renders the environment with the agent taking noops
done = False
while not done:
    env.render()
    # a dictionary of actions. Try indexing it and changing values.
    action = env.action_space.noop()
    obs, reward, done, info = env.step(action)
```

See complete solution code here (Python file) or an interactive version here (Jupyter Notebook).

1.8 Using Minecraft Commands

1.8.1 Introduction

MineRL provides support for sending Minecraft commands. In addition to opening up numerous custom environment possibilities (Minecraft commands can be used to move players, summon or destroy mobs and blocks, reset player health/food, apply potions effects, and much more), this feature can be *very* useful for speeding up training.

> **Warning:** This feature is in BETA; it comes with a number of restrictions.
>
> Only messages from the first agent are supported in the multiagent setting.
>
> You must add the `ChatAction` handler to your envspec.
>
> You can only execute one chat action per time step,

1.8.2 How Can MC Commands speed up training?

Consider an agent attempting the Navigate task. After each attempt to get to the objective the Minecraft world is reset. Resetting the world is very computationally costly and it would be better to just reset the position, health, and food of the agent.

This could be accomplished with the following Minecraft commands:

```
# teleport all agents to (x=0, z=0)
/tp @a 0 ~ 0

# reset health of all agents
/effect @a minecraft:instant_health 1 100 true

# reset food of all agents
/effect @a minecraft:saturation 1 255 true
```

1.8.3 Adding the `ChatAction` to your envspec

In order to send Minecraft commands, you need to add the `ChatAction` handler to your environment's envspec. See this tutorial on how to make custom environments and envspecs.

The `ChatAction` allows the sending of regular Minecraft chat messages as well as Minecraft commands. This can be accomplished by adding the `ChatAction` handler to your envspec:

```
def create_actionables(self) -> List[Handler]:
    return super().create_actionables() + [
        # enable chat
        handlers.ChatAction()
    ]
```

1.8.4 Abstracted Command Sending

All environments which use the `ChatAction` handler will support the `set_next_chat_message` function. This function takes a string and sends it as a chat message the next time the environment is stepped:

```
# no actions
actions = {}
env.set_next_chat_message("/gamemode @a adventure")
# sets the gamemode of all players to adventure
env.step(actions)
# the chat message is not executed again;
# it gets cleared each time step() is called
env.step(actions)
env.set_next_chat_message("/tp @r 320 54 66")
# teleports a random agent to the given coordinates
env.step(actions)
```

1.8.5 Advanced use

If for some reason you need to execute multiple commands in the *same* time step, you can either spawn in a chain of Minecraft Command Blocks or load a world from the file with a chain of command blocks. This level of complexity shouldn't be needed but could be useful if you need to execute many distinct commands and don't want to spread them over multiple time steps.

1.9 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 Diamond Competition, many environments are provided for training. However, competition agents will only be evaluated in the `MineRLObtainDiamond-v0` (Intro track) and `MineRLObtainDiamondVectorObf-v0` (Research track) environments which have **sparse** rewards. For more details see *MineRLObtainDiamond-v0* and *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()`.

1.10 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!

1.10.1 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!

1.10.2 Spaces

Enum Spaces

Some observation and action spaces are `Enum` types. Examples include the *equip observation* and the *equip action*.

Observation and action spaces that are `Enum` are encoded as strings by default (e.g. "none", "log", and "sandstone#2") when they are returned from `env.step()` and `env.reset()`, or yielded from *`minerl.data.DataPipeline.batch_iter()`*.

When building an action to pass into `env.step(act)`, the Enum component of the action dict can be encoded as either a string or an integer.

Tip: The Enum integer value that corresponds to each Enum string value can be accessed via `Enum.values_map[string_value]`. For example, to get the integer value corresponding to the equip action "dirt" in `MineRLObtainDiamond` or `MineRLBasaltBuildVillageHouse`, you can call `env.action_space.spaces["equip"].values_map["dirt"]`.

1.10.3 Observations

Visual Observations - pov, third-person

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!

Equip Observations - equipped_items

equipped_items.mainhand.type : Enum('none', 'air', ..., 'other'))

This observation is an Enum type. See *Enum Spaces* for more information.

The type of the item that the player has equipped in the mainhand slot. If the mainhand slot is empty then the value is 'air'. If the mainhand slot contains an item not inside this observation space, then the value is 'other'.

Type
np.int64

Shape
[1]

1.10.4 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 - `equip` and `use`

`equip : Enum('none', 'air', ..., 'other'))`

This is action is an Enum type. See *Enum Spaces* for more information.

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. `air` matches any empty slot in an agent's inventory and functions as an un-equip, or equip-nothing action.

Type
`np.int64`

Shape
[1]

Note: `equip 'none'` and `equip 'other'` are both no-op actions. In other words, they leave the currently equipped item unchanged. However, in the MineRL dataset, `other` takes on a special meaning. `other` is the wildcard equip action that is recorded in the dataset whenever a player equipped an item that wasn't included in this action space's Enum.

Warning: *env.step(act)* typically will not process the equip action for two ticks (i.e., you will not see the observation value equipped_items change until two more calls to *env.step*.)

This is due to a limitation with the current version of Malmo, our Minecraft backend.

`use : Discrete(1) [use]`

This action is equivalent to right-clicking in Minecraft. It causes the agent to use the item it is holding in the *mainhand slot*, or to open doors or gates when it is facing an applicable Minecraft structure.

Type
`np.int64`

Shape
[1]

1.11 MineRL Diamond Competition Intro Track Environments

1.11.1 MineRLTreechop-v0

In treechop, the agent must collect 64 *minecraft: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())
```

