Rog RL Documentation

Release 0.1.0

Sharada Mohanty

Oct 29, 2020

Contents:

1 Installation **1**
- 1.1 Stable release . . . 1
- 1.2 From sources . . . 1

2 Usage **3**

3 rog_rl **5**
- 3.1 rog_rl package . . . 5

4 Contributing **15**
- 4.1 Types of Contributions . . . 15
- 4.2 Get Started! . . . 16
- 4.3 Pull Request Guidelines . . . 17
- 4.4 Tips . . . 17
- 4.5 Deploying . . . 17

5 Credits **19**
- 5.1 Development Lead . . . 19
- 5.2 Contributors . . . 19

6 History **21**
- 6.1 0.1.0 (2020-04-02) . . . 21

7 Indices and tables **23**

Index **25**

CHAPTER 1

Installation

1.1 Stable release

To install Rog RL, run this command in your terminal:

```
$ pip install rog_rl
```

This is the preferred method to install Rog RL, 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.

1.2 From sources

The sources for Rog RL can be downloaded from the Gitlab repo.

You can either clone the public repository:

```
$ git clone git://gitlab.aicrowd.com/rog-rl/rog-rl
```

Or download the tarball:

```
$ curl -OJL https://gitlab.aicrowd.com/rog-rl/rog-rl/tarball/master
```

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

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

CHAPTER 2

Usage

To use Rog RL in a project::

```
import rog_rl
```

CHAPTER 3

rog_rl

3.1 rog_rl package

3.1.1 Submodules

3.1.2 rog_rl.agent module

class rog_rl.agent.**DiseaseSimAgent**(*unique_id*, *model*, *prob_agent_movement=0.0*, *moore=True*)

Bases: mesa.agent.Agent

moore = True

move_to(*new_position*)

Move the agent to a new location on the grid and do other associated house keeping tasks

- Update global observation in model

pos = None

prob_agent_movement = 0.0

process_state_transitions()

random_move()

set_state(*new_state: rog_rl.agent_state.AgentState*)

step()

A single step of the agent.

trigger_infection(*prob_infection=1.0*)

Attempts to trigger an infection, and if infection is triggered, then it returns True, else returns False.

3.1.3 rog_rl.agent_event module

class rog_rl.agent_event.**AgentEvent**(*previous_state=<AgentState.SUSCEPTIBLE: 0>*, *new_state=<AgentState.SUSCEPTIBLE: 0>*, *update_timestep=-1*)

Bases: object

mark_as_executed()
Mark that this event has been executed

mark_as_pending()
Mark that the execution of this event is pending

3.1.4 rog_rl.agent_state module

class rog_rl.agent_state.**AgentState**
Bases: enum.Enum

An enumeration.

EXPOSED = 1

INFECTIOUS = 2

RECOVERED = 4

SUSCEPTIBLE = 0

SYMPTOMATIC = 3

VACCINATED = 5

3.1.5 rog_rl.benchmark module

rog_rl.benchmark.**performance_metrics**(*render_on=False*)

rog_rl.benchmark.**profile**(*filename*)

3.1.6 rog_rl.cli module

Console script for rog_rl.

3.1.7 rog_rl.colors module

class rog_rl.colors.**ANSI_COLOR_MAP**
Bases: object

BACK_BLACK = '\x1b[40m'

BACK_BLUE = '\x1b[44m'

BACK_CYAN = '\x1b[46m'

BACK_GREEN = '\x1b[42m'

BACK_MAGENTA = '\x1b[45m'

BACK_RED = '\x1b[41m'

BACK_RESET = '\x1b[49m'

BACK_WHITE = '\x1b[47m'

BACK_YELLOW = '\x1b[43m'

FORE_BLACK = '\x1b[30m'

FORE_BLUE = '\x1b[34m'

FORE_CYAN = '\x1b[36m'

FORE_GREEN = '\x1b[32m'

FORE_MAGENTA = '\x1b[35m'

FORE_RED = '\x1b[31m'

FORE_RESET = '\x1b[39m'

FORE_WHITE = '\x1b[37m'

FORE_YELLOW = '\x1b[33m'

class rog_rl.colors.**ColorMap**(*mode='rgb'*)

Bases: object

get_color(*d*)

class rog_rl.colors.**Colors**

Bases: object

Reference : https://materialuicolors.co/ # Level : 600

Can potentially use : https://github.com/secretBiology/SecretColors/

AMBER = (255, 179, 0)

