Simulation-based Testing with BeamNG.tech

Tutorial @ SBST 2021

Alessio Gambi, Pascale Maul, Marc Müller

The Team

Dr. Alessio Gambi

University of Passau, Germany

The Team

Message

Pascale Maul · 3rd

MSc Informatik

Dresden, Saxony, Germany · 37 connections · **Contact info**

BeamNG GmbH

Universität Bremen

Message

Marc M. · 3rd

Programmer at BeamNG GmbH

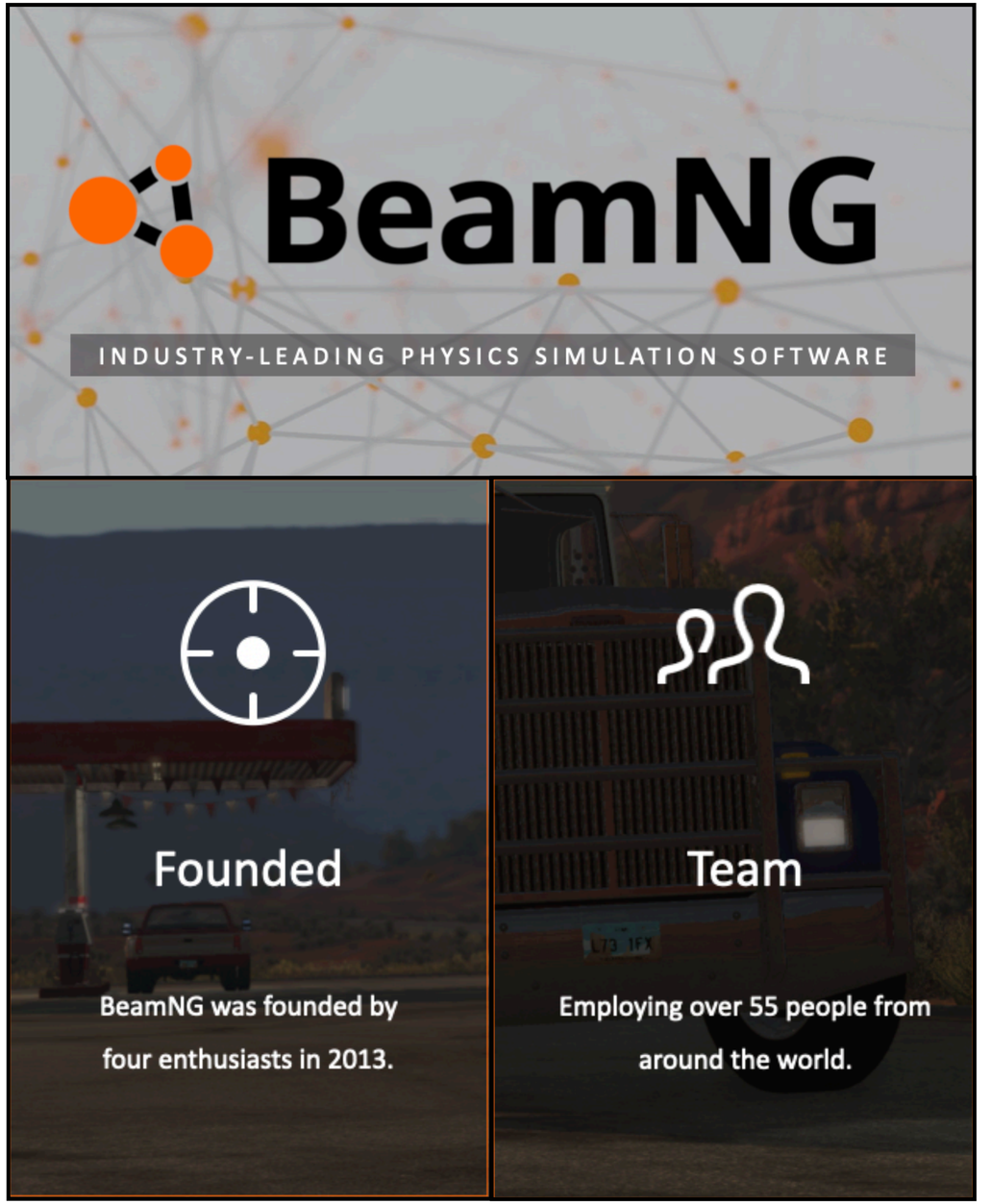
Bremen, Bremen, Germany · 63 connections · **Contact info**

BeamNG GmbH

Universität des Saarlandes

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) with overlapping work in the **gaming** industry
- Specialized on driving simulations, but "can simulate everything" (e.g., mods exists for drones, pedestrians, & more)



BeamNG/BeamNGpy: Python A

github.com/BeamNG/BeamNGpy

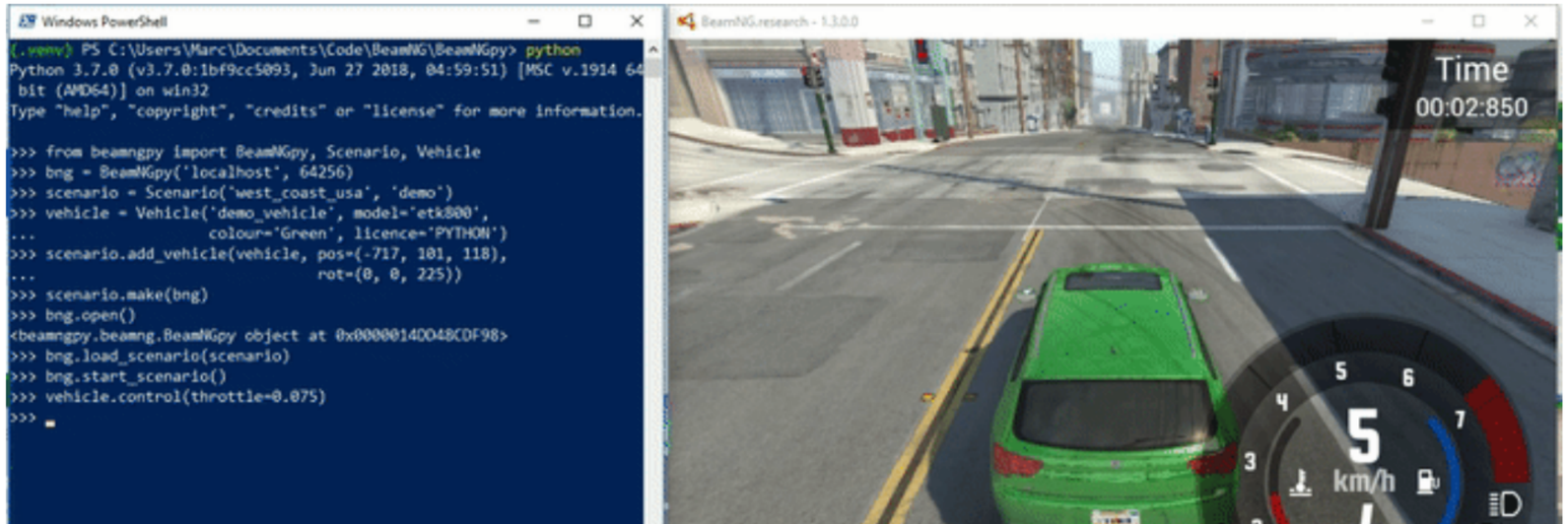
BeamNGpy

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:



BeamNG/BeamNGpy: Python A

README.md

github.com/BeamNG/BeamNGpy

BeamNGpy

BeamNG.research

Documentation

Abou

BeamN

game B

It allows

```
(.venv) PS C:\Users\Marc\Documents\Code\BeamNG\BeamNGpy> python
Python 3.7.0 (v3.7.0:1bf9cc5093, 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 0x0000014DD48CDF98>
>>> bng.load_scenario(scenario)
>>> bng.start_scenario()
>>> vehicle.control(throttle=0.075)
>>> _
```

Spoiler Alert:

Linux support will be available soon!