(continues on next page)

(continued from previous page)

```
    # 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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.2 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({
    "compass": {
            "angle": "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")

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

1.11.3 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({
    "compass": {
            "angle": "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("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
```

(continues on next page)

(continued from previous page)

```
#####################################

# 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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.4 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({
    "compass": {
            "angle": "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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.5 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({
    "compass": {
            "angle": "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("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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.6 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 only once per item the first time it obtains that 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=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "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")
```

(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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.7 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=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "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.
```

(continues on next page)

(continued from previous page)

```
    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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.8 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=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.11.9 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 every time it obtains an 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=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12 MineRL Diamond Competition Research Track Environments

1.12.1 MineRLTreechopVectorObf-v0

In treechop, the agent must collect 64 *minecraft: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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.2 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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.3 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

#######################################
```

(continues on next page)

(continued from previous page)

```
# Sample some data from the dataset!
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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.4 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()
```

(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("MineRLNavigateExtremeVectorObf-v0")

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

1.12.5 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()
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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.6 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 only once per item the first time it obtains that 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" />
```

(continues on next page)

(continued from previous page)

```
<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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.7 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
```

(continues on next page)

(continued from previous page)

```
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")

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

1.12.8 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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.12.9 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 every time it obtains an 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")
```

(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.batch_iter(num_epochs=1, batch_size=32):
    # Do something
```

1.13 MineRL BASALT Competition Environments

1.13.1 MineRLBasaltFindCave-v0

After spawning in a plains biome, explore and find a cave. When inside a cave, throw a snowball to end episode.

Observation Space

```
Dict({
    "equipped_items": {
            "mainhand": {
                    "damage": "Box(low=-1, high=1562, shape=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "type": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,
↪other,snowball,stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)"
            }
    },
    "inventory": {
            "bucket": "Box(low=0, high=2304, shape=())",
            "carrot": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "fence": "Box(low=0, high=2304, shape=())",
            "fence_gate": "Box(low=0, high=2304, shape=())",
            "snowball": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "stone_shovel": "Box(low=0, high=2304, shape=())",
            "water_bucket": "Box(low=0, high=2304, shape=())",
            "wheat": "Box(low=0, high=2304, shape=())",
            "wheat_seeds": "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,))",
    "equip": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,other,snowball,
↪stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)",
    "use": "Discrete(2)"
})
```

Starting Inventory

```
Dict({
    "snowball": 1
})
```

Usage

```
import gym
import minerl

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

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

1.13.2 MineRLBasaltMakeWaterfall-v0

After spawning in an extreme hills biome, use your waterbucket to make an beautiful waterfall. Then take an aesthetic "picture" of it by moving to a good location, positioning player's camera to have a nice view of the waterfall, and throwing a snowball. Throwing the snowball ends the episode.

Observation Space

```
Dict({
    "equipped_items": {
            "mainhand": {
                    "damage": "Box(low=-1, high=1562, shape=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "type": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,
↪other,snowball,stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)"
            }
    },
    "inventory": {
            "bucket": "Box(low=0, high=2304, shape=())",
            "carrot": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "fence": "Box(low=0, high=2304, shape=())",
            "fence_gate": "Box(low=0, high=2304, shape=())",
            "snowball": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "stone_shovel": "Box(low=0, high=2304, shape=())",
            "water_bucket": "Box(low=0, high=2304, shape=())",
            "wheat": "Box(low=0, high=2304, shape=())",
            "wheat_seeds": "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,))",
    "equip": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,other,snowball,
↪stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
```

```
    "sprint": "Discrete(2)",
    "use": "Discrete(2)"
})
```

Starting Inventory

```
Dict({
    "cobblestone": 20,
    "snowball": 1,
    "stone_pickaxe": 1,
    "stone_shovel": 1,
    "water_bucket": 1
})
```

Usage

```
import gym
import minerl

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

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

1.13.3 MineRLBasaltCreateVillageAnimalPen-v0

After spawning in a plains village, surround two or more animals of the same type in a fenced area (a pen), constructed near the house. You can't have more than one type of animal in your enclosed area. Allowed animals are chickens, sheep, cows, and pigs.

Do not harm villagers or existing village structures in the process.

Throw a snowball to end the episode.

Observation Space

```
Dict({
    "equipped_items": {
            "mainhand": {
                    "damage": "Box(low=-1, high=1562, shape=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "type": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,
↪other,snowball,stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)"
            }
    },
    "inventory": {
            "bucket": "Box(low=0, high=2304, shape=())",
            "carrot": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "fence": "Box(low=0, high=2304, shape=())",
            "fence_gate": "Box(low=0, high=2304, shape=())",
            "snowball": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "stone_shovel": "Box(low=0, high=2304, shape=())",
            "water_bucket": "Box(low=0, high=2304, shape=())",
            "wheat": "Box(low=0, high=2304, shape=())",
            "wheat_seeds": "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,))",
    "equip": "Enum(air,bucket,carrot,cobblestone,fence,fence_gate,none,other,snowball,
↪stone_pickaxe,stone_shovel,water_bucket,wheat,wheat_seeds)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)",
    "use": "Discrete(2)"
})
```

Starting Inventory

```
Dict({
    "carrot": 1,
    "fence": 64,
    "fence_gate": 64,
    "snowball": 1,
    "wheat": 1,
    "wheat_seeds": 1
})
```

Usage

```
import gym
import minerl

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

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

1.13.4 MineRLBasaltBuildVillageHouse-v0

Build a house in the style of the village without damaging the village. Give a tour of the house and then throw a snowball to end the episode.

