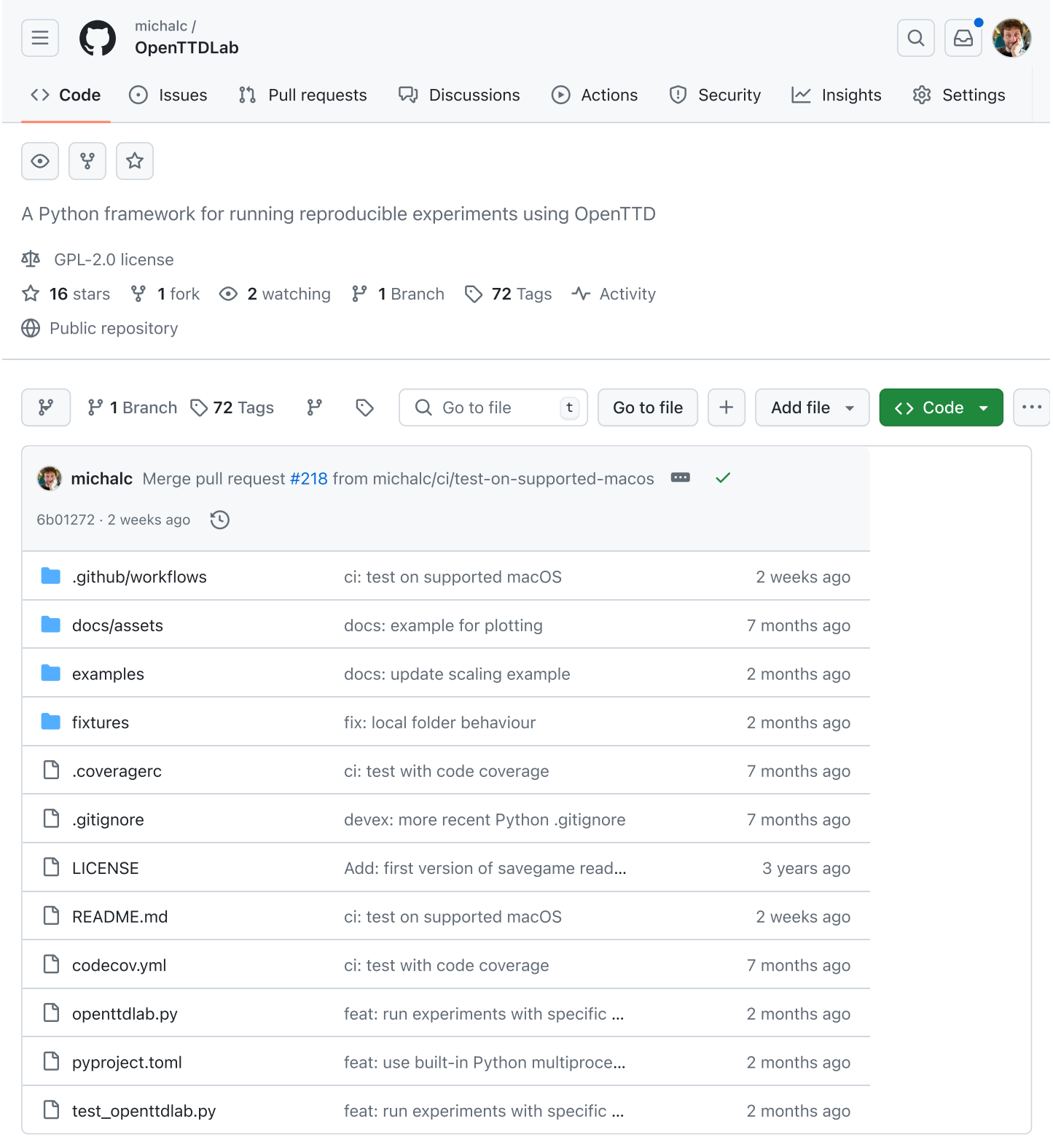
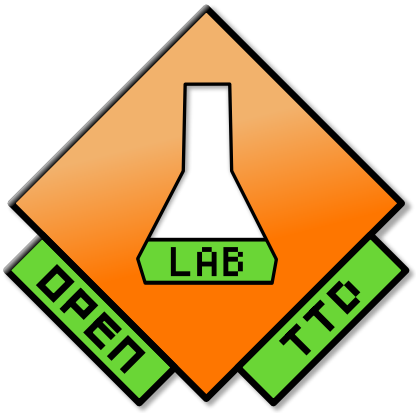

