

Simulation-based Testing with BeamNG.tech

Tutorial @ SBST 2021

Alessio Gambi, Pascale Maul, Marc Müller

The Team



Dr. Alessio Gambi

University of Passau, Germany



Pascale Maul · 3rd

MSc Informatik

Dresden, Saxony, Germany · 37 connections · [Contact info](#)

[Message](#)

...

BeamNG GmbH

Universität Bremen

The Team



Marc M. · 3rd

Programmer at BeamNG GmbH

Bremen, Bremen, Germany · 63 connections · [Contact info](#)

[Message](#)

...

BeamNG GmbH

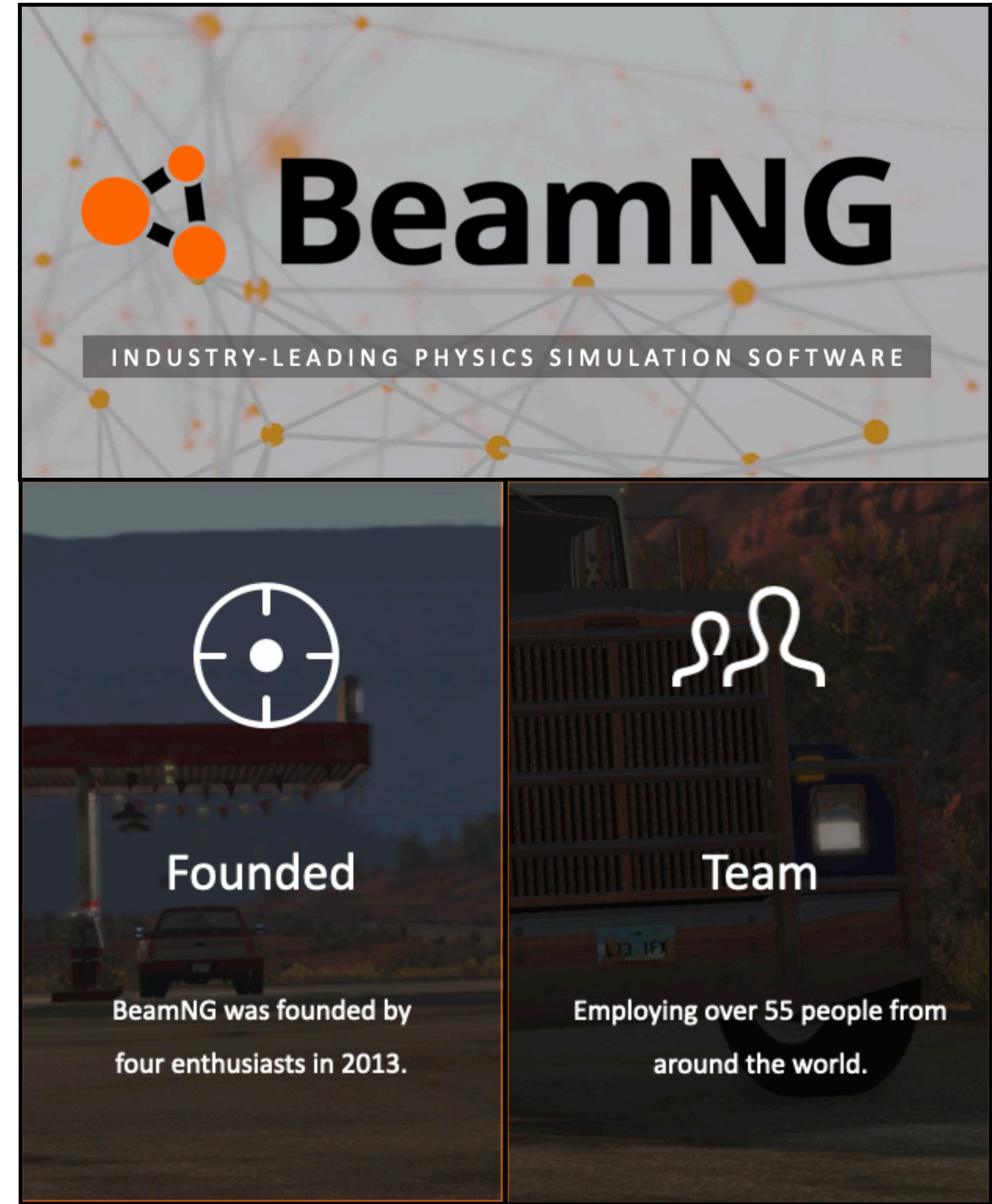
Universität des Saarlandes

Simulation-based Testing with BeamNG.tech

Simulation-based Testing with **BeamNG.tech**

What's BeamNG.tech?

- Photo-realistic, accurate, soft-body physics "engine capable of real-time simulation of vehicle dynamics and damage, all on consumer-grade hardware"
- Developed by **BeamNG GmbH**, an international tech company (based in Germany) also working in the **gaming** industry
- Specialized on driving simulations, but "can simulate everything" (e.g., mods exists for drones, pedestrians, & more)









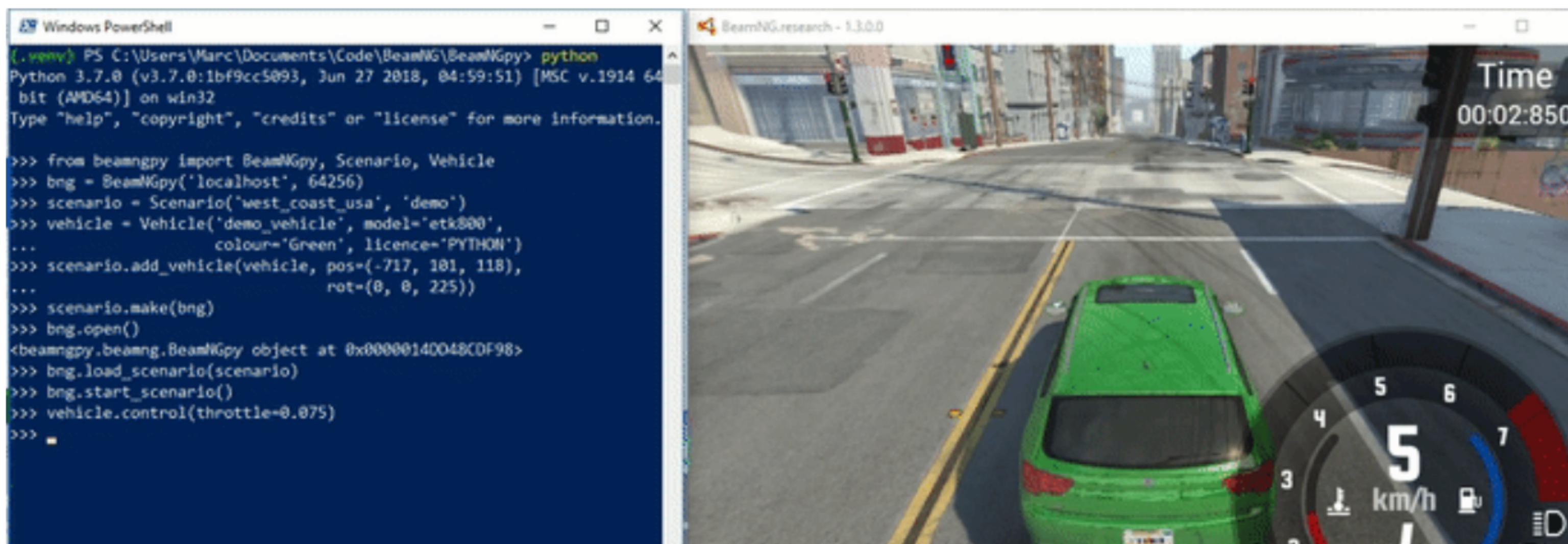
BeamNGpy

[BeamNG.research](#)[Documentation](#)

About

BeamNGpy is an official library providing a Python interface to BeamNG.research, the research-oriented fork of the video game BeamNG.drive.

It allows remote control of the simulation, including vehicles contained in it:



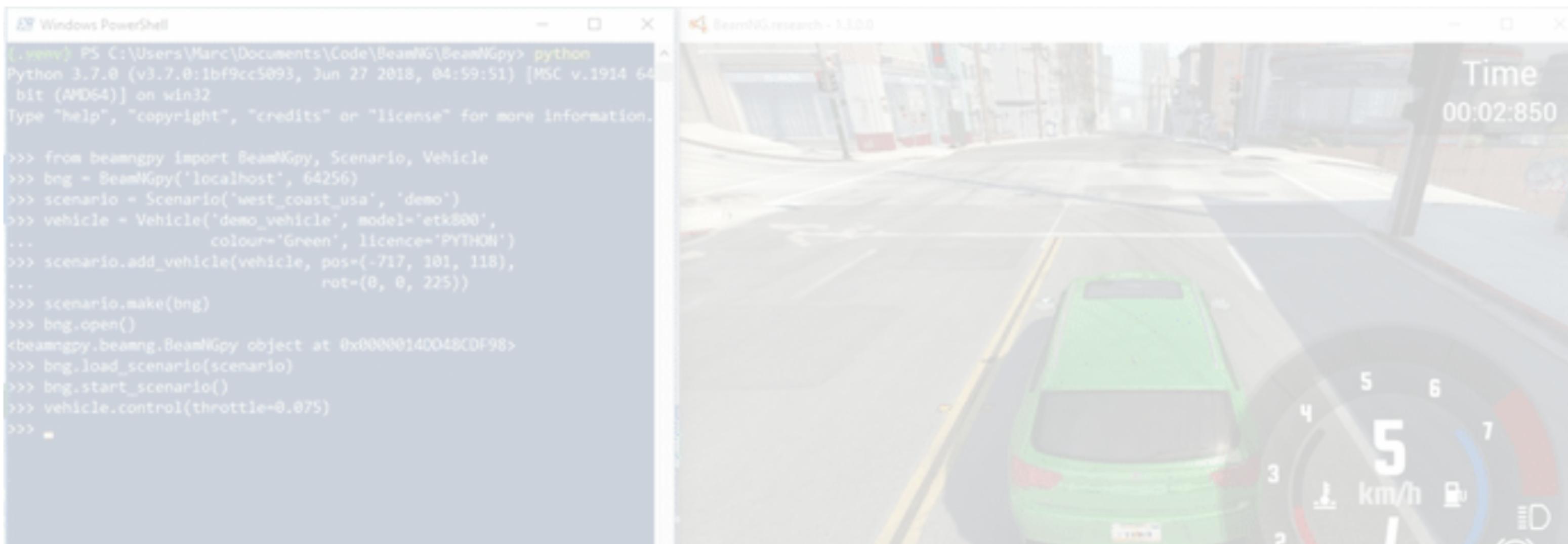
[README.md](#)github.com/BeamNG/BeamNGpy

BeamNGpy

[BeamNG.research](#)[Documentation](#)

BeamNGpy is an official library providing a Python interface to BeamNG.research, the research-oriented fork of the video game BeamNG.drive.

It allows remote control of the simulation, including vehicles contained in it:



```
Windows PowerShell
PS C:\Users\Marc\Documents\Code\BeamNG\BeamNGpy> python
Python 3.7.0 (v3.7.0:bf9cc5093, Jun 27 2018, 04:59:51) [MSC v.1914 64
bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.

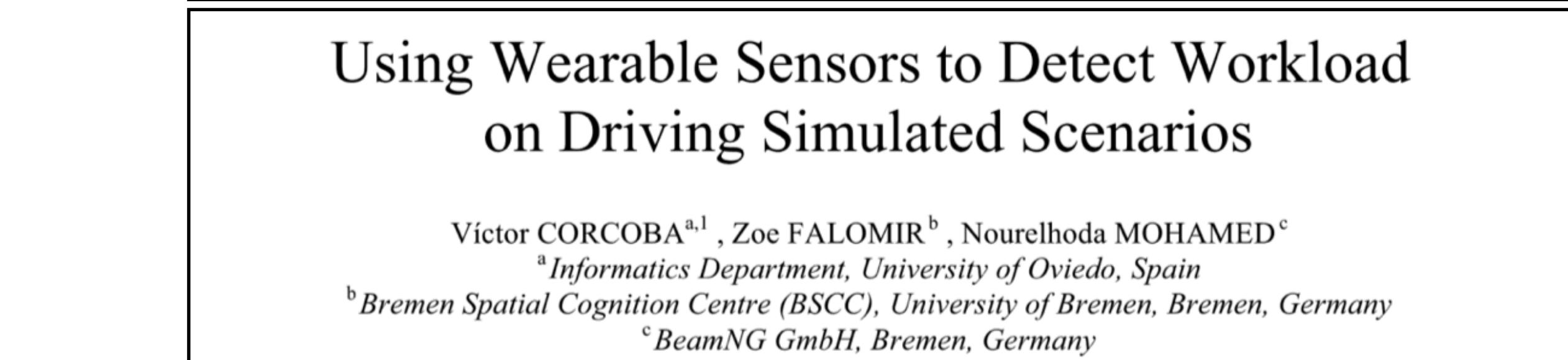
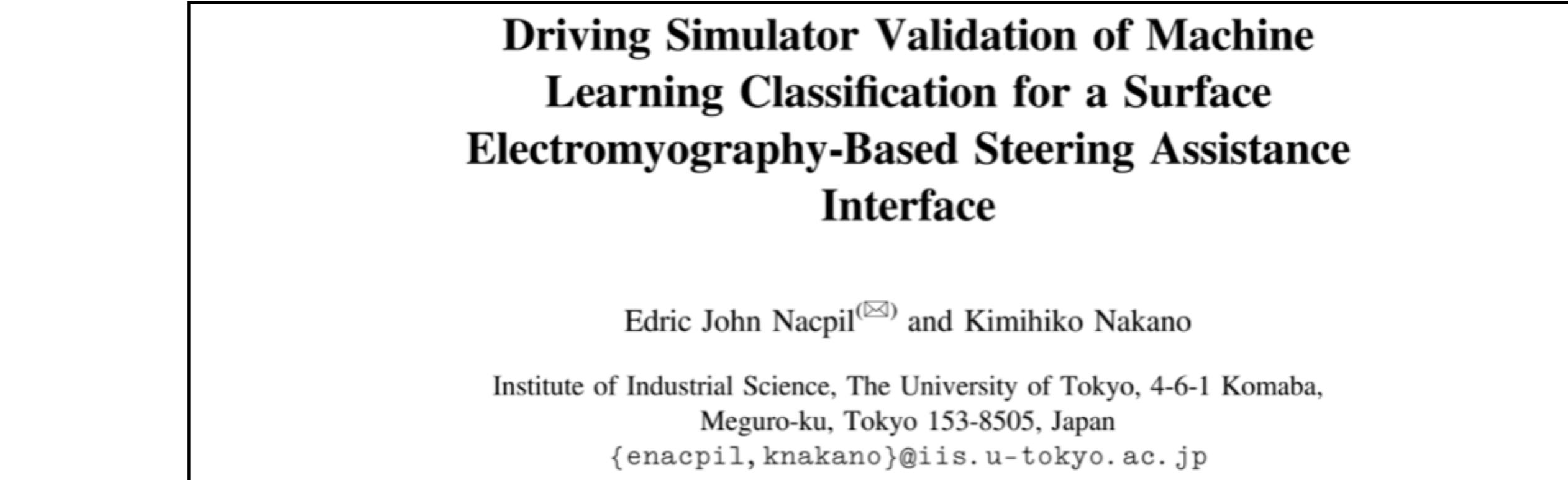
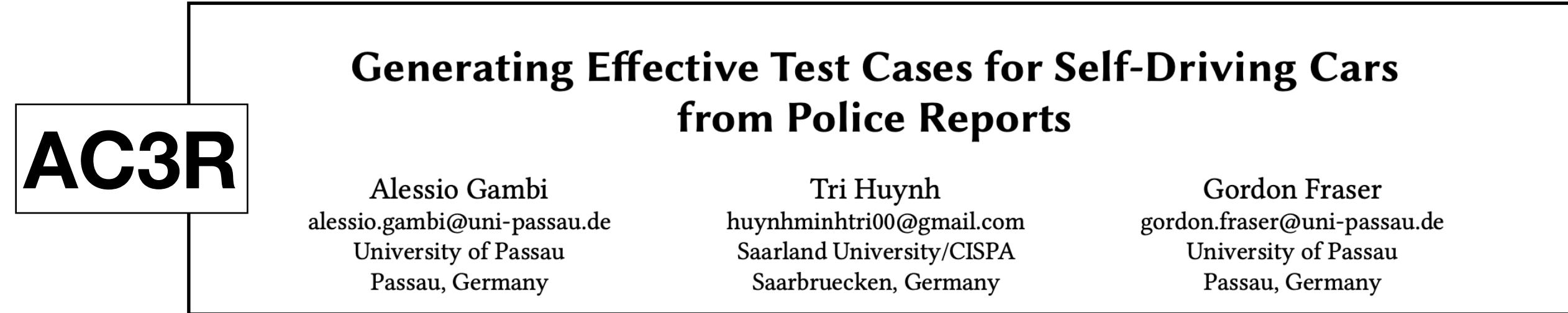
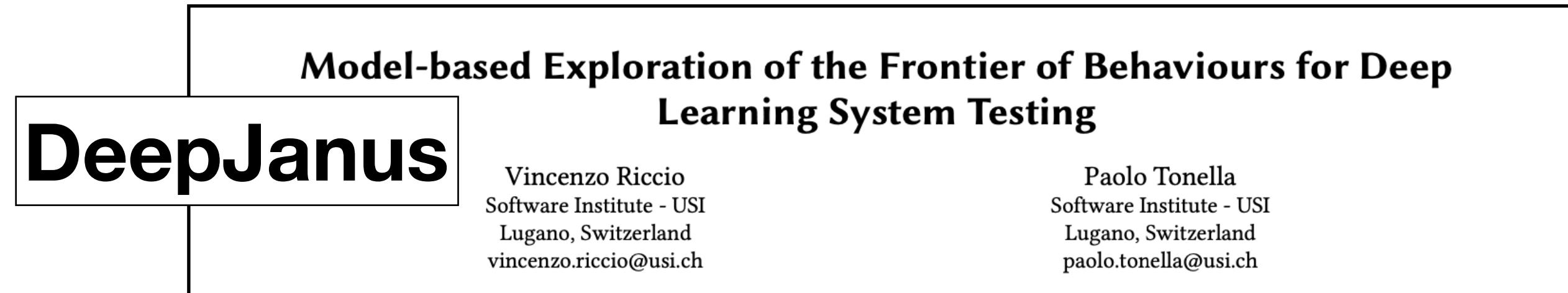
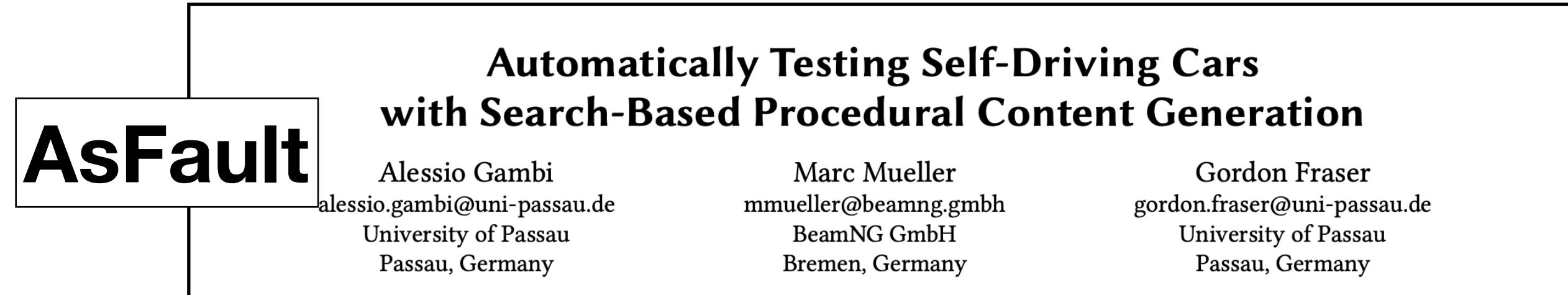
>>> from beamngpy import BeamNGpy, Scenario, Vehicle
>>> bng = BeamNGpy('localhost', 64256)
>>> scenario = Scenario('west_coast_usa', 'demo')
>>> vehicle = Vehicle('demo_vehicle', model='etk800',
...                     colour='Green', licence='PYTHON')
>>> scenario.add_vehicle(vehicle, pos=(-717, 101, 118),
...                       rot=(0, 0, 225))
>>> scenario.make(bng)
>>> bng.open()
<beamngpy.beamng.BeamNGpy object at 0x00000140048CDF98>
>>> bng.load_scenario(scenario)
>>> bng.start_scenario()
>>> vehicle.control(throttle=0.075)
>>> =
```

Who uses it?

- Researchers:
 - Papers in major SE/ST conferences
 - Several Tools for SBST
- Students:
 - More than 20+ Master and Bachelor theses (UniPassau, USI, ZHAW)
 - Advanced seminars

Who uses it?

- Researchers:
 - Papers in major SE/ST conferences
 - Several Tools for SBST
- Students:
 - More than 20+ Master and Bachelor theses (UniPassau, USI, ZHAW)
 - Advanced seminars



Who uses it?

- Researchers:
 - Papers in major SE/ST conferences
 - Several Tools for SBST
- Students:
 - More than 20+ Master and Bachelor theses (UniPassau, USI, ZHAW)
 - Advanced seminars

DEEPHYPERION: Exploring the Feature Space of Deep Learning-Based Systems through Illumination Search

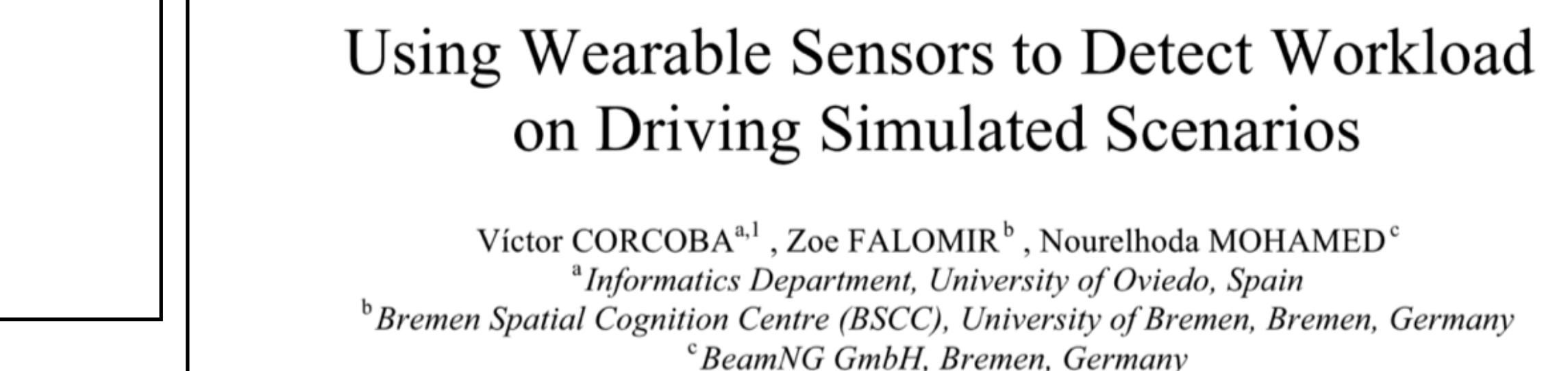
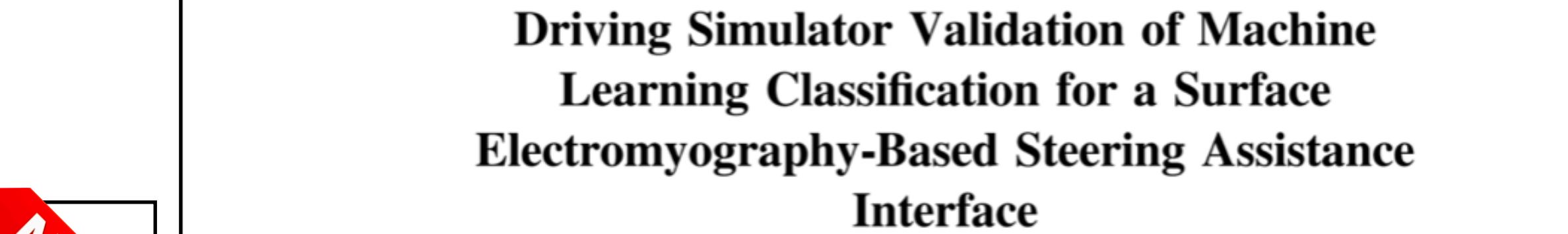
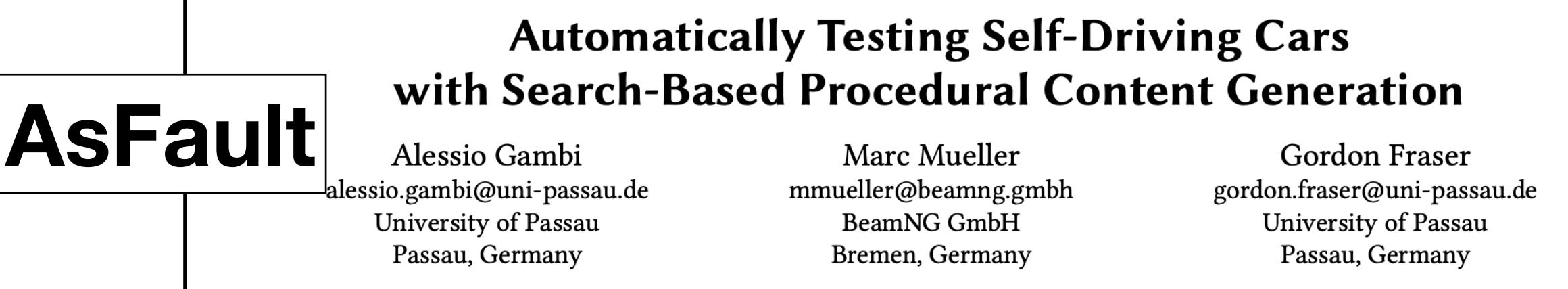
Tahereh Zohdinasab
Università della Svizzera Italiana
Lugano, Switzerland
tahereh.zohdinasab@usi.ch

Alessio Gambi
University of Passau
Passau, Germany
alessio.gambi@uni-passau.de

Vincenzo Riccio
Università della Svizzera Italiana
Lugano, Switzerland
vincenzo.riccio@usi.ch

Paolo Tonella
Università della Svizzera Italiana
Lugano, Switzerland
paolo.tonella@usi.ch

NEW



Who uses it?