Note: In the observation and action spaces, the following (internal Minecraft) item IDs can be interpreted as follows:

- `log#0` is oak logs.

- `log#1` is spruce logs.
- `log2#0` is acacia logs.
- `planks#0` is oak planks.
- `planks#1` is spruce planks.
- `planks#4` is acacia planks.
- `sandstone#0` is cracked sandstone.
- `sandstone#2` is smooth sandstone.

Tip: You can find detailed information on which materials are used in each biome-specific village (plains, savannah, taiga, desert) here: https://minecraft.fandom.com/wiki/Village/Structure_(old)/Blueprints#Village_generation

---

Observation Space

```
Dict({
    "equipped_items": {
            "mainhand": {
                    "damage": "Box(low=-1, high=1562, shape=())",
                    "maxDamage": "Box(low=-1, high=1562, shape=())",
                    "type": "Enum(acacia_door,acacia_fence,cactus,cobblestone,dirt,fence,
↪flower_pot,glass,ladder,log#0,log#1,log2#0,none,other,planks#0,planks#1,planks#4,red_
↪flower,sand,sandstone#0,sandstone#2,sandstone_stairs,snowball,spruce_door,spruce_fence,
↪stone_axe,stone_pickaxe,stone_stairs,torch,wooden_door,wooden_pressure_plate)"
            }
    },
    "inventory": {
            "acacia_door": "Box(low=0, high=2304, shape=())",
            "acacia_fence": "Box(low=0, high=2304, shape=())",
            "cactus": "Box(low=0, high=2304, shape=())",
            "cobblestone": "Box(low=0, high=2304, shape=())",
            "dirt": "Box(low=0, high=2304, shape=())",
            "fence": "Box(low=0, high=2304, shape=())",
            "flower_pot": "Box(low=0, high=2304, shape=())",
            "glass": "Box(low=0, high=2304, shape=())",
            "ladder": "Box(low=0, high=2304, shape=())",
            "log#0": "Box(low=0, high=2304, shape=())",
            "log#1": "Box(low=0, high=2304, shape=())",
            "log2#0": "Box(low=0, high=2304, shape=())",
            "planks#0": "Box(low=0, high=2304, shape=())",
            "planks#1": "Box(low=0, high=2304, shape=())",
            "planks#4": "Box(low=0, high=2304, shape=())",
            "red_flower": "Box(low=0, high=2304, shape=())",
            "sand": "Box(low=0, high=2304, shape=())",
            "sandstone#0": "Box(low=0, high=2304, shape=())",
            "sandstone#2": "Box(low=0, high=2304, shape=())",
            "sandstone_stairs": "Box(low=0, high=2304, shape=())",
            "snowball": "Box(low=0, high=2304, shape=())",
            "spruce_door": "Box(low=0, high=2304, shape=())",
            "spruce_fence": "Box(low=0, high=2304, shape=())",
```

```
            "stone_axe": "Box(low=0, high=2304, shape=())",
            "stone_pickaxe": "Box(low=0, high=2304, shape=())",
            "stone_stairs": "Box(low=0, high=2304, shape=())",
            "torch": "Box(low=0, high=2304, shape=())",
            "wooden_door": "Box(low=0, high=2304, shape=())",
            "wooden_pressure_plate": "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,))",
    "equip": "Enum(acacia_door,acacia_fence,cactus,cobblestone,dirt,fence,flower_pot,
↪glass,ladder,log#0,log#1,log2#0,none,other,planks#0,planks#1,planks#4,red_flower,sand,
↪sandstone#0,sandstone#2,sandstone_stairs,snowball,spruce_door,spruce_fence,stone_axe,
↪stone_pickaxe,stone_stairs,torch,wooden_door,wooden_pressure_plate)",
    "forward": "Discrete(2)",
    "jump": "Discrete(2)",
    "left": "Discrete(2)",
    "right": "Discrete(2)",
    "sneak": "Discrete(2)",
    "sprint": "Discrete(2)",
    "use": "Discrete(2)"
})
```

Starting Inventory

```
Dict({
    "acacia_door": 64,
    "acacia_fence": 64,
    "cactus": 3,
    "cobblestone": 64,
    "dirt": 64,
    "fence": 64,
    "flower_pot": 3,
    "glass": 64,
    "ladder": 64,
    "log#0": 64,
    "log#1": 64,
    "log2#0": 64,
    "planks#0": 64,
    "planks#1": 64,
    "planks#4": 64,
    "red_flower": 3,
    "sand": 64,
    "sandstone#0": 64,
    "sandstone#2": 64,
    "sandstone_stairs": 64,
    "snowball": 1,
```

```
    "spruce_door": 64,
    "spruce_fence": 64,
    "stone_axe": 1,
    "stone_pickaxe": 1,
    "stone_stairs": 64,
    "torch": 64,
    "wooden_door": 64,
    "wooden_pressure_plate": 64
})
```