BLUE = (30, 136, 229)

BLUE_GREY = (84, 110, 122)

BROWN = (109, 76, 65)

CYAN = (0, 172, 193)

DEEP_ORANGE = (244, 81, 30)

DEEP_PURPLE = (94, 53, 177)

GREEN = (67, 160, 71)

GREY = (117, 117, 117)

INDIGO = (57, 73, 171)

LIGHT_BLUE = (3, 155, 229)

LIGHT_GREEN = (124, 179, 66)

LIGHT_GREY = (234, 237, 237)

LIME = (192, 202, 51)

ORANGE = (251, 140, 0)

PINK = (216, 27, 96)

PURPLE = (142, 36, 170)

RED = (229, 57, 53)

```
TEAL = (0, 137, 123)

WHITE = (255, 255, 255)

YELLOW = (253, 216, 53)
```

3.1.8 rog_rl.contact_network module

class rog_rl.contact_network.**ContactNetwork**
Bases: object

This keeps a record of all the "contacts" that happen in a single simulation

compute_R0()
Returns the value of R0 based on all the registered infections

register_contact(*agent_a*, *agent_b*)

register_infection_spread(*agent_a*, *agent_b*)
Register the fact that agent_a infected agent_b

3.1.9 rog_rl.disease_planner module

class rog_rl.disease_planner.**DiseasePlannerBase**(*random=False*)
Bases: object

This class plans the schedule of different state transitions for a disease

get_disease_plan(*base_timestep=0*)
Plans out the schedule of the state transitions for a particular agent using a particular disease model.

It returns a list of AgentEvent objects which have to be "executed" by the Agent at the right moment.

sample_disease_progression()

class rog_rl.disease_planner.**SEIRDiseasePlanner**(*latent_period_mu=8*, *latent_period_sigma=4*, *incubation_period_mu=20*, *incubation_period_sigma=12*, *recovery_period_mu=56*, *recovery_period_sigma=4*, *random=False*)
Bases: rog_rl.disease_planner.DiseasePlannerBase

This class plans the schedule of different state transitions for a disease

build_disease_plan(*disease_progression*, *base_timestep=0*)

get_disease_plan(*base_timestep=0*)
It returns a list of AgentEvent objects which have to be "executed" by the Agent at the right moment.

sample_disease_progression()
Plans out the schedule of the state transitions for a particular agent using a particular disease model.

class rog_rl.disease_planner.**SimpleSEIRDiseasePlanner**(*latent_period=2*, *incubation_period=5*, *recovery_period=14*, *random=False*)
Bases: rog_rl.disease_planner.SEIRDiseasePlanner

This class plans the schedule of different state transitions for a disease

3.1.10 rog_rl.env module

class rog_rl.env.**ActionType**
Bases: enum.Enum

An enumeration.

STEP = 0

VACCINATE = 1

class rog_rl.env.**RogSimEnv**(*config={}*)
Bases: gym.core.Env

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

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

dummy_env_step()
Implements a fake env.step for faster Integration Testing with RL experiments framework

get_current_game_metrics(*dummy_simulation=False*)
Returns a dictionary containing important game metrics

get_current_game_score()
Returns the current game score

The game score is currently represented as : (percentage of susceptibles left in the population)

initialize_renderer(*mode='human'*)

render(*mode='human'*)
This methods provides the option to render the environment's behavior to a window which should be readable to the human eye if mode is set to 'human'.

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

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

Returns: 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.

set_renderer(*renderer*)

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)

update_renderer(*mode='human'*)
Updates the latest board state on the renderer

3.1.11 rog_rl.model module

class rog_rl.model.**DiseaseSimModel**(*width=50*, *height=50*, *population_density=0.75*, *vaccine_density=0*, *initial_infection_fraction=0.1*, *initial_vaccination_fraction=0.0*, *prob_infection=0.2*, *prob_agent_movement=0.0*, *disease_planner_config={'incubation_period_mu': 20*, *'incubation_period_sigma': 0*, *'latent_period_mu': 8*, *'latent_period_sigma': 0*, *'recovery_period_mu': 56*, *'recovery_period_sigma': 0}*, *max_timesteps=200*, *early_stopping_patience=14*, *toric=True*, *seed=None*)

Bases: mesa.model.Model

The model class holds the model-level attributes, manages the agents, and generally handles the global level of our model.