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

AsFault

Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation

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

Marc Mueller
mmueller@beamng.gmbh
BeamNG GmbH
Bremen, Germany

Gordon Fraser
gordon.fraser@uni-passau.de
University of Passau
Passau, Germany

DeepJanus

Model-based Exploration of the Frontier of Behaviours for Deep Learning System Testing

Vincenzo Riccio
Software Institute - USI
Lugano, Switzerland
vincenzo.riccio@usi.ch

Paolo Tonella
Software Institute - USI
Lugano, Switzerland
paolo.tonella@usi.ch

AC3R

Generating Effective Test Cases for Self-Driving Cars from Police Reports

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

Tri Huynh
huynhminhtri00@gmail.com
Saarland University/CISPA
Saarbruecken, Germany

Gordon Fraser
gordon.fraser@uni-passau.de
University of Passau
Passau, Germany

Driving Simulator Validation of Machine Learning Classification for a Surface Electromyography-Based Steering Assistance Interface

Edric John Nacpil(✉) and Kimihiko Nakano

Institute of Industrial Science, The University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
{enacpil, knakano}@iis.u-tokyo.ac.jp

Using Wearable Sensors to Detect Workload on Driving Simulated Scenarios

Víctor CORCOBA[a,1], Zoe FALOMIR[b], Nourelhoda MOHAMED[c]
[a] *Informatics Department, University of Oviedo, Spain*
[b] *Bremen Spatial Cognition Centre (BSCC), University of Bremen, Bremen, Germany*
[c] *BeamNG GmbH, Bremen, Germany*

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

AsFault

Automatically Testing Self-Driving Cars with Search-Based Procedural Content Generation

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

Marc Mueller
mmueller@beamng.gmbh
BeamNG GmbH
Bremen, Germany

Gordon Fraser
gordon.fraser@uni-passau.de
University of Passau
Passau, Germany

DeepJanus

Model-based Exploration of the Frontier of Behaviours for Deep Learning System Testing

Vincenzo Riccio
Software Institute - USI
Lugano, Switzerland
vincenzo.riccio@usi.ch

Paolo Tonella
Software Institute - USI
Lugano, Switzerland
paolo.tonella@usi.ch

AC3R

Generating Effective Test Cases for Self-Driving Cars from Police Reports

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

Tri Huynh
huynhminhtri00@gmail.com
Saarland University/CISPA
Saarbruecken, Germany

Gordon Fraser
gordon.fraser@uni-passau.de
University of Passau
Passau, Germany

Driving Simulator Validation of Machine Learning Classification for a Surface Electromyography-Based Steering Assistance Interface

Edric John Nacpil(✉) and Kimihiko Nakano

Institute of Industrial Science, The University of Tokyo, 4-6-1 Komaba, Meguro-ku, Tokyo 153-8505, Japan
{enacpil, knakano}@iis.u-tokyo.ac.jp

Using Wearable Sensors to Detect Workload on Driving Simulated Scenarios

Víctor CORCOBA[a,1], Zoe FALOMIR[b], Nourelhoda MOHAMED[c]
[a] *Informatics Department, University of Oviedo, Spain*
[b] *Bremen Spatial Cognition Centre (BSCC), University of Bremen, Bremen, Germany*
[c] *BeamNG GmbH, Bremen, Germany*

NEW

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

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

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

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

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, also you will use it, maybe?

Goals of this Tutorial

It's a *Beginners* tutorial

- Part I: Get a grip on BeamNG.tech (5min)
- Part II: Basics of simulation-based tests (15min)
- Part III: Examples of automated test generation (10min)

Goals of this Tutorial

It's a *Beginners* tutorial

- Part I: Get a grip on BeamNG.tech (5min)
- Part II: Basics of simulation-based tests (15min)
- Part III: Examples of automated test generation (10min)

***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

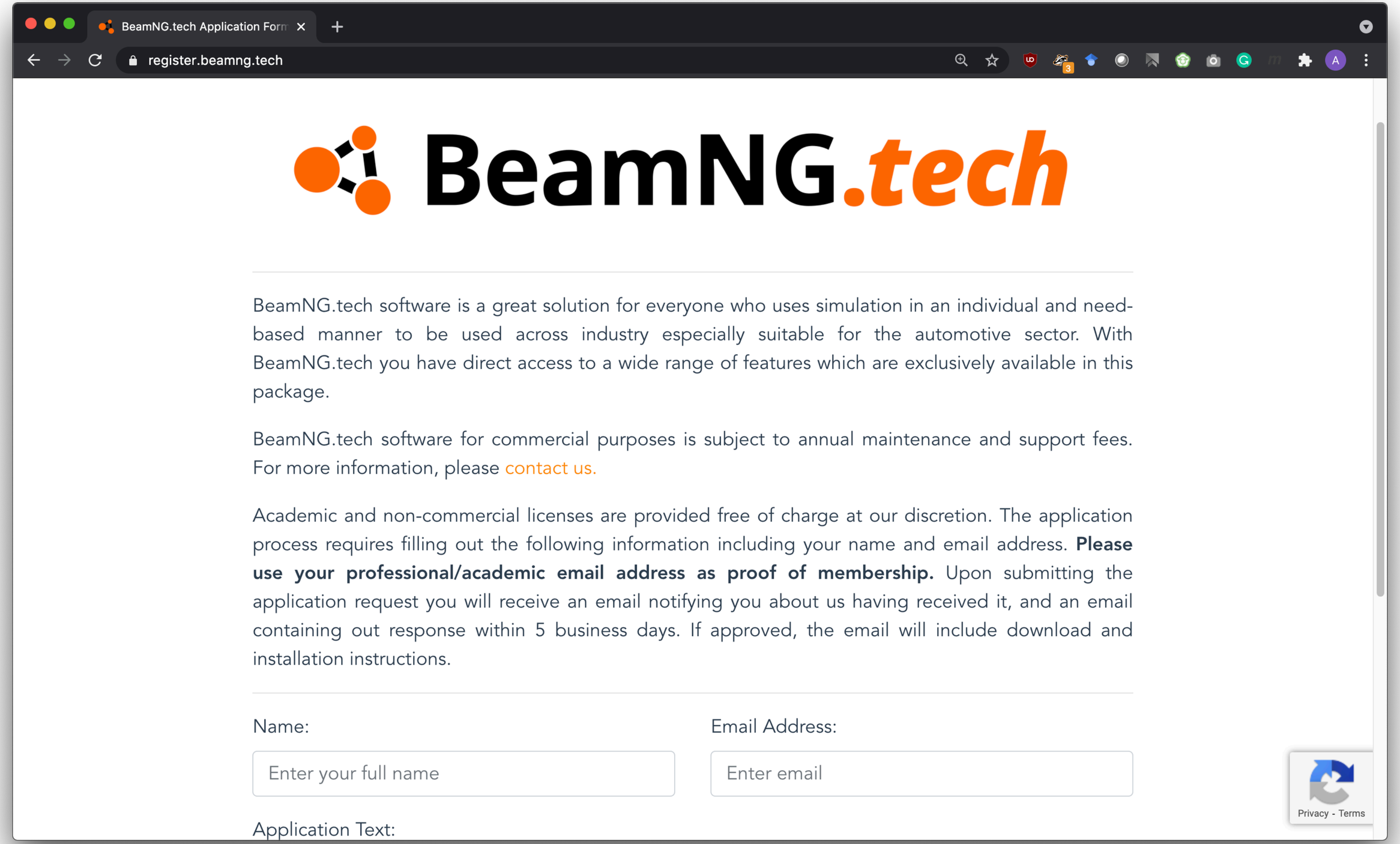
https://github.com/se2p/sbst-2021-tutorial

- 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/


BeamNG.tech Application Form
register.beamng.tech
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 out 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:
Privacy - Terms

Obtaining the Simulator

Getting Started

- The simulator can be obtained **for free** after registering at https://register.beamng.tech/

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 expand 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 expand 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"