SBST Tool Competition 2021

Sebastiano Panichella*, Alessio Gambi†, Fiorella Zampetti‡ and Vincenzo Riccio§

*Zurich University of Applied Science (ZHAW), Zurich, Switzerland

Email: panc@zhaw.ch

†University of Passau, Passau, Germany

Email: alessio.gambi@uni-passau.de

‡University of Sannio, Benevento, Italy

Email: fiorella.zampetti@unisannio.it

§Software Institute - USI, Lugano, Switzerland

Email: vincenzo.riccio@usi.ch

Who uses it?

SBST Tool Competition 2021

Sebastiano Panichella*, Alessio Gambi†, Fiorella Zampetti‡ and Vincenzo Riccio§

*Zurich University of Applied Science (ZHAW), Zurich, Switzerland

Email: panc@zhaw.ch

†University of Passau, Passau, Germany

Email: alessio.gambi@uni-passau.de

‡University of Sannio, Benevento, Italy

Email: fiorella.zampetti@unisannio.it

§Software Institute - USI, Lugano, Switzerland

Email: vincenzo.riccio@usi.ch

After today, maybe also you will use it
Software Institute USI - Lugano, Switzerland
vincenzo.riccio@usi.ch

After today, maybe also you will use it

Goals of this Tutorial

It's a *Beginners* tutorial

- Part I: Get a grip on BeamNG.tech
- Part II: Basics of simulation-based tests
- Part III: Examples of automated test generation

Goals of this Tutorial

It's a *Beginners* tutorial

- Part I: Get a grip on BeamNG.tech
- Part II: Basics of simulation-based tests
- Part III: Examples of automated test generation

Disclaimer: We show only a fraction of the possibilities enabled by BeamNG.tech. The simulator is designed to be extensible and most of its code is **open source**.

Part I: Getting Started

Resources

<https://github.com/se2p/sbst-2021-tutorial>

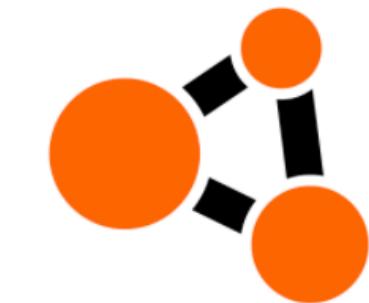
- Setup instructions
- Code examples
- Copy of the slides
- Links to additional resources



Obtaining the Simulator

Getting Started

- The simulator can be obtained **for free** after registering at <https://register.beamng.tech/>
- Use a valid **university** email address for the registration



BeamNG.tech

BeamNG.tech software is a great solution for everyone who uses simulation in an individual and need-based manner to be used across industry especially suitable for the automotive sector. With BeamNG.tech you have direct access to a wide range of features which are exclusively available in this package.

BeamNG.tech software for commercial purposes is subject to annual maintenance and support fees. For more information, please [contact us](#).

Academic and non-commercial licenses are provided free of charge at our discretion. The application process requires filling out the following information including your name and email address. **Please use your professional/academic email address as proof of membership.** Upon submitting the application request you will receive an email notifying you about us having received it, and an email containing our response within 5 business days. If approved, the email will include download and installation instructions.

Name:

Enter your full name

Email Address:

Enter email

Application Text:

Obtaining the Simulator

Getting Started

- The simulator can be obtained **for free** after registering at <https://register.beamng.tech/>
- Use a valid **university** email address for the registration

Obtaining the Simulator

Getting Started

- The simulator can be obtained **for free** after registering at <https://register.beamng.tech/>
- Use a valid **university** email address for the registration
- After registration you should get a confirmation email (check the spam folder!) with a `tech.key` file and the link to download the simulator.
- In this tutorial, we use the **BeamNG.tech version v0.21.3.0**

Installing the Simulator

Getting Started

- The simulator comes as *tar-ball* and does **not** require any specific installation, just extract it somewhere it can fit (~16.5 GB)
- **Avoid special characters and spaces in paths**
- We will refer to this folder as <BNG_HOME>
 - For this tutorial, we use "C:\\BeamNG.tech.v0.21.3.0"

Installing the Simulator

Getting Started

- The simulator comes as *tar-ball* and does **not** require any specific installation, just extract it somewhere it can fit (~16.5 GB)
- **Avoid special characters and spaces** in paths
- We will refer to this folder as <BNG_HOME>
 - For this tutorial, we use "C:\\BeamNG.tech.v0.21.3.0"

Officially, the simulator can run only under Windows, but you can try it also on Mac OS using Parallels (note, Parallels is **shareware**)

Installing the Simulator

Getting Started

- The simulator comes as *tar-ball* and does not require any specific installation, just extract it to your desktop.

Spoiler Alert:

- A
 - V
 - W
 - L
- Linux support will be available soon!*

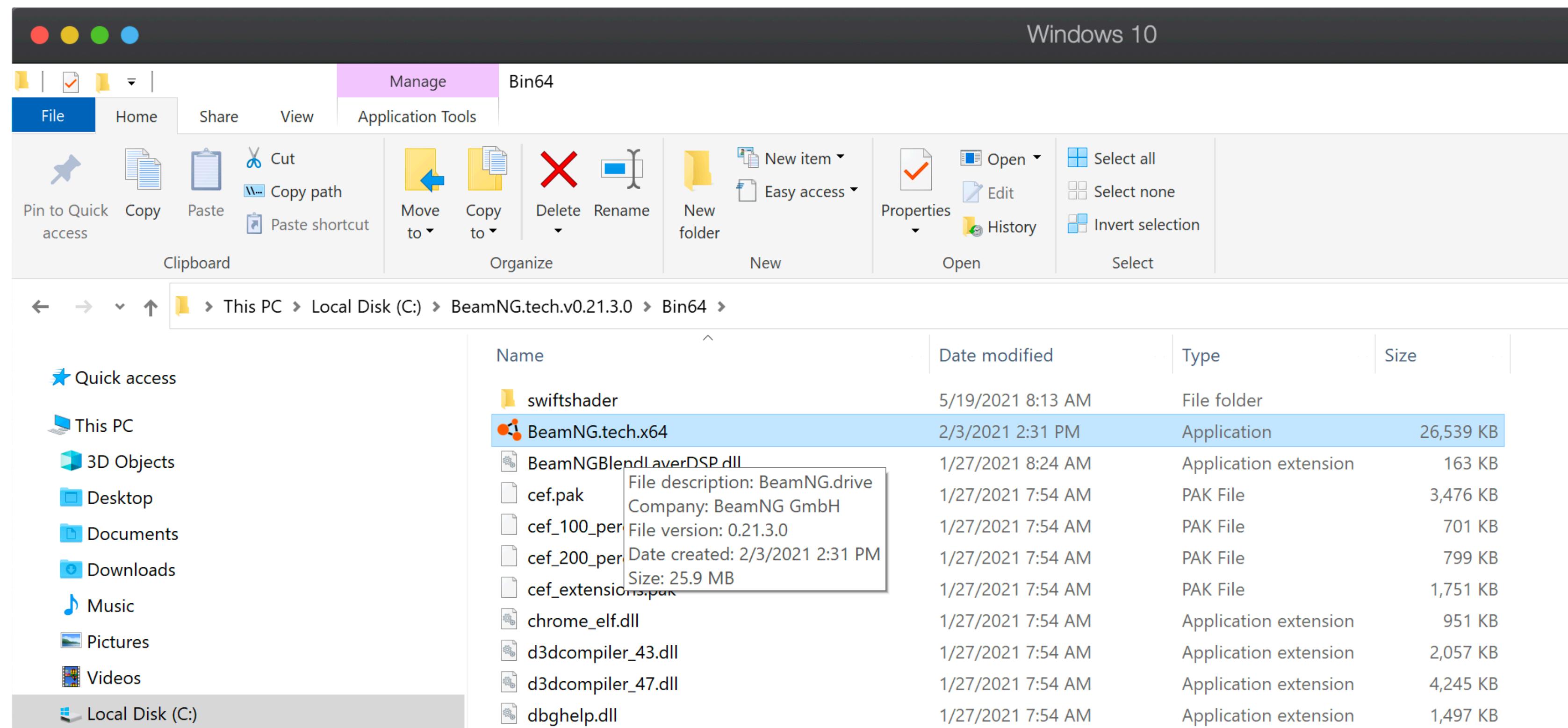
Officially, the simulator can run only under Windows, but you can try it also on Mac OS using Parallels (note, this is **shareware**)

Running the Simulator

Getting Started

- The simulator can be **started manually** by double-clicking on

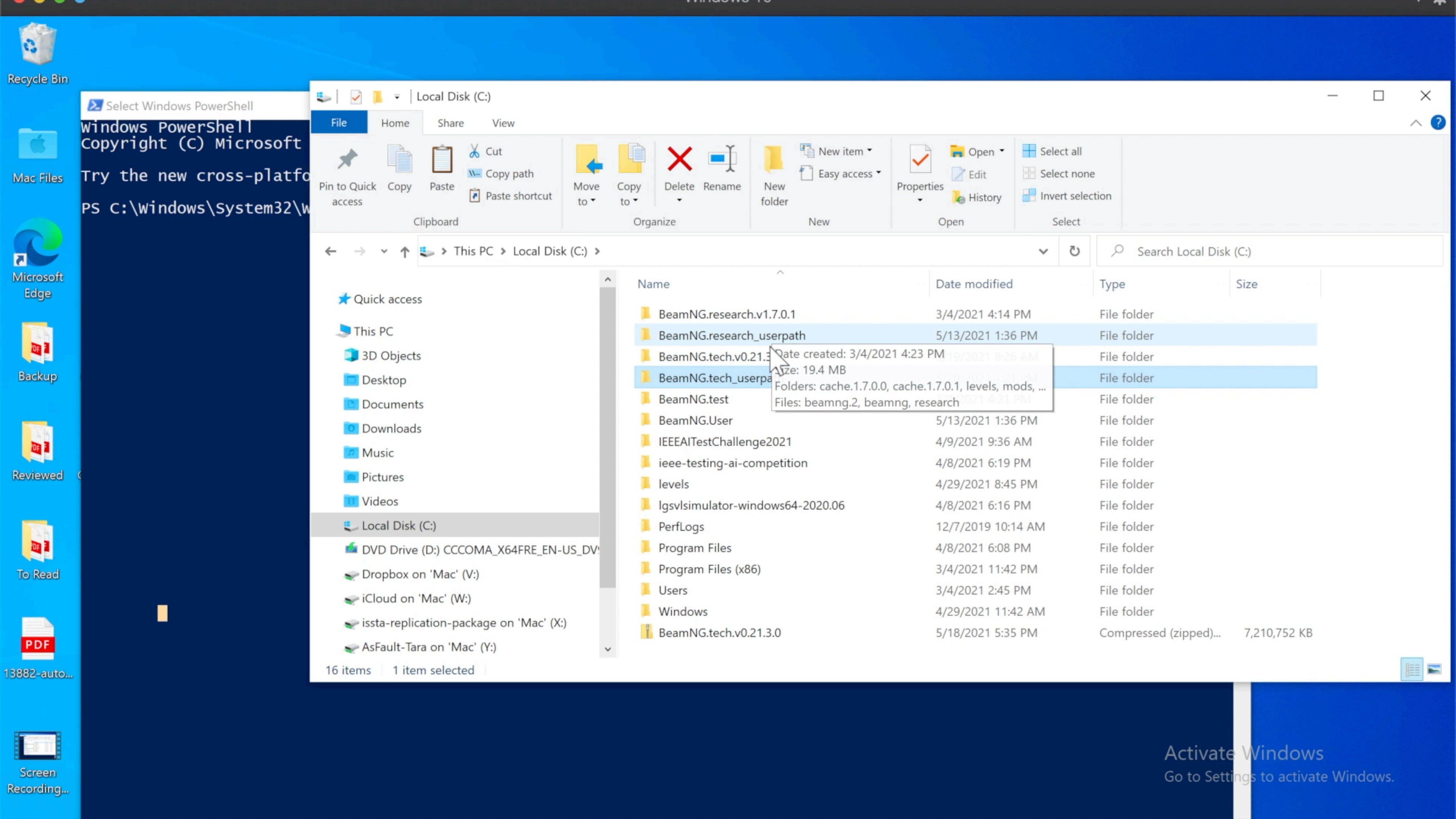
`<BNG_HOME>\Bin64\BeamNG.tech.x64.exe`

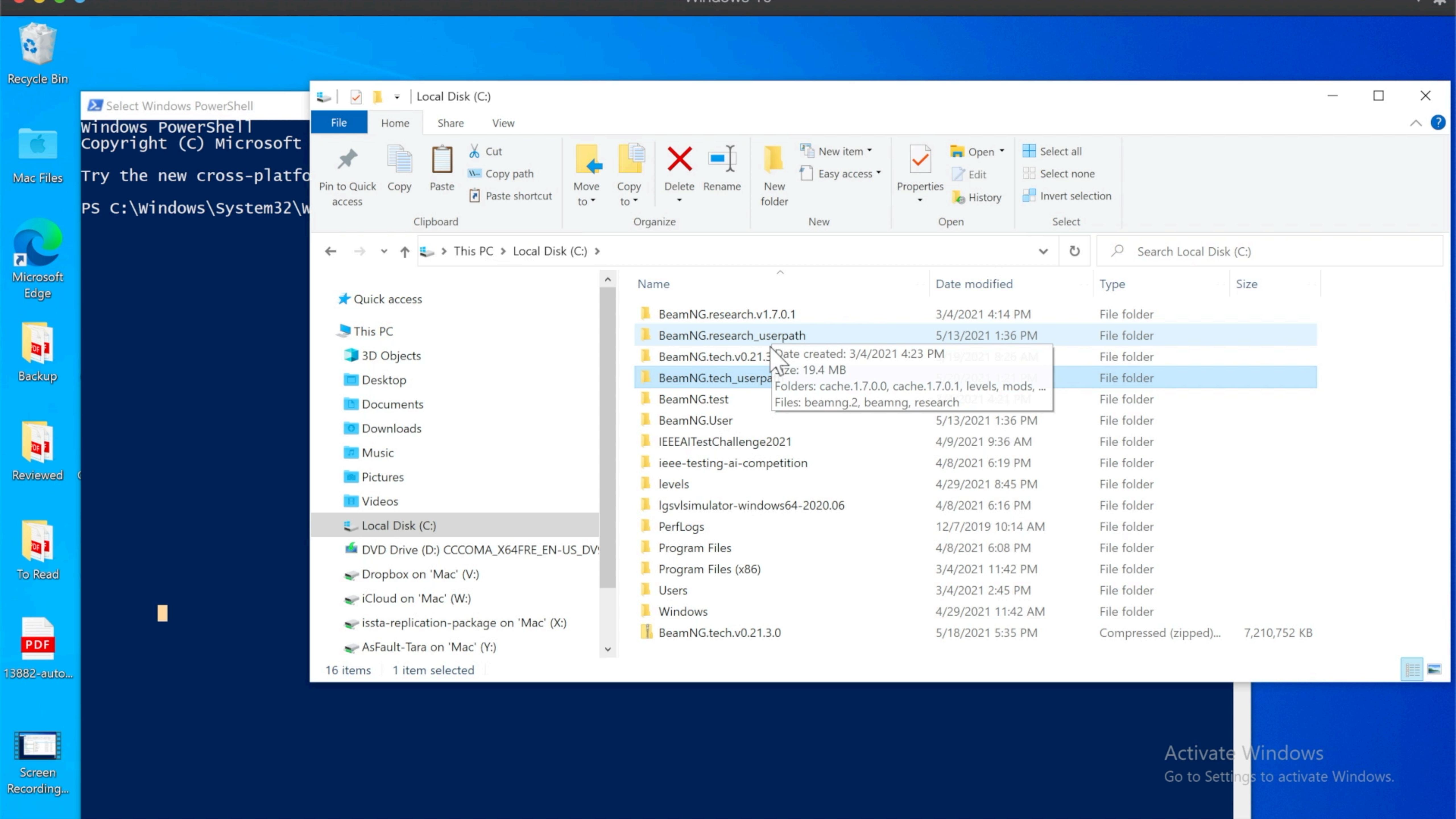


Running the Simulator

Getting Started

- The simulator can be **started manually** by double-clicking on
`<BNG_HOME>\Bin64\BeamNG.tech.x64.exe`
- Note that starting the simulator manually will **not automatically start the Python API** so you will not be able to control it remotely.
- However, you can still use the simulator to **browse existing maps** and **create new content** using the various editors





Activate Windows
Go to Settings to activate Windows.

Running the Simulator

Getting Started

- The simulator can be **started** from Powershell using the following command:

```
<BNG_HOME>\Bin64\BeamNG.tech.x64.exe -console -rport 64256  
-nosteam -physicsfps 4000 -userpath <BNG_USER>  
-lua "registerCoreModule('util/researchGE')"
```

- <BNG_USER> is the *work dir* of the simulator, you can choose any folder on your system as long as that is writable and have no spaces or special characters in its name and path
- <BNG_USER> must contain the registration key (i.e., tech.key) otherwise the simulator will **not** work

Running the Simulator

Getting Started

- The simulator can be **started** (and **controlled**) using its Python API, BeamNGpy
- BeamNGpy is available on PyPI as `beamngpy`
- Running the simulator using the Python API requires to set an env variable called `BNG_HOME` pointing to `<BNG_HOME>` or provide this value as input parameter.
- We tested the examples using Python versions 3.7 and 3.9
- `<BNG_USER>` instead is not mandatory. If not specified, BeamNGpy defaults to `~\\Documents\\BeamNG.tech_userpath`
 - Remember: **no special characters or empty space**

Installing the Dependencies

- `Code\README.md` contains the verbose descriptions on how to install the Python dependencies, please refer to that if you face issues.
- **TL;DR**
 - create a new virtual environment: `py.exe -m venv .venv`
 - activate it: `.\.venv\Scripts\activate`
 - update pip: `py.exe -m pip install --upgrade pip`
 - update utilities: `pip install --upgrade setuptools wheel`
 - Install the library: `pip install beamngpy==1.19.1`

Running the Simulator

Assumption: BeamNGpy correctly installed

```
from beamngpy import BeamNGpy, Scenario, Road

# Specify location of BeamNG home and BeamNG user folders
BNG_HOME = "C:\\BeamNG.tech.v0.21.3.0"
BNG_USER = "C:\\BeamNG.tech_userpath"

beamng = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)

# Start BeamNG by setting launch to True
bng = beamng.open(launch=True)

try:
    input('Press enter when done...')
finally:
    bng.close()
```

example1.py

Running the Simulator

Assumption: BeamNGpy correctly installed

```
from beamngpy import BeamNGpy, Scenario, Road

# Specify where BeamNG home and user are
BNG_HOME = "C:\\BeamNG.tech.v0.21.3.0"
BNG_USER = "C:\\BeamNG.tech_userpath"

# Use Python Context Manager so it automatically closes
with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER):
    input('Press enter when done...')
```