michalc /
OpenTTDLab
Code
Issues
Pull requests
Discussions
Actions
Security
Insights
Settings
A Python framework for running reproducible experiments using OpenTTD
GPL-2.0 license
16 stars
1 fork
2 watching
1 Branch
72 Tags
Activity
Public repository
1 Branch
72 Tags
Go to file
t
Go to file
Add file
Code
michalc Merge pull request #218 from michalc/ci/test-on-supported-macos
6b01272 · 2 weeks ago
.github/workflows ci: test on supported macOS 2 weeks ago
docs/assets docs: example for plotting 7 months ago
examples docs: update scaling example 2 months ago
fixtures fix: local folder behaviour 2 months ago
.coveragerc ci: test with code coverage 7 months ago
.gitignore devex: more recent Python .gitignore 7 months ago
LICENSE Add: first version of savegame read... 3 years ago
README.md ci: test on supported macOS 2 weeks ago
codecov.yml ci: test with code coverage 7 months ago
openttdlab.py feat: run experiments with specific ... 2 months ago
pyproject.toml feat: use built-in Python multiproce... 2 months ago
test_openttdlab.py feat: run experiments with specific ... 2 months ago
README
License



OpenTTDLab - *Run reproducible experiments using OpenTTD*

PyPI package v0.0.72 | Test suite passing | Code coverage 92%

OpenTTDLab is a Python framework for using OpenTTD to run reproducible experiments and extracting results from them, with as few manual steps as possible. OpenTTDLab can also be used to help run regression tests of OpenTTD AIs, parse OpenTTD savegame files, and download content from BaNaNaS.

OpenTTDLab is based on Patric Stout's OpenTTD Savegame Reader.

> **Caution**
>
> OpenTTDLab currently does not work with OpenTTD 14.0 or later. The latest version of OpenTTD known to work is 13.4.

Contents

Features

OpenTTDLab essentially turns OpenTTD into a simulator - and through AIs and AI libraries it allows you to experiment with different techniques of building supply chains and study their effects. In more detail OpenTTDLab:

- Allows you to easily run OpenTTD in a headless mode (i.e. without a graphical interface) over a variety of configurations.
- And allows you to do this from Python code - for example from a Jupyter Notebook.
- As is typical from Python code, it is cross platform - allowing to share code snippets between macOS, Windows, and Linux, even though details like how to install and start OpenTTD are different on each platform.
- Downloads (and caches) OpenTTD, OpenGFX, AIs, and AI libraries - no need to download these separately or through OpenTTD's built-in content browser.
- Transparently parallelises runs of OpenTTD, by default up to the number of CPUs. (Although with fairly poor scaling properties.)
- Results are extracted from OpenTTD savegames as plain Python dictionaries and lists - reasonably convenient for importing into tools such as pandas for analysis or visualisation.

Installation

OpenTTDLab is distributed via PyPI, and so can usually be installed using pip.

```
python -m pip install OpenTTDLab
```

When run on macOS, OpenTTDLab has a dependency that pip does not install: 7-zip. To install 7-zip, first install Homebrew, and then use Homebrew to install the p7zip package that contains 7-zip.

```
brew install p7zip
```

You do not need to separately download or install OpenTTD (or OpenGFX) in order to use OpenTTDLab. OpenTTDLab itself handles downloading them.