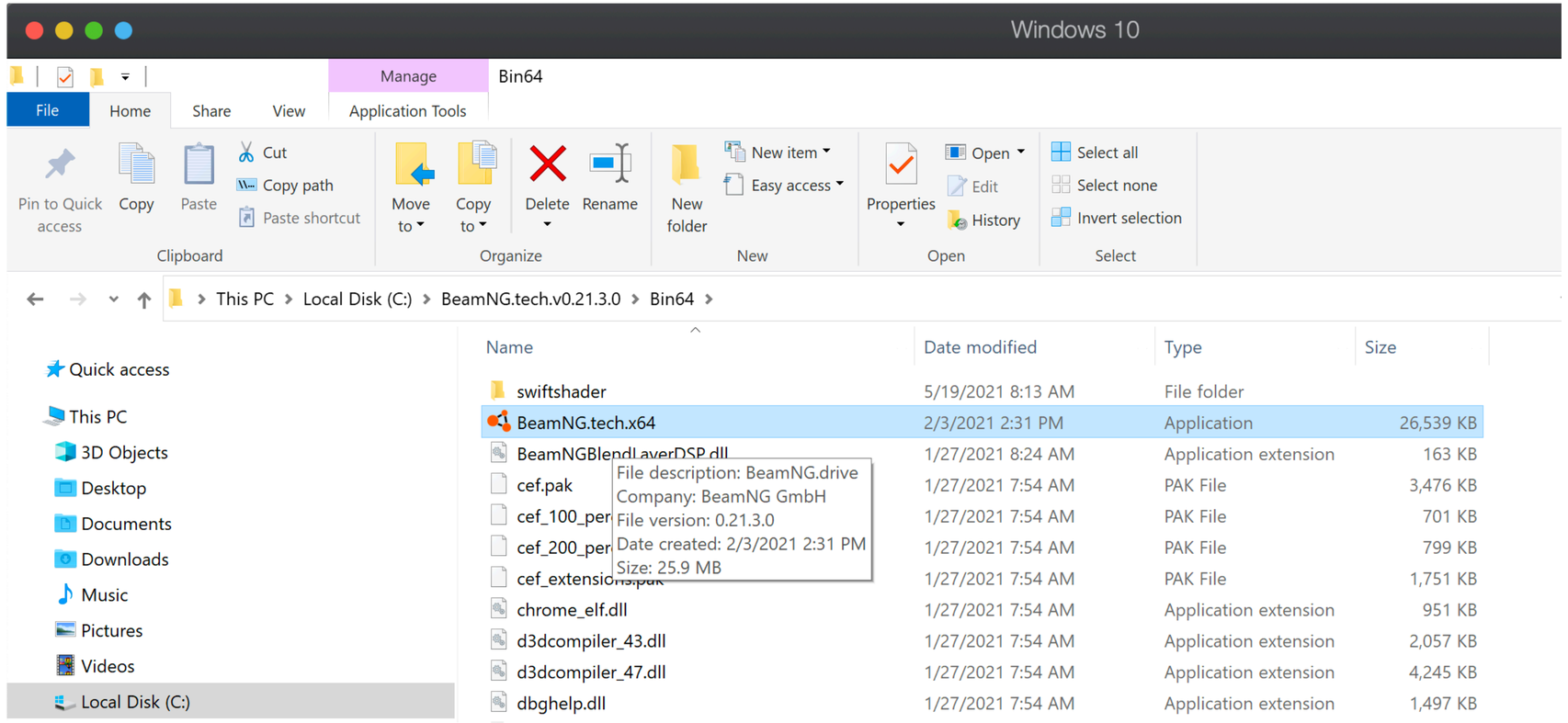
Officially, the simulator can run only under Windows, but you can try it also on Mac OS using Parallels (note, this is **shareware**)