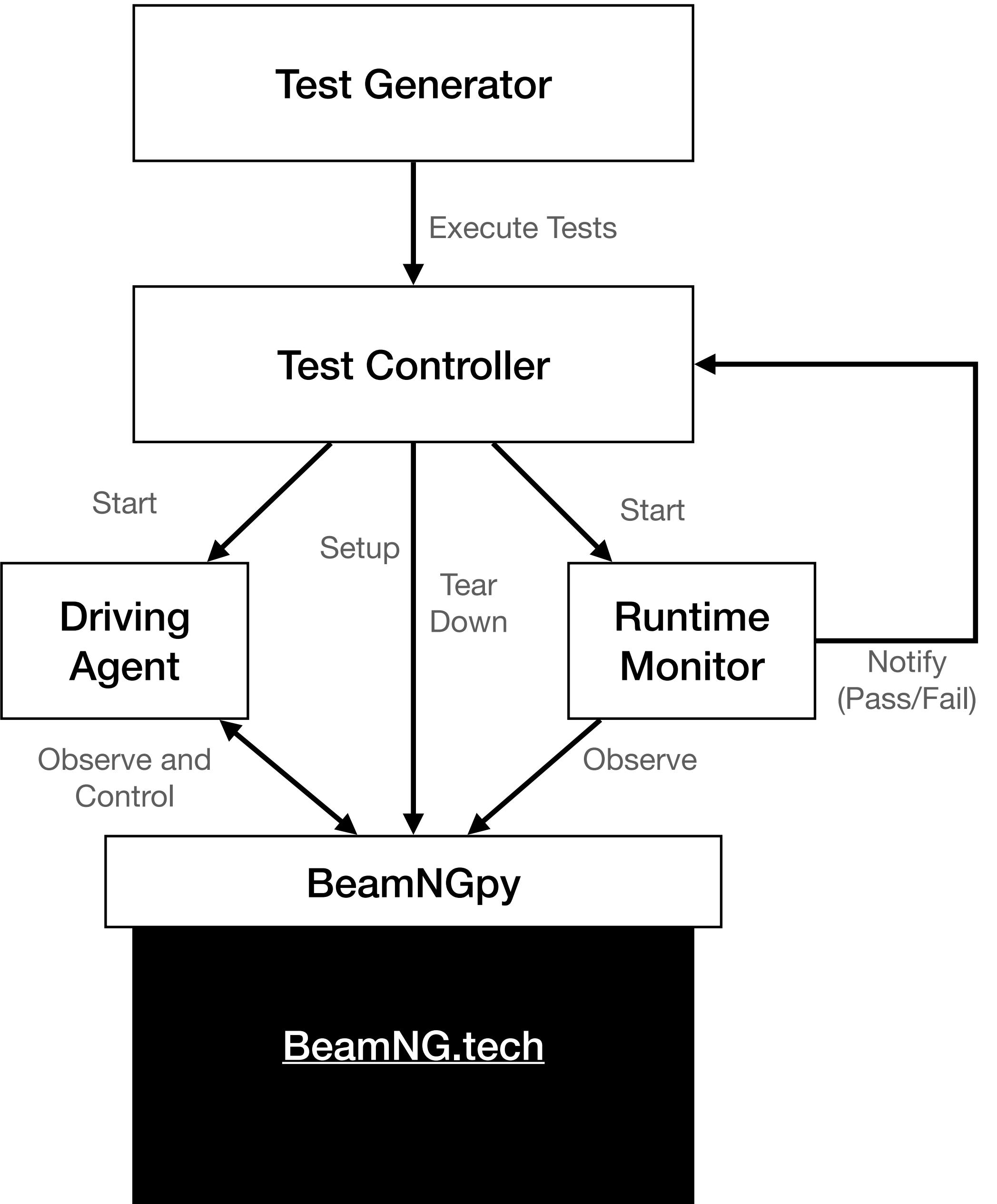
Part II

Simulation-Based Testing

Introduction

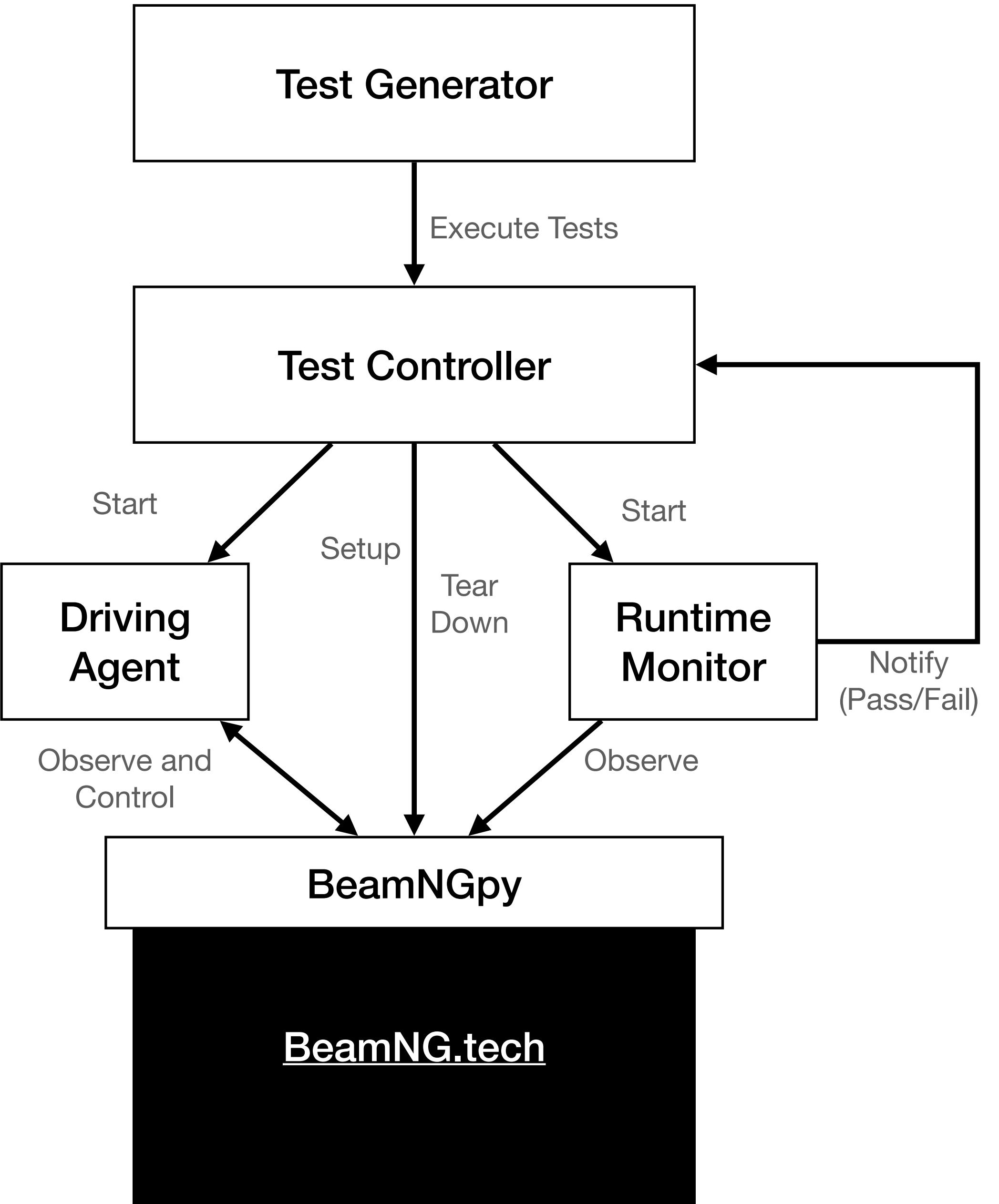
- Simulation-based testing can be used to implement **X-in-the-loop** testing
 - We focus on Software-in-the-loop (SIL)
- Test subjects are **continuous controllers** and may be based on **ML/AI**
 - Notoriously difficult to test by traditional means

Reference Architecture



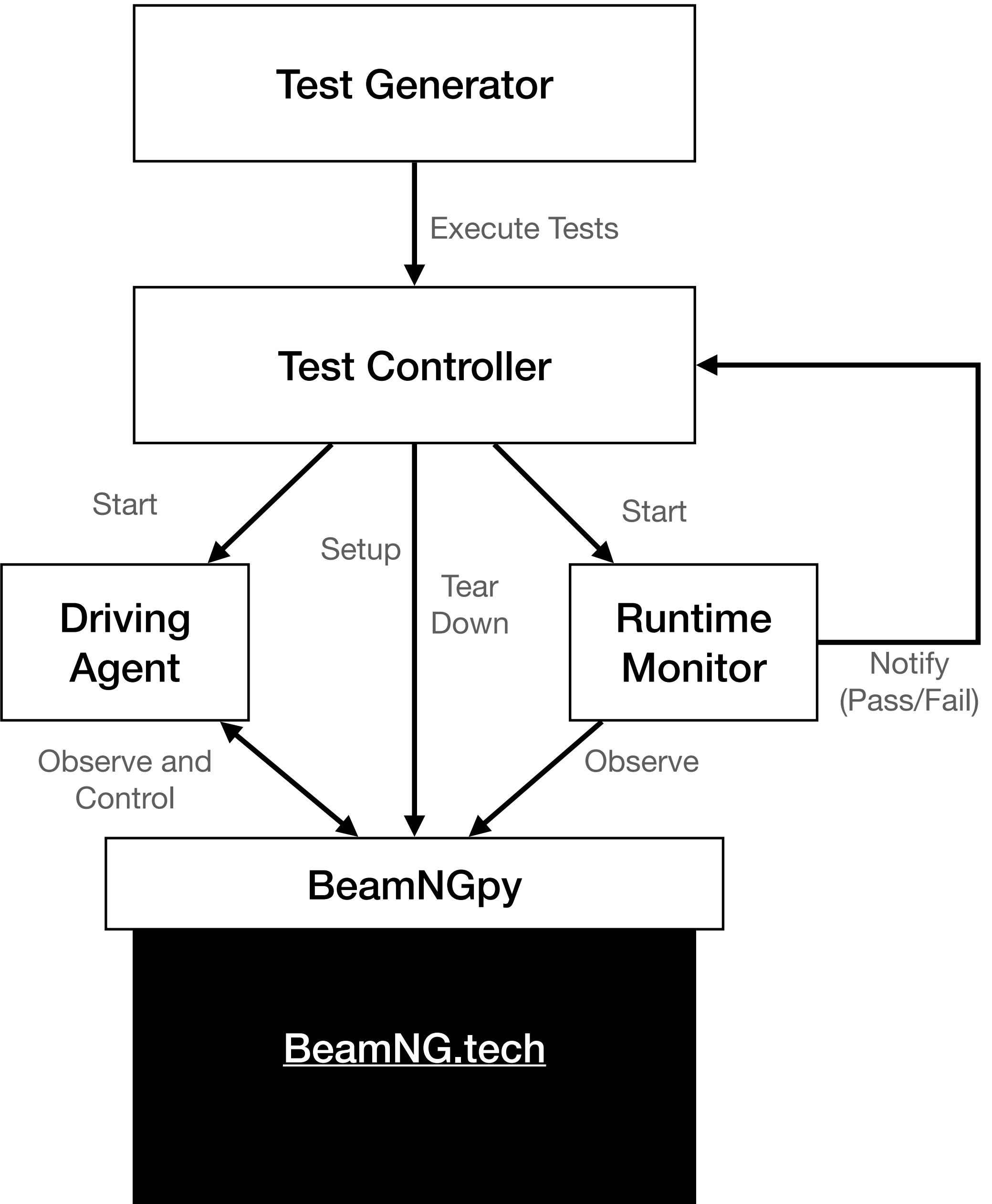
Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.



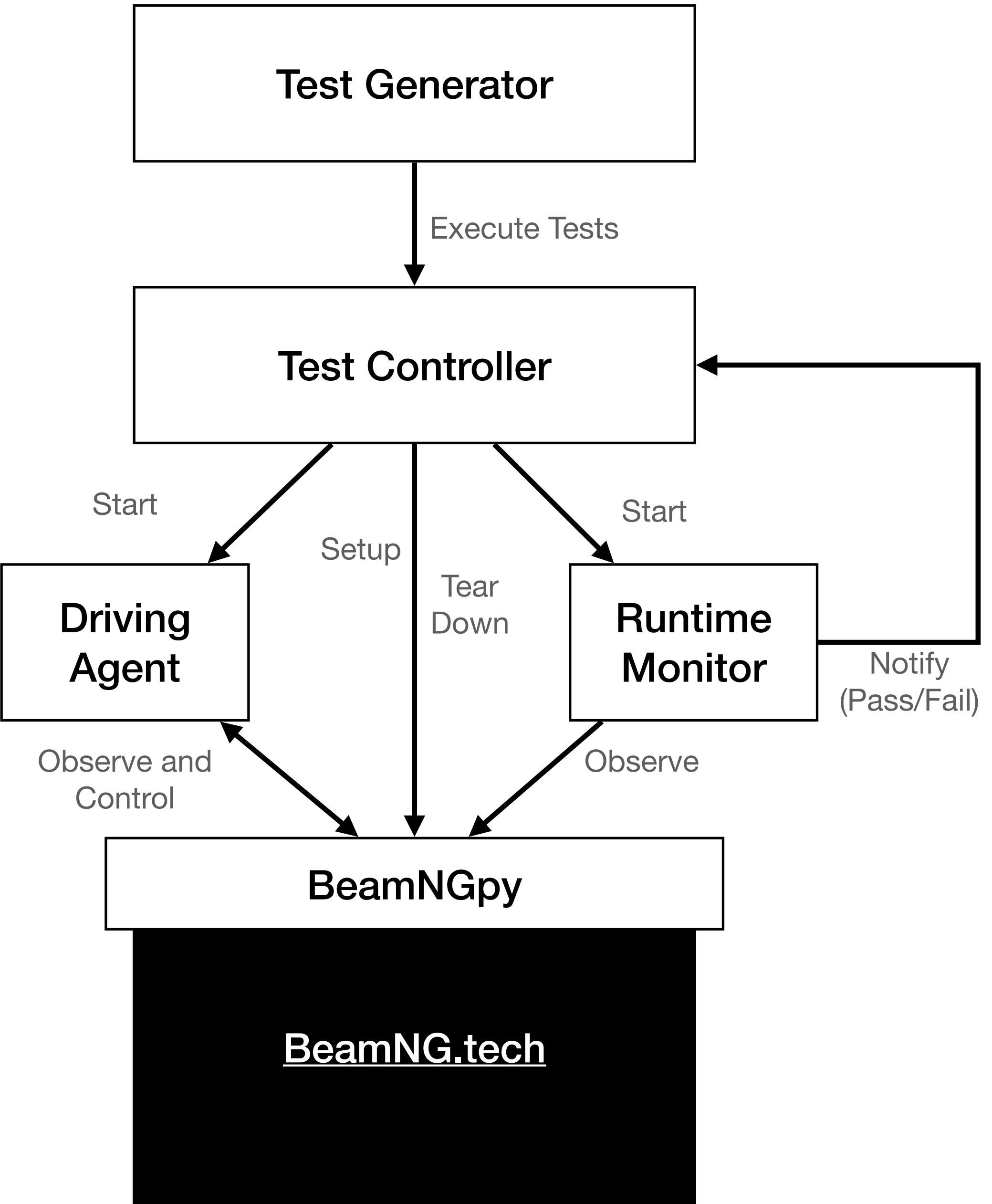
Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.
- **Test Controller** to start/stop simulation, start/stop tests (e.g., pytest, unittest)



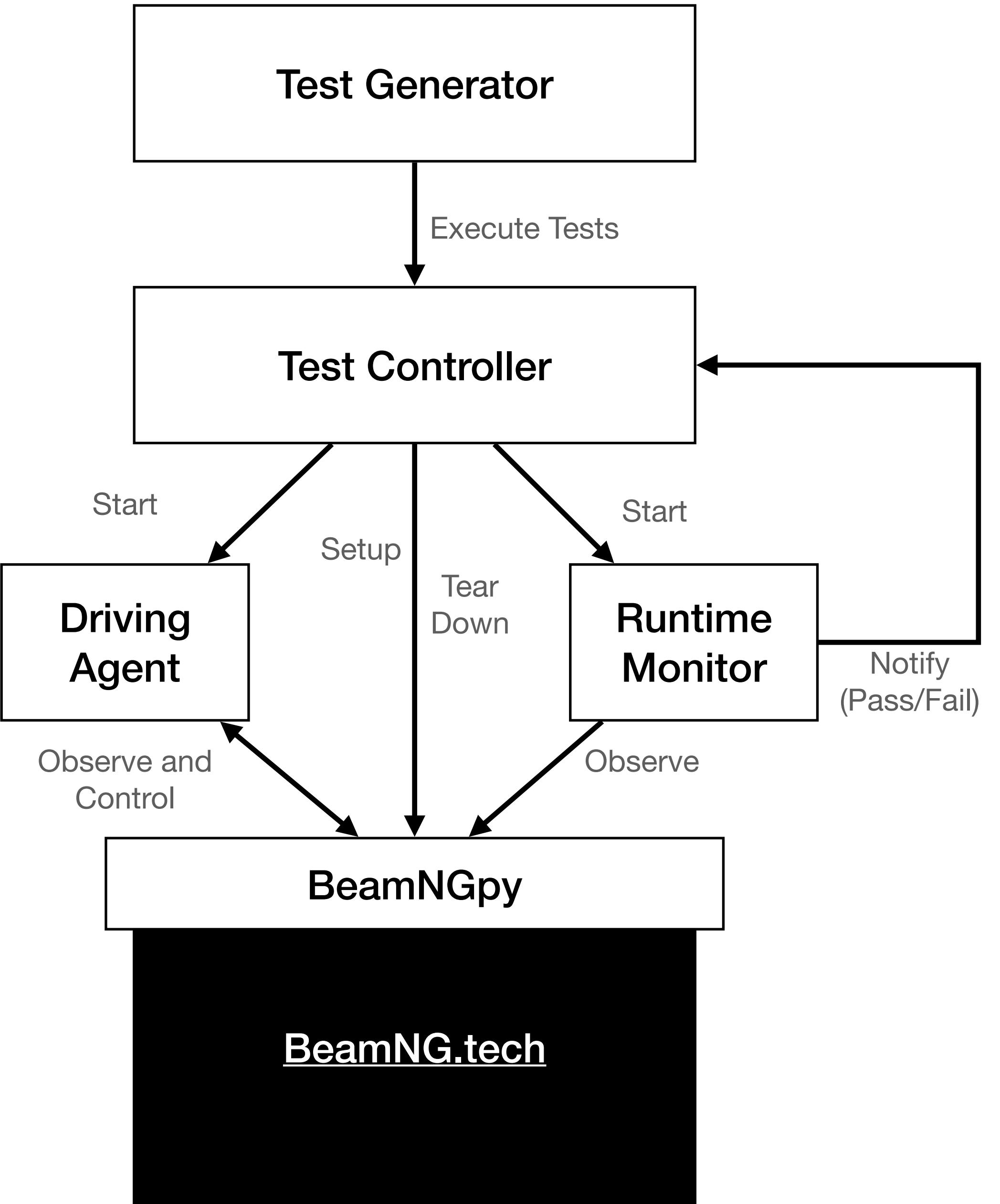
Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.
- **Test Controller** to start/stop simulation, start/stop tests (e.g., pytest, unittest)
- **BeamNGpy API** to BeamNG.tech, allow for multiple simultaneous connections.



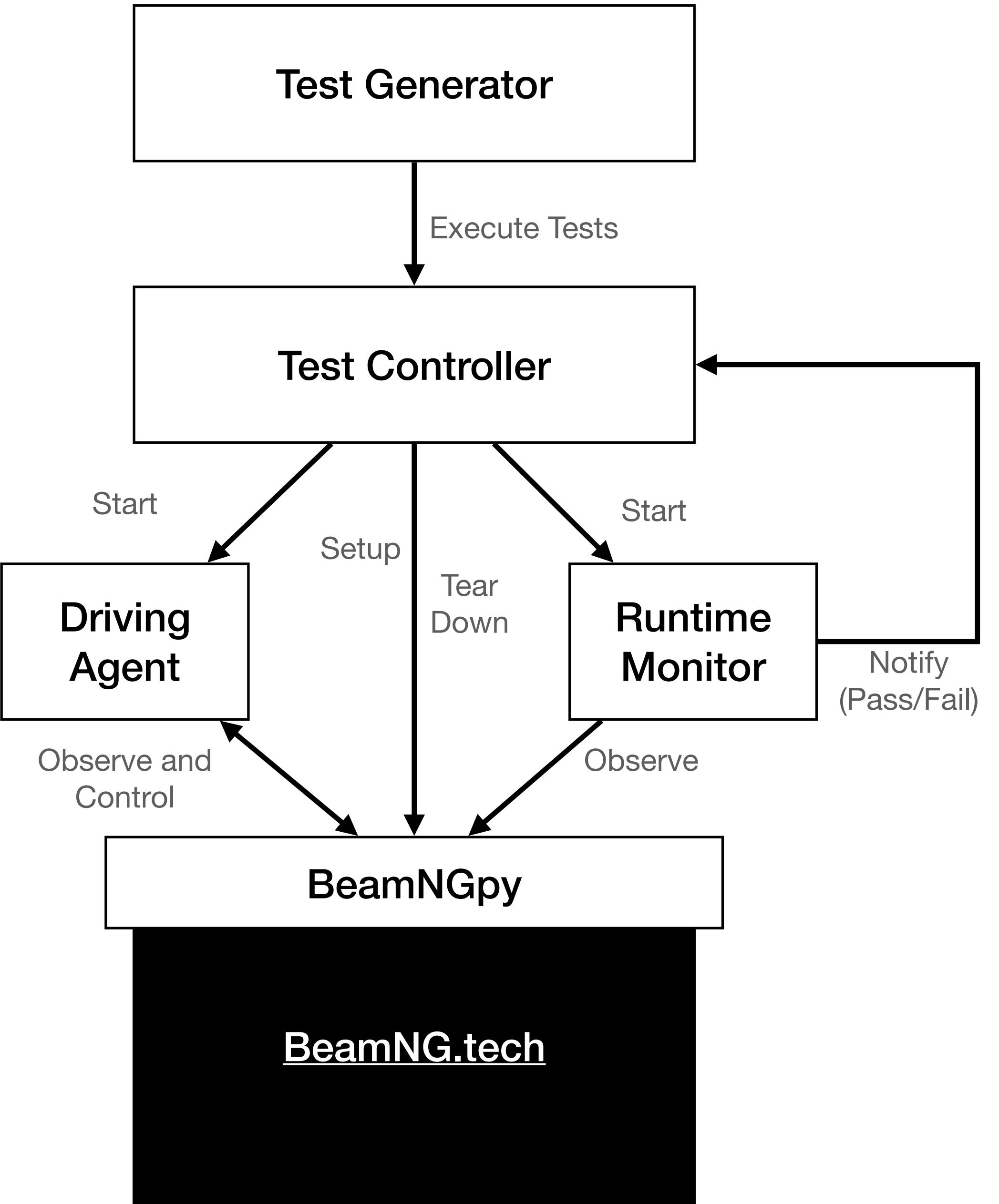
Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.
- **Test Controller** to start/stop simulation, start/stop tests (e.g., pytest, unittest)
- **BeamNGpy API** to BeamNG.tech, allow for multiple simultaneous connections.
- **BeamNG.tech** is the simulator (considered as black-box)



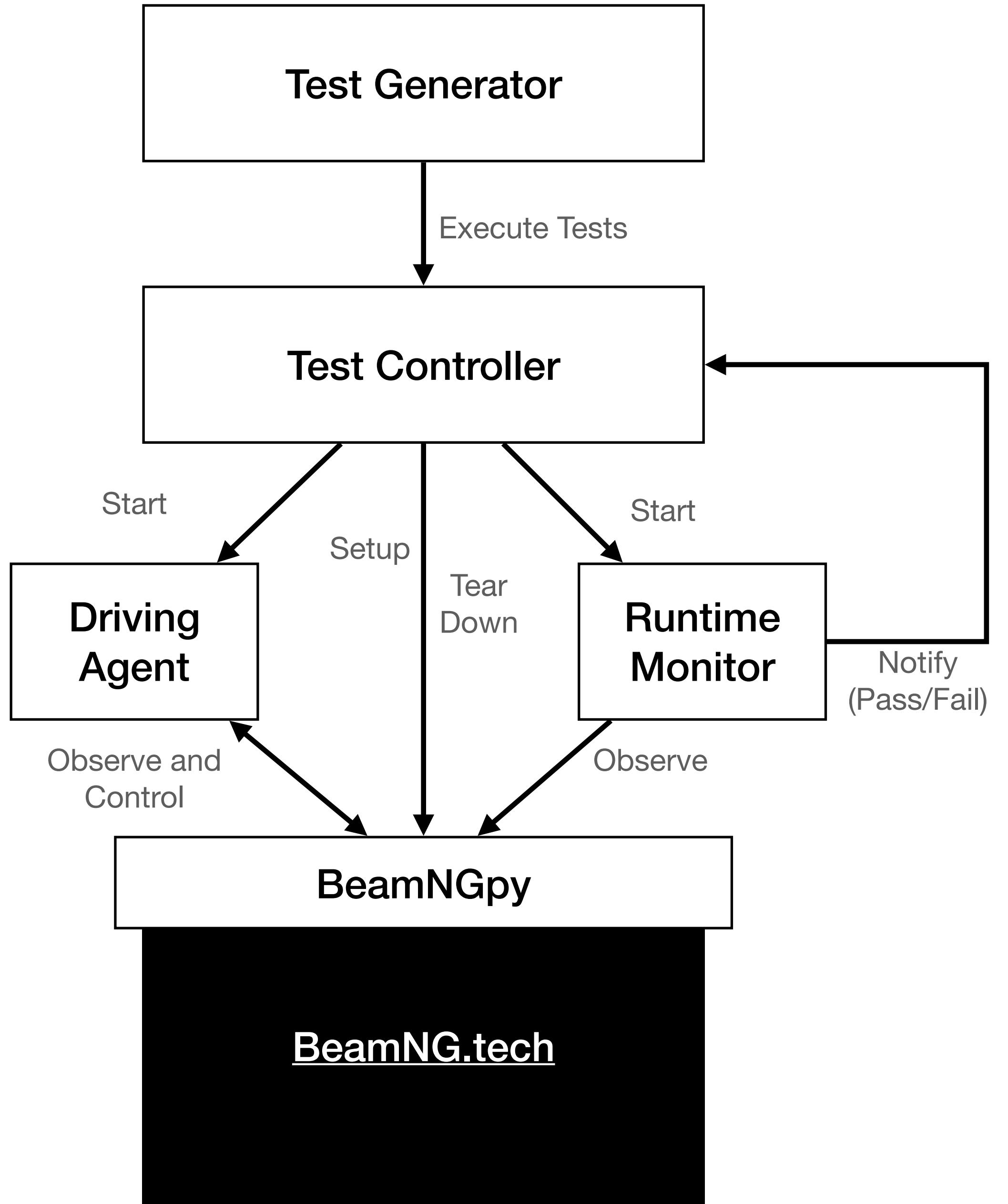
Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.
- **Test Controller** to start/stop simulation, start/stop tests (e.g., pytest, unittest)
- **BeamNGpy API** to BeamNG.tech, allow for multiple simultaneous connections.
- **BeamNG.tech** is the simulator (considered as black-box)
- **Runtime Monitor** to monitor execution and check oracles



