

RODENT Python SDK: cheat sheet

Version 0.4.2 For Google Colab or any Python 3.9+

Starter notebook: on the website, open an experiment and click **Use in Python**, then **Open in Google Colab**. Or open https://colab.research.google.com/github/HuzaifaMajeed/rodent-sdk-releases/blob/main/notebooks/getting_started.ipynb

Training notebook: https://colab.research.google.com/github/HuzaifaMajeed/rodent-sdk-releases/blob/main/notebooks/train_your_rat.ipynb

The left column is what you type; the right column is what it does. *Italic words* are placeholders for your own names or values. The names of doors, stimuli and regions come from printing your experiment (section 6).

1 Get started

Code	What it does
<code>!pip install rodent-sdk==0.4.2</code>	Install the SDK (in Colab; elsewhere, leave out the !).
<code>!pip install "rodent-sdk[train]==0.4.2"</code>	The same, plus what training needs (Gymnasium, NumPy, pandas, matplotlib).
<code>import rodent_sdk as rodent</code>	Load the SDK.
<code>rodent.__version__</code>	The installed version, for example "0.4.2".
<code>rodent.templates()</code>	A list of the ready-made arena names.
<code>rodent.TEMPLATES</code>	The same names, as a fixed tuple.
<code>rodent.build_info()</code>	How this SDK was built: website address, simulator pack and Godot version.
<code>rodent.DEFAULT_GODOT_VERSION</code>	The Godot version the simulator runs on ("4.7.1").
Arenas: <code>complex_habitat</code> , <code>mcsf_sprint_one</code> , <code>open_field</code> , <code>light_dark_box</code> , <code>t_maze</code> , <code>four_room_arena</code> .	

2 Quick runs without the website

Code	What it does
<code>run = rodent.run("arena", steps=5000, seed=42)</code>	Run an arena in one line. Nothing is saved online. Also takes <code>changes=</code> (section 9).
<code>exp = rodent.experiment("arena", seed=42, name="name")</code>	An experiment you can change before running. <code>seed</code> and <code>name</code> are optional.
<code>exp = rodent.Experiment.from_template("arena", seed=42)</code>	The same as <code>rodent.experiment</code> .

3 Sign in

Code	What it does
<code>client = rodent.login()</code>	Sign in. In Colab it reads <code>RODENT_EMAIL</code> and <code>RODENT_PASSWORD</code> from Secrets; otherwise it asks.
<code>client = rodent.login("email", "password")</code>	Sign in with the details typed in (avoid this in shared notebooks).
<code>print(client)</code>	Who is signed in, and their role.
<code>client.user</code>	Your profile: email, role and id.
<code>client.user_id</code>	Your account id.

4 Projects

Code	What it does
<code>client.projects()</code>	Your projects.
<code>project = client.project("name or id")</code>	Open a project. If two share a name, use the id.
<code>project = client.create_project("name", template="arena", description="text")</code>	Create a project with a baseline experiment. You become its lead. Optional <code>scopes=("light",)</code> limits what may be customised.
<code>project.name, project.id, project.description</code>	The project's name, id and description.
<code>project.is_lead</code>	True if you lead the project.
<code>project.scopes</code>	The modules that may be customised, for example <code>["all"]</code> .
<code>project.choice_label()</code>	A short label: your role and the modules.

5 Experiments on the website

Code	What it does
<code>project.experiments()</code>	The project's experiments, most recently edited first.
<code>exp = project.experiment("name or id")</code>	Open an experiment.
<code>p, e = "code".split("/") exp = client.project(p).experiment(e)</code>	Open the experiment whose code you copied from Use in Python on the website.
<code>exp = project.create_experiment("name", template="arena", seed=7)</code>	Create an experiment from an arena. Project leads only; <code>seed</code> is optional.
<code>exp.id</code>	The experiment's id.
<code>exp.status</code>	"draft" or "published".
<code>exp.version, exp.revision</code>	The published version number, and how many times it has been saved.
<code>exp.choice_label()</code>	A short label: status, version and when it was last updated.