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

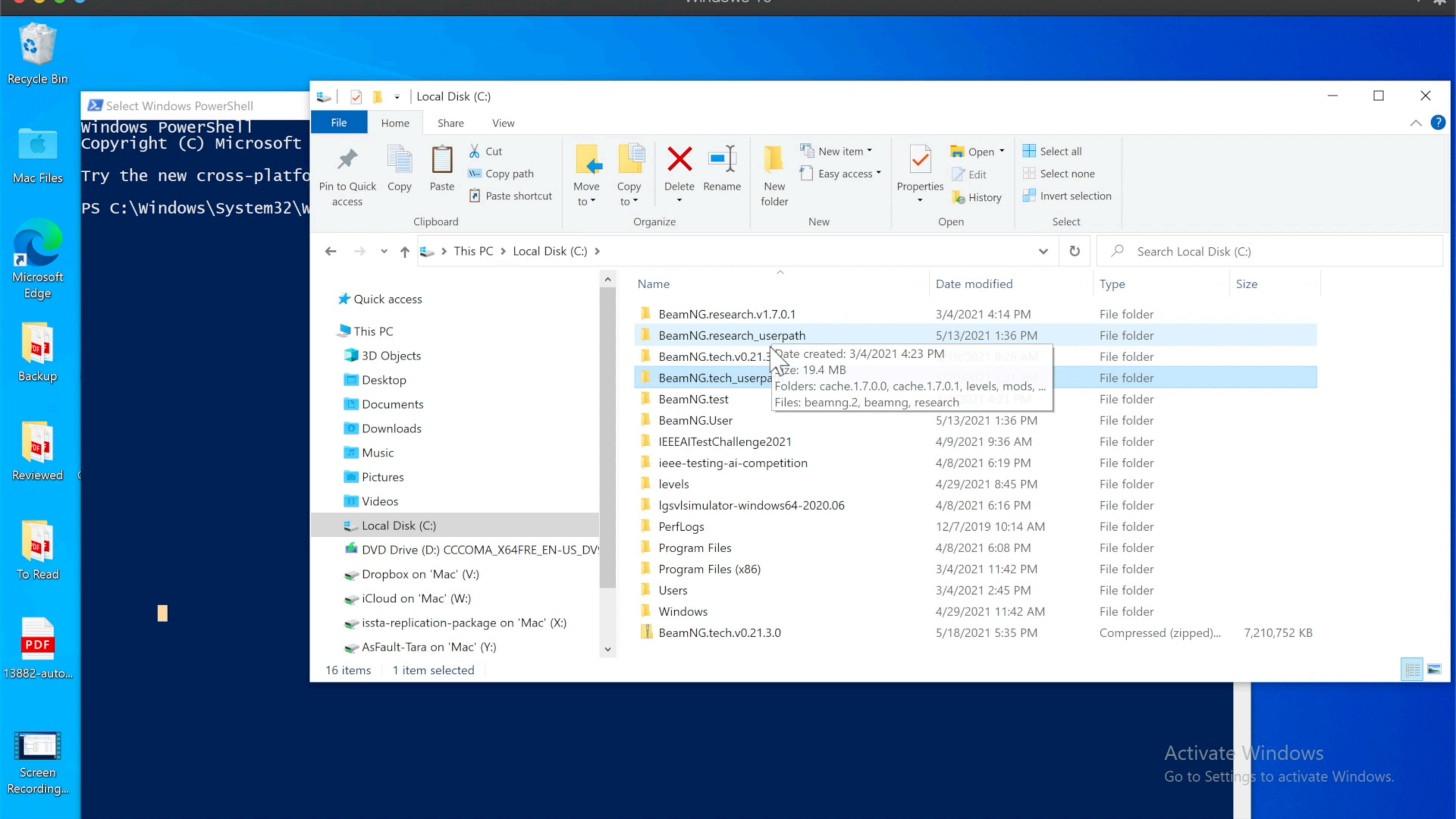
Getting Started

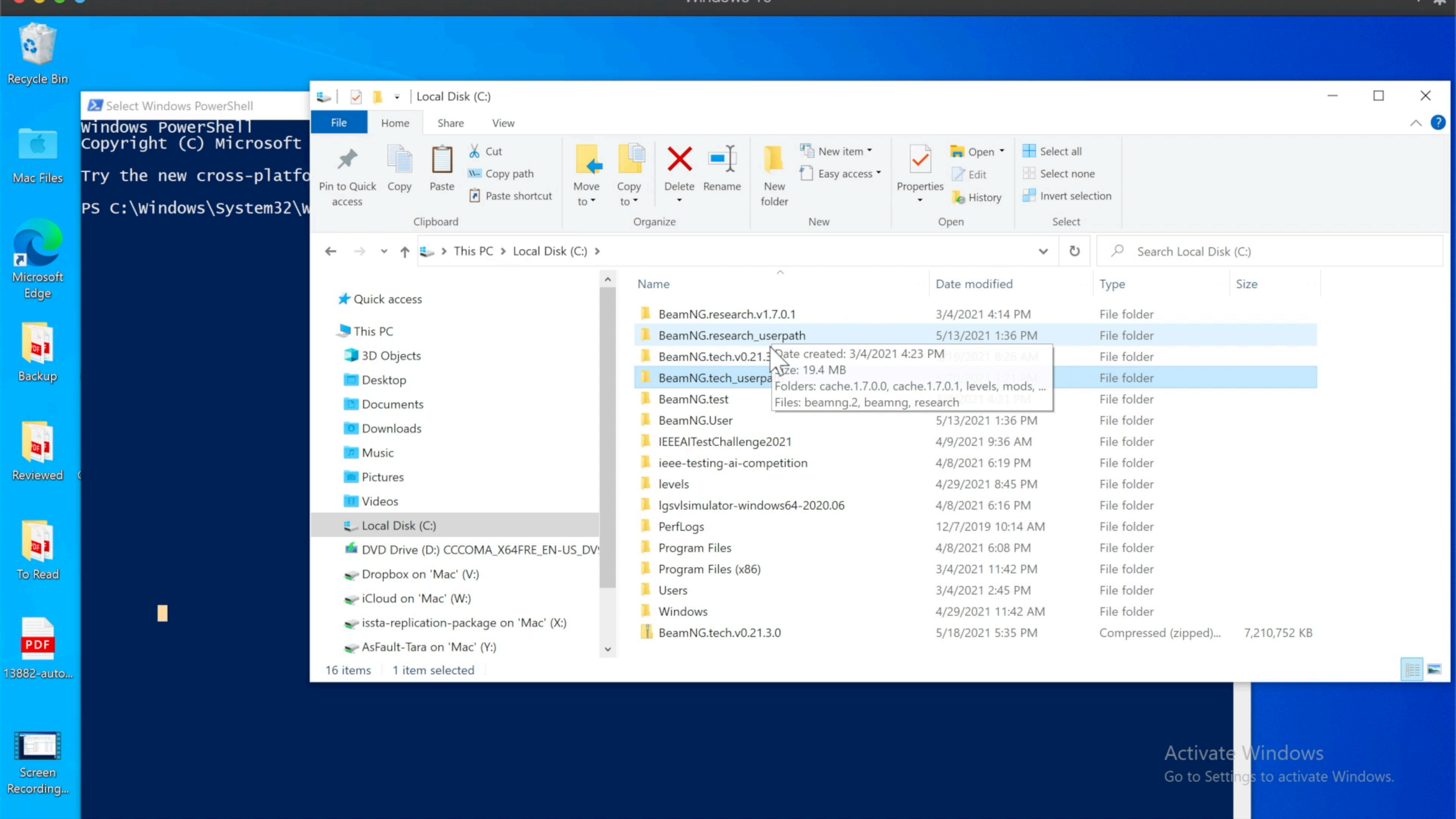
- The simulator can be **started manually** by double-clicking on

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

- Note that starting the simulator manually will **not automatically start the Python API** so you will not able to control remotely the simulator.

- However, you can still use the simulator to browse existing maps and create new content using the various editors


Recycle Bin
Mac Files
Microsoft Edge
Backup
Reviewed
To Read
13882-auto...
Screen Recording...
Select Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft
Try the new cross-platfo
PS C:\Windows\System32\W
Local Disk (C:)
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC
Local Disk (C:)
Search Local Disk (C:)
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCCOMA_X64FRE_EN-US_DV
Dropbox on 'Mac' (V:)
iCloud on 'Mac' (W:)
issta-replication-package on 'Mac' (X:)
AsFault-Tara on 'Mac' (Y:)
Name
Date modified
Type
Size
BeamNG.research.v1.7.0.1 3/4/2021 4:14 PM File folder
BeamNG.research_userpath 5/13/2021 1:36 PM File folder
BeamNG.tech.v0.21.3 File folder
BeamNG.tech_userpa File folder
Date created: 3/4/2021 4:23 PM
Size: 19.4 MB
Folders: cache.1.7.0.0, cache.1.7.0.1, levels, mods, ...
Files: beamng.2, beamng, research
BeamNG.test File folder
BeamNG.User 5/13/2021 1:36 PM File folder
IEEEAITestChallenge2021 4/9/2021 9:36 AM File folder
ieee-testing-ai-competition 4/8/2021 6:19 PM File folder
levels 4/29/2021 8:45 PM File folder
lgsvlsimulator-windows64-2020.06 4/8/2021 6:16 PM File folder
PerfLogs 12/7/2019 10:14 AM File folder
Program Files 4/8/2021 6:08 PM File folder
Program Files (x86) 3/4/2021 11:42 PM File folder
Users 3/4/2021 2:45 PM File folder
Windows 4/29/2021 11:42 AM File folder
BeamNG.tech.v0.21.3.0 5/18/2021 5:35 PM Compressed (zipped)... 7,210,752 KB
16 items
1 item selected
Activate Windows
Go to Settings to activate Windows.


Recycle Bin
Mac Files
Microsoft Edge
Backup
Reviewed
To Read
13882-auto...
Screen Recording...
Select Windows PowerShell
Windows PowerShell
Copyright (C) Microsoft
Try the new cross-platfo
PS C:\Windows\System32\W
Local Disk (C:)
File
Home
Share
View
Pin to Quick access
Copy
Paste
Cut
Copy path
Paste shortcut
Clipboard
Move to
Copy to
Delete
Rename
Organize
New folder
New item
Easy access
New
Properties
Open
Edit
History
Open
Select all
Select none
Invert selection
Select
This PC
Local Disk (C:)
Search Local Disk (C:)
Quick access
This PC
3D Objects
Desktop
Documents
Downloads
Music
Pictures
Videos
Local Disk (C:)
DVD Drive (D:) CCCOMA_X64FRE_EN-US_DV
Dropbox on 'Mac' (V:)
iCloud on 'Mac' (W:)
issta-replication-package on 'Mac' (X:)
AsFault-Tara on 'Mac' (Y:)
Name
Date modified
Type
Size
BeamNG.research.v1.7.0.1 3/4/2021 4:14 PM File folder
BeamNG.research_userpath 5/13/2021 1:36 PM File folder
BeamNG.tech.v0.21.3 File folder
BeamNG.tech_userpa File folder
Date created: 3/4/2021 4:23 PM
Size: 19.4 MB
Folders: cache.1.7.0.0, cache.1.7.0.1, levels, mods, ...
Files: beamng.2, beamng, research
BeamNG.test File folder
BeamNG.User 5/13/2021 1:36 PM File folder
IEEEAITestChallenge2021 4/9/2021 9:36 AM File folder
ieee-testing-ai-competition 4/8/2021 6:19 PM File folder
levels 4/29/2021 8:45 PM File folder
lgsvlsimulator-windows64-2020.06 4/8/2021 6:16 PM File folder
PerfLogs 12/7/2019 10:14 AM File folder
Program Files 4/8/2021 6:08 PM File folder
Program Files (x86) 3/4/2021 11:42 PM File folder
Users 3/4/2021 2:45 PM File folder
Windows 4/29/2021 11:42 AM File folder
BeamNG.tech.v0.21.3.0 5/18/2021 5:35 PM Compressed (zipped)... 7,210,752 KB
16 items
1 item selected
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** start