Reference Architecture

- **Test Generator** (either human or algorithm) to generate the tests.
- **Test Controller** to start/stop simulation, start/stop tests (e.g., pytest, unittest)
- **BeamNGpy API** to BeamNG.tech, allow for multiple simultaneous connections.
- **BeamNG.tech** is the simulator (considered as black-box)
- **Runtime Monitor** to monitor execution and check oracles
- **Driving Agent** is the test subject that gets sensor data and drives the ego-car



Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

Fresh vs Shared Simulator Instances

- One can start a new instance of the simulator **for each test** or start only one instance of the simulator and **reuse it for all the tests**
- Starting a fresh instance may reduce flakyness/pollution, but it takes more time
- One can also run **multiple concurrent instances** of the simulation (not shown in this tutorial) provided enough computing resources are available for running the simulators **and** the test subjects (GPU-Intensive)

Fresh Instance

```
class StartFreshInstanceOfTheSimulator(unittest.TestCase):

    def test_that_simulation_start(self):
        with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER):
            do_something()

    def test_that_simulation_restart(self):
        with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER):
            do_something_else()
```

simple-test.py

Fresh Instance (2)

```
class StartFreshInstanceOfTheSimulatorUsingSetupAndTearDown(unittest.TestCase):

    def setUp(self):
        beamng = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)
        self.bng = beamng.open(launch=True)

    def tearDown(self):
        if self.bng:
            self.bng.close()

    def test_that_simulation_start(self):
        do_something()

    def test_that_simulation_restart(self):
        do_something_else()
```

simple-test.py

Shared Instance

simple-test.py

```
class SharedInstance(unittest.TestCase):
    beamng = None

    @classmethod
    def setUpClass(cls):
        bng = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)
        cls.beamng = bng.open(launch=True)

    @classmethod
    def tearDownClass(cls):
        if cls.beamng:
            cls.beamng.close()
```

Shared Instance

simple-test.py

```
class SharedInstance(unittest.TestCase):  
    beamng = None  
  
    @classmethod  
    def setUpClass(cls):  
        bng = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)  
        cls.beamng = bng.open(launch=True)
```

```
@classmethod  
def tearDownClass(cls):  
    if cls.beamng:  
        cls.beamng.close()
```

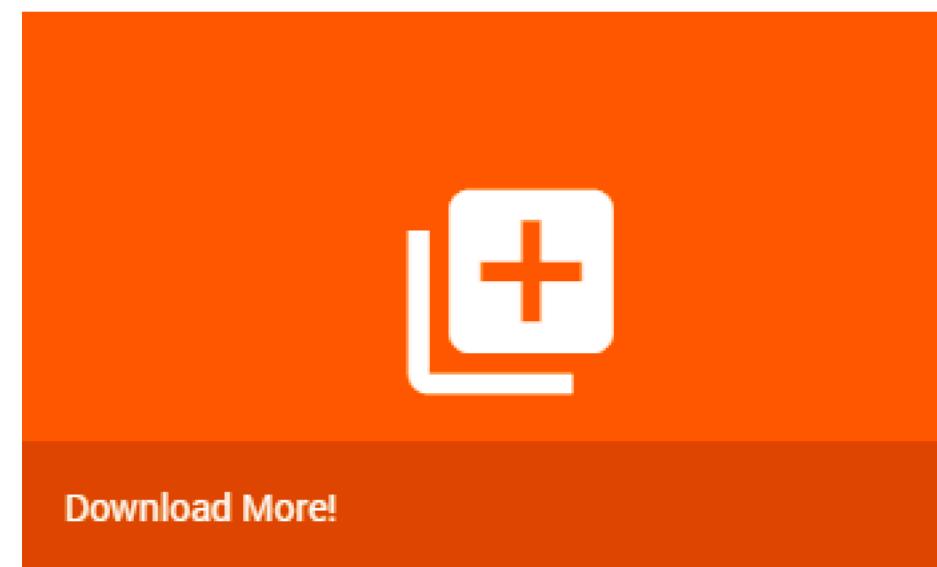
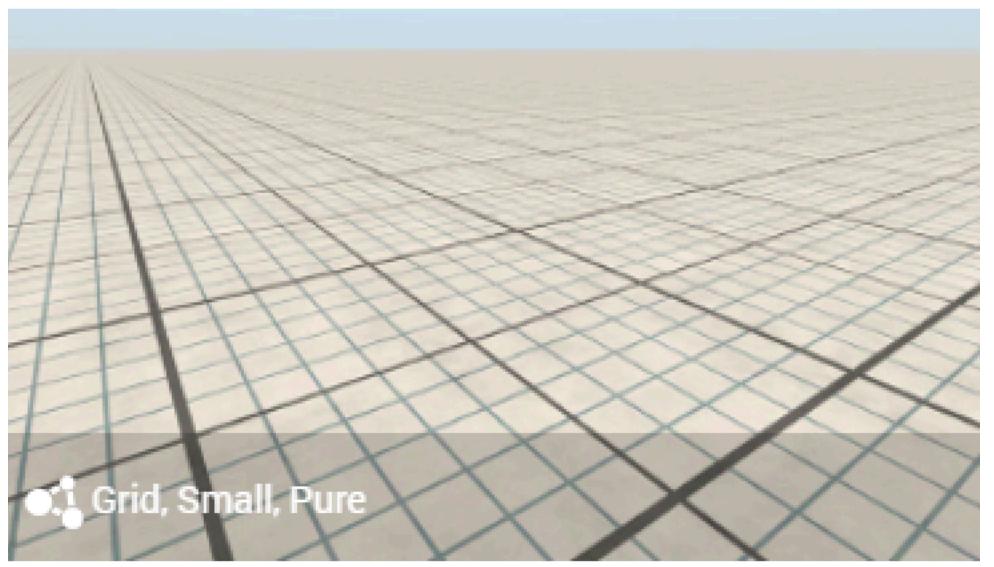
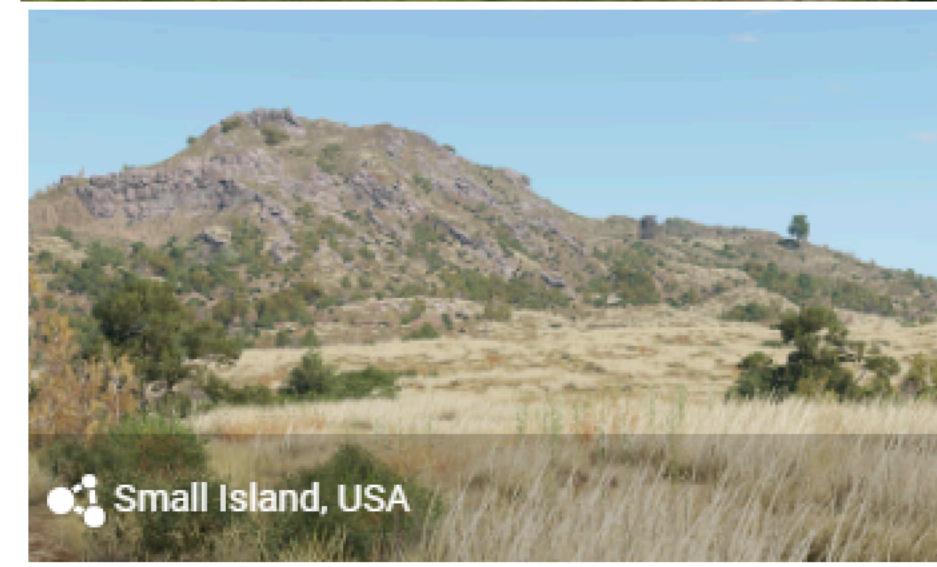
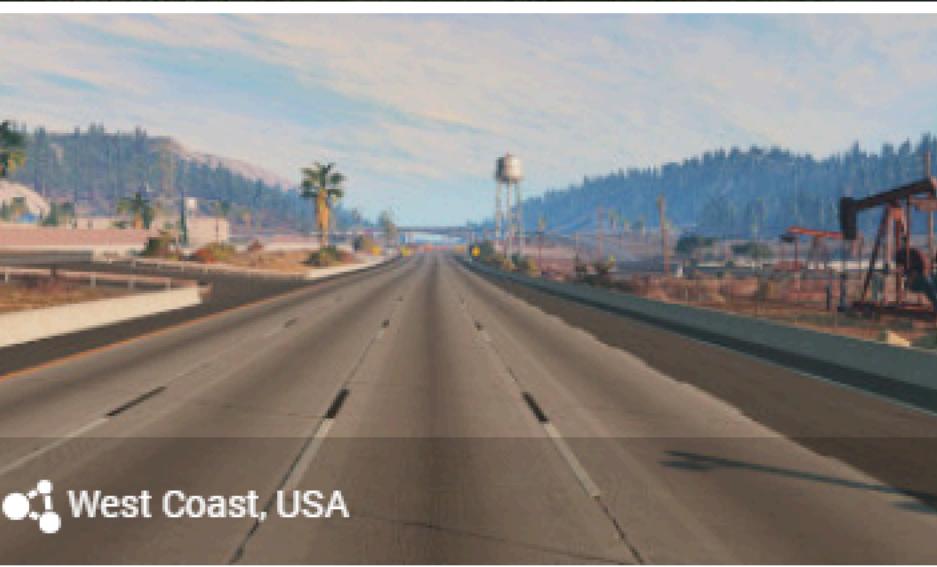
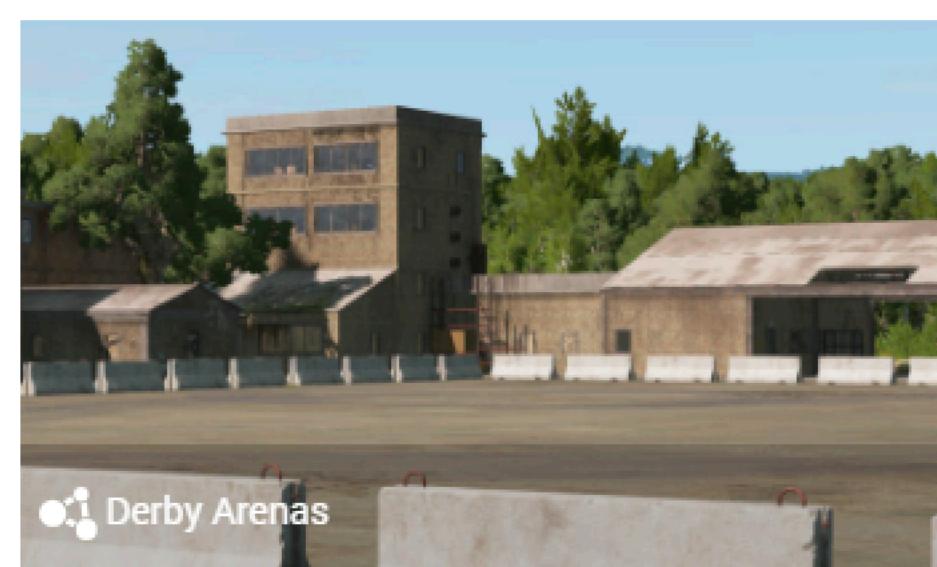
```
def test_that_can_connect_to_simulator(self):  
    client_a = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)  
    try:  
        client_a.open(launch=False, deploy=False)  
        do_something()  
    finally:  
        client_a.skt.close()  
  
def test_that_can_reconnect_to_simulator(self):  
    client_b = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)  
    try:  
        client_b.open(launch=False, deploy=False)  
        do_something_else()  
    finally:  
        client_b.skt.close()
```

Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

Levels and Scenarios

- Simulation-based tests with BeamNG.tech take place inside **scenarios**
- Scenarios are defined in the context of **levels**
- Levels contain **terrains**, **maps**, **props/arts**, **materials**, and other meta-data required to execute the simulation
- **Levels** must be **manually** defined, **scenarios** can be defined **programmatically**
- Currently, BeamNG.tech contains 19 levels (more can be downloaded)
- Levels are locate inside the `levels` folders inside `<BNG_HOME>` and `<BNG_USER>`



This PC > Local Disk (C:) > BeamNG.tech.v0.21.3.0 > levels >

Name	Date modified
automation_test_track	5/19/2021 8:14 AM
autotest	5/19/2021 8:14 AM
Cliff	5/19/2021 8:14 AM
derby	5/19/2021 8:15 AM
driver_training	5/19/2021 8:15 AM
east_coast_usa	5/21/2021 11:43 AM
garage	5/19/2021 8:16 AM
glow_city	5/19/2021 8:16 AM
GridMap	5/19/2021 8:17 AM
hirochi_raceway	5/19/2021 8:17 AM
Industrial	5/19/2021 8:18 AM
italy	5/19/2021 8:20 AM
jungle_rock_island	5/19/2021 8:21 AM
showroom_v2_white	5/19/2021 8:21 AM
small_island	5/19/2021 8:22 AM
smallgrid	5/21/2021 12:35 PM
template	5/19/2021 8:22 AM
Utah	5/19/2021 8:23 AM
west_coast_usa	5/19/2021 8:24 AM

Driving on Existing Roads

- Almost all the **existing** levels come with **maps** that include **roads**
- Maps can be **queried** using BeamNGpy but are not yet full HD Maps
- All the levels can be loaded and used "as they are" or can be customized

```
with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:  
    level_name='Utah'  
    scenario_name='ai_test'  
    scenario = Scenario(level_name, scenario_name)  
    scenario.make(bng)  
    bng.load_scenario(scenario)  
    bng.start_scenario()  
    input('Press enter when done...')
```

Procedural Road Generation

- Roads can be **procedurally generated** and added to existing scenarios
- Roads **geometry** is defined as sequences of road nodes, i.e., points (x, y, z) plus width
- Roads have **attributes**, including id, material, lane configuration, and direction

Procedural Road Generation

- Roads can be **procedurally generated** and added to existing scenarios
- Roads **geometry** is defined as sequences of road nodes, i.e., points (x, y, z) plus width
- Roads have **attributes**, including id, material, lane configuration, and direction

```
road_nodes = [(0, 30, 0, 8), (20, 30, 0, 8), (40, 30, 0, 8), (60, 30, 0, 8)]  
# TIG level courtesy of Precrime group, Lugano  
scenario = Scenario('tig', 'test_scenario_1')  
road = Road('road_rubber_sticky', rid='road_1')  
road.nodes.extend(road_nodes)
```

```
scenario.add_road(road)  
scenario.make(bng)  
bng.load_scenario(scenario)  
bng.start_scenario()
```

load_scenario_and_roads.py

Procedural Road Generation

- Roads can be **procedurally generated** and added to existing scenarios
- Roads **geometry** is defined as sequences of road nodes, i.e., points (x, y, z) plus width
- Roads have **attributes**, including id, material, lane configuration, and direction

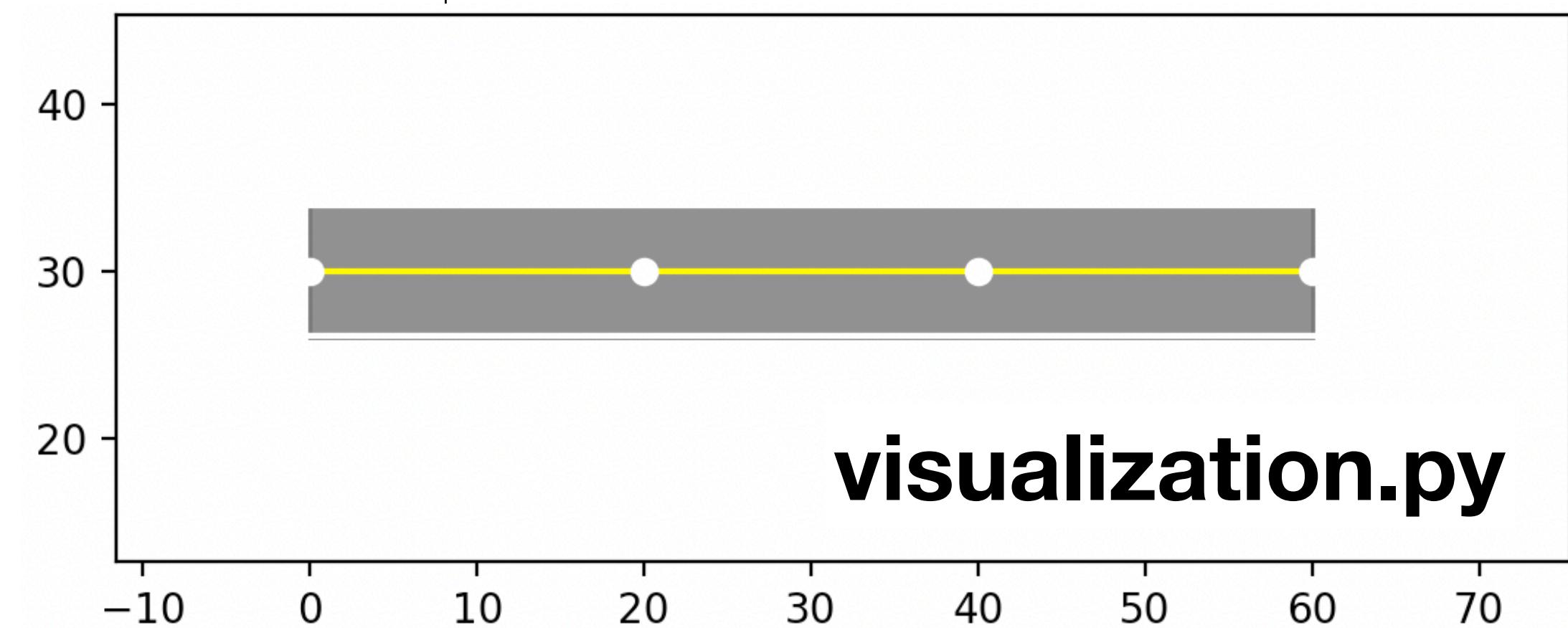
```
road_nodes = [(0, 30, 0, 8), (20, 30, 0, 8), (40, 30, 0, 8), (60, 30, 0, 8)]  
# TIG level courtesy of Precrime group, Lugano  
scenario = Scenario('tig', 'test_scenario_1')  
road = Road('road_rubber_sticky', rid='road_1')  
road.nodes.extend(road_nodes)
```

```
scenario.add_road(road)
```

```
scenario.make(bng)
```

```
bng.load_scenario(scenario)
```

```
bng.start_scenario()
```



visualization.py

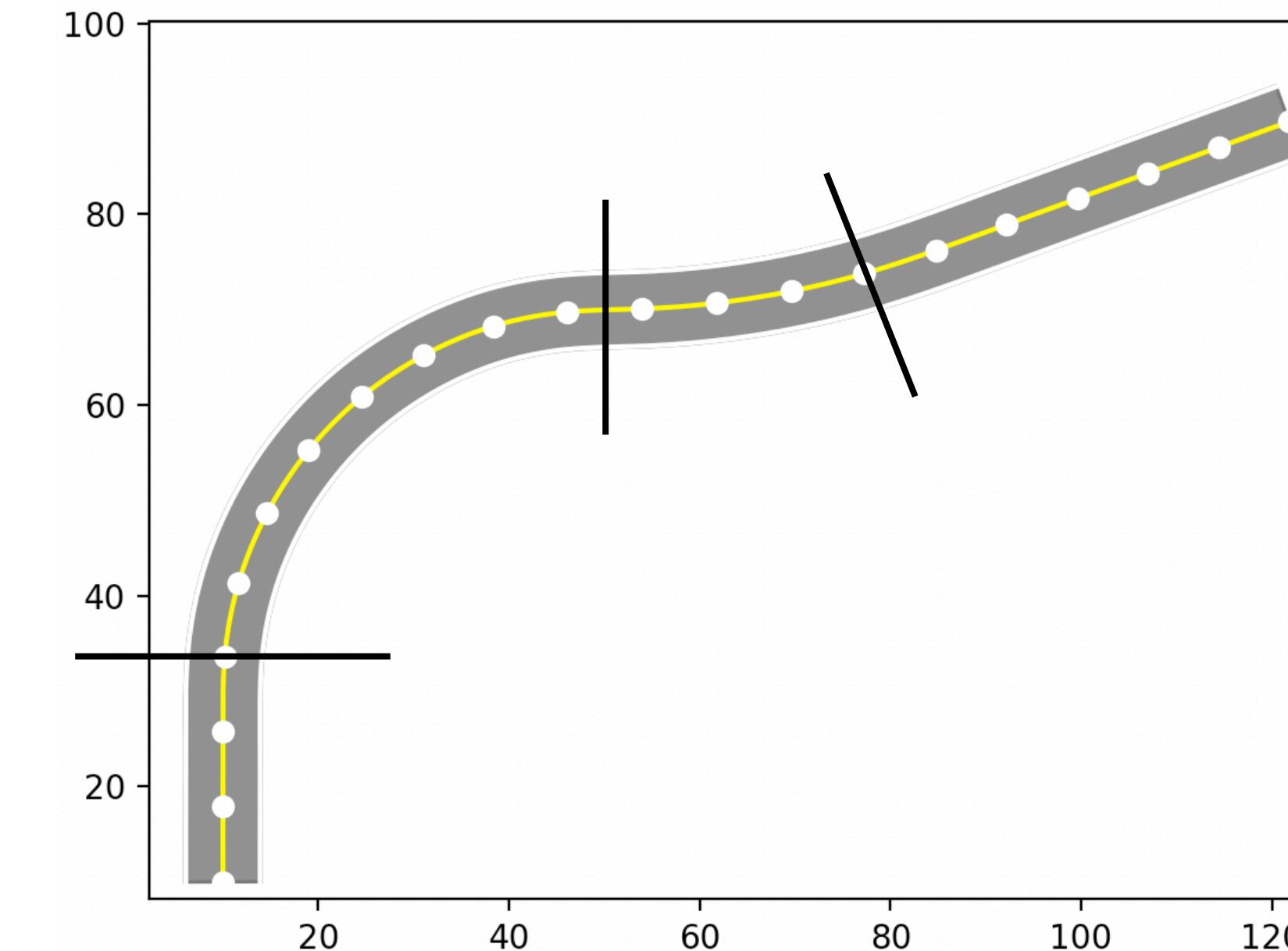
load_scenario_and_roads.py

Procedural Road Generation (2)

- One technique to generate roads is to define **sequences of road segments** such as *straight* and *turns*, and generate the road nodes from them given an initial location and direction (**AsFault**)

```
road_segments.append([
    {'type': 'straight', 'length': 20.0},
    {'type': 'turn', 'angle': -90.0, 'radius': 40.0},
    {'type': 'turn', 'angle': +20.0, 'radius': 100.0},
    {'type': 'straight', 'length': 40.0}
])
```

trajectory_generator.py



- Many other techniques are available, implemented by tools like **AC3R**, **DeepJanus**, **DeepHyperion**, and the participants of the **SBST Tool Competition**

Lane Markings

- Roads are rendered using **textures** associated to the **materials** they are made of (e.g, `road_rubber_sticky`)
- Textures do not come automatically with **lane markings**, in that case the generated roads will be plain vanilla



Lane Markings as Roads

```
right_marking = Road('line_white', rid='right')
right_marking.nodes.extend(rm_nodes)
scenario.add_road(right_marking)
```

```
left_marking = Road('line_white', rid='left')
left_marking.nodes.extend(lm_nodes)
scenario.add_road(left_marking)
```