There is only one model-level parameter: how many agents the model contains. When a new model is started, we want it to populate itself with the given number of agents.

The scheduler is a special model component which controls the order in which agents are activated.

get_observation()

get_population_fraction_by_state(*state: rog_rl.agent_state.AgentState*)

get_scheduler()

initialize_agents(*infection_fraction*, *vaccination_fraction*)
Intializes the intial agents on the grid

initialize_contact_network()
Initializes the contact network

initialize_datacollector()
Setup the initial datacollector

initialize_disease_planner()
Initializes a disease planner that the Agents can use to "schedule" infection progressions

initialize_grid()
Initializes the initial Grid

initialize_observation()
Observation is a nd-array of shape (width, height, num_states) where each AgentState will be marked in a separate challenge for each of the cells

initialize_scheduler()
Initializes the scheduler

is_running()

propagate_infections()
Propagates infection during a single simulation step

simulation_completion_checks()

Simulation is complete if :

- if the timesteps have exceeded the number of max_timesteps

or - the fraction of susceptible population is <= 0 or - the fraction of susceptible population has not changed since the last N timesteps

step()
A model step. Used for collecting data and advancing the schedule

tick()
a mirror function for the internal step function to help avoid confusion in the RL codebases (with the RL step)

vaccinate_cell(*cell_x*, *cell_y*)
Vaccinates an agent at cell_x, cell_y, if present

Response with : (is_vaccination_successful, vaccination_response) of types (boolean, VaccinationResponse)

3.1.12 rog_rl.renderer module

class rog_rl.renderer.**ANSIRenderer**
Bases: object

clear_screen()

close()

render(*grid*)

render_grid(*grid*)
Renders the Grid in ANSI

render_stats()

setup(*mode='ansi'*)

setup_stats()

update_stats(*key*, *value*)

class rog_rl.renderer.**Renderer**(*grid_size=(30, 30)*)
Bases: object

close()

convert_gym_color(*color: rog_rl.colors.Colors*)

draw_cell(*cell_x*, *cell_y*, *color=False*)

draw_grid(*color*)

draw_standard_line(*color*, *start_coord*, *end_coord*)

draw_standard_rect(*color*, *rect_dims*)

draw_stats()

get_cell_base(*cell_x*, *cell_y*)

get_grid_height()

get_grid_width()

post_render(*return_rgb_array=False*)
Some part of the code is taken from the file https://github.com/openai/gym/blob/master/gym/envs/classic_control/rendering.py The render method of class *viewer* clears the window. This also results in any text on the screen to be lost Hence we copy the contents of the *render* function and modify it

pre_render()

prepare_render()

setup(*mode='human'*)

setup_constants()

setup_stats()

update_stats(*key*, *value*)

3.1.13 rog_rl.scheduler module

class rog_rl.scheduler.**CustomScheduler**(*model: mesa.model.Model*)
Bases: mesa.time.RandomActivation

add(*agent: mesa.agent.Agent*) → None
Add an Agent object to the schedule.

Args: agent: An Agent to be added to the schedule. NOTE: The agent must have a step() method.

get_agent_count_by_state(*state: rog_rl.agent_state.AgentState*) → int
Returns the current number of agents in a particular state.

get_agent_fraction_by_state(*state: rog_rl.agent_state.AgentState*) → int
Returns the current number of agents in a particular state.

get_agents_by_state(*state: rog_rl.agent_state.AgentState*)

remove(*agent: mesa.agent.Agent*) → None
Remove all instances of a given agent from the schedule.

Args: agent: An agent object.

update_agent_state_in_registry(*agent: mesa.agent.Agent*, *previous_state: rog_rl.agent_state.AgentState*) → None

3.1.14 rog_rl.server module

Configure visualization elements and instantiate a server

rog_rl.server.**agent_potrayal**(*agent*)

rog_rl.server.**build_server**(*grid_width=50*, *grid_height=50*)

3.1.15 rog_rl.vaccination_response module

class rog_rl.vaccination_response.**VaccinationResponse**
Bases: enum.Enum

An enumeration.