6 Look at an experiment

Code	What it does
<code>print(exp)</code>	A one-line description.
<code>exp.name</code>	The experiment's name (you can also set it).
<code>exp.seed</code>	The seed.
<code>exp.regions</code>	The names of the arena's regions.
<code>exp.doors</code>	The doors, and whether each is open.
<code>exp.stimuli</code>	The lights, sounds and odours.
<code>exp.treatment</code>	Treatment condition, dose, unit and step.
<code>exp.config</code>	The complete setup, as a dictionary.
<code>exp.to_dict()</code>	A copy of the complete setup.
One door (<code>door = exp.doors["door"]</code>):	
Code	What it does
<code>door.id, door.label</code>	The door's name, and the label shown on the website.
<code>door.open</code>	True if open at the start. Set it to open or close the door.
<code>door.enabled</code>	False leaves the door out of the run.
One stimulus (<code>s = exp.stimuli["stimulus"]</code>):	
Code	What it does
<code>s.id, s.type</code>	The stimulus's name, and "light", "sound" or "odour".
<code>s.active</code>	True if on at the start.
<code>s.intensity</code>	Strength: lux for lights, 0 to 1 for sounds and odours.
<code>s.radius</code>	Reach, in centimetres.
<code>s.enabled</code>	False leaves the stimulus out of the run.
Treatment (<code>t = exp.treatment</code>):	
Code	What it does
<code>t.condition</code>	"no_treatment", "placebo" or "medicine".
<code>t.dose, t.unit</code>	The dose, and its unit such as "mg".
<code>t.administration_step</code>	The step at which it is given.

7 Change an experiment

Code	What it does
<code>exp.doors["door"].open = False</code>	Close a door at the start (True opens it).
<code>exp.doors["door"].enabled = False</code>	Remove a door from the run.
<code>exp.stimuli["stimulus"].active = True</code>	Switch a stimulus on at the start.
<code>exp.stimuli["stimulus"].intensity = 250</code>	Change its strength.
<code>exp.stimuli["stimulus"].radius = 15</code>	Change its reach, in centimetres.
<code>exp.stimuli["stimulus"].enabled = False</code>	Remove a stimulus from the run.
<code>exp.stimuli["stimulus"].enabled = True</code>	Put a stimulus in the arena. A source works only when it is enabled <i>and</i> active; the templates' odour and sound sources start disabled.
<code>exp.treatment.condition = "placebo"</code>	Change the treatment condition.
<code>exp.treatment.dose = 2.5</code>	Change the dose (also <code>.unit</code> and <code>.administration_step</code>).
<code>exp.seed = 7</code>	Change the seed (a whole number, 1 or more).
<code>exp.name = "new name"</code>	Rename the experiment.
<code>exp.validate()</code>	Problems the simulation would reject; an empty list means fine.

8 Keep changes on the website

Code	What it does
<code>exp.has_unsaved_changes</code>	Whether you have changes that are not saved.
<code>exp.save()</code>	Store your changes in this draft. The website checks them against your role.
<code>exp.reload()</code>	Discard unsaved changes and load the saved version again.
<code>copy = exp.save_as_new("new name")</code>	Copy the experiment, with your changes, into a new draft. Works for any editor.
<code>exp.publish()</code>	Save and freeze this draft as the next published version.

9 Run

Code	What it does
<code>run = exp.run(steps=5000)</code>	Run it. One step is 0.1 s, so 5000 steps is 500 s.
<code>run = exp.run(steps=5000, changes=changes)</code>	Run with doors or stimuli changed at chosen steps (below).
<code>print(run)</code>	A one-line result.

Changing doors or stimuli during a run, at chosen steps (**True** opens or switches on):

```
run = exp.run(steps=5000, changes={
    1500: {"doors": {"north_gate": False}},
    3000: {"doors": {"bridge_entry_gate": True},
          "stimuli": {"bridge_light": False}},
})
```

10 Results

Code	What it does
<code>run.summary()</code>	Steps, duration, distance, interventions, and per region: time, entries and first entry.
<code>run.region_table()</code>	The per-region results as a table.
<code>run.rows()</code>	Every step as a list of rows (the same columns as the CSV).
<code>run.to_dataframe()</code>	Every step as a pandas table.
<code>run.plot()</code>	The rat's path over the arena. Pass <code>ax=</code> to draw into your own figure.
<code>run.door_states()</code>	Each door's final state, and the set of doors changed during the run.
<code>run.seed</code> , <code>run.steps</code>	The seed and the number of steps.
<code>run.fixed_delta</code>	Seconds per step (0.1 by default).
<code>run.config</code>	The exact setup the run used.