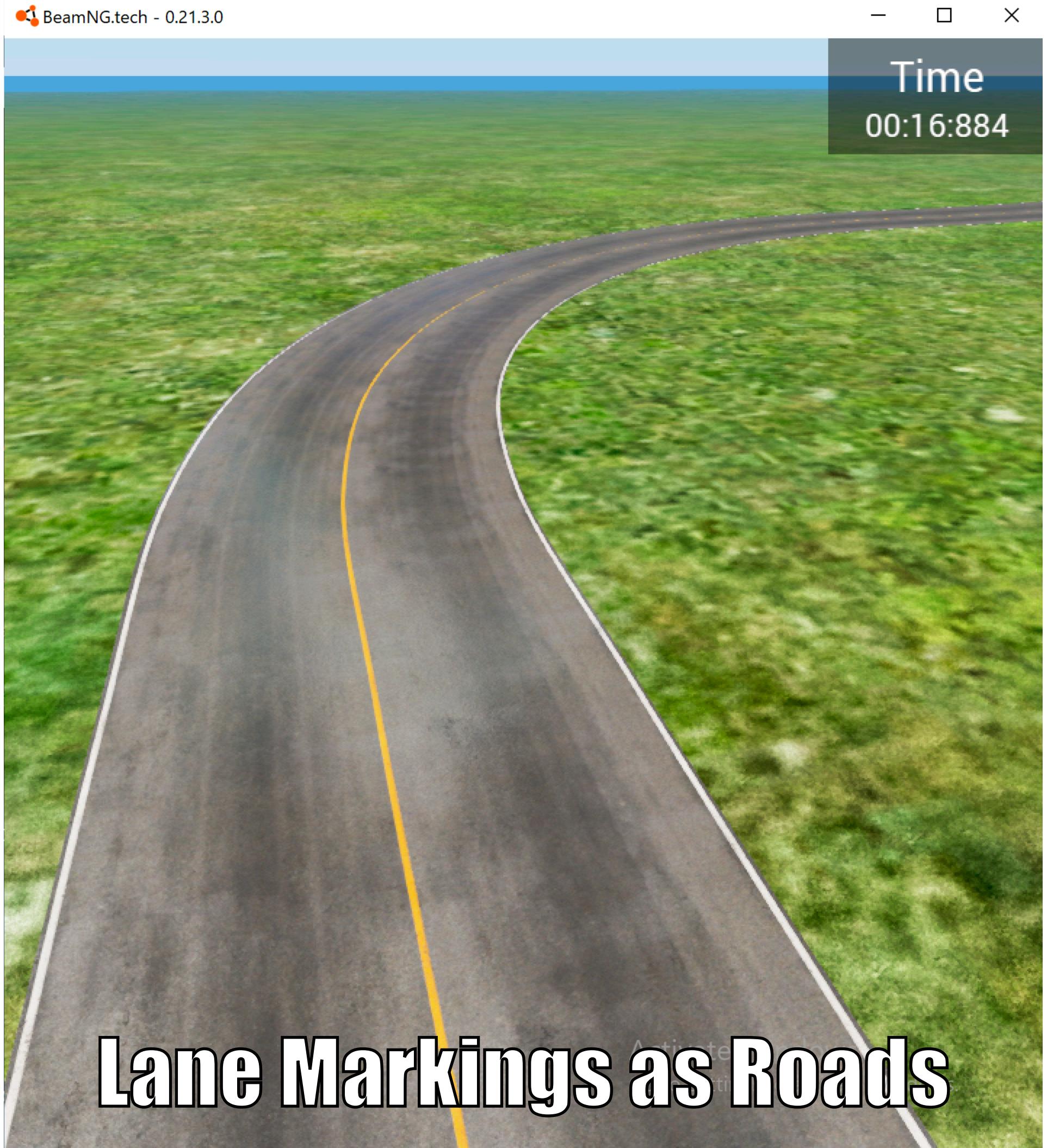
```
central_marking = Road('line_yellow', rid='central')
central_marking.nodes.extend(cm_nodes)
scenario.add_road(central_marking)
```

```
road = Road('road_rubber_sticky', rid='road')
road.nodes.extend(road_nodes)
scenario.add_road(road)
```

- Lane markings can be generated by *stacking* "thin roads" on top of the base pavement, so their textures are rendered together (**AsFault**, **AC3R**)
- A similar technique can be used to add skid marks, dirt, and other effects on the road

trajectory_generator.py

Example



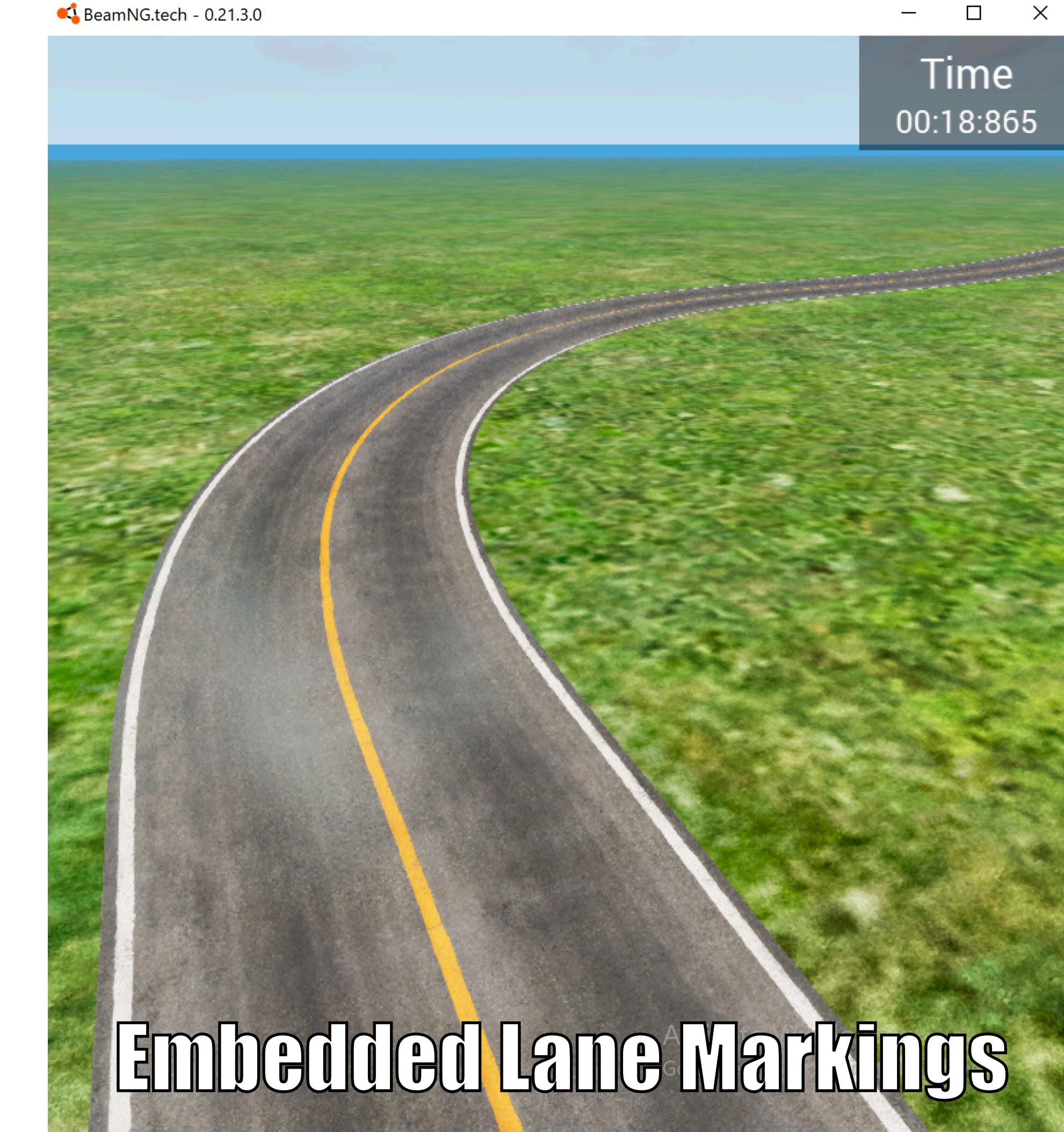
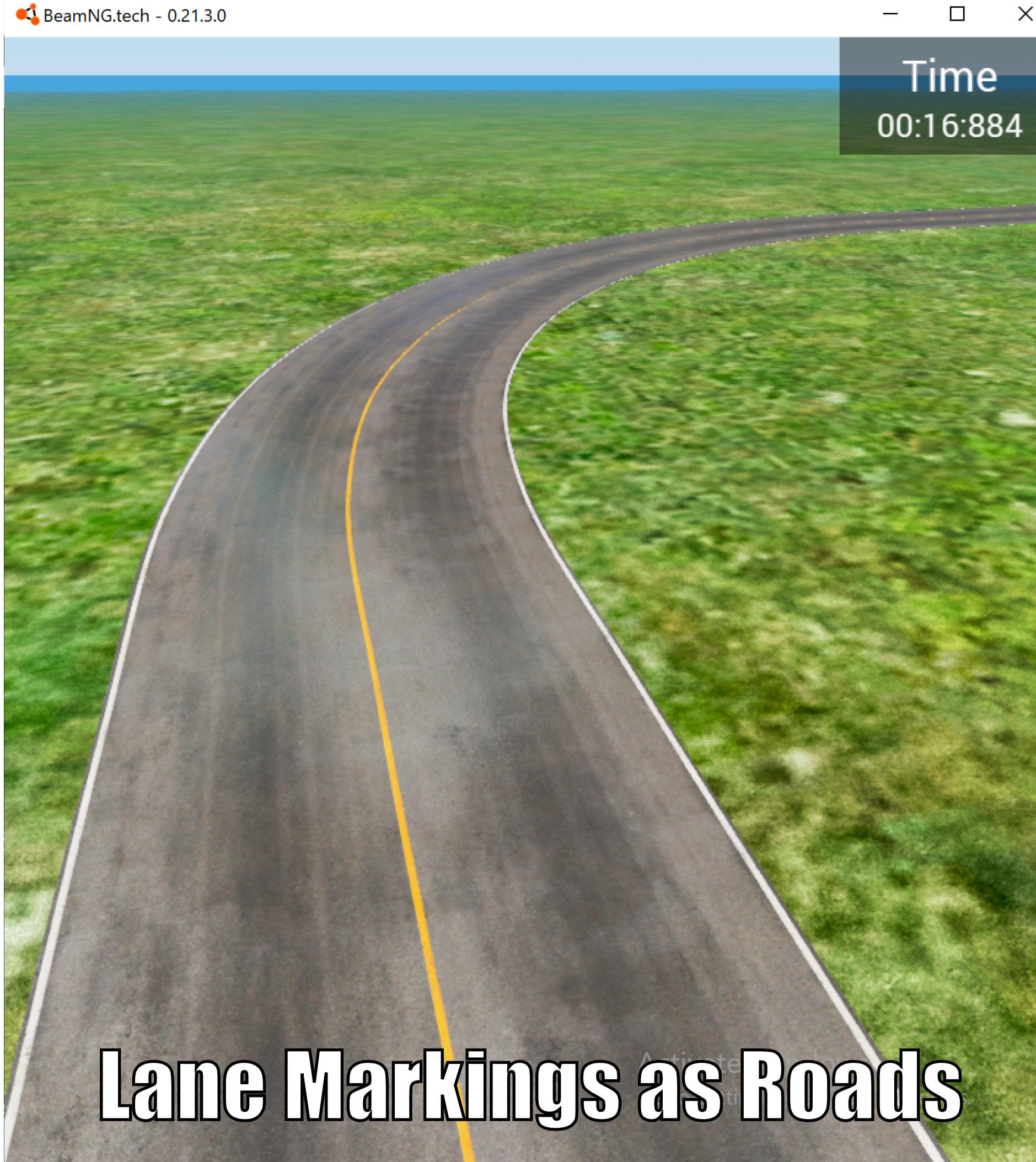
Embedded Lane Markings

- Lane markings can be also generated by *embedding* them in the textures associated to the road materials (**DeepJanus**, **DeepHyperion**, **Tool Competition Pipeline**)

```
scenario = Scenario('tig', 'test_scenario_2')
# This material comes already with lane markings
road = Road('tig_road_rubber_sticky', rid='road_1')
road.nodes.extend(road_nodes)
scenario.add_road(road)
```



Example



Further Environment Customization

- BeamNGpy also supports changing some aspects of the environment, including **time of the day** (night/day-light)



Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

Placing Vehicles and Objects

- BeamNG allows multiple **vehicles** and **objects** to be placed at arbitrary locations and rotated in any direction
- **Vehicles** and complex objects composed of (moving) parts have a **model**. Models are listed in the `<BNG_HOME>\vehicles` folder.
- **Objects** are static (e.g., concrete barriers, buildings) and can be found inside the various `<BNG_HOME>**\art**` folders or procedurally generated

`place_vehicles_and_obstacles.py`

Name
autobello
ball
barrels
barrier
barstow
blockwall
bluebuck
bollard
boxutility
boxutility_large
burnside
cannon
caravan
christmas_tree
citybus
common
cones
coupe
dryvan
etk800
etkc

Placing Vehicles

Ego-car: beginning of the road, in the middle of the right lane

```
ego_pos = translate(start_of_the_road, 0.0, -lane_width*0.5)
```

Move the car a little inside the road

```
ego_pos = translate(ego_pos, length_car, 0.0)
```

```
ego_vehicle = Vehicle('ego', model='etk800', licence='ego', color="red")
```

```
self.scenario.add_vehicle(ego_vehicle, pos=(ego_pos.x, ego_pos.y, ego_pos.z),  
rot_quat=direction_of_the_road)
```



Create a yellow small car in front of the ego-car, in same lane, following the same direction

```
heading_vehicle_pos = translate(ego_pos, +20.0, 0.0)
```

```
heading_vehicle = Vehicle('heading', model='autobello', licence='heading', color="yellow")
```

```
self.scenario.add_vehicle(heading_vehicle, pos=(heading_vehicle_pos.x, heading_vehicle_pos.y, ground_level),  
rot=None, rot_quat=direction_of_the_road)
```

Create a bus in front of the ego, but on the opposite lane, following the opposite direction

```
opposite_vehicle_pos = translate(ego_pos, +10.0, +lane_width)
```

```
opposite_vehicle = Vehicle('opposite', model='citybus', licence='opposite', color="white")
```

```
self.scenario.add_vehicle(opposite_vehicle, pos=(opposite_vehicle_pos.x, opposite_vehicle_pos.y, ground_level),  
rot=None, rot_quat=opposite_direction_of_the_road)
```



Windows PowerShell

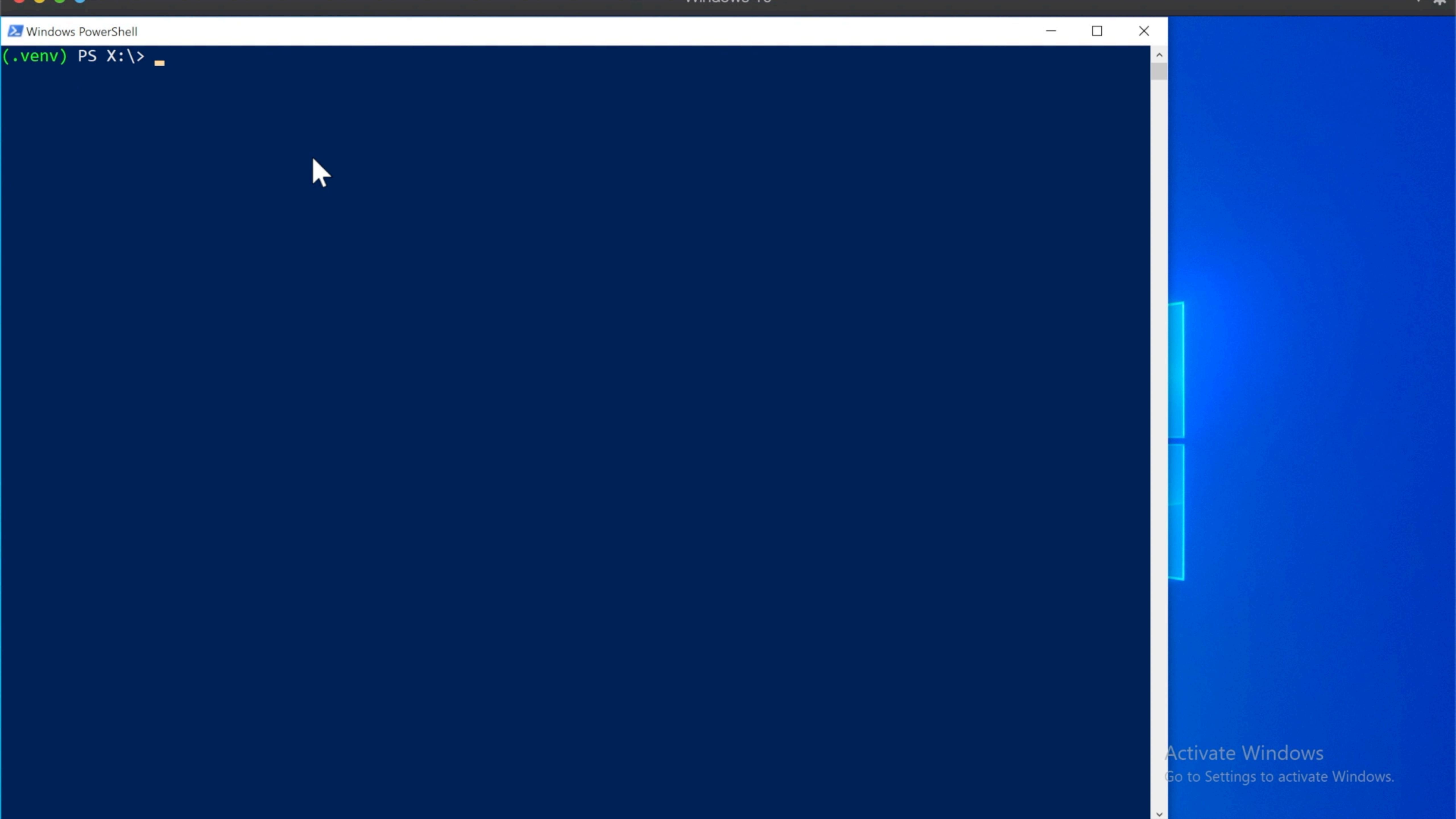
(.venv) PS X:\> ■

Windows 10

— □ ×



Activate Windows
Go to Settings to activate Windows.



Windows PowerShell

(.venv) PS X:\> ■

Windows 10

— □ ×



Activate Windows
Go to Settings to activate Windows.

Placing Vehicles

- Vehicles can collide with other vehicles and objects, but **cannot** be placed "mid-air" nor in the ground. Otherwise, they **fall and crash or crash immediately**

#Adding a vehicle in the wrong place, i.e., mid-air, will cause it to fall and break

```
def test_generate_ground_and_flying_vehicles(self):
```

```
    ground_level = -28.0
```

Create a vehicle and put it on the ground, ground is at -28.0 in tig level

```
    vehicle = Vehicle('vehicle', model='etk800', licence='ground', color="red")
```

```
    self.scenario.add_vehicle(vehicle, pos=(0, 0, ground_level), rot=None, rot_quat=(0, 0, 1, 0))
```

*# Create a vehicle at 0, so 28.0 meters **above** the ground of tig level*

```
flying_vehicle = Vehicle('flying_vehicle', model='etk800', licence='flying', color="yellow")
```

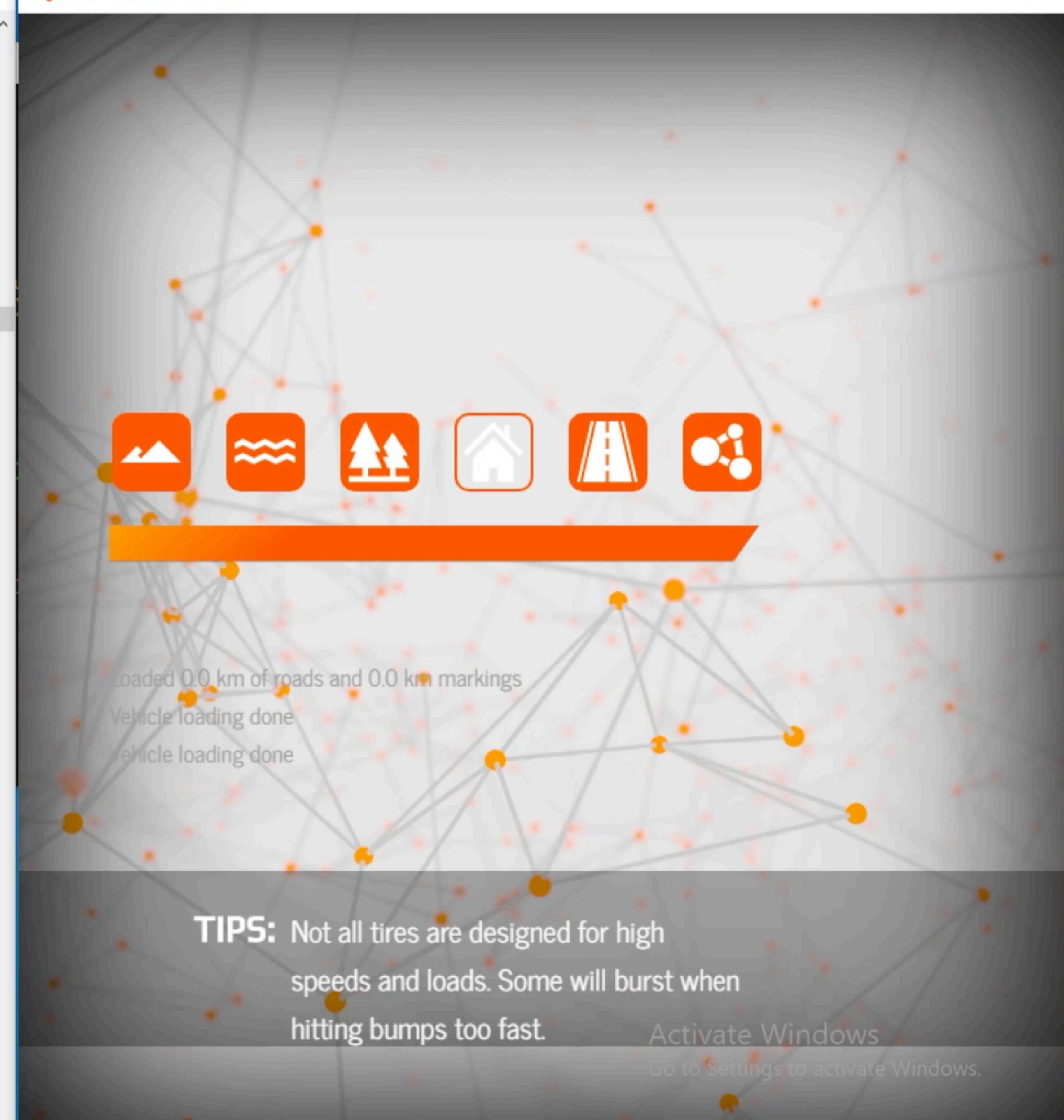
```
self.scenario.add_vehicle(flying_vehicle, pos=(0, 10, 0), rot=None, rot_quat=(0, 0, 1, 0), cling=False)
```

Windows PowerShell

```
0511062949842
> Polling sensors
Current speed Km/h 28.773642602046085
Controlling car with: Steering: -0.08810234814882278 ; Throttle: 0.0723
2407781701777
> Polling sensors
Current speed Km/h 28.734028671539257
Controlling car with: Steering: -0.09031359106302261 ; Throttle: 0.0744
6078449503446
> Polling sensors
Current speed Km/h 28.773406254540863
Car reached target location. Exit
C:\Users\gambitemp\AppData\Local\Programs\Python\Python37\lib\subprocess.py:883: ResourceWarning: subprocess 8332 is still running
    ResourceWarning, source=self)
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
C:\Users\gambitemp\AppData\Local\Programs\Python\Python37\lib\unittest\ca
se.py:628: ResourceWarning: unclosed <socket.socket fd=956, family=Addres
sFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1'
, 51015), raddr=('127.0.0.1', 64256)>
    testMethod()
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
.
-----
Ran 1 test in 223.547s

OK
(.venv-37) PS X:\> deactivate
PS X:\> .\venv\Scripts\activate
(.venv) PS X:\> py.exe PlaceVehiclesAndObstacles
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\python.exe: can
't open file 'X:\PlaceVehiclesAndObstacles': [Errno 2] No such file or di
rectory
(.venv) PS X:\> py.exe .\place_vehicles_and_obstacles.py PlaceVehiclesAnd
Obstacles
```