Running the Simulator

Getting Started

- The simulator can be **started** (and **controlled**) using its Python API: `BeamNGpy`
- `BeamNGpy` is open source and available both on GitHub and 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.
- `<BNG_USER>` instead is not mandatory. If not specified, `BeamNGpy` will default its value 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

```python
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

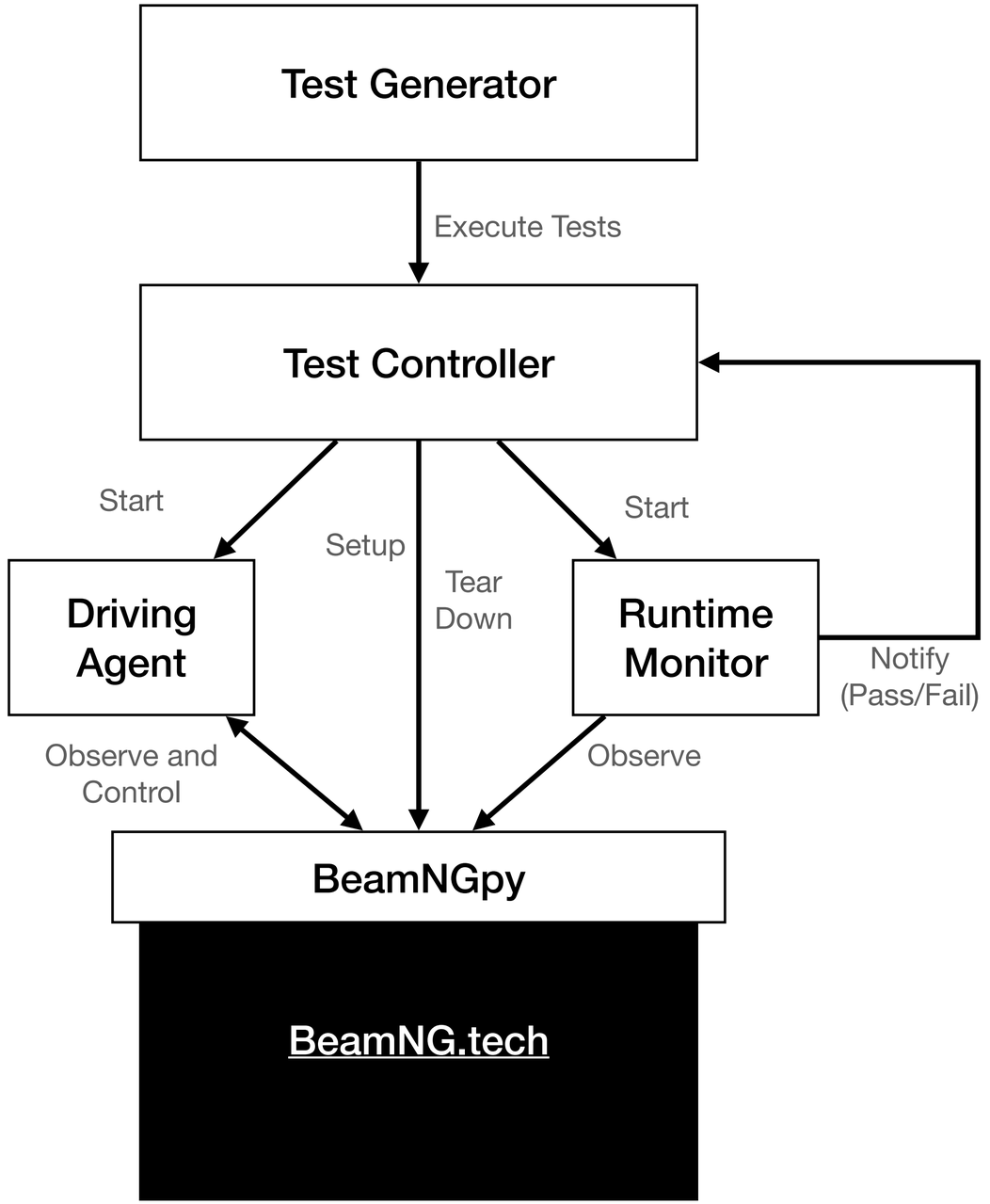
Introduction

- 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

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



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

```
def do_the_sleep(for_seconds):
    for i in reversed(range(1, for_seconds, )):
        print(i)
        sleep(1)

class StartFreshInstanceOfTheSimulator(unittest.TestCase):

    def test_that_simulation_start(self):
        with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER):
            do_the_sleep(5)

    def test_that_simulation_restart(self):
        with BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER):
            do_the_sleep(3)
```

simple-test.py

Fresh Instance (2)

```python
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_the_sleep(5)

    def test_that_simulation_restart(self):
        do_the_sleep(3)
```

simple-test.py

```python
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

Shared Instance

simple-test.py

```python
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()
```

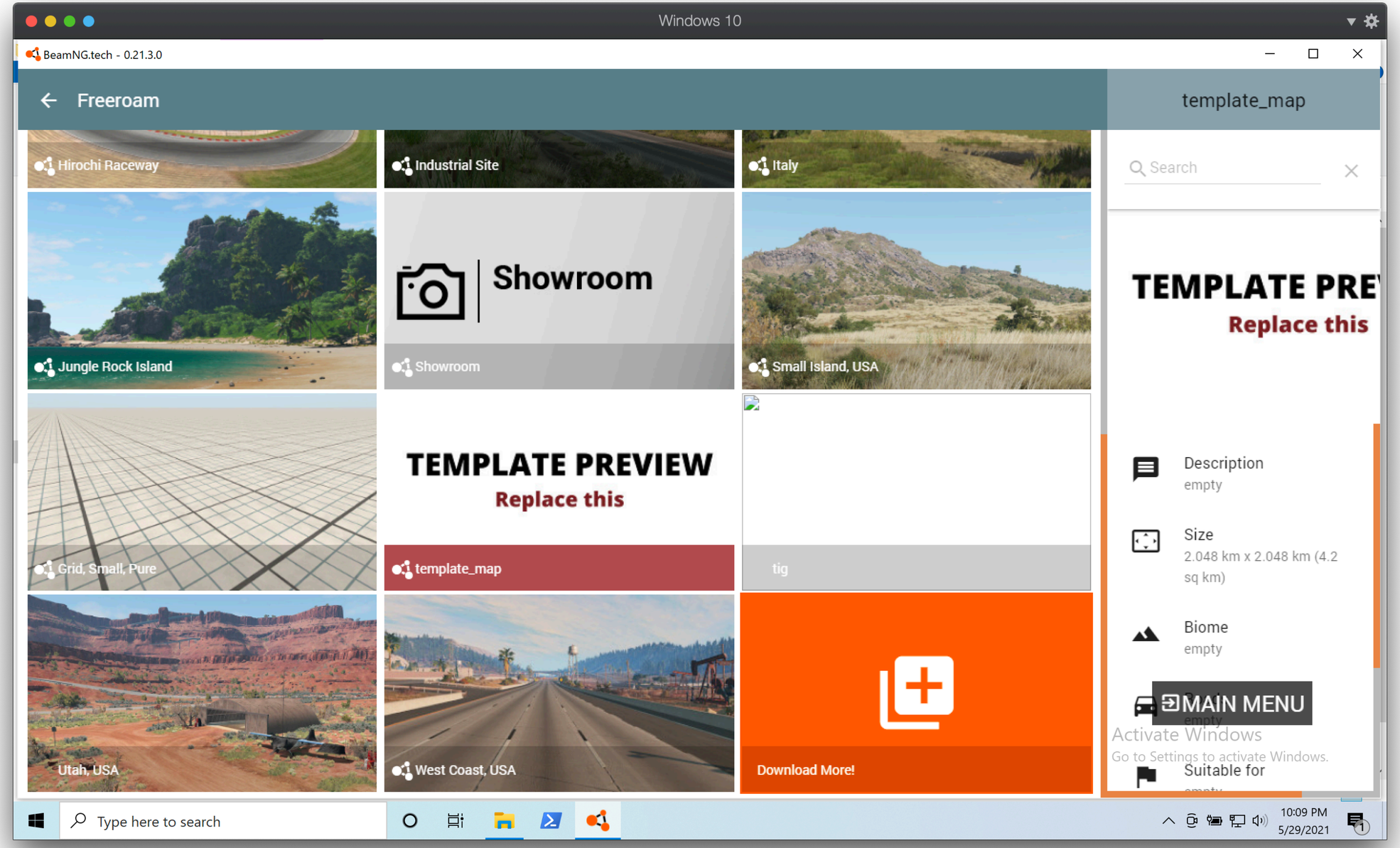
```python
def test_that_can_connect_to_simulator(self):
    client_a = BeamNGpy('localhost', 64256, home=BNG_HOME, user=BNG_USER)
    try:
        client_a.open(launch=False, deploy=False)
        do_the_sleep(2)
    finally:
        client_a.skt.close()

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_the_sleep(5)
    finally:
        client_b.skt.close()
```