Running experiments

The core function of OpenTTD is the `run_experiments` function.

```
from openttdlab import run_experiments, bananas_ai

# Run experiments...
results = run_experiments(
    openttd_version='13.4',  # ... for a specific versions of OpenTTD
    opengfx_version='7.1',   # ... and a specific versions of OpenGFX
    experiments=(
       {
         # ... for random seeds
         'seed': seed,
         # ... running specific AIs. In this case a single AI, with no
         # parameters, fetching it from https://bananas.openttd.org/package/ai
         'ais': (
             bananas_ai('54524149', 'trAIns', ai_params=()),
         ),
         # ... each for a number of (in game) days
         'days': 365 * 4 + 1,
       }
       for seed in range(0, 10)
    ),
)
```

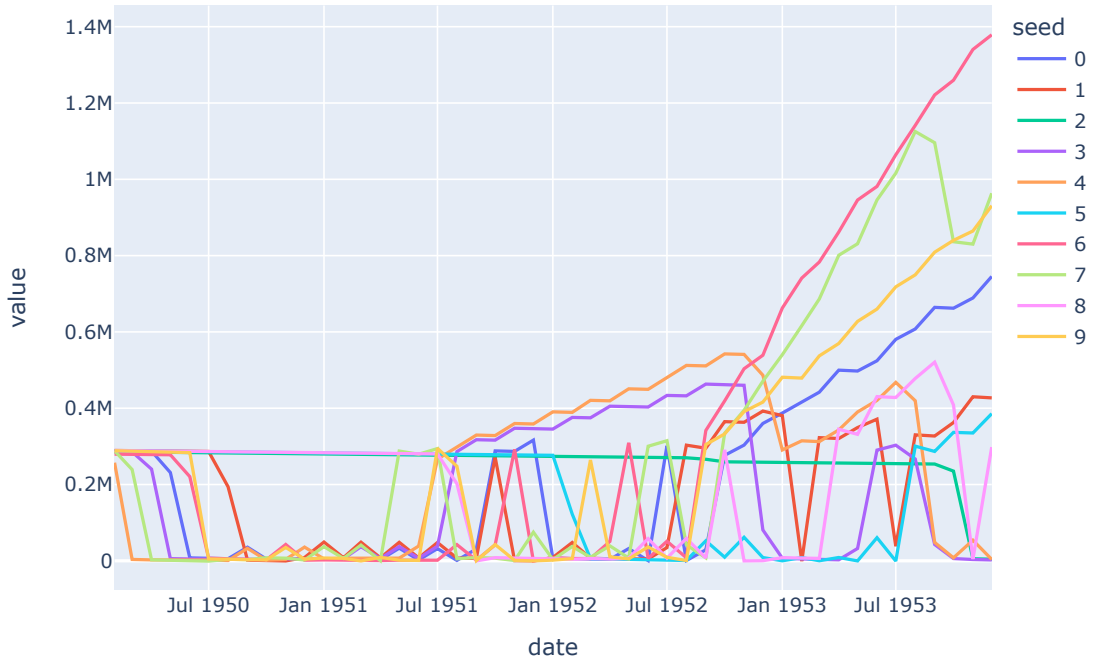
Plotting results

OpenTTD does not require any particular library for plotting results. However, pandas and Plotly Express are common options for plotting from Python. For example if you have a `results` object from `run_experiments` as in the above example, the following code

```
import pandas as pd
import plotly.express as px

df = pd.DataFrame(
    {
        'seed': row['experiment']['seed'],
        'date': row['date'],
        'money': row['chunks']['PLYR']['0']['money'],
    }
    for row in results
)
df = df.pivot(index='date', columns='seed', values='money')
fig = px.line(df)
fig.show()
```

should output a plot much like this one.