BeamNG.tech - 0.21.3.0

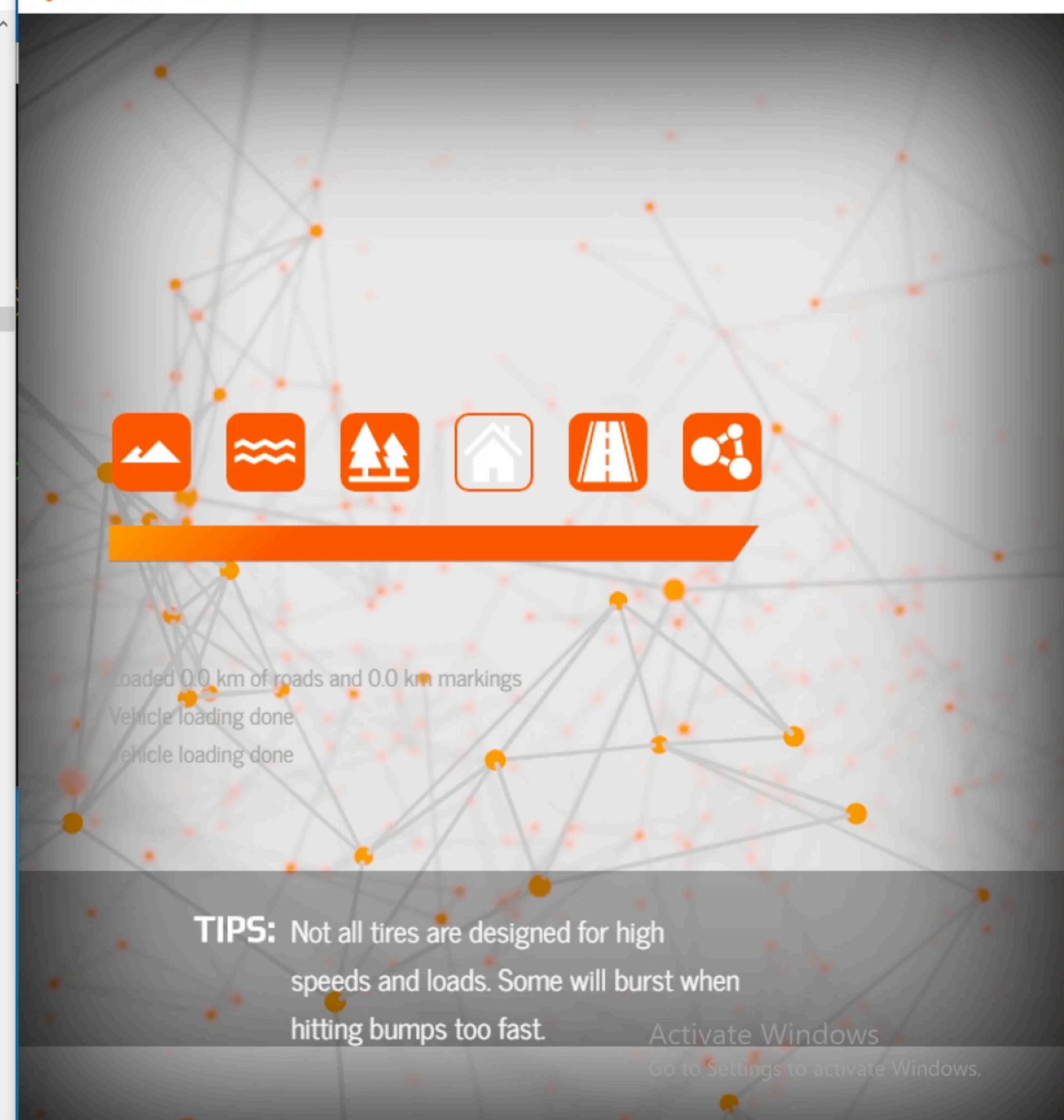


Windows PowerShell

```
0511062949842
> Polling sensors
Current speed Km/h 28.773642602046085
Controlling car with: Steering: -0.08810234814882278 ; Throttle: 0.0723
2407781701777
> Polling sensors
Current speed Km/h 28.734028671539257
Controlling car with: Steering: -0.09031359106302261 ; Throttle: 0.0744
6078449503446
> Polling sensors
Current speed Km/h 28.773406254540863
Car reached target location. Exit
C:\Users\gambitemp\AppData\Local\Programs\Python\Python37\lib\subprocess.py:883: ResourceWarning: subprocess 8332 is still running
    ResourceWarning, source=self)
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
C:\Users\gambitemp\AppData\Local\Programs\Python\Python37\lib\unittest\ca
se.py:628: ResourceWarning: unclosed <socket.socket fd=956, family=Addres
sFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1'
, 51015), raddr=('127.0.0.1', 64256)>
    testMethod()
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
.
-----
Ran 1 test in 223.547s

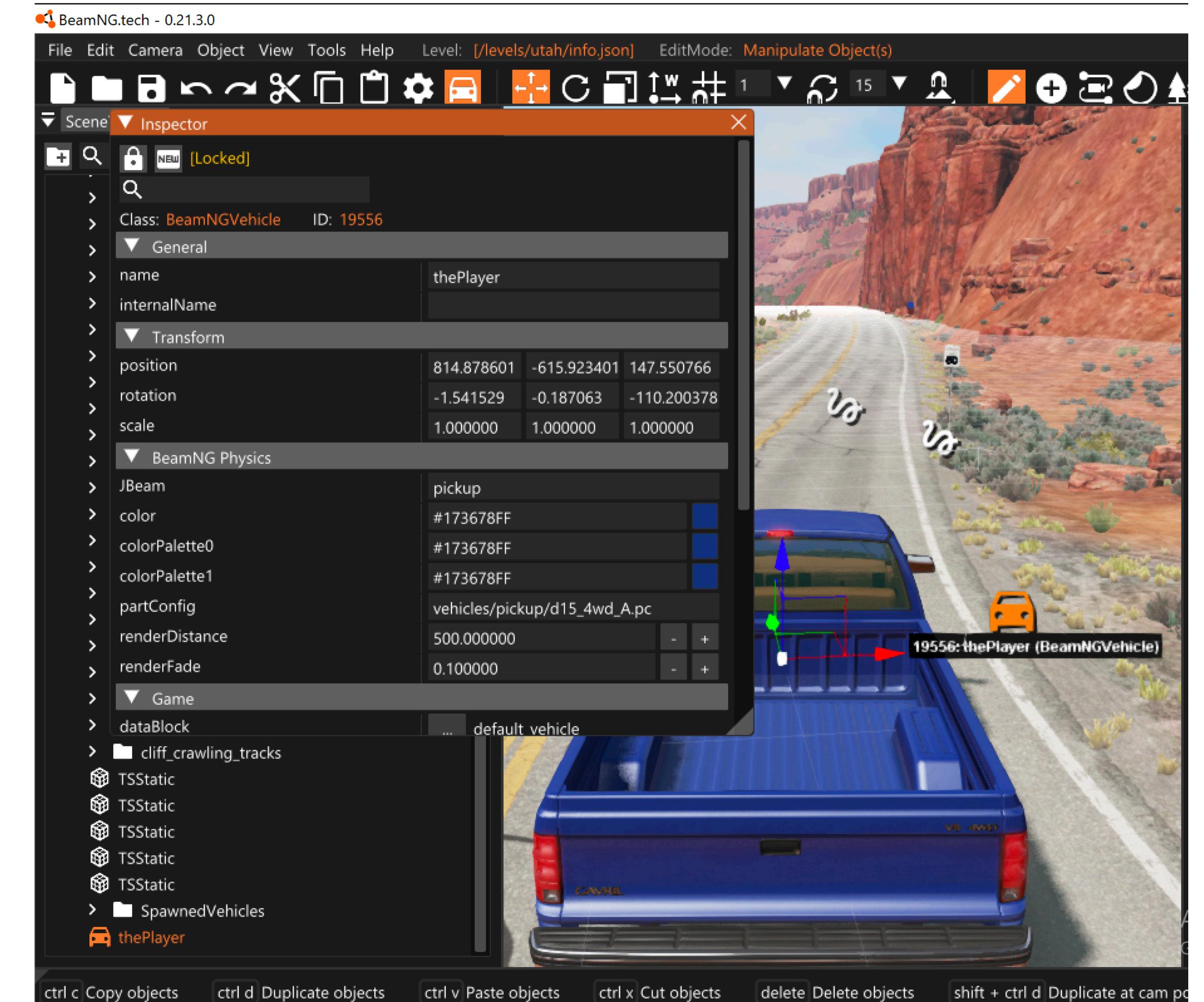
OK
(.venv-37) PS X:\> deactivate
PS X:\> .\venv\Scripts\activate
(.venv) PS X:\> py.exe PlaceVehiclesAndObstacles
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\python.exe: can
't open file 'X:\PlaceVehiclesAndObstacles': [Errno 2] No such file or di
rectory
(.venv) PS X:\> py.exe .\place_vehicles_and_obstacles.py PlaceVehiclesAnd
Obstacles
```

BeamNG.tech - 0.21.3.0



Obtaining Object Locations

- One can **query** the simulator to obtain locations and geometry of the roads in the map and accessing info about the other elements loaded by the simulator
- One can **pause** the simulator (Press J), open the **world editor** (Press F11), and use the **inspector tool** to find out the location of the simulated entities



Placing Objects

- Objects can (but do not have to) collide with vehicles and other objects, and can be placed "mid-air", on the ground, and in the ground.
- **Simple** geometrical objects (e.g., sphere) can be **procedurally generated**
- **Complex** objects (e.g., barrier) must exist (i.e., as .dae) and can be **imported**

```
# A procedurally generated speed bump
# spanning the entire road width
bump = ProceduralBump(name='bump',
pos=(pos.x, pos.y, pos.z), rot=None,
rot_quat=(0, 0, 0, 1),
width=1.0, upper_length= 2*(lane_width-0.1),
length=2*(lane_width+0.1), upper_width=0.4,
height=0.1, material="bumber")
self.scenario.add_procedural_mesh(bump)
```

```
# An imported concrete barrier
barrier = StaticObject(name='barrier',
pos=(pos.x, pos.y, pos.z),
rot=None,
rot_quat=facing_of_the_road,
scale=(1, 1, 1),
shape='/levels/west_coast_usa/art/shapes/race\
/concrete_road_barrier_a.dae')
self.scenario.add_object(barrier)
```

```
2 import common
3
4 from beamngpy import BeamNGpy, Scenario, Road, Vehicle
5 from beamngpy import ProceduralCone, ProceduralBump, ProceduralCube
6
7 from beamngpy import StaticObject
8
9 from shapely.geometry import Point, LineString
10 from shapely.affinity import translate, rotate
11
12 # Specify where BeamNG home and user are
13 BNG_HOME = "C:\\BeamNG.tech.v0.21.3.0"
14 BNG_USER = "C:\\BeamNG.tech_userpath"
15
16
17 > class PlaceVehiclesAndObstacles(unittest.TestCase):
18     def test_cylinder(self):
19         scenario = self._create_scenario()
20
21         # Create a cylinder
22         cylinder = StaticObject(
23             shape=shapely.geometry.Point(0, 0, 0),
24             size=(1, 1, 1))
25
26         # Create a road
27         road = Road(
28             geometry=LineString([(0, 0), (10, 0)]),
29             height=0)
30
31         # Create a scenario
32         scenario.add_road(road)
33         scenario.add_cylinder(cylinder)
34
35         # Run simulation
36         beamngpy = BeamNGpy(scenario=scenario)
37         beamngpy.start()
38
39         # Check if cylinder is placed correctly
40         cylinder_center = cylinder.get_center()
41         assert cylinder_center[0] == 5.0
42         assert cylinder_center[1] == 0.0
43         assert cylinder_center[2] == 0.5
44
45         # Clean up
46         beamngpy.stop()
47
48     def _create_scenario(self):
49         scenario = Scenario()
50
51         # Create a road
52         road = Road(
53             geometry=LineString([(0, 0), (10, 0)]),
54             height=0)
55
56         # Create a scenario
57         scenario.add_road(road)
58
59         return scenario
60
61
62 if __name__ == '__main__':
63     unittest.main()
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
```

```
2 import common
3
4 from beamngpy import BeamNGpy, Scenario, Road, Vehicle
5 from beamngpy import ProceduralCone, ProceduralBump, ProceduralCube
6
7 from beamngpy import StaticObject
8
9 from shapely.geometry import Point, LineString
10 from shapely.affinity import translate, rotate
11
12 # Specify where BeamNG home and user are
13 BNG_HOME = "C:\\BeamNG.tech.v0.21.3.0"
14 BNG_USER = "C:\\BeamNG.tech_userpath"
15
16
17 > class PlaceVehiclesAndObstacles(unittest.TestCase):
18     def test_cylinder(self):
19         scenario = self._create_scenario()
20
21         # Create a cylinder
22         cylinder = StaticObject(
23             shape=shapely.geometry.Polygon([
24                 Point(0, 0),
25                 Point(1, 0),
26                 Point(1, 1),
27                 Point(0, 1),
28                 Point(0, 0)
29             ]))
30
31         # Rotate the cylinder
32         cylinder = rotate(cylinder, 45)
33
34         # Translate the cylinder
35         cylinder = translate(cylinder, xoff=0.5, yoff=0.5)
36
37         # Add the cylinder to the scenario
38         scenario.add_static_object(cylinder)
39
40         # Create a road
41         road = Road()
42
43         # Add the road to the scenario
44         scenario.add_road(road)
45
46         # Run the scenario
47         scenario.run()
48
49         # Check if the cylinder is in the scenario
50         cylinder_in_scenario = cylinder in scenario.static_objects
51
52         self.assertTrue(cylinder_in_scenario)
53
54
55     def _create_scenario(self):
56         scenario = Scenario()
57
58         # Set the BeamNG home path
59         scenario.set_home(BNG_HOME)
60
61         # Set the BeamNG user path
62         scenario.set_user(BNG_USER)
63
64         # Set the BeamNG version
65         scenario.set_version("v0.21.3.0")
66
67         # Set the BeamNG physics engine
68         scenario.set_physics("beamng")
69
70         # Set the BeamNG simulation mode
71         scenario.set_mode("normal")
72
73         # Set the BeamNG simulation time step
74         scenario.set_time_step(0.01)
75
76         # Set the BeamNG simulation frame rate
77         scenario.set_frame_rate(60)
78
79         # Set the BeamNG simulation gravity
80         scenario.set_gravity(9.81)
81
82         # Set the BeamNG simulation temperature
83         scenario.set_temperature(25)
84
85         # Set the BeamNG simulation pressure
86         scenario.set_pressure(101325)
87
88         # Set the BeamNG simulation density
89         scenario.set_density(1.225)
90
91         # Set the BeamNG simulation viscosity
92         scenario.set_viscosity(0.001)
93
94         # Set the BeamNG simulation air density
95         scenario.set_air_density(1.225)
96
97         # Set the BeamNG simulation air viscosity
98         scenario.set_air_viscosity(0.001)
99
100        # Set the BeamNG simulation air pressure
101        scenario.set_air_pressure(101325)
102
103        # Set the BeamNG simulation air density
104        scenario.set_air_density(1.225)
105
106        # Set the BeamNG simulation air viscosity
107        scenario.set_air_viscosity(0.001)
108
109        # Set the BeamNG simulation air pressure
110        scenario.set_air_pressure(101325)
111
112        # Set the BeamNG simulation air density
113        scenario.set_air_density(1.225)
114
115        # Set the BeamNG simulation air viscosity
116        scenario.set_air_viscosity(0.001)
117
118        # Set the BeamNG simulation air pressure
119        scenario.set_air_pressure(101325)
120
121        # Set the BeamNG simulation air density
122        scenario.set_air_density(1.225)
123
124        # Set the BeamNG simulation air viscosity
125        scenario.set_air_viscosity(0.001)
126
127        # Set the BeamNG simulation air pressure
128        scenario.set_air_pressure(101325)
129
130        # Set the BeamNG simulation air density
131        scenario.set_air_density(1.225)
132
133        # Set the BeamNG simulation air viscosity
134        scenario.set_air_viscosity(0.001)
135
136        # Set the BeamNG simulation air pressure
137        scenario.set_air_pressure(101325)
138
139        # Set the BeamNG simulation air density
140        scenario.set_air_density(1.225)
141
142        # Set the BeamNG simulation air viscosity
143        scenario.set_air_viscosity(0.001)
144
145        # Set the BeamNG simulation air pressure
146        scenario.set_air_pressure(101325)
147
148        # Set the BeamNG simulation air density
149        scenario.set_air_density(1.225)
150
151        # Set the BeamNG simulation air viscosity
152        scenario.set_air_viscosity(0.001)
153
154        # Set the BeamNG simulation air pressure
155        scenario.set_air_pressure(101325)
156
157        # Set the BeamNG simulation air density
158        scenario.set_air_density(1.225)
159
160        # Set the BeamNG simulation air viscosity
161        scenario.set_air_viscosity(0.001)
162
163        # Set the BeamNG simulation air pressure
164        scenario.set_air_pressure(101325)
165
166        # Set the BeamNG simulation air density
167        scenario.set_air_density(1.225)
168
169        # Set the BeamNG simulation air viscosity
170        scenario.set_air_viscosity(0.001)
171
172        # Set the BeamNG simulation air pressure
173        scenario.set_air_pressure(101325)
174
175        # Set the BeamNG simulation air density
176        scenario.set_air_density(1.225)
177
178        # Set the BeamNG simulation air viscosity
179        scenario.set_air_viscosity(0.001)
180
181        # Set the BeamNG simulation air pressure
182        scenario.set_air_pressure(101325)
183
184        # Set the BeamNG simulation air density
185        scenario.set_air_density(1.225)
186
187        # Set the BeamNG simulation air viscosity
188        scenario.set_air_viscosity(0.001)
189
190        # Set the BeamNG simulation air pressure
191        scenario.set_air_pressure(101325)
192
193        # Set the BeamNG simulation air density
194        scenario.set_air_density(1.225)
195
196        # Set the BeamNG simulation air viscosity
197        scenario.set_air_viscosity(0.001)
198
199        # Set the BeamNG simulation air pressure
200        scenario.set_air_pressure(101325)
201
202        # Set the BeamNG simulation air density
203        scenario.set_air_density(1.225)
204
205        # Set the BeamNG simulation air viscosity
206        scenario.set_air_viscosity(0.001)
207
208        # Set the BeamNG simulation air pressure
209        scenario.set_air_pressure(101325)
210
211        # Set the BeamNG simulation air density
212        scenario.set_air_density(1.225)
213
214        # Set the BeamNG simulation air viscosity
215        scenario.set_air_viscosity(0.001)
216
217        # Set the BeamNG simulation air pressure
218        scenario.set_air_pressure(101325)
219
220        # Set the BeamNG simulation air density
221        scenario.set_air_density(1.225)
222
223        # Set the BeamNG simulation air viscosity
224        scenario.set_air_viscosity(0.001)
225
226        # Set the BeamNG simulation air pressure
227        scenario.set_air_pressure(101325)
228
229        # Set the BeamNG simulation air density
230        scenario.set_air_density(1.225)
231
232        # Set the BeamNG simulation air viscosity
233        scenario.set_air_viscosity(0.001)
234
235        # Set the BeamNG simulation air pressure
236        scenario.set_air_pressure(101325)
237
238        # Set the BeamNG simulation air density
239        scenario.set_air_density(1.225)
240
241        # Set the BeamNG simulation air viscosity
242        scenario.set_air_viscosity(0.001)
243
244        # Set the BeamNG simulation air pressure
245        scenario.set_air_pressure(101325)
246
247        # Set the BeamNG simulation air density
248        scenario.set_air_density(1.225)
249
250        # Set the BeamNG simulation air viscosity
251        scenario.set_air_viscosity(0.001)
252
253        # Set the BeamNG simulation air pressure
254        scenario.set_air_pressure(101325)
255
256 >     if __name__ == '__main__':
257         unittest.main()
258
```

Windows PowerShell

Ran 1 test in 223.547s

OK

```
(.venv-37) PS X:\> deactivate
PS X:\> .\venv\Scripts\activate
(.venv) PS X:\> py.exe PlaceVehiclesAndObstacles
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\python.exe: can't open file 'X:\PlaceVehiclesAndObstacles': [Errno 2] No such file or directory
(.venv) PS X:\> py.exe .\place_vehicles_and_obstacles.py PlaceVehiclesAndObstacles
Press enter when done...
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\subprocess.py:1052: ResourceWarning: subprocess 8760 is still running
    _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\unittest\case.py:550: ResourceWarning: unclosed <socket.socket fd=1280, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 51031), raddr=('127.0.0.1', 64256)>
    method()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
.Press enter when done...quit()
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\subprocess.py:1052: ResourceWarning: subprocess 4472 is still running
    _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation traceback
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\unittest\case.py:550: ResourceWarning: unclosed <socket.socket fd=1160, family=AddressFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1', 51037), raddr=('127.0.0.1', 64256)>
    method()
ResourceWarning: Enable tracemalloc to get the object allocation traceback
```



Windows PowerShell

Ran 1 test in 223.547s

OK

```
(.venv-37) PS X:\> deactivate
PS X:\> .\venv\Scripts\activate
(.venv) PS X:\> py.exe PlaceVehiclesAndObstacles
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\python.exe: can't open file 'X:\PlaceVehiclesAndObstacles': [Errno 2] No such file or directory
(.venv) PS X:\> py.exe .\place_vehicles_and_obstacles.py PlaceVehiclesAnd
Obstacles
Press enter when done...
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\subprocess.py:1052: ResourceWarning: subprocess 8760 is still running
    _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\unittest\ca
se.py:550: ResourceWarning: unclosed <socket.socket fd=1280, family=Addre
ssFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1
', 51031), raddr=('127.0.0.1', 64256)>
    method()
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
.Press enter when done...quit()
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\subprocess.py:1052: ResourceWarning: subprocess 4472 is still running
    _warn("subprocess %s is still running" % self.pid,
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
C:\Users\gambitemp\AppData\Local\Programs\Python\Python39\lib\unittest\ca
se.py:550: ResourceWarning: unclosed <socket.socket fd=1160, family=Addre
ssFamily.AF_INET, type=SocketKind.SOCK_STREAM, proto=0, laddr=('127.0.0.1
', 51037), raddr=('127.0.0.1', 64256)>
    method()
ResourceWarning: Enable tracemalloc to get the object allocation tracebac
k
```



Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

Positive vs Negative Oracles

- Testing **continuous controllers** requires the definition of **positive oracles** to establish when a test **passes** and **negative oracles** to decide when it **fails**
 - Examples of positive oracles are **target areas** that the ego-car must reach or traverse.
 - Examples of negative oracles are **timeouts**, **damages** (safety), **lack of movement** (liveliness), and "**forbidden**" **areas** (safety).
- Oracles can be checked **on-line** or **off-line**. On-line checking requires **runtime monitors**, off-line checking requires to **persist the data**.