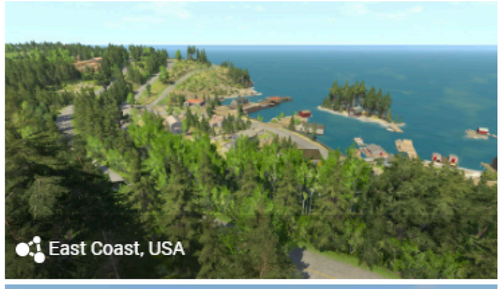
Anatomy of a Simulation-based Test

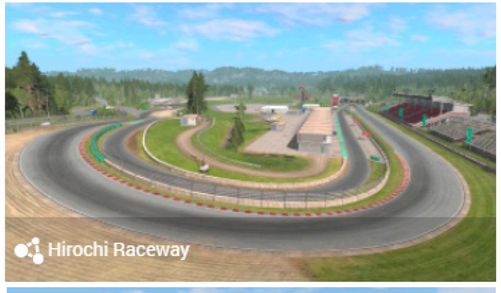
- **Precondition:** A running simulator
- Set the **environment** (terrain, map, roads)
- Set the test **initial conditions** (placement of vehicles, obstacles)
- Configure **runtime monitors** (positive/negative oracles)
- Configure the **test subject** (connect to ego-car, instruct about test goal/task)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

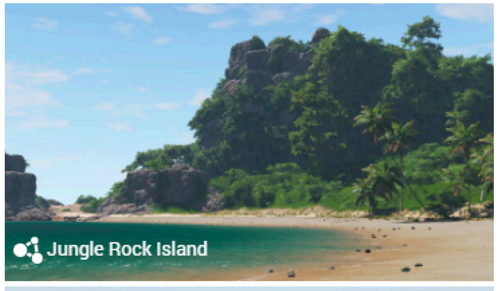
Levels and Scenarios

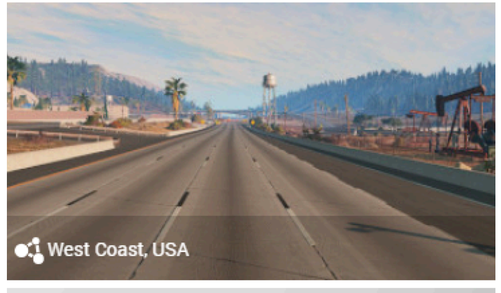
- Simulation-based tests with BeamNG take place inside **scenarios**
- Scenarios are defined in the context of **levels**
- Levels contain **terrains**, **maps**, **props/arts**, **materials**, and other meta-data required to execute the simulation
- **Levels** must be **manually** defined, **scenarios** 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>`

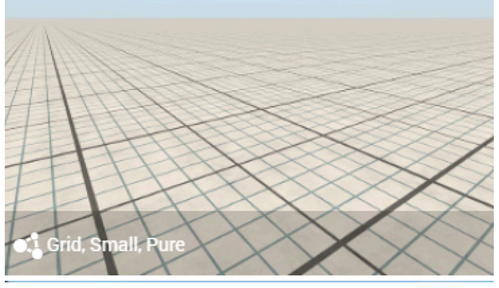

Windows 10
BeamNG.tech - 0.21.3.0
Freeroam
template_map
Hirochi Raceway
Industrial Site
Italy
Search
Jungle Rock Island
Showroom
Showroom
Small Island, USA
TEMPLATE PRE
Replace this
Grid, Small, Pure
TEMPLATE PREVIEW
Replace this
template_map
tig
Description
empty
Size
2.048 km x 2.048 km (4.2 sq km)
Biome
empty
MAIN MENU
Utah, USA
West Coast, USA
Download More!
Suitable for
Activate Windows
Go to Settings to activate Windows.
Type here to search
10:09 PM
5/29/2021

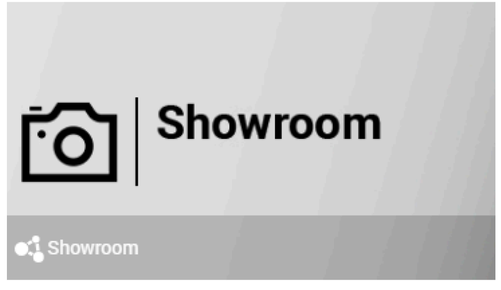












This PC › Local Disk (C:) › BeamNG.tech.v0.21.3.0 › levels ›

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

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

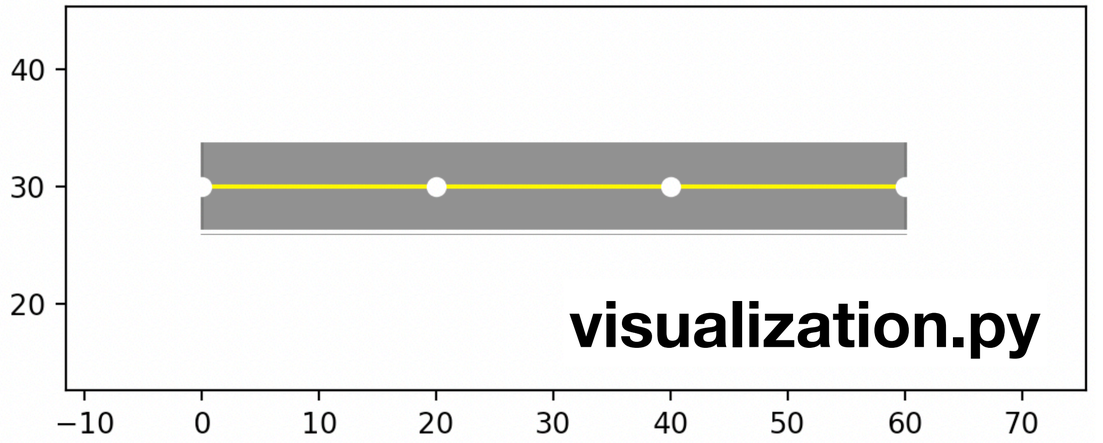
load_scenario_and_roads.py

Procedural Road Generation

- Roads can be **procedurally generated** and added to existing maps
- Roads geometry is defined as sequences of road nodes, i.e., points with `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



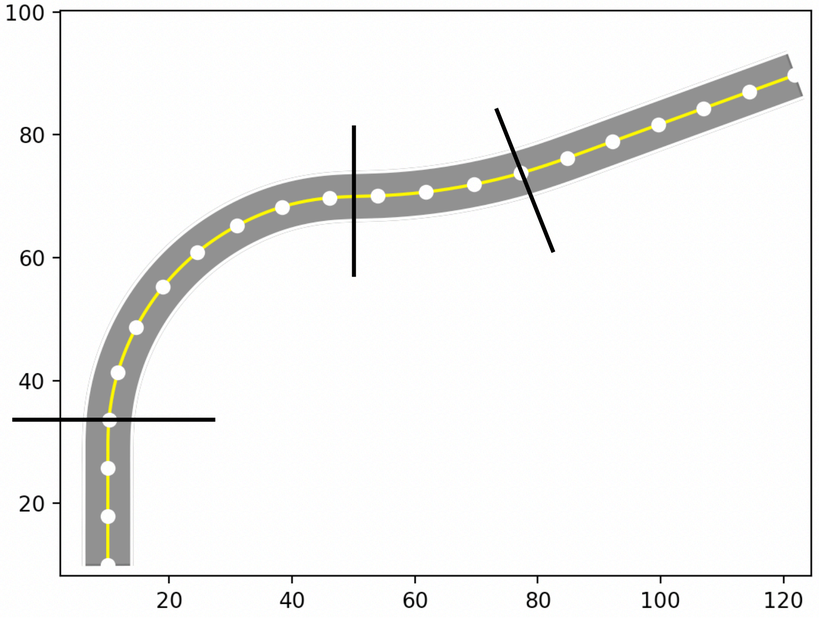
visualization.py

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 (**AsFault**)

```
driving_actions.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 **DeepJanus**, **DeepHyperion**, and the **Tool Competition Pipeline**

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