Examples

A few examples are available:

- A Jupyter notebook of the above example, briefly exploring the performance of trAIns
- A Jupyter notebook showing how the performance of OpenTTDLab scales with the number of workers
- A pytest .py file using OpenTTDLab as a test harness for an OpenTTD AI, run automatically using a GitHub action workflow

API

Core function

`run_experiments(...)`

The core function of OpenTTDLab is the `run_experiments` function, used to run an experiment and return results extracted from the savegame files that OpenTTD produces. It has the following parameters and defaults.

- `experiments=()`

  An iterable of the experiments to run. Each experiment should be a dictionary with the (string) keys:

  - `'ais'`

    The list of AIs to run in this experiment. See the Fetching AIs section for details on this parameter.

  - `'seed'`

    The integer seed of the random number generator for this experiment.

  - `'days'`

    The integer number of in-game days that this experiment will run for.

  - `openttd_config=''`

    OpenTTD config to run each experiment under. This must be in the openttd.cfg format. This is added to by OpenTTDLab before being passed to OpenTTD.

- `ais_libraries=()`

  The list of AI libraries to have available to AI code. See the Fetching AI libraries section for details on this parameter.

- `result_processor=lambda r: (r,)`

  A function that takes a single result row, which is a parsed save game file from an experiment, alongside other metadata describing the experiment, and returns it processed in some way. The function should return an iterable of zero or more rows that will appear in the the return value of `run_experiments`.

  This is typically used to reduce memory usage with high numbers of experiments where only a small amount of data is needed for analysis.

- `final_screenshot_directory=None`

  The directory to save a PNG screenshot of the entire map at the end of each run. Each is named in the format `<seed>.png`, where `<seed>` is the experiment's seed of the random number generator. If `None`, then no screenshots are saved.

  For technical reasons, a window will briefly appear while each screenshot is being saved. This can be avoided when running on Linux if `xvfb-run` is installed and available in the path.

- `max_workers=None`

  The maximum number of workers to use to run OpenTTD in parallel. If `None`, then `os.cpu_count()` defined how many workers run.

- `openttd_version=None`

The version of OpenTTD to use. If `None`, the latest version available at `openttd_base_url` is used.

> **Caution** OpenTTDLab currently does not work with OpenTTD 14.0 or later. The latest version of OpenTTD known to work is 13.4.

- `opengfx_version=None`

  The version of OpenGFX to use. If `None`, the latest version available at `opengfx_base_url` is used.

- `openttd_base_url='https://cdn.openttd.org/openttd-releases/`

  The base URL used to fetch the list of OpenTTD versions, and OpenTTD binaries.

- `opengfx_base_url='https://cdn.openttd.org/opengfx-releases/`

  The URL used to fetch the list of OpenGFX versions, and OpenGFX binaries.

- `get_http_client=lambda: httpx.Client(transport=httpx.HTTPTransport(retries=3)`

  The HTTP client used to make HTTP requests when fetching OpenTTD, OpenGFX, or AIs. Note that the `bananas_ai` function uses a raw TCP connection in addition to HTTP requests, and so not all outgoing connections use the client specified by this.

Configuring AIs

The value of the `ais` key of each dictionary in the `experiments` parameter configures which AIs will run, how their code will be located, their names, and what parameters will be passed to each of them when they start. In more detail, the `ais` parameter must be an iterable of the return value of any of the the following 4 functions.

> Important
>
> The `ai_name` argument passed to each of the following functions must exactly match the name of the corresponding AI as published. If it does not match, the AI will not be started.

> Important
>
> The return value of each of the following is opaque: it should not be used in client code, other than by passing into `run_experiments` as part of the `ais` parameter.

`bananas_ai(unique_id, ai_name, ai_params=(), md5=None)`

Defines an AI by the `unique_id` and `ai_name` of an AI published through OpenTTD's content service at https://bananas.openttd.org/package/ai. This allows you to quickly run OpenTTDLab with a published AI. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

The `unique_id` is sometimes surfaced as the "Content Id", but it should not include its `ai/` prefix.