Columns of `rows()`, `to_dataframe()` and the CSV: `step`, `time`, `x`, `y` (cm), `region_id`, `active_stimuli`, `interventions`.

11 Save and download

Code	What it does
<code>run.save()</code>	Save to the website, under Review for this experiment. Returns the saved run's id.
<code>run.save(allow_unsaved=True)</code>	Save even though the run used unsaved changes. The website would replay it on the saved setup, so avoid this.
<code>run.save_csv("run.csv")</code>	Save every step as a CSV file.
<code>run.save_json("run.json")</code>	Save the complete run, in the website's replay format.
<code>rodent.RunResult.load_json("run.json")</code>	Open a saved JSON run.
<code>exp.replays()</code>	Runs saved to the website for this experiment, newest first.
<code>exp.load_replay("id")</code>	Download a saved run to analyse it.

To download a file from Colab to your computer:

```
from google.colab import files
files.download("run.csv")
```

12 Train your own rat

Your own model (for example a PyTorch network) controls the rat and learns from rewards. Each try is an **episode**. The environment uses the Gymnasium format, so training loops written for Gymnasium work unchanged.

Code	What it does
<code>env = exp.training_env(max_steps=1000, reward=reward)</code>	A training environment: your model drives the rat. One episode is <code>max_steps</code> steps (1000 is 100s).
<code>reward=function</code>	Your idea of good: a function of the step (below) that returns a number.
<code>done=function</code>	Optional: end an episode early, for example when the rat reaches the goal.
<code>actions="discrete"</code>	Five moves (stay, forward, left, right, run) instead of a turn and a speed.
<code>start="random"</code>	Start each episode at a random free spot. Also <code>(x, y)</code> , or <code>{"random": True, "position": [x, y]}</code> for a fixed spot and random heading.
<code>log="folder"</code>	Keep every episode so any can be rebuilt later. In Colab, a Google Drive folder survives a disconnect.
<code>copies=2</code>	Run two simulators side by side (Colab has two cores).
<code>seed=n, max_turn=radians, stimulus_reactions=True, extra_obs=function</code>	Episode <code>n</code> uses seed <code>seed + n</code> ; turn limit per step; keep the built-in reactions to stimuli; add your own observation values.
<code>obs, info = env.reset()</code>	Start the next episode.
<code>obs, reward, terminated, truncated, info = env.step(action)</code>	One step. <code>action</code> is <code>[turn, speed]</code> (turn <code>-1</code> to <code>1</code> , speed <code>0</code> to <code>1</code>) or a move number.
<code>env.observation_names, env.obs_size</code>	The 31 numbers the rat senses, in order (table below).
<code>env.action_space, env.observation_space</code>	The Gymnasium spaces.
<code>env.episode</code>	The current episode's number (from 1).
<code>env.close()</code>	Stop the simulator.

What your `reward` and `done` functions receive for each step:

Code	What it does
<code>step.region, step.entered_region</code>	The region the rat is in, and whether it just entered it.
<code>step.position, step.heading</code>	Where it is (cm) and which way it faces (radians).
<code>step.hit_wall, step.moved</code>	Whether a wall stopped it, and how far it moved (cm).
<code>step.light, step.sound, step.odour</code>	What it senses here, 0 to 1.
<code>step.step, step.action, step.events, step.treatment</code>	The step number, the action taken, door and stimulus changes, and the treatment.

What the rat senses (`obs`, 31 numbers): position (`x, y`, 0 to 1 of the arena); heading (`sin, cos`); speed; last turn; distance to the nearest wall or closed door in 8 directions around it (0 touching, 1 at 50 cm or more); touching a wall; then for light, sound and odour in turn: strength here, at its left side, at its right side, and the direction it comes from (`sin, cos`, relative to the heading); and how far through the episode it is. Light is blocked by walls and bounces once; sound and odour are weakened by every wall they cross; odour spreads as a gas.