```
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**)
- A similar technique can be used to add skid marks, dirt, and other effects on the road

trajectory_generator.py

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)
```

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)

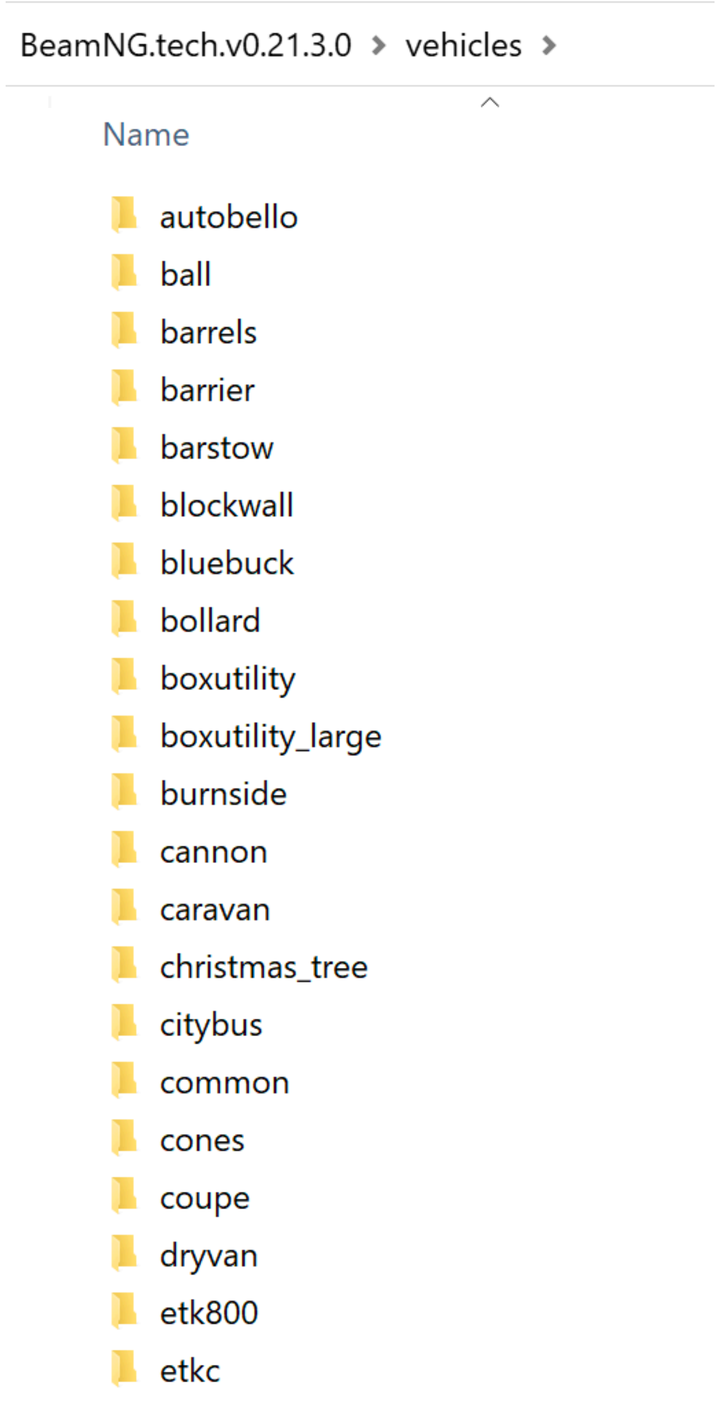
Placing Vehicles and Objects

- BeamNG allows multiple **vehicles** and **objects** to be **placed** at arbitrary locations and **rotated** in any direction

- **Vehicles** have a **model.** Models are listed in the `<BNG_HOME>\vehicles` folder. The folder contains also other complex objects composed of parts (e.g., a *cannon* really?!).

- **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



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)
```

Placing Vehicles

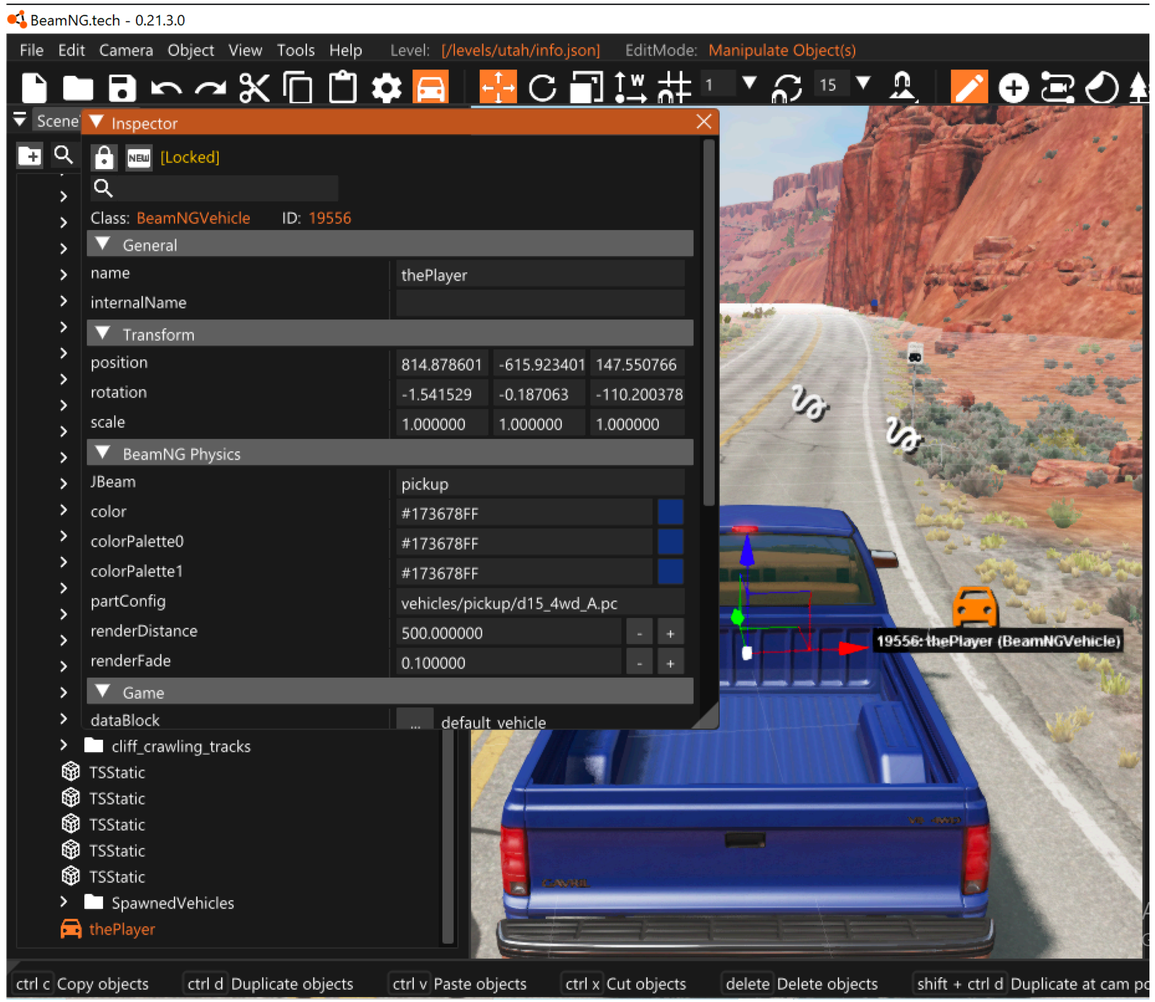
- Vehicles can collide with other vehicles and objects, but **cannot** be placed "mid-air" nor in the ground. Otherwise, they **fall and crash** or **crash immediately**

```
#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)
```

