OpenAI Gym Environments for Donkey CarDocumentation

Release 1.0.13

Leigh Johnson

Aug 04, 2019

Contents:

CHAPTER 1

OpenAI Gym Environments for Donkey Car

Donkey Car OpenAI Gym

- Free software: MIT license
- Documentation: https://gym-donkeycar.readthedocs.io/en/latest/

1.1 Installation

- Install with pip

```
pip install gym-donkeycar
```

- Download simulator binaries: https://github.com/tawnkramer/donkey_gym/releases

1.2 Example Usage

```
import gym

env = gym.make("donkey-generated-track-v0)
```

1.3 Environments

- "donkey-warehouse-v0"
- "donkey-generated-roads-v0"
- "donkey-avc-sparkfun-v0"
- "donkey-generated-track-v0"

1.4 Credits

Original Source Code

Tawn Kramer - https://github.com/tawnkramer/donkey_gym Roma Sokolkov - https://github.com/r7vme/donkey_gym

Release Engineer

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install OpenAI Gym Environments for Donkey Car, run this command in your terminal:

```
$ pip install gym_donkeycar
```

This is the preferred method to install OpenAI Gym Environments for Donkey Car, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for OpenAI Gym Environments for Donkey Carcan be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/leigh-johnson/gym_donkeycar
```

Or download the tarball:

```
$ curl  -OL https://github.com/leigh-johnson/gym_donkeycar/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use OpenAI Gym Environments for Donkey Carin a project:

```
import gym_donkeycar
```

CHAPTER 4

gym_donkeycar

4.1 gym_donkeycar package

4.1.1 Subpackages

gym_donkeycar.core package

Submodules

gym_donkeycar.core.fps module

class gym_donkeycar.core.fps.**FPSTimer**
Bases: object

on_frame()

reset()

gym_donkeycar.core.tcp_server module

author: Tawn Kramer date: 16 October 2018 file: tcp_server.py notes: a tcp socket server to talk to the unity donkey simulator

class gym_donkeycar.core.tcp_server.**IMesgHandler**
Bases: object

on_close()

on_connect(*socketHandler*)

on_disconnect()

on_recv_message(*message*)

class gym_donkeycar.core.tcp_server.**SimHandler**(*sock*, *chunk_size=16384*, *msg_handler=None*)

Bases: asyncore.dispatcher

Handles messages from a single TCP client.

handle_close()

handle_json_message(*chunk*)
We are expecing a json object

handle_read()
Read an incoming message from the client and put it into our outgoing queue. handle_read should only be called when the given socket has data ready to be processed.

handle_write()
Write as much as possible of the most recent message we have received. This is only called by async manager when the socket is in a writable state and when self.writable return true, that yes, we have data to send.

queue_message(*msg*)

writable()
We want to write if we have received data.

class gym_donkeycar.core.tcp_server.**SimServer**(*address*, *msg_handler*)
Bases: asyncore.dispatcher

Receives network connections and establishes handlers for each client. Each client connection is handled by a new instance of the SteeringHandler class.

handle_accept()

handle_close()

gym_donkeycar.core.tcp_server.**replace_float_notation**(*string*)
Replace unity float notation for languages like French or German that use comma instead of dot. This convert the json sent by Unity to a valid one. Ex: “test”: 1,2, “key”: 2 -> “test”: 1.2, “key”: 2

Parameters **string** – (str) The incorrect json string

Returns (str) Valid JSON string

Module contents

gym_donkeycar.envs package

Submodules

gym_donkeycar.envs.donkey_env module

file: donkey_env.py author: Tawn Kramer date: 2018-08-31

class gym_donkeycar.envs.donkey_env.**AvcSparkfunEnv**
Bases: *gym_donkeycar.envs.donkey_env.DonkeyEnv*

class gym_donkeycar.envs.donkey_env.**DonkeyEnv**(*level*, *time_step=0.05*, *frame_skip=2*)
Bases: gym.core.Env

OpenAI Gym Environment for Donkey

ACTION_NAMES = ['steer', 'throttle']

STEER_LIMIT_LEFT = -1.0

STEER_LIMIT_RIGHT = 1.0

THROTTLE_MAX = 5.0

THROTTLE_MIN = 0.0

VAL_PER_PIXEL = 255

close()
Override close in your subclass to perform any necessary cleanup.

Environments will automatically close() themselves when garbage collected or when the program exits.

is_game_over()

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

render(*mode='human'*, *close=False*)
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.

Args: mode (str): the mode to render with

Example:

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

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

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

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

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

reset()
Resets the state of the environment and returns an initial observation.

Returns: observation (object): the initial observation.

seed(*seed=None*)
Sets the seed for this env's random number generator(s).