AGENT_EXPOSED = 2

AGENT_INFECTIOUS = 3

AGENT_RECOVERED = 5

AGENT_SYMPTOMATIC = 4

AGENT_VACCINATED = 6

AGENT_VACCINES_EXHAUSTED = 7

CELL_EMPTY = 1

VACCINATION_SUCCESS = 0

3.1.16 rog_rl.visualization module

class rog_rl.visualization.**CustomTextGrid**(*grid*, *converter=None*)
Bases: mesa.visualization.TextVisualization.TextGrid

grid = None

render(*endl='\n'*)
What to show when printed.

3.1.17 Module contents

Top-level package for Rog RL.

CHAPTER 4

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:

4.1 Types of Contributions

4.1.1 Report Bugs

Report bugs at https://gitlab.aicrowd.com/rog-rl/rog-rl/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.

4.1.2 Fix Bugs

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

4.1.3 Implement Features

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

4.1.4 Write Documentation

Rog RL could always use more documentation, whether as part of the official Rog RL docs, in docstrings, or even on the web in blog posts, articles, and such.

4.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://gitlab.aicrowd.com/rog-rl/rog-rl/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 :)

4.2 Get Started!

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

1. Fork the *RogRL* repo on gitlab.
2. Clone your fork locally:

```
$ git clone git@gitlab.aicrowd.com:your_name_here/RogRL.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 RogRL
$ cd RogRL/
$ 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 rog_rl tests
$ python setup.py test or pytest
$ tox
```

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

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

```
$ 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 gitlab website.

4.3 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 3.5, 3.6, 3.7 and 3.8, and for PyPy. Check https://travis-ci.com/gitlab/spMohanty/RogRL/pull_requests and make sure that the tests pass for all supported Python versions.

4.4 Tips

To run a subset of tests:

```
$ pytest -k agent_state
```

4.5 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:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```

Travis will then deploy to PyPI if tests pass.

CHAPTER 5

Credits

5.1 Development Lead

- Sharada Mohanty <spmohanty91@gmail.com>

5.2 Contributors

None yet. Why not be the first?

CHAPTER 6

History

6.1 0.1.0 (2020-04-02)

- First release on PyPI.

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Index

A

ActionType (*class in rog_rl.env*), 9
add() (*rog_rl.scheduler.CustomScheduler method*), 12
AGENT_EXPOSED (*rog_rl.vaccination_response.VaccinationResponse attribute*), 12
AGENT_INFECTIOUS (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
agent_potrayal() (*in module rog_rl.server*), 12
AGENT_RECOVERED (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
AGENT_SYMPTOMATIC (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
AGENT_VACCINATED (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
AGENT_VACCINES_EXHAUSTED (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
AgentEvent (*class in rog_rl.agent_event*), 6
AgentState (*class in rog_rl.agent_state*), 6
AMBER (*rog_rl.colors.Colors attribute*), 7
ANSI_COLOR_MAP (*class in rog_rl.colors*), 6
ANSIRenderer (*class in rog_rl.renderer*), 11

B

BACK_BLACK (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_BLUE (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_CYAN (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_GREEN (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_MAGENTA (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_RED (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_RESET (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 6
BACK_WHITE (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
BACK_YELLOW (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
BLUE (*rog_rl.colors.Colors attribute*), 7
BLUE_GREY (*rog_rl.colors.Colors attribute*), 7
BROWN (*rog_rl.colors.Colors attribute*), 7
build_disease_plan() (*rog_rl.disease_planner.SEIRDiseasePlanner method*), 8
build_server() (*in module rog_rl.server*), 12

C

CELL_EMPTY (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
clear_screen() (*rog_rl.renderer.ANSIRenderer method*), 11
close() (*rog_rl.env.RogSimEnv method*), 9
close() (*rog_rl.renderer.ANSIRenderer method*), 11
close() (*rog_rl.renderer.Renderer method*), 11
ColorMap (*class in rog_rl.colors*), 7
Colors (*class in rog_rl.colors*), 7
compute_R0() (*rog_rl.contact_network.ContactNetwork method*), 8
ContactNetwork (*class in rog_rl.contact_network*), 8
convert_gym_color() (*rog_rl.renderer.Renderer method*), 11
CustomScheduler (*class in rog_rl.scheduler*), 12
CustomTextGrid (*class in rog_rl.visualization*), 13
CYAN (*rog_rl.colors.Colors attribute*), 7

D

DEEP_ORANGE (*rog_rl.colors.Colors attribute*), 7
DEEP_PURPLE (*rog_rl.colors.Colors attribute*), 7
DiseasePlannerBase (*class in rog_rl.disease_planner*), 8
DiseaseSimAgent (*class in rog_rl.agent*), 5
DiseaseSimModel (*class in rog_rl.model*), 10
draw_cell() (*rog_rl.renderer.Renderer method*), 11

draw_grid() (*rog_rl.renderer.Renderer method*), 11
draw_standard_line() (*rog_rl.renderer.Renderer method*), 11
draw_standard_rect() (*rog_rl.renderer.Renderer method*), 11
draw_stats() (*rog_rl.renderer.Renderer method*), 11
dummy_env_step() (*rog_rl.env.RogSimEnv method*), 9

E

EXPOSED (*rog_rl.agent_state.AgentState attribute*), 6

F

FORE_BLACK (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_BLUE (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_CYAN (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_GREEN (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_MAGENTA (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_RED (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_RESET (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_WHITE (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7
FORE_YELLOW (*rog_rl.colors.ANSI_COLOR_MAP attribute*), 7

G

get_agent_count_by_state() (*rog_rl.scheduler.CustomScheduler method*), 12
get_agent_fraction_by_state() (*rog_rl.scheduler.CustomScheduler method*), 12
get_agents_by_state() (*rog_rl.scheduler.CustomScheduler method*), 12
get_cell_base() (*rog_rl.renderer.Renderer method*), 11
get_color() (*rog_rl.colors.ColorMap method*), 7
get_current_game_metrics() (*rog_rl.env.RogSimEnv method*), 9
get_current_game_score() (*rog_rl.env.RogSimEnv method*), 9
get_disease_plan() (*rog_rl.disease_planner.DiseasePlannerBase method*), 8
get_disease_plan() (*rog_rl.disease_planner.SEIRDiseasePlanner method*), 8
get_grid_height() (*rog_rl.renderer.Renderer method*), 11
get_grid_width() (*rog_rl.renderer.Renderer method*), 12
get_observation() (*rog_rl.model.DiseaseSimModel method*), 10
get_population_fraction_by_state() (*rog_rl.model.DiseaseSimModel method*), 10
get_scheduler() (*rog_rl.model.DiseaseSimModel method*), 10
GREEN (*rog_rl.colors.Colors attribute*), 7
GREY (*rog_rl.colors.Colors attribute*), 7