Sensors

- Oracles can be defined on **objects' state** (e.g., pos, rotation), **absolute values** (simulation time), **physical quantities** (e.g., speed, acceleration, forces), and other **sensor values** (e.g., pixel-perfect annotations, steering wheel position)
- The data required to evaluate the oracles can be collected by **polling** the simulator via the **sensors** abstraction
- Sensors must be **attached** to vehicles
- BeamNGpy implements many sensors, including State, Timer, Camera, Electrics, Lidar, and more

A Simple Sensor: State

```
from beampy.sensors import State

# Attach a State sensor to the vehicle
state_sensor = State()
self.ego_vehicle.attach_sensor('state', state_sensor)

with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:
    self.scenario.make(bng)
    bng.load_scenario(self.scenario)
    bng.start_scenario()
    for i in range(1, TIMEOUT):
        sleep(10) # Wait
        self.ego_vehicle.poll_sensors() # Poll Data
```

Position-based Oracle

```
# This class implements an oracle to check if the vehicle reached a target location (circular area)
# It uses the State sensor to get the position of the vehicle
from shapely.geometry import Point

class TargetAreaOracle():

    def __init__(self, target_position, radius, state_sensor):
        self.target_position = Point(target_position)
        self.radius = radius
        self.state_sensor = state_sensor

    def check(self):
        # Get current position from the state_sensor and measure distance to target
        distance_to_goal = self.target_position.distance(Point(self.state_sensor.data['pos']))
        return distance_to_goal < self.radius
```

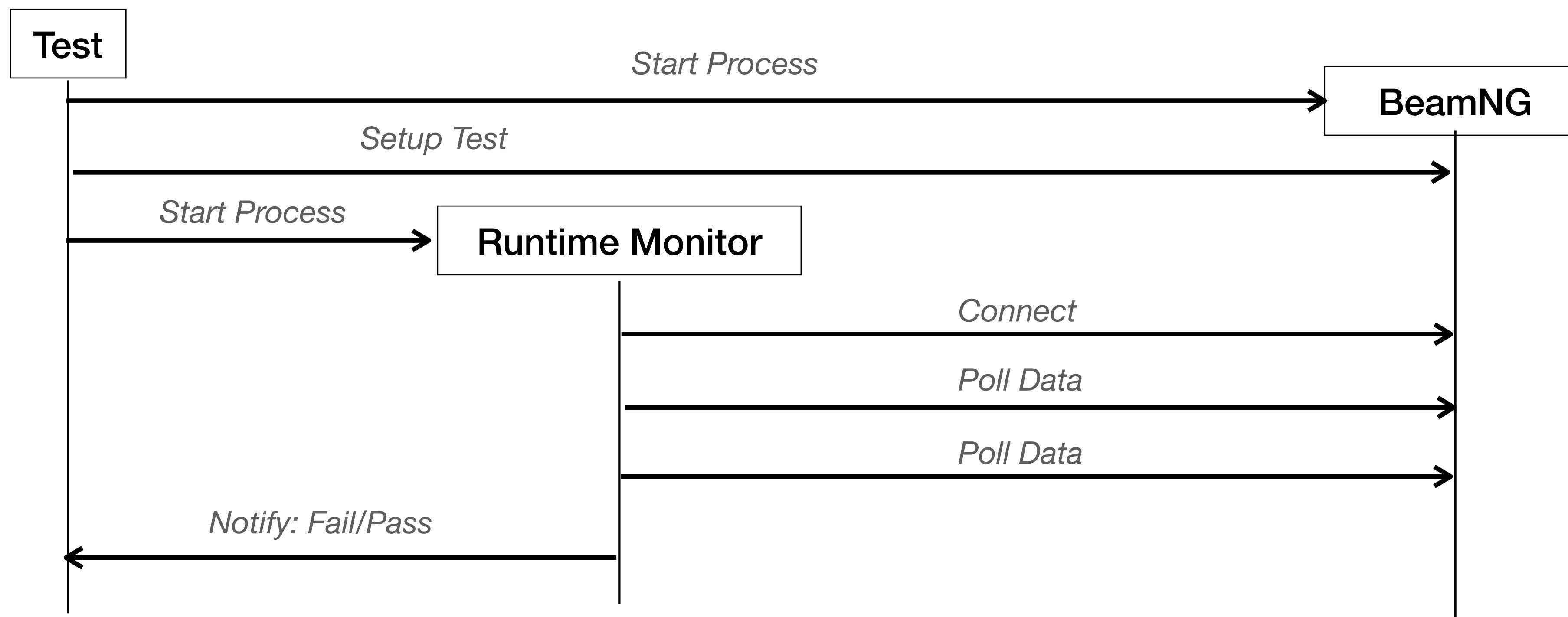
Putting All Together

```
# Attach a State sensor to the vehicle
state_sensor = State()
self.ego_vehicle.attach_sensor('state', state_sensor)
# Define the oracle
target_position = (60, 30, ground_level)
radius = 2 * lane_width + 0.2
target_area_reached_oracle = TargetAreaOracle(target_position, radius, state_sensor)

with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:
    self.scenario.make(bng)
    bng.load_scenario(self.scenario)
    bng.start_scenario()
    for i in range(1, TIMEOUT):
        sleep(10) # Wait
        self.ego_vehicle.poll_sensors() # Poll Data
        # Check Oracle
        if target_area_reached_oracle.check():
            print("Car reached target location")
    return
```

Runtime Monitoring

- During the simulation execution, data can be gathered directly in the **main process** (i.e., test controller) or in a **background (sub-)process**.
- Gathering data directly in the main process is **simpler to implement** since it does not require to setup complex inter-process communications (as illustrated below)



Synchronous Simulations

- Collecting large amount of data (e.g., Camera images, Point-clouds) at high frequency may introduce large performance overhead, impacting the achievable Frame-per-Second (FPS) and overall quality of the simulation.
- Likewise, performance overhead impact the **quality of control** because it introduces lags/delays in controllers/drivers reaction
- **Synchronous simulations** can be used to reduce the impact of processing data on the critical path by executing a number of **simulation steps** and then **pausing** waiting for the "clients" to read and process the data before continuing
- Synchronous simulation may improves **reproducibility** and reduce noise and *flakyness* (especially when paired with **deterministic** simulations)

Synchronous Simulations

```
with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:  
    # Define the simulation frequency: 60 FPS  
    bng.set_steps_per_second(60)  
    bng.set_deterministic()  
  
    # Pause the simulation  
    bng.pause()  
  
    for i in range(1,10):  
        # Progress it for 30 steps, 30/60=0.5 sec  
        bng.step(30)  
        # Pretend this is a long-running computation  
        sleep(2)  
  
        # Eventually restore the standard simulation behavior  
        bng.resume()
```

Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, traffic)

The Ego-Car, a.k.a. the Test Subject

- The logic controlling the ego-car can be executed **inside** the simulator (e.g., BeamNG driving agent) or **outside**
- Outside the simulator, the driving agent can run inside the **main** process (e.g., **DeepJanus**, **DeepJanus**, **Tool Competition Pipeline**) or in a **separated** process (e.g., **AsFault**).
- As runtime monitors do, **driving agents collect** data using **sensors** attached to the Vehicle(s).
- Different than runtime monitors, driving agent **control** the ego-car (i.e., steering, throttle, brake)

Define the Test Goal

- Simulation-based tests usually take the form of **driving tasks**, such as follow the lane or avoid an obstacle, that the test subject must successfully complete
- Instructing the driving agent/AI on what to do is **problem and application specific**
- Sometimes the driving task is "**implicitly**" defined:
 - For testing lane keeping systems, **AsFault**, provides a list of way points for the ego-car to follow, while **DeepJanus** puts the car on a single road, and the ego-car has to follow it

Using BeamNG Driving AI

Tool Competition Pipeline

- The logic controlling vehicles is embedded in the simulator and exposed via BeamNGpy
 - Set destination waypoint (must be **reachable** from ego-vehicle current position)
 - Set risk factor, max speed, and other properties

```
r_nodes = generate_road_nodes()  
# Define the destination point  
dest = (r_nodes[-1][0], r_nodes[-1][1], r_nodes[-1][2])  
scenario.add_checkpoints([dest], [(1, 1, 1)], ids=["dest_wp"])  
with BeamNGpy('localhost', 64256) as bng:  
    # Make and start the scenario  
    scenario.make(bng)  
    bng.load_scenario(scenario)  
    bng.start_scenario()  
    bng.pause()  
    # Configure BeamNG driving AI to drive to the destination  
    ego_vehicle.ai_set_mode('manual')  
    ego_vehicle.ai_set_waypoint('dest_wp')  
    ego_vehicle.ai_drive_in_lane('true')  
    ego_vehicle.ai_set_speed(50.0 / 3.6, mode='limit')
```

End-to-end Driving in Main Process

DeepJanus, DeepHyperion

- The logic controlling the vehicles is executed **inside the main process** along with the runtime monitor/oracles
 - Data are collected using the sensors and sent to the driving agent that uses the API to **drive the vehicle** by setting steering, brake, and throttle.
- Simple setup, but **requires installing the driving agent dependencies** along with the dependencies of the other components
 - For instance, NVidia Dave2 (from Precrime) requires tensorflow, keras, open-cv, and other libraries

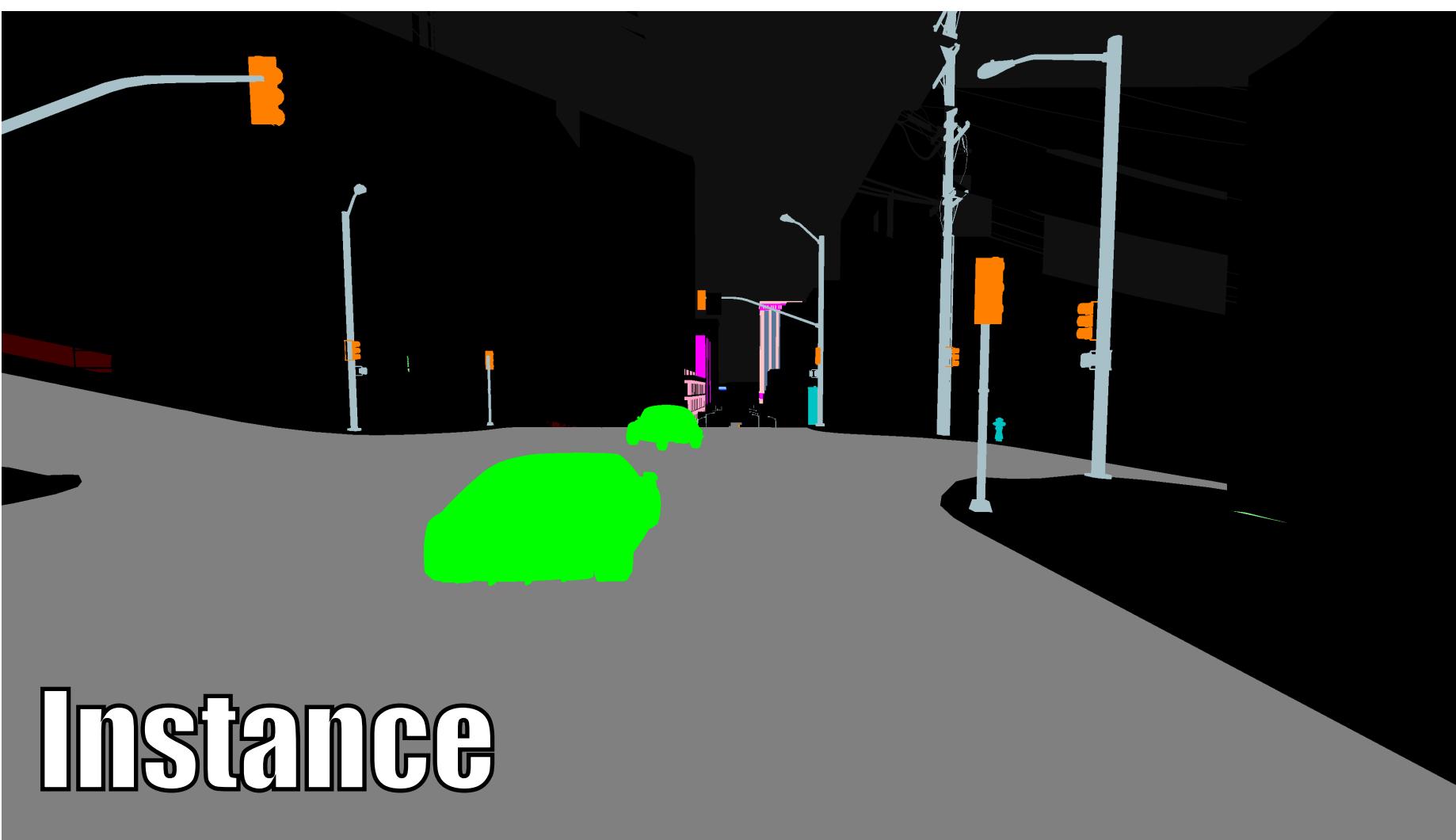
Camera Sensor



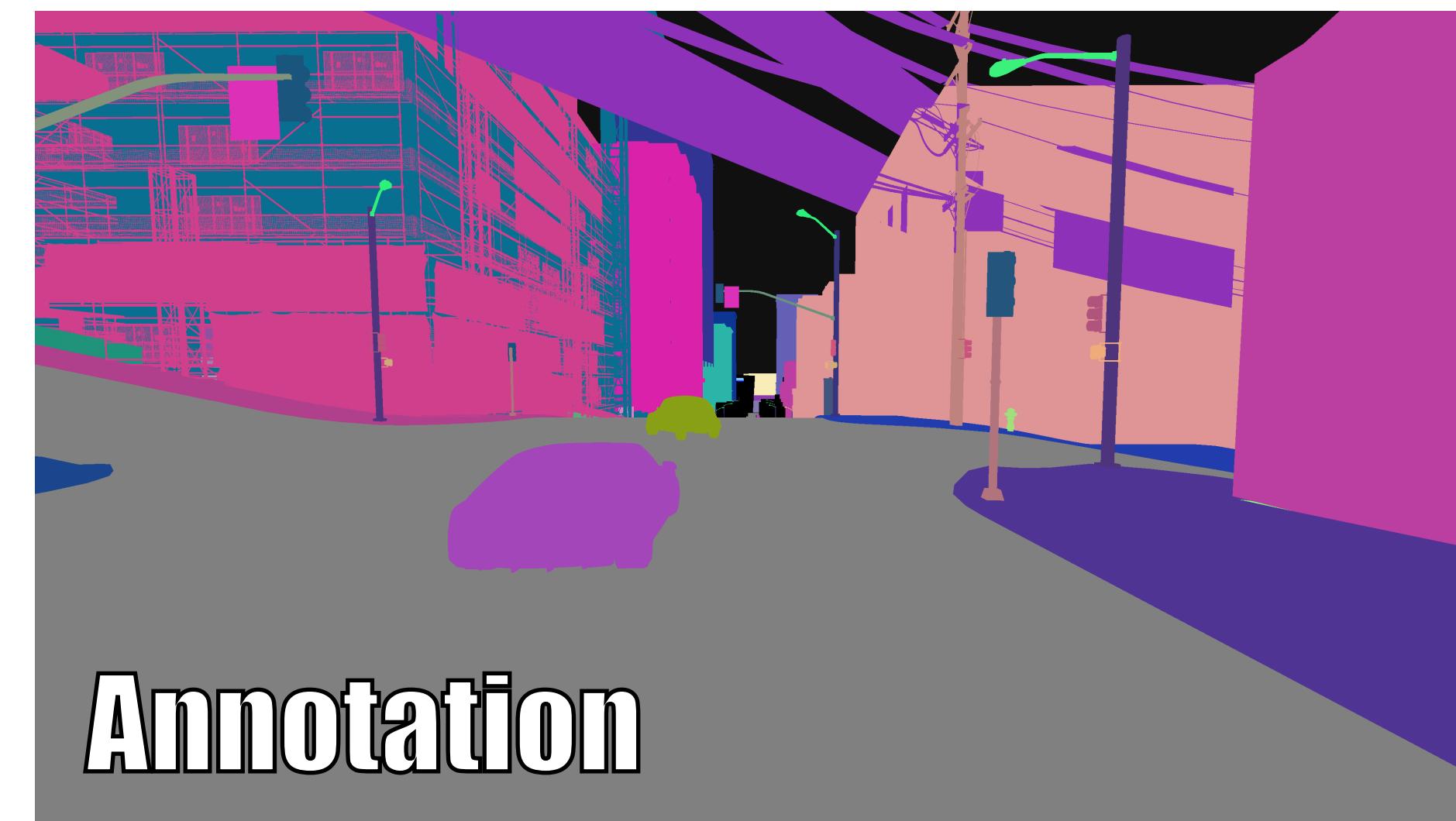
Colour



Depth



Instance



Annotation

DeepJanus*

DeepHyperion*

* *Code simplified to ease the presentation*

https://github.com/testingautomated-usi/DeepJanus/tree/master/DeepJanus-BNG/self_driving

DeepJanus*

DeepHyperion*

* Code simplified to ease the presentation

class NvidiaPrediction:

nvidia_prediction.py

```
def __init__(self, model, config: Config):
    self.model = model
    self.config = config
    self.speed_limit = config.MAX_SPEED

def predict(self, image, car_state):
    # Predict the steering angle from the image
    image = np.asarray(image)
    image = preprocess(image)
    image = np.array([image])
    steering_angle = float(self.model.predict(image, batch_size=1))
    # Adjust the speed
    speed = car_state.vel_kmh
    if speed > self.speed_limit:
        self.speed_limit = self.config.MIN_SPEED # slow down
    else:
        self.speed_limit = self.config.MAX_SPEED
    throttle = 1.0 - steering_angle ** 2 - (speed / self.speed_limit) ** 2
    return steering_angle, throttle
```

DeepJanus*

DeepHyperion*

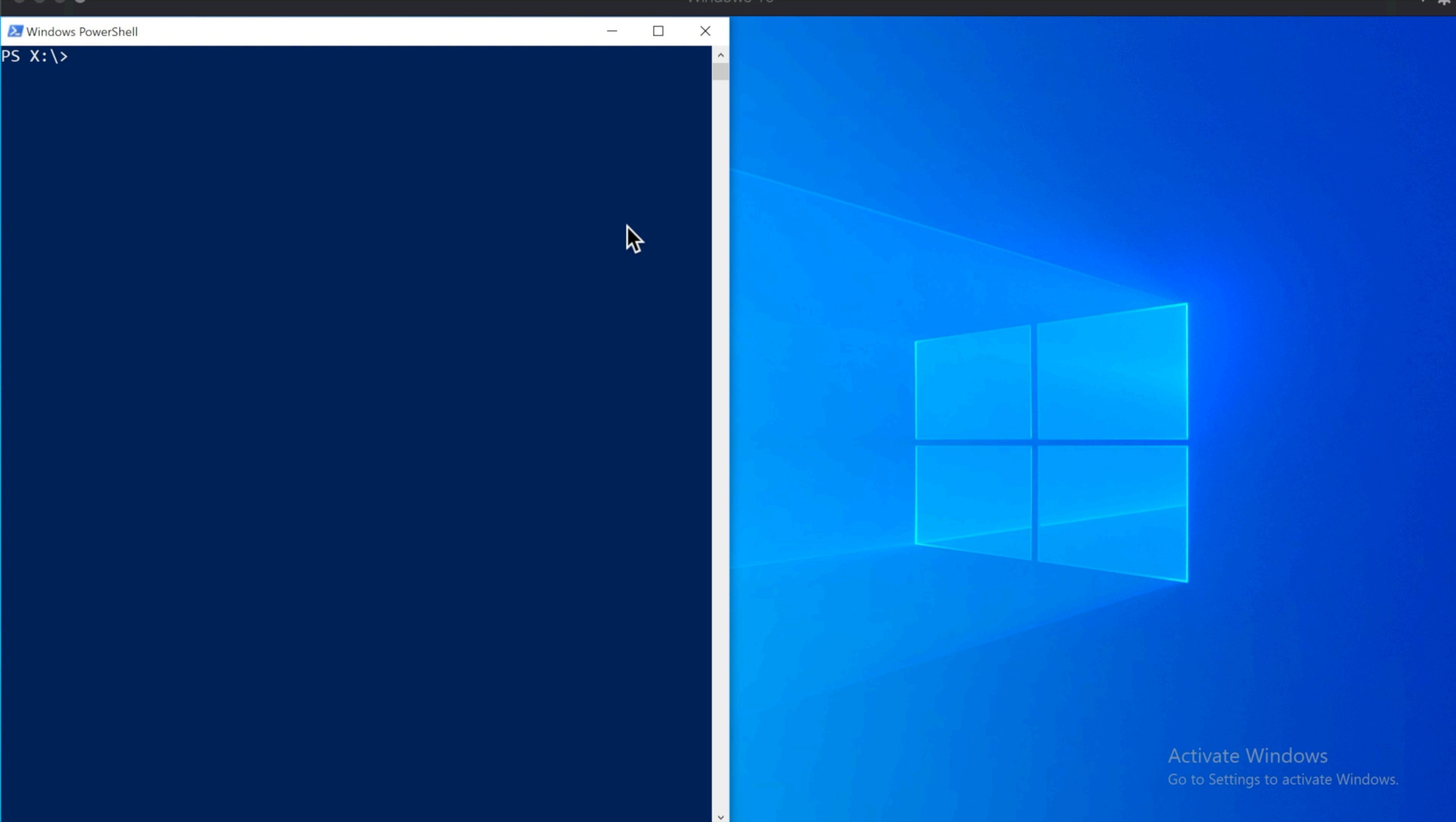
* Code simplified to ease the presentation

nvidia_prediction.py

```
class NvidiaPrediction:  
  
    def __init__(self, model, config: Config):  
        self.model = model  
        self.config = config  
        self.speed_limit = config.MAX_SPEED  
  
    def predict(self, image, car_state):  
        # Predict the steering angle from the image  
        image = np.asarray(image)  
        image = preprocess(image)  
        image = np.array([image])  
        steering_angle = float(self.model.predict(image, batch_size=1))  
        # Adjust the speed  
        speed = car_state.vel_kmh  
        if speed > self.speed_limit:  
            self.speed_limit = self.config.MIN_SPEED # slow down  
        else:  
            self.speed_limit = self.config.MAX_SPEED  
        throttle = 1.0 - steering_angle ** 2 - (speed / self.speed_limit) ** 2  
        return steering_angle, throttle
```

beamng_nvidia_runner.py

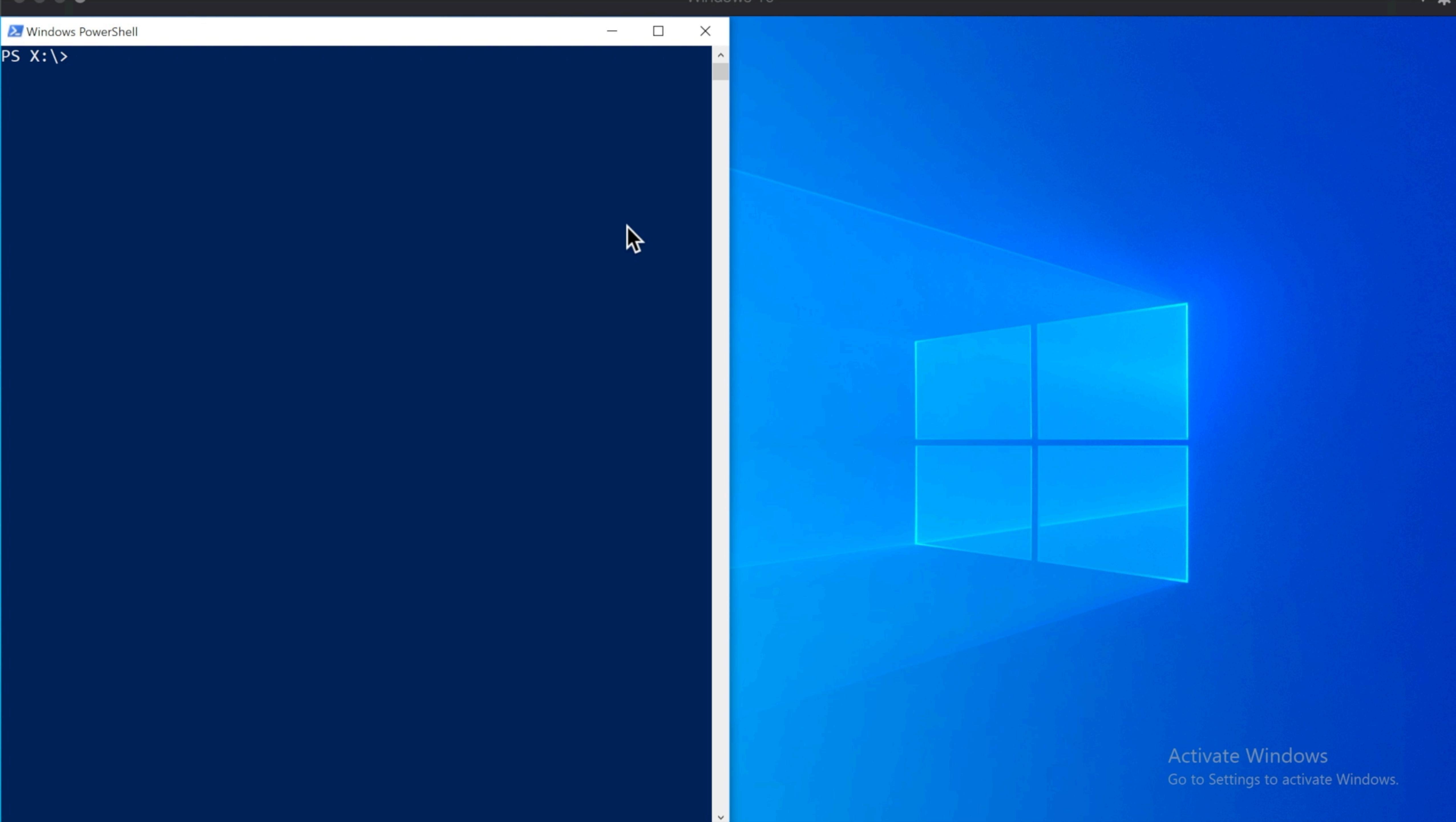
```
def run_simulation(self, nodes):  
    # Setup the test  
    self.setup_road_nodes(nodes)  
    self.vehicle_start_pose = _get_vehicle_start_pose()  
    # Define the destination  
    waypoint_goal = BeamNGWaypoint('waypoint_goal', get_node_coords(nodes[-1]))  
    # Setup runtime monitoring  
    vehicle_state_reader = VehicleStateReader(self.vehicle, self.beamng,  
                                              additional_sensors=BeamNGCarCameras())  
    sim_data_collector = SimulationDataCollector(self.vehicle, self.beamng,  
                                                vehicle_state_reader=vehicle_state_reader)  
    # Start runtime monitoring  
    sim_data_collector.get_simulation_data().start()  
    brewer.bring_up() # Start simulation  
    self.model = load_model(self.model_file) # Load the model  
    predict = NvidiaPrediction(self.model, self.config)  
    while True:  
        # Poll sensors  
        sim_data_collector.collect_current_data(oob_bb=False)  
        last_state: SimulationDataRecord = sim_data_collector.states[-1]  
        # Check oracles  
        if points_distance(last_state.pos, waypoint_goal.position) < 6.0 or last_state.is_oob:  
            break  
        # Make prediction based on camera image  
        img = vehicle_state_reader.sensors['cam_center']['colour'].convert('RGB')  
        steering_angle, throttle = predict.predict(img, last_state)  
        # Apply control command  
        self.vehicle.control(throttle=throttle, steering=steering_angle, brake=0)  
        # Progress with the simulation  
        beamng.step(steps)
```



Windows PowerShell

PS X:\>

Activate Windows
Go to Settings to activate Windows.