13 Look at any episode

Code	What it does
<code>env.log.plot()</code>	Learning curves: reward per episode, and time in each region.
<code>env.log.table()</code>	One row per episode: seed, steps, reward, distance, time in each region.
<code>ep = env.log.episode(522)</code>	Episode 522 rebuilt exactly, with its full recording. The model is not needed.
<code>ep.plot(steps=(400, 600))</code>	Its path between two steps.
<code>ep.at(450)</code>	The rat after 450 steps: position, region, what it sensed, the action it took next, the reward.
<code>ep.animate(steps=(400, 600))</code>	A 2D animation that plays in Colab.
<code>ep.rewards, ep.total_reward</code>	The reward at every step, and in total.
<code>ep.save()</code>	Save it to Review on the website (website experiments only).
<code>ep.review_link(450)</code>	A link that opens it in 3D on the website at that step.
<code>env.log.compare([1, 240, 480])</code>	Several episodes side by side.
<code>log = rodent.open_log("folder")</code>	Open a log later, in a new session.

An episode is also a run, so `summary()`, `region_table()`, `to_dataframe()` and `save_csv()` work on it.

14 Use and share a trained rat

Code	What it does
<code>run = exp.run(steps=5000, model=policy)</code>	A normal run where your model drives the rat. <i>policy</i> is a function from <code>obs</code> to an action, or a PyTorch network. A network with an <code>act()</code> method chooses its own moves, as in training, seeded so the run repeats; otherwise its highest-scoring move is taken.
<code>exp.run(..., episode_length=1000)</code>	Tell it the training episode length, so the progress input means the same.
<code>rodent.compare(built_in=run_a, trained=run_b)</code>	Time in each region and distance, side by side.
<code>project.save_model("name", network, notes="...", training=env.log)</code>	Save the network's weights to the project as the next version, with how it was trained.
<code>project.models(), project.model("name")</code>	The project's saved models.
<code>model.versions(), model.version(3), model.latest</code>	Its versions.
<code>model.latest.load_into(MyNetwork())</code>	Load the weights into your network (always loaded safely, weights only). The network must have the same shape as the one saved.
<code>model.latest.download("weights.pt")</code>	Download the weights file, checked against its fingerprint.
<code>from rat_model import RatModel</code>	The training notebook's model (a PPO actor-critic) from its file <code>rat_model.py</code> . Replace the file to train your own model with the same notebook.
<code>run.save(); run.review_link()</code>	Save a model-driven run; Review shows the model and version.
<code>exp.stimulus_map("light")</code>	A heat map of what the rat would sense across the arena. Also "sound", and "odour" with <code>after_steps=300</code> to let the gas spread.

15 Step by step: the simulation engine

For moving through a run one step at a time, for example to try your own logic. Most people never need this.

Code	What it does
<code>eng = rodent.Engine()</code>	Start a simulator. Use with <code>rodent.Engine()</code> as <code>eng</code> : so it closes itself.
<code>eng.load_config(exp.to_dict())</code>	Load an experiment's setup.
<code>eng.load("experiment.json")</code>	Load a setup from a file.
<code>eng.reset()</code>	Back to step 0.
<code>eng.step(10)</code>	Advance by a number of steps (1 if left out).
<code>eng.observe()</code>	The current state: the rat's position, heading, region, nearby stimuli and more.
<code>eng.act({"heading_delta": 0.3})</code>	Turn the rat (radians), then advance one step. {"position": [x, y]} moves it. Add <code>advance=False</code> to stay on this step. Recorded as an intervention.
<code>eng.set_door("door", False)</code>	Close (or open) a door now.
<code>eng.set_stimulus("stimulus", True)</code>	Switch a stimulus on (or off) now.
<code>eng.results()</code>	The recording and measures so far. <code>include_recording=False</code> is lighter.
<code>eng.info</code>	What the simulator supports, and its versions.
<code>eng.env_reset(seed=1) / eng.env_step(turn, speed)</code>	The commands a training environment uses: start an episode with your model in control, then move it one step.
<code>eng.sense_map(cell=2.0)</code>	Light, sound and odour strength on a grid over the arena, now.
<code>eng.close()</code>	Stop the simulator.
<code>exp = rodent.experiment("arena", engine=eng)</code>	Use your engine for runs (<code>login</code> and <code>run</code> also take <code>engine=</code>).

16 Simulator files and settings

The SDK downloads Godot and the simulator pack the first time, then reuses them.

Code	What it does
<code>rodent.ensure_godot()</code>	Download Godot now, if needed. Returns where it is.
<code>rodent.ensure_pack()</code>	Download the simulator pack now, if needed, and check it.
<code>rodent.Engine(pack="file.pck", godot="path")</code>	Use your own pack or Godot (developers).