If you pass the full MD5 hex string of a specific version of AI as `md5`, for example previously returned from the `download_from_bananas` function, the corresponding version will be used. Otherwise, the latest version will be used.

`local_folder(folder_path, ai_name, ai_params=()))`

Defines an AI by the `folder_path` to a local folder that contains the AI code of an AI with name `ai_name`. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

`local_file(path, ai_name, ai_params=())`

Defines an AI by the local path to a .tar AI file that contains the AI code. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

`remote_file(url, ai_name, ai_params=())`

Fetches the AI by the URL of a tar.gz file that contains the AI code. For example, a specific GitHub tag of a repository that contains its code. The `ai_params` parameter is an optional parameter of an iterable of `(key, value)` parameters passed to the AI on startup.

Configuring AI libraries

The `ai_libraries` parameter of `run_experiments` ensures that AI libraries are available to the AIs running. In more detail, the `ais_libraries` parameter must be an iterable, where each item the the return value of the `bananas_ai_library` function described below.

Note that for AIs specified by `bananas_ai` OpenTTDLab automatically downloads all of their AI library dependencies without them having to be specified through the `ai_libraries` parameter. This includes all transitive AI library dependencies - AI libraries needed by AI libraries needed by AIs, and so on.

Similarly for AI libraries specified by `bananas_ai_library` - OpenTTDLab automatically downloads of their AI library dependencies.

`bananas_ai_library(unique_id, ai_library_name, md5=None)`

Fetches the AI library defined by `unique_id` and `ai_name` of a library published through OpenTTD's content service at https://bananas.openttd.org/package/ai-library.

The `unique_id` is sometimes surfaced as the "Content Id", but it should not include its `ai-library/` prefix.

If you pass the full MD5 string of a specific version of AI library as `md5`, for example previously returned from the `download_from_bananas` function, this will fetch this corresponding version from BaNaNaS. Otherwise, the latest version is fetched.

Parsing savegame files

`parse_savegame(chunks: Iterable[bytes])`

Under the hood the `run_experiments` handles the generation and parsing of savegame files, but if you have your own savegame files generated separately, the `parse_savegame` function is exposed that can extract data from them.

It takes an iterable of `bytes` instances of a savegame file, and returns a nested dictionary of parsed data.

```
from openttdlab import parse_savegame

with open('my.sav') as f:
   parsed_savegame = parse_savegame(iter(lambda: f.read(65536), b''))
```

Downloading from BaNaNaS

> Important
>
> Please do not use this to try to download all content from BaNaNaS. See this discussion about writing a client for BaNaNaS for more details.

> Important
>
> Please note the license of each piece of content you download, and adhere to its rules. As examples, licenses may require you to attribute the author, they can restrict you from distributing any modifications you make, they can restrict you from using the content for commercial purposes, or they can require you to make the source available if you distribute a compiled version.

`download_from_bananas(content_id: str, md5: Optional[str]=None)`

This function is a Python BaNaNaS client for downloading the content from BaNaNaS. Given a content id, it returns an iterable of that content and all of its direct and transitive dependencies.

```
from openttdlab import download_from_bananas

with download_from_bananas('ai/41444d4c') as files:
    for content_id, filename, license, partial_or_full_md5, get_data in files:
        with get_data() as chunks:
            with open(filename, 'wb') as f:
                for chunk in chunks:
                    f.write(chunk)
```

Each `chunks` iterable are the binary chunks of the non-compressed `.tar` file of the content. Also, under the hood `download_from_bananas` transparently caches content where possible. This is the main reason for using context managers as in the above example - they allow for robust cleanup of resources and caching of data once the data has been iterated over.