Note: Some environments use multiple pseudorandom number generators. We want to capture all such seeds used in order to ensure that there aren't accidental correlations between multiple generators.

Returns:

list<bigint>: Returns the list of seeds used in this env's random number generators. The first value in the list should be the "main" seed, or the value which a reproducer should pass to 'seed'. Often, the main seed equals the provided 'seed', but this won't be true if seed=None, for example.

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

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

Args: action (object): an action provided by the agent

Returns: observation (object): 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)

class gym_donkeycar.envs.donkey_env.**GeneratedRoadsEnv**
Bases: *gym_donkeycar.envs.donkey_env.DonkeyEnv*

class gym_donkeycar.envs.donkey_env.**GeneratedTrackEnv**
Bases: *gym_donkeycar.envs.donkey_env.DonkeyEnv*

class gym_donkeycar.envs.donkey_env.**WarehouseEnv**
Bases: *gym_donkeycar.envs.donkey_env.DonkeyEnv*

gym_donkeycar.envs.donkey_ex module

exception gym_donkeycar.envs.donkey_ex.**SimFailed**
Bases: Exception

gym_donkeycar.envs.donkey_proc module

file: donkey_proc.py author: Felix Yu date: 2018-09-12

class gym_donkeycar.envs.donkey_proc.**DonkeyUnityProcess**
Bases: object

quit()
Shutdown unity environment

start(*sim_path*, *headless=False*, *port=9090*)

gym_donkeycar.envs.donkey_sim module

file: donkey_sim.py author: Tawn Kramer date: 2018-08-31

class gym_donkeycar.envs.donkey_sim.**DonkeyUnitySimContoller**(*level*, *time_step=0.05*, *hostname='0.0.0.0'*, *port=9090*, *max_cte=5.0*, *loglevel='INFO'*, *cam_resolution=(120, 160, 3)*)
Bases: object

calc_reward(*done*)

get_sensor_size()

is_game_over()

observe()

quit()

render(*mode*)

reset()

take_action(*action*)

wait_until_loaded()

class gym_donkeycar.envs.donkey_sim.**DonkeyUnitySimHandler**(*level*, *time_step=0.05*, *max_cte=5.0*, *cam_resolution=None*)

Bases: *gym_donkeycar.core.tcp_server.IMesgHandler*

calc_reward(*done*)

determine_episode_over()

get_sensor_size()

is_game_over()

observe()

on_car_loaded(*data*)

on_connect(*socketHandler*)

on_disconnect()

on_recv_message(*message*)

on_recv_scene_names(*data*)

on_scene_selection_ready(*data*)

on_telemetry(*data*)

queue_message(*msg*)

reset()

send_control(*steer*, *throttle*)

send_get_scene_names()

send_load_scene(*scene_name*)

send_reset_car()

take_action(*action*)

Module contents

4.1.2 Module contents

Top-level package for OpenAI Gym Environments for Donkey Car.

CHAPTER 5

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

5.1 Types of Contributions

5.1.1 Report Bugs

Report bugs at https://github.com/leigh-johnson/gym_donkeycar/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

5.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

5.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

5.1.4 Write Documentation

OpenAI Gym Environments for Donkey Carcould always use more documentation, whether as part of the official OpenAI Gym Environments for Donkey Cardocs, in docstrings, or even on the web in blog posts, articles, and such.

5.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/leigh-johnson/gym_donkeycar/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

5.2 Get Started!

Ready to contribute? Here's how to set up *gym_donkeycar* for local development.

1. Fork the *gym_donkeycar* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/gym_donkeycar.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv gym_donkeycar
$ cd gym_donkeycar/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 gym_donkeycar tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

5.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.6, 3.7, and for PyPy. Check https://travis-ci.org/leigh-johnson/gym_donkeycar/pull_requests and make sure that the tests pass for all supported Python versions.

5.4 Tips

To run a subset of tests:

```
$ py.test tests.test_gym_donkeycar
```

5.5 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed (including an entry in HISTORY.rst). Then run:

```
$ bumpversion patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

WARNING: Ensure tests have passed on *branch* before cutting a release with *bumpversion && git push –tags*.

If the deploy build flakes, you will need run *bumpversion* and *git push –tags* again.

CHAPTER 6

Credits

6.1 Contributors

- Tawn Kramer @tawnkramer (original source: https://github.com/tawnkramer/donkey_gym)
- Roma Sokolkov @r7vme (https://github.com/r7vme/donkey_gym)
- Leigh Johnson @leigh-johnson <leigh@data-literate.com>

CHAPTER 7

History

7.1 1.0.0 (2019-07-26)

- First release on PyPI.

7.2 1.0.1 - 1.0.11 (2019-08-04)

- Testing out deploy system
- Update credits/authors
- flake8

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Python Module Index

g

Index

O

Q

R

S

T

V

W