Usage

```
import gym
import minerl

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

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

1.14 Performance tips

1.14.1 Slowdown in obfuscated environments

Obfuscated environments, like `MineRLObtainDiamondVectorObf-v0` make extensive use of `np.dot` function, which by default is parallelized over multiple threads. Since the vectors/matrices are small, the overhead from this outweighs benefits, and the environment appears much slower than it really is.

To speed up obfuscated environments, try setting environment variable `OMP_NUM_THREADS=1` to restrict Numpy to only use one thread.

1.14.2 Faster alternative to xvfb

Running MineRL on xvfb will slow it down by 2-3x as the rendering is done on CPU, not on the GPU. A potential alternative is to use a combination of VirtualGL and virtual displays from nvidia tools.

Note that this may interfere with your display/driver setup, and may not work on cloud VMs (`nvidia-xconfig` is not available).

Following commands outline the procedure. You may need to adapt it to suit your needs. After these commands, run `export DISPLAY=:0` and you should be ready to run MineRL. The Minecraft window will be rendered in a virtual display.

All credits go to Tencent researchers who kindly shared this piece of information!

```
sudo apt install lightdm libglu1-mesa mesa-utils xvfb xinit xserver-xorg-video-dummy

sudo nvidia-xconfig -a --allow-empty-initial-configuration --virtual=1920x1200 --busid␣
↪PCI:0:8:0
cd /tmp
wget https://nchc.dl.sourceforge.net/project/virtualgl/2.6.3/virtualgl_2.6.3_amd64.deb
sudo dpkg -i virtualgl_2.6.3_amd64.deb

sudo service lightdm stop
sudo vglserver_config
sudo service lightdm start
```

1.15 Links to papers and projects

Here you can find useful links to the presentations, code and papers of the finalists in previous MineRL competitions, as well as other publications and projects that use MineRL.

To see all papers that cite MineRL, check Google Scholar. You can also create alerts there to get notified whenever a new citation appears.

If you want to add your paper/project here, do not hesitate to create a pull request in the main repository!

1.15.1 Presentations

- MineRL 2019 - Finalists presentations at NeurIPS 2019
- MineRL 2019 - 1st place winners presentation, longer one (slides in English, talk in Russian)
- MineRL 2020 - Round 1 finalists presentations at NeurIPS 2020
- MineRL 2020 - Round 2 finalists presentations at Microsoft AI and Gaming Research Summit 2021

1.15.2 MineRL papers

- MineRL: A Large-Scale Dataset of Minecraft Demonstrations
- The MineRL 2019 Competition on Sample Efficient Reinforcement Learning using Human Priors
- Retrospective Analysis of the 2019 MineRL Competition on Sample Efficient Reinforcement Learning
- The MineRL 2020 Competition on Sample Efficient Reinforcement Learning using Human Priors
- Towards robust and domain agnostic reinforcement learning competitions: MineRL 2020

1.15.3 2019 competitor code/papers

- 1st place: paper.
- 2nd place: paper, code.
- 3rd place: paper, code.
- 4th place: code.
- 5th place: paper, code.

1.15.4 2020 competitor code/papers

- 1st place: paper.
- 2nd place: code.
- 3rd place: code.

1.15.5 Other papers that use the MineRL environment

- PiCoEDL: Discovery and Learning of Minecraft Navigation Goals from Pixels and Coordinates (CVPR Embodied AI Workshop, 2021)
- Universal Value Iteration Networks: When Spatially-Invariant Is Not Universal (AAAI, 2020)
- Multi-task curriculum learning in a complex, visual, hard-exploration domain: Minecraft
- Follow up paper from the #1 team in 2019 (obtains diamond): paper, code.
- Align-RUDDER: Learning From Few Demonstrations by Reward Redistribution (obtains diamond): paper, code.

1.15.6 Other

- Data analysis for vector obfuscation/kmeans
- Malmo and MineRL tutorial

1.16 Windows FAQ

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

1.16.1 The `The system cannot find the path specified` error (installing)

If during installation you get errors regarding missing files or unspecified paths, followed by a long path string, you might be limited by the `MAX_PATH` setting on windows. Try removing this limitation with these instructions.

1.16.2 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()
```

1.17 `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.

1.17.1 MineRLEnv

class minerl.env._singleagent.**_SingleAgentEnv**(**args*, ***kwargs*)

Bases: _MultiAgentEnv

The single agent version of the MineRLEnv.

THIS CLASS SHOULD NOT BE INSTANTIATED DIRECTLY USE ENV SPEC.

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() → Dict[str, Any]

Reset the environment.

Sets-up the Env from its specification (called everytime the env is reset.)

Returns
The first observation of the environment.

step(*single_agent_action: Dict[str, Any]*) → Tuple[Dict[str, Any], float, bool, Dict[str, Any]]

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)

1.17.2 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'

KEEP_ALIVE_PYRO_FREQUENCY = 5

MAXINSTANCES = None

MINECRAFT_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.4.4/lib/python3.7/site-packages/minerl/env/../Malmo/Minecraft'

REMOTE = False

SCHEMAS_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.4.4/lib/python3.7/site-packages/minerl/env/../Malmo/Schemas'

STATUS_DIR = '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.4.4/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 set_valid_jdwp_port_for_instance(*instance*) → None

Find a valid port for JDWP (Java Debug Wire Protocol), so that the instance can be debugged with an attached debugger. The port is set in the instance, so that other instances can check whether the port is reserved. :param instance: Instance to find and port for, and where we will set the jdwp port.

classmethod shutdown()

class minerl.env.malmo.**MinecraftInstance**(*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

property actor_name

property client_socket

client_socket_close()

client_socket_recv_message()

client_socket_send_message(*msg*)

client_socket_shutdown(*param*)

close()

Closes the object.

create_multiagent_instance_socket(*socktime*)

get_output()

property had_to_clean

has_client_socket()

property host

property jdwp_port

JDWP (Java Debug Wire Protocol) port, if any, so the instance can be debugged with an attached debugger.

kill()

Kills the process (if it has been launched.)

launch(*daemonize=False*, *replaceable=True*)

property port

release_lock()

property status_dir

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*)

1.18 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.