If you don't pass `md5`, `download_from_bananas` will return details for the *latest* version of the content. And if the content has a known and acceptable license, `partial_or_full_md5` will contain the full MD5 for the content, which can then be subsequently passed back into `download_from_bananas` to download the same version later. If the file does not have a known license, `download_from_bananas` will contain the only the first 8 characters of the MD5. This means that `download_from_bananas` is deterministic only for content that has an acceptable license, and if you pass an MD5 previously retrieved from a call to `download_from_bananas`.

Note that the function `run_experiments` that uses `bananas_ai` or `bananas_ai_library` will handle automatically downloading from BaNaNaS, so this function is usually only useful if you would like to run experiments without using the `bananas_*` functions, or report on the filename (which includes the version of each piece of content) or the MD5 sum of the file.

Tips for repeatability, reproducibility, and replicability

OpenTTDLab should be able to help with all the 3Rs of repeatability, reproducibility and (although to a lesser degree) replicability of simulations. Definitions of the 3Rs are taken from https://www.acm.org/publications/policies/artifact-review-and-badging-current.

Repeatability

> "For computational experiments, this means that a researcher can reliably repeat her own computation."

The run_experiments function is the primary way OpenTTLab supports repeatability - with a single function call you can run a range of experiments. But to make sure you get the same results on each invocation:

- Pin to a specific OpenTTD and OpenGFX version.
- If fetching AIs and AI libraries from BaNaNaS, use the MD5 to ensure the same versions are used (at the moment the only way of discovering these is to initially separately call the download_from_bananas_function).
- If fetching AIs and AI libraries from another remote source, make sure that source is immutable.

- For all your own code store in version control, e.g. git, and make sure to note version/commit IDs used to generate results.
- Use a virtual environment to pin Python version and all dependencies (most importantly of OpenTTDLab).
- Use fixed random seeds.
- (Out of paranoia mostly, note the OS and its version, and CPU type.)

Reproducibility

> "For computational experiments, this means that an independent group can obtain the same result using the author's own artifacts."

To help others reproduce your results:

- Take all the repeatability steps above, and make sure to communicate all the noted details, e.g. what versions where pinned, along with any results.
- Make any artifacts that you have created available alongide the results / linked from the results. This can include the source of AIs you have used or created, the Python code to run OpenTTDLab, and the Python code to analyse the results.

In general, bitwise reproducibility is not expected, but with fixed random seeds it "should" be the case with OpenTTD/OpenTTDLab (depending on the analysis performed).

Replicability

> "For computational experiments, this means that an independent group can obtain the same result using artifacts which they develop completely independently."

This is arguably the most difficult of the 3Rs to plan for, and for other researchers to achieve. To help others replicate your results:

- Take all the repeatability and reproducibility steps above.
- Note and communicate the high level algorithms involved, so others can create their own implementations of them. For example, if you have written an AI, then precisely communicate the algorithm that AI takes so others can implement it, and then run it with OpenTTDLab.

  For *full* replicability, it could be argued that OpenTTD itself, or something equivalent, would also have to be implemented. While this sounds unfeasbile, depending on what the AIs involve and what you argue the results are, it might be feasible. However, for this level of replicability OpenTTDLab is unlikely to be helpful.

Compatibility

- OpenTTD versions between 12.0 and 13.4 (OpenTTD >= 14.0 is not currently supported. See this discussion on the changes in OpenTTD 14.0.)
- Linux (tested on Ubuntu 20.04), Windows (tested on Windows Server 2019), or macOS (tested on macOS 12)
- Python >= 3.8.0 (tested on 3.8.0 and 3.12.0)

Licenses and attributions

TL;DR

OpenTTDLab is licensed under the GNU General Public License version 2.0.

In more detail

Releases 72

v0.0.72 Latest
on Jun 16

+ 71 releases

Packages

No packages published
Publish your first package

Contributors 3

 **michalc** Michal Charemza

 TrueBrain Patric Stout

 **BasicBeluga** Ian Earle

Deployments 73

pypi 2 months ago

+ 72 deployments

Languages

Python 98.4% **Squirrel** 1.6%