Environment variables: `RODENT_CACHE` (where downloads go), `RODENT_PACK` and `RODENT_GODOT` (your own files), `RODENT_SUPABASE_URL` and `RODENT_SUPABASE_KEY` (a different website, also `login(url=, key=)`).

17 Direct website access (advanced)

These reach the website's database directly, within your permissions. The functions above cover normal use.

Code	What it does
<code>client.get("table", {"column": "eq.value"})</code>	Read rows, for example <code>client.get("replays", {"experiment_id": "eq."+exp.id})</code> .
<code>client.insert("table", row)</code>	Add a row.
<code>client.update("table", filters, changes)</code>	Change rows.
<code>client.rpc("function", arguments)</code>	Call a website database function.

18 Good to know

- `True` and `False` start with a capital letter. Names go in quotes.
- The same experiment and seed always give the same run.
- Save your changes before running if you want to save the run to the website.
- Published versions cannot be changed; continue with `save_as_new`.
- If someone saved after you loaded, reload and redo your change.
- Keep saved runs under about 5,000 steps in busy arenas: each step now records more about the stimuli.
- If saving a run says `statement timeout`, run `run.save()` again; nothing was half-saved.
- New projects start without stimuli; add them on the website (Setup, Stimuli) or use a template experiment.
- Each training run needs its own `log` folder; a folder made with a different experiment or settings is refused.
- If your account has two-factor sign-in, `rodent.login()` asks for the six-digit code (or pass `totp_code="..."`).
- The built-in rat is a placeholder model, not a validated biological result. Your own model's rat follows the same walls, doors and speed limits.
- Training speed: about 1,000 steps a second per simulator, plus your model's time; `copies=2` runs two side by side.

19 When something is refused

Messages say what to do next. To keep going after one:

```
try:
    exp.save()
except rodent.RodentError as problem:
    print(problem)
```

Code	What it does
<code>rodent.RodentAuthError</code>	A login problem.
<code>rodent.RodentPermissionError</code>	Your role does not allow this.
<code>rodent.RodentDownloadError</code>	The simulator could not be downloaded, or the download was damaged. Run it again.
<code>rodent.RodentError</code>	Anything else on the website side (also catches the three above).
<code>rodent.RodentCommandError</code>	The simulator rejected a setup or command; <code>.problems</code> lists why.

20 Recipes

Open a project and experiment, or create them if they do not exist

```
names = [p.name for p in client.projects()]
if "My study" in names:
    project = client.project("My study")
else:
    project = client.create_project("My study", template="four_room_arena")

names = [e.name for e in project.experiments()]
if "Trial A" in names:
    exp = project.experiment("Trial A")
else:
    exp = project.create_experiment("Trial A", template="four_room_arena")
```

Compare two conditions

```
dim = exp.save_as_new("Dim light")
dim.stimuli["bridge_light"].intensity = 50
dim.save()
print(exp.run(steps=5000).region_table())
print(dim.run(steps=5000).region_table())
```

Repeat with several seeds

```
for seed in [1, 2, 3, 4, 5]:
    result = rodent.run("complex_habitat", steps=3000, seed=seed)
    print(seed, round(result.summary()["distance_cm"]))
```

Train a small PyTorch rat to find the dark room

```
import math, torch
exp = rodent.experiment("light_dark_box", seed=1)
def reward(step):
    if step.region == "dark_room":
        return 1.0
    x, y = step.position
    return -math.hypot(50 - x, 50 - y) / 100    # warmer near the door
env = exp.training_env(max_steps=150, reward=reward, actions="discrete",
                       start={"random": True, "position": [12, 15]}, log="training")
policy = torch.nn.Sequential(torch.nn.Linear(env.obs_size, 64), torch.nn.Tanh(),
                              torch.nn.Linear(64, 5))
# ...your training loop (the training notebook has a complete one)...
run = exp.run(steps=300, model=lambda obs: int(policy(torch.as_tensor(obs)).argmax()),
              actions="discrete", start=[12, 15])
```

Heat map of where the rat spent time

```
import matplotlib.pyplot as plt
data = run.to_dataframe()
plt.hist2d(data["x"], data["y"], bins=40)
plt.gca().invert_yaxis()
plt.show()
```