1.18.1 MineRLv0

class minerl.data.**DataPipeline**(*data_directory: <module 'posixpath' from '/home/docs/checkouts/readthedocs.org/user_builds/minerl/envs/v0.4.4/lib/python3.7/posixpath*
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*)

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 episode, it will be appended with a new episode.

If you wish to obtain metadata of the episodes, consider using *load_data* instead.

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.

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*, *include_monitor_data=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.
- **include_monitor_data** (*bool, optional*) – Whether to include all of the monitor data from the environment. 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`

observation 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: Optional[str] = None*, *environment: Optional[str] = None*, *competition: Optional[str] = None*, *resolution: str = 'low'*, *texture_pack: int = 0*, *update_environment_variables: bool = True*, *disable_cache: bool = False*) → None

Low-level interface for downloading MineRL dataset.

Using the *python -m minerl.data.download* CLI script is preferred because it performs more input validation and hides internal-use arguments.

Run this command with *environment=None* and *competition=None* to download a minimal dataset with 2 demonstrations from each environment. Provide the *environment* or *competition* arguments to download a full dataset for a particular environment or competition.

Parameters

- **directory** – Destination folder for downloading MineRL datasets. If None, then use the *MINERL_DATA_ROOT* environment variable, or error if this environment variable is not set.
- **environment** – The name of a MineRL environment or None. If this argument is the name of a MineRL environment and *competition* is None, then this function downloads the full dataset for the specifies MineRL environment.

  If both *environment=None* and *competition=None*, then this function downloads a minimal dataset.
- **competition** – The name of a MineRL competition ("diamond" or "basalt") or None. If this argument is the name of a MineRL environment and *competition* is None, then this function downloads the full dataset for the specified MineRL competition.

  If both *environment=None* and *competition=None*, then this function downloads a minimal dataset.
- **resolution** – For internal use only. One of ['low', 'high'] corresponding to video resolutions of [64x64,1024x1024] respectively (note: high resolution is not currently supported).
- **texture_pack** – For internal use only. 0: default Minecraft texture pack, 1: flat semi-realistic texture pack.
- **update_environment_variables** – For internal use only. If True, then export of MINERL_DATA_ROOT environment variable (note: for some os this is only for the current shell).
- **disable_cache** – If False (default), then the tar download and other temporary download files are saved inside *directory*.

  If disable_cache is False on a future call to this function and temporary download files are detected, then the download is resumed from previous download progress. If disable_cache is False on a future call to this function and the completed tar file is detected, then the download is skipped entirely and we immediately extract the tar to *directory*.

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

1.19 minerl.herobraine

1.19.1 Handlers

In addition to the default environments MineRL provides, you can use a variety of custom handlers to build your own. See the Custom Environment Tutorial to understand how to use these handlers. The following is documentation on all handlers which MineRL currently supports.

1.19.2 Agent Handlers

Agent Handlers allow you to modify various properties of the agent (e.g. items in inventory, starting health, what gives the agent reward).

Agent Start Handlers

Agent start handlers define agent start conditions such as inventory items and health.

When used to create a Gym environment, they should be passed to `create_agent_start`

class minerl.herobraine.hero.handlers.agent.start.**AgentStartBreakSpeedMultiplier**(*multiplier=1.0*)

Bases: `Handler`

Sets the break speed multiplier (how fast the agent can break blocks)

See here for more information: https://minecraft.fandom.com/el/wiki/Breaking

Example usage:

```
AgentStartBreakSpeedMultiplier(2.0)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**AgentStartNear**(*anchor_name='MineRLAgent0'*, *min_distance=2*, *max_distance=10*, *max_vert_distance=3*)

Bases: Handler

Starts agent near another agent

Example usage:

```
AgentStartNear("MineRLAgent0", min_distance=2, max_distance=10, max_vert_distance=3)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**AgentStartPlacement**(*x*, *y*, *z*, *yaw*, *pitch=0.0*)

Bases: Handler

Sets for the agent start location

Example usage:

```
AgentStartPlacement(x=5, y=70, z=4, yaw=0, pitch=0)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**InventoryAgentStart**(*inventory: Dict[int, Dict[str, Union[str, int]]]*)

Bases: Handler

Sets the start inventory of the agent by slot id.

Example usage:

```
InventoryAgentStart({
    0: {'type':'dirt', 'quantity':10},
    # metadata specifies the type of planks (e.g. oak, spruce)
    1: {'type':'planks', 'metadata': 1, 'quantity':5},
    5: {'type':'log', 'quantity':1},
    6: {'type':'log', 'quantity':2},
    32: {'type':'iron_ore', 'quantity':4
})
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**RandomInventoryAgentStart**(*inventory: Dict[str, Union[str, int]]*, *use_hotbar: bool = False*)

Bases: *InventoryAgentStart*

Sets the agent start inventory by randomly distributing items throughout its inventory slots. Note: This has no effect on inventory observation handlers.