Windows PowerShell

PS X:\>

Activate Windows
Go to Settings to activate Windows.

Driving the ego-car from Another Process

AsFault + Path Planner

- The logic controlling the vehicles is executed **in a separate process** that uses BeamNGpy to **query the map and scenario and control the vehicle**
- Driving agent dependencies are **kept separated from the test code**, but this setup requires IPC and some form of synchronization between the test controller and the test subjects
- A path planner can control the vehicle by providing the BeamNG driving agent with a **trajectory** to follow (i.e., locations in time)

AsFault

```
def start_controller(self):          beamer.py
    l.info('Calling process: %s', self.ctrl)
    self.ctrl_process = subprocess.Popen(self.ctrl)

def kill_controller(self):
    if self.ctrl_process:
        l.info('Terminating process %s', self.ctrl)
        TestRunner.kill_process(self.ctrl_process)
        self.ctrl_process = None

@staticmethod
def kill_process(process):
    if process:
        if os.name == 'nt':
            subprocess.call(['taskkill',
                            '/F', '/T', '/PID',
                            str(process.pid)])
        else:
            os.kill(process.pid, signal.SIGTERM)
```

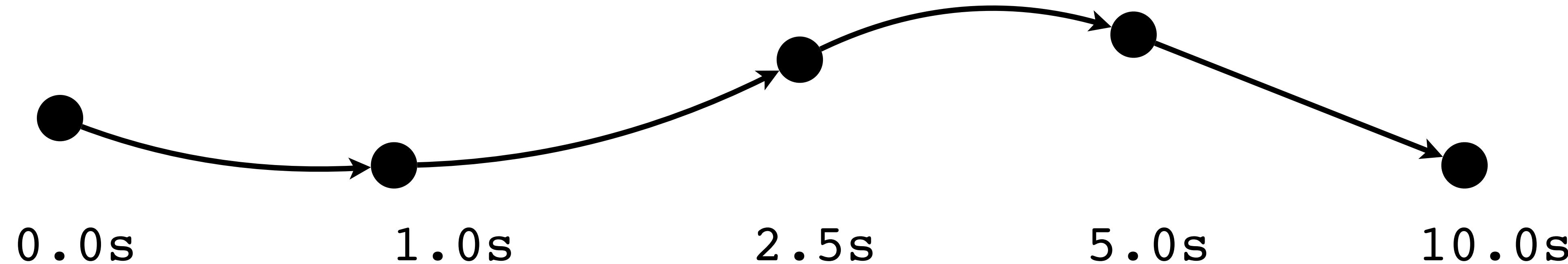
```
def run(self):
    self.bng = BeamNGpy('localhost', 64256)
    # Connect to the running BeamNG simulation
    self.bng = self.bng.open(launch=False, deploy=False)
    self.bng.pause()
    # Connect to the ego-car and configure sensors
    active_vehicles = bng.get_current_vehicles()
    self.vehicle = active_vehicles[car_model['id']]
    self.vehicle.attach_sensor('state', State())
    self.bng.connect_vehicle(self.vehicle)
    # Get initial state of the car
    self.bng.poll_sensors(self.vehicle)
    # Get geometry of the road
    road_geometry = self.bng.get_road_edges('the_road_id')
    # Compute the "optimal" trajectory
    driving_path = self._compute_driving_path(self.vehicle.state,
                                                road_geometry)
    self.ai_script = self.compute_ai_script(driving_path, self.car_model)
    # Configure BeamNG driving agent to follow the trajectory
    self.vehicle.ai_set_mode('disabled')
    self.vehicle.ai_set_script(self.script)
    # Resume the simulation (this will cause the car to move)
    self.bng.resume()
```

Anatomy of a Simulation-based Test

- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)
- Add **some dynamism** (NPC, random traffic)

Controlling NPC Vehicles

- Controlling non-playable character can be done using BeamNGpy and the available BeamNG driving AI by
 - **Setting the checkpoints** and the other configurations (e.g., speed limit, risk factor)
 - **Generating a script** that defines the trajectory **implicitly** by specifying a sequence of **nodes** (absolute positions) that must be reached by the vehicle at a given time (computed from the beginning of the simulation)



```
INTER_NODE_DISTANCE = 20.0
# Define a straight trajectory and constant speed
trajectory = []
for i in range(0, 5):
    node = {
        'x': INTER_NODE_DISTANCE * i + INITIAL_DISTANCE,
        'y': 30 - LANE_WIDTH * 0.5,
        'z': GROUND_LEVEL + 1.0,
        't': 2.5 * i
    }
    trajectory.append(node)
```

```
heading_vehicle = Vehicle('ego', model='etk800', licence='NPC', color="red")
scenario.add_vehicle(heading_vehicle, pos=initial_position, rot=None, rot_quat=direction_of_the_road)
scenario.make(bng)
bng.load_scenario(scenario)
bng.start_scenario()
```

```
# Configure the NPC
heading_vehicle.ai_set_mode('disabled')
heading_vehicle.ai_set_script(trajectory)
```

Part III

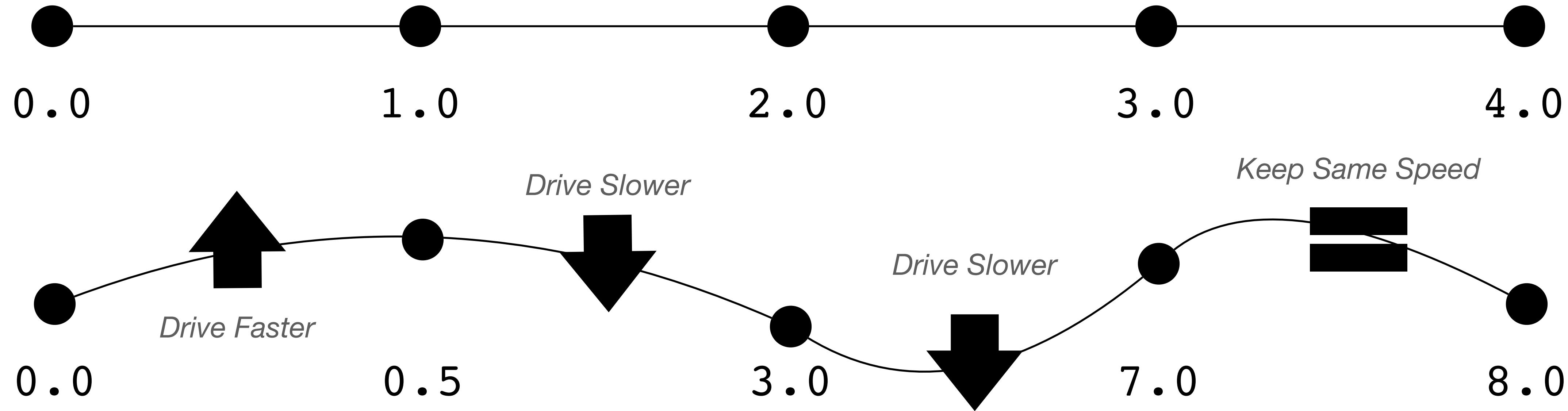
Automated test generation

Automated Test Generation

- Automated test generation can take **many** forms depending on the testing goal to achieve and the test subject. Standard examples are:
 - **Parameter exploration** (one abstract scenario, many concrete scenarios)
 - **Procedural content generation** (many abstract scenarios, many concrete scenarios)
- We show two cases:
 - Testing car-following (parameter exploration, inspired by EuroNCAP)
 - Testing lane-keeping (procedural content generation, inspired by AsFault)

Follow the Drunk Driver

- We start with the ego-car behind a NPC car and configure the NPC to follow a straight trajectory at a constant speed
- We **mutate** the trajectory by changing the **lateral position** of the node and the **inter-time** between consecutive nodes (i.e., the NPC car speed)
- We implement a basic **EA 1+1** (pop size = 1, only mutation)



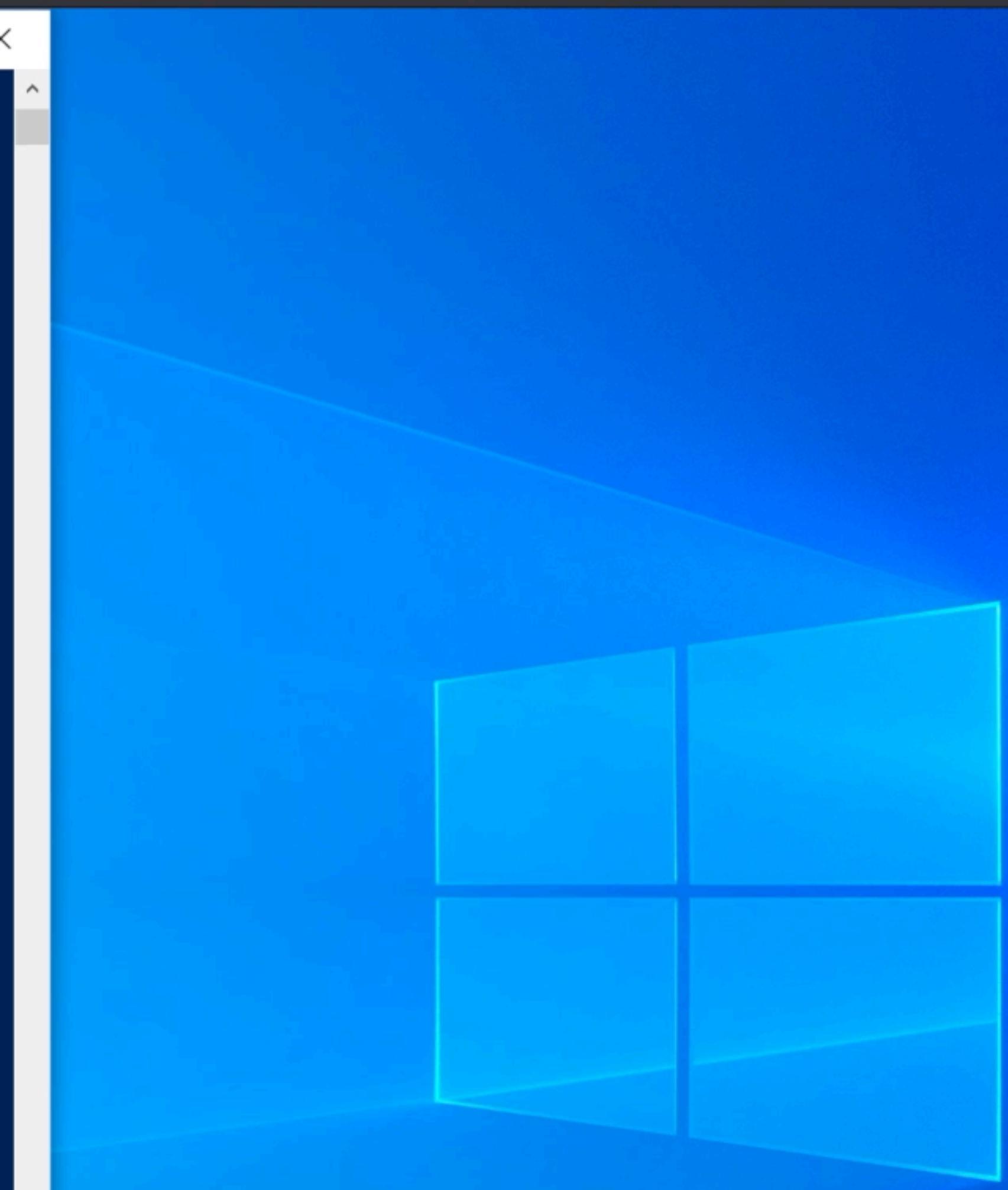
```
40 # https://stackoverflow.com/questions/6822725/rolling-or-sliding-window-iterator
41 def window(iterable, size):...
42
43
44
45
46
47 def execute_experiment(individual):...
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84 ► if __name__ == "__main__":
85     # This is the "main" Bng Client that starts and stop the simulator
86     with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:
87         main()
88
```

```
40 # https://stackoverflow.com/questions/6822725/rolling-or-sliding-window-iterator
41 def window(iterable, size):...
42
43
44
45
46
47 def execute_experiment(individual):...
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84 ► if __name__ == "__main__":
85     # This is the "main" Bng Client that starts and stop the simulator
86     with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER) as bng:
87         main()
88
```

Select Windows PowerShell

d-----	5/30/2021	1:08 AM	__pycache__
d-----	5/23/2021	9:17 PM	BeamNGpy
-----	5/27/2021	9:38 PM	4500 load_scenarios_and_roads.py
-----	4/29/2021	4:50 PM	34655136 self-driving-car-178-2020.h5
-----	5/20/2021	2:52 PM	376 example1.py
-----	5/29/2021	11:35 PM	9162 test_oracles.py
-----	5/18/2021	2:59 PM	1654 sudden_breaking.ipynb
-----	5/27/2021	9:31 PM	10593 runtime_monitoring_and_sync_simulation.py
-----	5/29/2021	11:41 PM	10299 notescratch.py
-----	5/29/2021	3:04 PM	6269 intersection_ai_examples.ipynb
-----	5/18/2021	2:59 PM	4364 intersection.ipynb
-----	5/30/2021	1:38 AM	9600 automation_example_2.py
-----	5/30/2021	12:48 AM	8124 visualization.py
-----	5/28/2021	5:47 PM	443 requirements-37.txt
-----	5/21/2021	9:30 AM	5187 README.md
-----	5/30/2021	1:44 AM	10350 automation_example_1.py
-----	5/21/2021	1:29 PM	8421 common.py
-----	5/29/2021	3:46 PM	1023 time-of-day.py
-----	5/21/2021	12:07 PM	2341 simple_test.py
-----	5/28/2021	11:02 PM	235 fork_test.py
-----	5/28/2021	11:52 PM	3591 self_driving.py
-----	5/29/2021	1:58 AM	8637 internal_vs_external_ai.py
-----	5/30/2021	1:33 AM	5907 trajectory_generator.py
-----	5/24/2021	3:26 PM	12154 place_vehicles_and_obsstacles.py
-----	4/29/2021	4:50 PM	5506 udacity_utils.py

(.venv) PS X:\>



Activate Windows

Go to Settings to activate Windows.

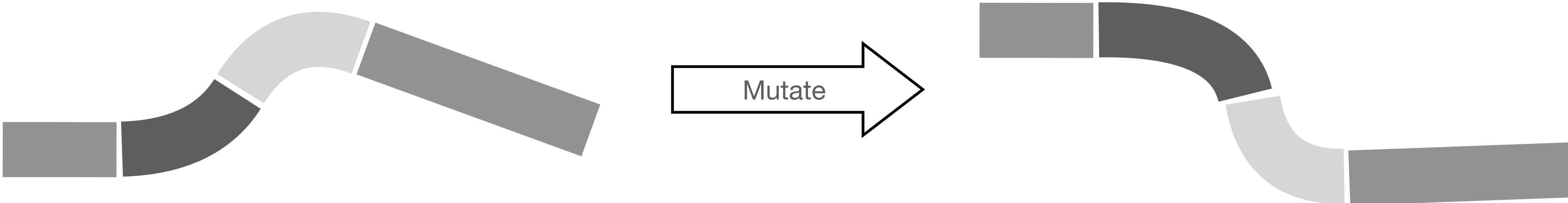
Select Windows PowerShell

```
-->-----  
d---- 5/30/2021 1:08 AM      __pycache__  
d---- 5/23/2021 9:17 PM      BeamNGpy  
----- 5/27/2021 9:38 PM      4500 load_scenarios_and_roads.py  
----- 4/29/2021 4:50 PM      34655136 self-driving-car-178-2020.h5  
----- 5/20/2021 2:52 PM      376 example1.py  
----- 5/29/2021 11:35 PM     9162 test_oracles.py  
----- 5/18/2021 2:59 PM      1654 sudden_breaking.ipynb  
----- 5/27/2021 9:31 PM      10593 runtime_monitoring_and_sync_simulation.py  
----- 5/29/2021 11:41 PM     10299 notescratch.py  
----- 5/29/2021 3:04 PM      6269 intersection_ai_examples.ipynb  
----- 5/18/2021 2:59 PM     4364 intersection.ipynb  
----- 5/30/2021 1:38 AM      9600 automation_example_2.py  
----- 5/30/2021 12:48 AM     8124 visualization.py  
----- 5/28/2021 5:47 PM      443 requirements-37.txt  
----- 5/21/2021 9:30 AM      5187 README.md  
----- 5/30/2021 1:44 AM      10350 automation_example_1.py  
----- 5/21/2021 1:29 PM      8421 common.py  
----- 5/29/2021 3:46 PM      1023 time-of-day.py  
----- 5/21/2021 12:07 PM     2341 simple_test.py  
----- 5/28/2021 11:02 PM     235 fork_test.py  
----- 5/28/2021 11:52 PM     3591 self_driving.py  
----- 5/29/2021 1:58 AM      8637 internal_vs_external_a.py  
----- 5/30/2021 1:33 AM      5907 trajectory_generator.py  
----- 5/24/2021 3:26 PM      11154 place_vehicles_and_obstacles.py  
----- 4/29/2021 4:50 PM      5506 udacity_utils.py
```

(.venv) PS X:\>

Procedurally Generate Roads

- We abstract roads as sequence of segments (straights, left-turn, right-turn) and procedurally generate "concrete" roads (**AsFault**)
- We implement a simple Genetic Algorithm
- Roads have **fixed** number of segments
- We evolve the roads by **replacing** segments or **mutating** their attributes



```
89
90     def mutate(individual):...
105
106
107     def execute_experiment(individual, road_visualizer=None):...
207
208
209     # tournament selection
210     def selection(pop, scores, k=3):...
218
219
220     # https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/
221     def main():...
272
273
274 ► if __name__ == "__main__":...
278
```

```
89
90     def mutate(individual):...
105
106
107     def execute_experiment(individual, road_visualizer=None):...
207
208
209     # tournament selection
210     def selection(pop, scores, k=3):...
218
219
220     # https://machinelearningmastery.com/simple-genetic-algorithm-from-scratch-in-python/
221     def main():...
272
273
274 ► if __name__ == "__main__":...
278
```

```
(.venv-37) PS X:\>  
Rec(.venv-37) PS X:\> deactivate  
PS X:\> ls
```

Directory: X:\

Mode	LastWriteTime	Length	Name
---	---	---	---
d----	5/30/2021 12:46 AM		__pycache__
Md----	5/23/2021 9:17 PM		BeamNGpy
-----	5/27/2021 9:38 PM	4500	load_scenarios_and_roads.py
-----	4/29/2021 4:50 PM	34655136	self-driving-car-178-2020.h5
-----	5/20/2021 2:52 PM	376	example1.py
-----	5/29/2021 11:35 PM	9162	test_oracles.py
B-----	5/18/2021 2:59 PM	1654	sudden_breaking.ipynb
-----	5/27/2021 9:31 PM	10593	runtime_monitoring_and_sync_simulation.py
-----	5/29/2021 11:41 PM	10299	notescratch.py
-----	5/29/2021 3:04 PM	6269	intersection_ai_examples.ipynb
-----	5/18/2021 2:59 PM	4364	intersection.ipynb
Re-----	5/30/2021 1:04 AM	8862	automation_example_2.py
-----	5/30/2021 12:48 AM	8124	visualization.py
-----	5/28/2021 5:47 PM	443	requirements-37.txt
-----	5/21/2021 9:30 AM	5187	README.md
T-----	5/29/2021 12:43 PM	10309	automation_example_1.py
-----	5/21/2021 1:29 PM	8421	common.py
-----	5/29/2021 3:46 PM	1023	time-of-day.py
-----	5/21/2021 12:07 PM	2341	simple_test.py
-----	5/28/2021 11:02 PM	235	fork_test.py
-----	5/28/2021 11:52 PM	3591	self_driving.py
-----	5/29/2021 1:58 AM	8637	internal_vs_external_ai.py
-----	5/21/2021 10:19 PM	5787	trajectory_generator.py
-----	5/24/2021 3:26 PM	12154	place_vehicles_and_obstacles.py
-----	4/29/2021 4:50 PM	5506	udacity_utils.py

Activate Windows

Go to Settings to activate Windows.

```
(.venv-37) PS X:\>  
Rec(.venv-37) PS X:\> deactivate  
PS X:\> ls
```

Directory: X:\

Mode	LastWriteTime	Length	Name
---	-----	-----	-----
d----	5/30/2021 12:46 AM		__pycache__
Md----	5/23/2021 9:17 PM		BeamNGpy
-----	5/27/2021 9:38 PM	4500	load_scenarios_and_roads.py
-----	4/29/2021 4:50 PM	34655136	self-driving-car-178-2020.h5
-----	5/20/2021 2:52 PM	376	example1.py
-----	5/29/2021 11:35 PM	9162	test_oracles.py
B-----	5/18/2021 2:59 PM	1654	sudden_breaking.ipynb
-----	5/27/2021 9:31 PM	10593	runtime_monitoring_and_sync_simulation.py
-----	5/29/2021 11:41 PM	10299	notescratch.py
-----	5/29/2021 3:04 PM	6269	intersection_ai_examples.ipynb
-----	5/18/2021 2:59 PM	4364	intersection.ipynb
Re-----	5/30/2021 1:04 AM	8862	automation_example_2.py
-----	5/30/2021 12:48 AM	8124	visualization.py
-----	5/28/2021 5:47 PM	443	requirements-37.txt
-----	5/21/2021 9:30 AM	5187	README.md
T-----	5/29/2021 12:43 PM	10309	automation_example_1.py
-----	5/21/2021 1:29 PM	8421	common.py
-----	5/29/2021 3:46 PM	1023	time-of-day.py
-----	5/21/2021 12:07 PM	2341	simple_test.py
-----	5/28/2021 11:02 PM	235	fork_test.py
-----	5/28/2021 11:52 PM	3591	self_driving.py
-----	5/29/2021 1:58 AM	8637	internal_vs_external_ai.py
-----	5/21/2021 10:19 PM	5787	trajectory_generator.py
-----	5/24/2021 3:26 PM	12154	place_vehicles_and_obstacles.py
-----	4/29/2021 4:50 PM	5506	udacity_utils.py

Activate Windows

Go to Settings to activate Windows.

Useful Links

- <https://sbst21.github.io/>
- <https://documentation.beamng.com/>
- <https://github.com/se2p/sbst-2021-tutorial>
- <https://github.com/se2p/tool-competition-av>
- <https://github.com/BeamNG/BeamNGpy>
- <https://github.com/alessiogambi/AsFault/tree/asfault-deap/src>
- <https://github.com/testingautomated-usi/DeepJanus/tree/master/DeepJanus-BNG>
- <https://github.com/TriHuynh00/AC3R-Demo>

Simulation-based Testing with BeamNG.tech

Tutorial @ SBST 2021



Alessio Gambi
University of Passau



Pascale Maul
BeamNG



Marc Müller
BeamNG