grid (*rog_rl.visualization.CustomTextGrid attribute*), 13

I

INDIGO (*rog_rl.colors.Colors attribute*), 7
INFECTIOUS (*rog_rl.agent_state.AgentState attribute*), 6
initialize_agents() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_contact_network() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_datacollector() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_disease_planner() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_grid() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_observation() (*rog_rl.model.DiseaseSimModel method*), 10
initialize_renderer() (*rog_rl.env.RogSimEnv method*), 9
initialize_scheduler() (*rog_rl.model.DiseaseSimModel method*), 10
is_running() (*rog_rl.model.DiseaseSimModel method*), 10

L

LIGHT_BLUE (*rog_rl.colors.Colors attribute*), 7
LIGHT_GREEN (*rog_rl.colors.Colors attribute*), 7
LIGHT_GREY (*rog_rl.colors.Colors attribute*), 7
LIME (*rog_rl.colors.Colors attribute*), 7

M

mark_as_executed() (*rog_rl.agent_event.AgentEvent method*), 6
mark_as_pending() (*rog_rl.agent_event.AgentEvent method*), 6
moore (*rog_rl.agent.DiseaseSimAgent attribute*), 5
move_to() (*rog_rl.agent.DiseaseSimAgent method*), 5

O

ORANGE (*rog_rl.colors.Colors attribute*), 7

P

performance_metrics() (*in module rog_rl.benchmark*), 6
PINK (*rog_rl.colors.Colors attribute*), 7
pos (*rog_rl.agent.DiseaseSimAgent attribute*), 5
post_render() (*rog_rl.renderer.Renderer method*), 12
pre_render() (*rog_rl.renderer.Renderer method*), 12
prepare_render() (*rog_rl.renderer.Renderer method*), 12
prob_agent_movement (*rog_rl.agent.DiseaseSimAgent attribute*), 5
process_state_transitions() (*rog_rl.agent.DiseaseSimAgent method*), 5
profile() (*in module rog_rl.benchmark*), 6
propagate_infections() (*rog_rl.model.DiseaseSimModel method*), 10
PURPLE (*rog_rl.colors.Colors attribute*), 7

R

random_move() (*rog_rl.agent.DiseaseSimAgent method*), 5
RECOVERED (*rog_rl.agent_state.AgentState attribute*), 6
RED (*rog_rl.colors.Colors attribute*), 7
register_contact() (*rog_rl.contact_network.ContactNetwork method*), 8
register_infection_spread() (*rog_rl.contact_network.ContactNetwork method*), 8
remove() (*rog_rl.scheduler.CustomScheduler method*), 12
render() (*rog_rl.env.RogSimEnv method*), 9
render() (*rog_rl.renderer.ANSIRenderer method*), 11
render() (*rog_rl.visualization.CustomTextGrid method*), 13
render_grid() (*rog_rl.renderer.ANSIRenderer method*), 11
render_stats() (*rog_rl.renderer.ANSIRenderer method*), 11
Renderer (*class in rog_rl.renderer*), 11
reset() (*rog_rl.env.RogSimEnv method*), 9
rog_rl (*module*), 13
rog_rl.agent (*module*), 5
rog_rl.agent_event (*module*), 6
rog_rl.agent_state (*module*), 6
rog_rl.benchmark (*module*), 6
rog_rl.cli (*module*), 6
rog_rl.colors (*module*), 6
rog_rl.contact_network (*module*), 8
rog_rl.disease_planner (*module*), 8
rog_rl.env (*module*), 9
rog_rl.model (*module*), 10
rog_rl.renderer (*module*), 11
rog_rl.scheduler (*module*), 12
rog_rl.server (*module*), 12
rog_rl.vaccination_response (*module*), 12
rog_rl.visualization (*module*), 13
RogSimEnv (*class in rog_rl.env*), 9

S

sample_disease_progression() (*rog_rl.disease_planner.DiseasePlannerBase method*), 8
sample_disease_progression() (*rog_rl.disease_planner.SEIRDiseasePlanner method*), 8
seed() (*rog_rl.env.RogSimEnv method*), 9
SEIRDiseasePlanner (*class in rog_rl.disease_planner*), 8
set_renderer() (*rog_rl.env.RogSimEnv method*), 9
set_state() (*rog_rl.agent.DiseaseSimAgent method*), 5
setup() (*rog_rl.renderer.ANSIRenderer method*), 11
setup() (*rog_rl.renderer.Renderer method*), 12
setup_constants() (*rog_rl.renderer.Renderer method*), 12
setup_stats() (*rog_rl.renderer.ANSIRenderer method*), 11
setup_stats() (*rog_rl.renderer.Renderer method*), 12
SimpleSEIRDiseasePlanner (*class in rog_rl.disease_planner*), 8
simulation_completion_checks() (*rog_rl.model.DiseaseSimModel method*), 11
STEP (*rog_rl.env.ActionType attribute*), 9
step() (*rog_rl.agent.DiseaseSimAgent method*), 5
step() (*rog_rl.env.RogSimEnv method*), 9
step() (*rog_rl.model.DiseaseSimModel method*), 11
SUSCEPTIBLE (*rog_rl.agent_state.AgentState attribute*), 6

SYMPTOMATIC (*rog_rl.agent_state.AgentState attribute*), 6

T

TEAL (*rog_rl.colors.Colors attribute*), 7
tick() (*rog_rl.model.DiseaseSimModel method*), 11
trigger_infection() (*rog_rl.agent.DiseaseSimAgent method*), 5

U

update_agent_state_in_registry() (*rog_rl.scheduler.CustomScheduler method*), 12
update_renderer() (*rog_rl.env.RogSimEnv method*), 10
update_stats() (*rog_rl.renderer.ANSIRenderer method*), 11
update_stats() (*rog_rl.renderer.Renderer method*), 12

V

VACCINATE (*rog_rl.env.ActionType attribute*), 9
vaccinate_cell() (*rog_rl.model.DiseaseSimModel method*), 11
VACCINATED (*rog_rl.agent_state.AgentState attribute*), 6
VACCINATION_SUCCESS (*rog_rl.vaccination_response.VaccinationResponse attribute*), 13
VaccinationResponse (*class in rog_rl.vaccination_response*), 12

W

WHITE (*rog_rl.colors.Colors attribute*), 8

Y

YELLOW (*rog_rl.colors.Colors attribute*), 8