Example usage:

```
RandomInventoryAgentStart(
    {'dirt': 10, 'planks': 5}
)
```

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**RandomizedStartDecorator**

Bases: Handler

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**SimpleInventoryAgentStart**(*inventory: List[Dict[str, Union[str, int]]]*)

Bases: *InventoryAgentStart*

Sets the start inventory of the agent sequentially.

Example usage:

```
SimpleInventoryAgentStart([
    {'type':'dirt', 'quantity':10},
    {'type':'planks', 'quantity':5},
    {'type':'log', 'quantity':1},
    {'type':'iron_ore', 'quantity':4}
])
```

class minerl.herobraine.hero.handlers.agent.start.**StartingFoodAgentStart**(*food: int = 20, food_saturation: Optional[float] = None*)

Bases: Handler

Sets the starting food and/or food saturation of the agent.

Example usage:

```
StartingFoodAgentStart(food=2.5, food_saturation=1)
```

:param food: The amount of food the agent starts out with :param food_saturation: Determines how fast the hunger level depletes, defaults to 5

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.start.**StartingHealthAgentStart**(*max_health: float = 20, health: Optional[float] = None*)

Bases: Handler

Sets the starting health of the agent

Example usage:

```
StartingHealthAgentStart(max_health=20, health=2.5)
```

max_health sets the maximum amount of health the agent can have health sets amount of health the agent starts with (max_health if not specified)

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Agent Quit Handlers

These handlers cause the episode to terminate based on certain agent conditions.

When used to create a Gym environment, they should be passed to `create_agent_handlers`

class minerl.herobraine.hero.handlers.agent.quit.**AgentQuitFromCraftingItem**(*items: List[Dict[str, Union[str, int]]]*)

Bases: `Handler`

Terminates episode when agent crafts one of the items in `items`

Example usage:

```
AgentQuitFromCraftingItem([
    dict(type="iron_axe", amount=1), dict(type="diamond_block", amount=5)
])
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.quit.**AgentQuitFromPossessingItem**(*items: List[Dict[str, Union[str, int]]]*)

Bases: `Handler`

Terminates episode when agent obtains one of the items in `items`

Example usage:

```
AgentQuitFromPossessingItem([
    dict(type="golden_apple", amount=3), dict(type="diamond", amount=1)
])
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.quit.**AgentQuitFromTouchingBlockType**(*blocks: List[str]*)

Bases: `Handler`

Terminates episode when agent touches one of the blocks in `blocks`

Example usage:

```
AgentQuitFromTouchingBlockType([
    "gold_block", "oak_log"
])
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Reward Handlers

These handlers modify what things the agent gets rewarded for.

When used to create a Gym environment, they should be passed to `create_rewardables`

class minerl.herobraine.hero.handlers.agent.reward.**ConstantReward**(*constant*)

Bases: *RewardHandler*

A constant reward handler

from_hero(*obs_dict*)

By default hero will include the reward in the observation. This is just a pass through for convenience. :param obs_dict: :return: The reward

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

class minerl.herobraine.hero.handlers.agent.reward.**RewardForCollectingItems**(*item_rewards: List[Dict[str, Union[str, int]]]*)

Bases: _RewardForPosessingItemBase

The standard malmo reward for collecting item.

Example usage:

```
RewardForCollectingItems([
    dict(type="log", amount=1, reward=1.0),
])
```

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

class minerl.herobraine.hero.handlers.agent.reward.**RewardForCollectingItemsOnce**(*item_rewards: List[Dict[str, Union[str, int]]]*)

Bases: _RewardForPosessingItemBase

The standard malmo reward for collecting item once.

Example usage:

```
RewardForCollectingItemsOnce([
    dict(type="log", amount=1, reward=1),
])
```

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

class minerl.herobraine.hero.handlers.agent.reward.**RewardForDistanceTraveledToCompassTarget**(*reward_per_bl int, density: str = 'PER_TICK'*)

Bases: *RewardHandler*

Creates a reward which is awarded when the player reaches a certain distance from a target.

Example usage:

```
RewardForDistanceTraveledToCompassTarget(2)
```

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

reset()

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.reward.**RewardForMissionEnd**(*reward: int*, *description: str = 'out_of_time'*)

Bases: *RewardHandler*

Creates a reward which is awarded when a mission ends.

Example usage:

```
# awards a reward of 5 when mission ends
RewardForMissionEnd(reward=5.0, description="mission termination")
```

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.reward.**RewardForTouchingBlockType**(*blocks: List[Dict[str, Union[str, int, float]]]*)

Bases: *RewardHandler*

Creates a reward which is awarded when the player touches a block.

Example usage:

```
RewardForTouchingBlockType([
    {'type':'diamond_block', 'behaviour':'onceOnly', 'reward':'10'},
])
```

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

reset()

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.reward.**RewardHandler**

Bases: TranslationHandler

Specifies a reward handler for a task. These need to be attached to tasks with reinforcement learning objectives. All rewards need inherit from this reward handler #Todo: Figure out how this interplays with Hero, as rewards are summed.

from_hero(*obs_dict*)

By default hero will include the reward in the observation. This is just a pass through for convenience. :param obs_dict: :return: The reward

Action Handlers

Action handlers define what actions agents are allowed to take.

When used to create a gym, you should override create_actionables and pass the action handlers to this function. See the Custom Environment Tutorial for more.

Camera

class minerl.herobraine.hero.handlers.agent.actions.camera.**CameraAction**

Bases: Action

Uses <delta_pitch, delta_yaw> vector in degrees to rotate the camera. pitch range [-180, 180], yaw range [-180, 180]

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Craft

class minerl.herobraine.hero.handlers.agent.actions.craft.**CraftAction**(*items: list*, *_other=typing.Union[str, NoneType]*, *_default=typing.Union[str, NoneType]*)

Bases: ItemListAction

An action handler for crafting items

Note when used alongside Craft Item Nearby, block lists must be disjoint or from_universal will fire multiple times

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.agent.actions.craft.**CraftNearbyAction**(*items: list*, *_other=typing.Union[str, NoneType]*, *_default=typing.Union[str, NoneType]*)

Bases: *CraftAction*

An action handler for crafting items when agent is in view of a crafting table

Note when used along side Craft Item, item lists must be disjoint or from_universal will fire multiple times

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Equip

class minerl.herobraine.hero.handlers.agent.actions.equip.**EquipAction**(*items: list*, *_default='none'*, *_other='other'*)

Bases: ItemWithMetadataListAction

An action handler for observing a list of equipped items

from_universal(*obs*) → str

Converts a universal representation of the handler (e.g. universal action/observation)

logger = <Logger minerl.herobraine.hero.handlers.agent.actions.equip.EquipAction (WARNING)>

reset()

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Keyboard

class minerl.herobraine.hero.handlers.agent.actions.keyboard.**KeybasedCommandAction**(*command*, **keys*)

Bases: Action

A command action which is generated from human keypresses in anvil. Examples of such actions are movement actions, etc.

This is not to be confused with keyboard actions, whereby both anvil and malmo simulate and act on direct key codes.

Combinations of KeybasedCommandActions yield actions like:

```
{
    “move” : 1,
    “jump”: 1
}
```

where move and jump are the commands, which correspond to keys like 'W', 'SPACE', etc.

This is as opposed to keyboard actions (see the following class definition in keyboard.py) which yield actions like:

```
{
    "keyboard" : {
        "W" : 1,
        "A": 1,
        "S": 0,
        "E": 1,
        ...
    }
}
```

More information can be found in the unification document (internal).

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Notice how all of the instances of keybased command actions, of which there will be typically many in an environment spec, correspond to exactly the same XML stub.

This is discussed at length in the unification proposal and is a chief example of where manifest consolidation is needed.

Place

class minerl.herobraine.hero.handlers.agent.actions.place.**PlaceBlock**(*blocks: list*, *_other=typing.Union[str, NoneType]*, *_default=typing.Union[str, NoneType]*)

Bases: `ItemListAction`

An action handler for placing a specific block

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Smelt

class minerl.herobraine.hero.handlers.agent.actions.smelt.**SmeltItemNearby**(*items: list*, *_other=typing.Union[str, NoneType]*, *_default=typing.Union[str, NoneType]*)

Bases: *CraftAction*

An action handler for crafting items when agent is in view of a crafting table

Note when used along side Craft Item, block lists must be disjoint or from_universal will fire multiple times

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Chat

class minerl.herobraine.hero.handlers.agent.actions.chat.**ChatAction**

Bases: Action

Handler which lets agents send Minecraft chat messages

Note: this may currently be limited to the first agent sending messages (check Malmo for this)

This can be used to execute MINECRAFT COMMANDS !!!

Example usage:

```
ChatAction()
```

To summon a creeper, use this action dictionary:

```
{"chat": "/summon creeper"}
```

from_universal(*x*)

Converts a universal representation of the handler (e.g. universal action/observation)

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Observation Handlers

Observation handlers define what observation data agents receive (e.g. POV image, lifestats)

When used to create a gym, you should override create_observables and pass the observation handlers to this function. See the Custom Environment Tutorial for more.

Compass

class minerl.herobraine.hero.handlers.agent.observations.compass.**CompassObservation**(*angle=True*, *distance=False*)

Bases: TranslationHandlerGroup

Defines compass observations.

Parameters

- **angle** (*bool, optional*) – Whether or not to include angle observation. Defaults to True.
- **distance** (*bool, optional*) – Whether or not ot include distance observation. Defaults to False.

Example usage:

```
# A compass observation object which gives angle and distance information
CompassObservation(True, True)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Damage Source

class minerl.herobraine.hero.handlers.agent.observations.damage_source.**ObservationFromDamageSource**

Bases: TranslationHandlerGroup

Includes the most recent damage event including the amount, type, location, and other properties.

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Equipped Item

class minerl.herobraine.hero.handlers.agent.observations.equipped_item.**EquippedItemObservation**(*items: Sequence[str*, *mainhand: bool = True*, *offhand: bool = False*, *armor: bool = False*, *_default: str = 'none'*, *_other: str = 'other'*)

Bases: TranslationHandlerGroup

Enables the observation of equipped items in the main, offhand, and armor slots of the agent.

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Inventory

class minerl.herobraine.hero.handlers.agent.observations.inventory.**FlatInventoryObservation**(*item_list: Sequence[str]*, *_other='other'*)

Bases: TranslationHandler

Handles GUI Container Observations for selected items

add_to_mission_spec(*mission_spec*)

from_hero(*obs*)

Converts the Hero observation into a one-hot of the inventory items for a given inventory container. Ignores variant / color :param obs: :return:

from_universal(*obs*)

Converts a universal representation of the handler (e.g. universal action/observation)

logger = <Logger minerl.herobraine.hero.handlers.agent.observations.inventory.FlatInventoryObservation (WARNING)>

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Lifestats

class minerl.herobraine.hero.handlers.agent.observations.lifestats.**ObservationFromLifeStats**

Bases: TranslationHandlerGroup

Groups all of the lifestats observations together to correspond to one XML element.

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Location Stats

class minerl.herobraine.hero.handlers.agent.observations.location_stats.**ObservationFromCurrentLocation**

Bases: TranslationHandlerGroup

Includes the current biome, how likely rain and snow are there, as well as the current light level, how bright the sky is, and if the player can see the sky.

Also includes x, y, z, roll, and pitch

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Base Stats

class minerl.herobraine.hero.handlers.agent.observations.mc_base_stats.**ObserveFromFullStats**(*stat_key*)

Bases: TranslationHandlerGroup

Includes the use_item statistics for every item in MC that can be used

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

POV

class minerl.herobraine.hero.handlers.agent.observations.pov.**POVObservation**(*video_resolution: Tuple[int, int]*, *include_depth: bool = False*)

Bases: KeymapTranslationHandler

Handles POV observations.

from_hero(*obs*)

Converts a "hero" representation of an instance of this handler to a member of the space.

to_string()

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

1.19.3 Server Handlers

Server Start Handlers

Server start handlers allow you to set the initial state of the World (e.g. weather, time)

When used to create a Gym environment, they should be passed to `create_server_initial_conditions`

class minerl.herobraine.hero.handlers.server.start.**SpawningInitialCondition**(*allow_spawning: bool*)

Bases: `Handler`

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.start.**TimeInitialCondition**(*allow_passage_of_time: bool*, *start_time: Optional[int] = None*)

Bases: `Handler`

Sets the initial world time as well as whether time can pass.

Example usage:

```
# Sets time to morning and stops passing of time
TimeInitialCondition(False, 23000)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.start.**WeatherInitialCondition**(*weather: str*)

Bases: Handler

Sets the initial weather condition in the world.

Example usage:

```
# Sets weather to thunder
WeatherInitialCondition("thunder")
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

Server Quit Handlers

These handlers allow episode termination based on server conditions (e.g. time passed)

When used to create a Gym environment, they should be passed to `create_server_quit_producers`

class minerl.herobraine.hero.handlers.server.quit.**ServerQuitFromTimeUp**(*time_limit_ms: int*, *description='out_of_time'*)

Bases: Handler

Forces the server to quit after a certain time_limit_ms also specifies a description parameter for the xml.

Example usage

```
ServerQuitFromTimeUp(50000)
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.quit.**ServerQuitWhenAnyAgentFinishes**

Bases: Handler

Forces the server to quit if any of the agents involved quits. Has no parameters.

Example usage:

```
ServerQuitWhenAnyAgentFinishes()
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

World Handlers

World handlers provide a number of ways to generate and modify the Minecraft world (e.g. specifying the type of world to be created, like Superflat, or drawing shapes and blocks in the world).

When used to create a Gym environment, they should be passed to `create_server_world_generators`

class minerl.herobraine.hero.handlers.server.world.**BiomeGenerator**(*biome: Union[int, str]*, *force_reset: bool = True*)

Bases: `Handler`

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.world.**DefaultWorldGenerator**(*force_reset=True*, *generator_options: str = '{}'*)

Bases: `Handler`

Generates a world using minecraft procedural generation (this is the default world type in minecraft).

Parameters

- **force_reset** (*bool, optional*) – If the world should be reset every episode.. Defaults to True.
- **generator_options** – A JSON object specifying parameters to the procedural generator.

Example usage:

```
# Generates a default world that does not reset every episode (e.g. if blocks get␣
↪broken in one episode
# they will not be replaced in the next)
DefaultWorldGenerator(False, "")
```

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.world.**DrawingDecorator**(*to_draw: str*)

Bases: Handler

Draws shapes (e.g. spheres, cuboids) in the world.

Example usage:

```
# draws an empty square of gold blocks
DrawingDecorator('
    <DrawCuboid x1="3" y1="4" z1="3" x2="3" y2="6" z2="-3" type="gold_block"/>
    <DrawCuboid x1="3" y1="4" z1="3" x2="-3" y2="6" z2="3" type="gold_block"/>
    <DrawCuboid x1="-3" y1="4" z1="-3" x2="3" y2="6" z2="-3" type="gold_block"/>
    <DrawCuboid x1="-3" y1="4" z1="-3" x2="-3" y2="6" z2="3" type="gold_block"/>
')
```

See Project Malmo for more

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.world.**FileWorldGenerator**(*filename: str*, *destroy_after_use: bool = True*)

Bases: Handler

Generates a world from a file.

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.world.**FlatWorldGenerator**(*force_reset: bool = True*, *generatorString: str = ''*)

Bases: Handler

Generates a world that is a flat landscape.

Example usage:

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

class minerl.herobraine.hero.handlers.server.world.**VillageSpawnDecorator**

Bases: Handler

to_string() → str

The unique identifier for the agent handler. This is used for constructing action/observation spaces and unioning different env specifications.

xml_template() → str

Generates an XML representation of the handler.

This XML representation is templated via Jinja2 and has access to all of the member variables of the class.

Note: This is not an abstract method so that handlers without corresponding XML's can be combined in handler groups with group based XML implementations.

CHAPTER

TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

N

O

P

R

S

T