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)
```

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)

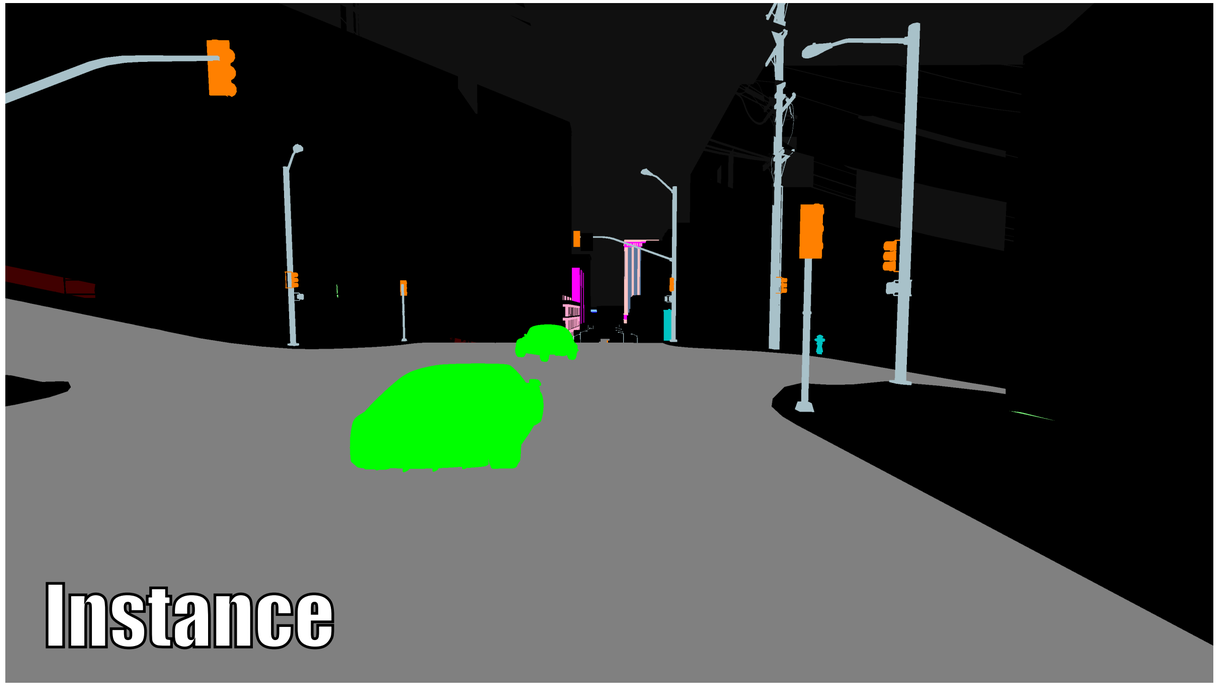
Positive vs Negative Oracles

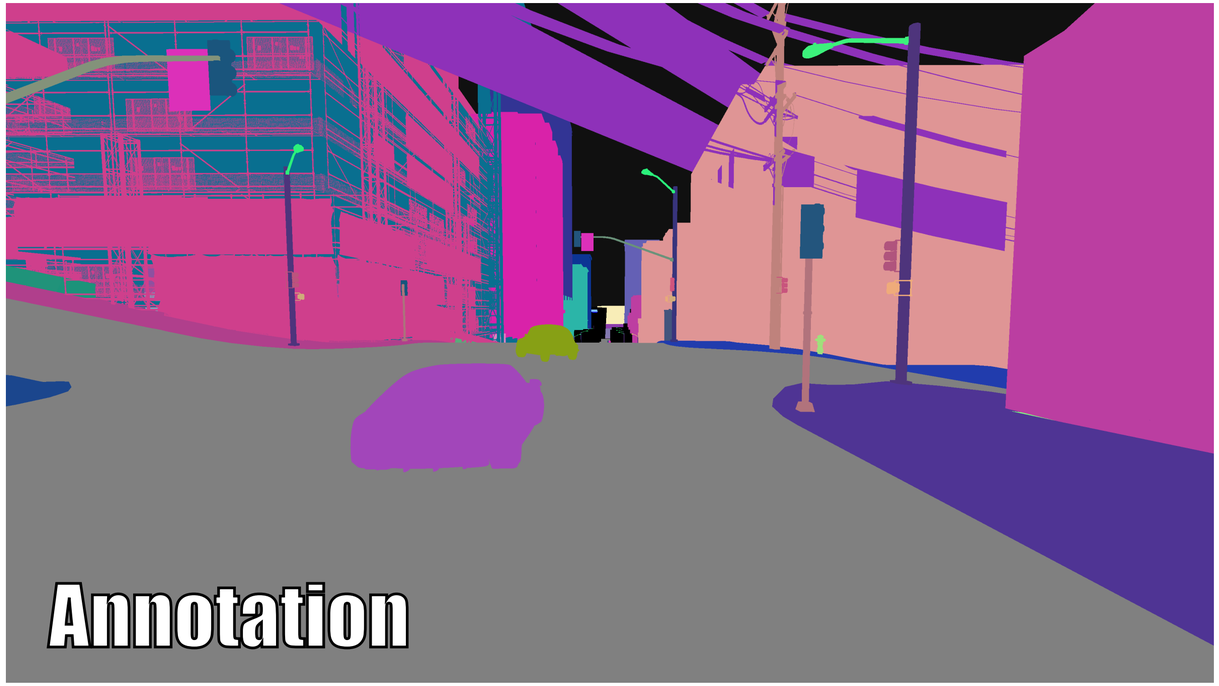
- Testing **continuous controllers** requires the definition of **positive oracles** to establish when a test **passes** and **negative oracles** to decide when it **fails**
- Examples of positive oracles are **target areas** that the ego-car must reach or traverse.
- Examples of negative oracles are **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, gear, steering wheel pos).
- The data required to evaluate the oracles can be collected by **polling** the simulator via the **sensors** abstractions.
- Sensors must be **attached** to vehicles
- BeamNGpy implements many sensors, including `State`, `Timer`, `Camera`, `Electrics`, `Lidar`, and more

Example Sensor: Camera





Example of Oracle based on Sensors

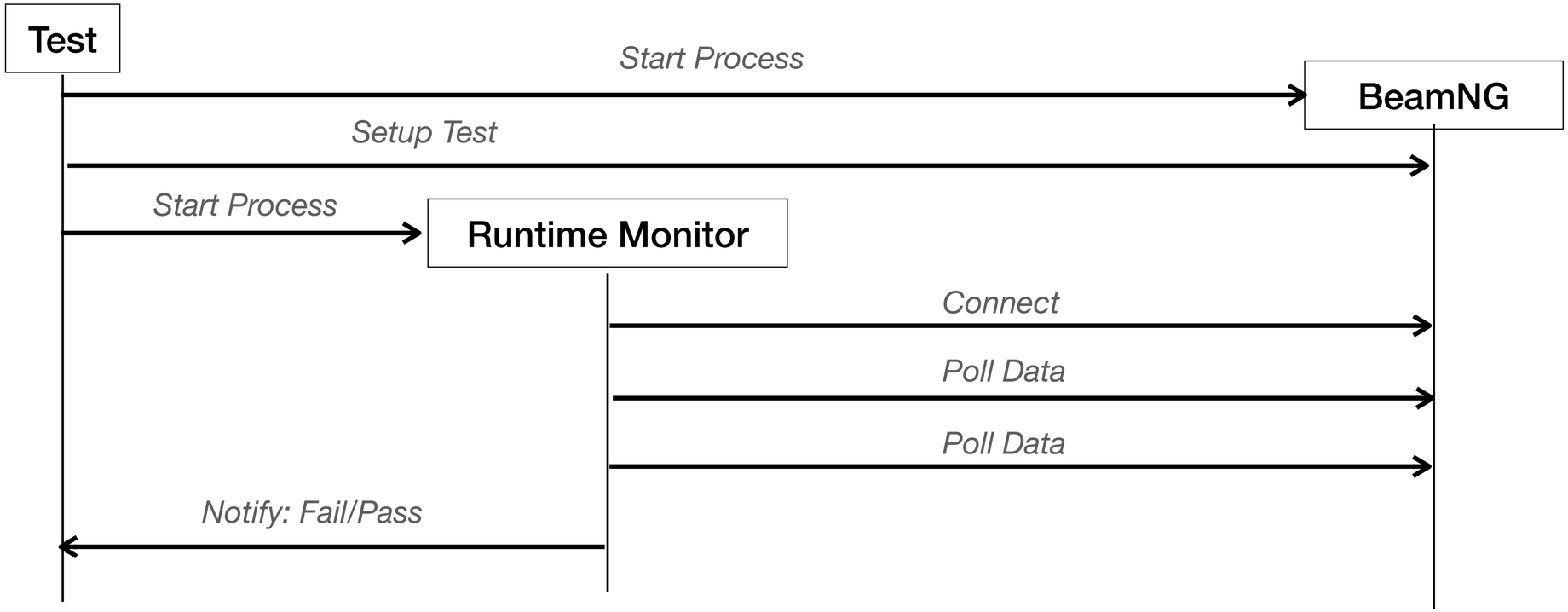
```python
# This class implements an oracle to check if the vehicle reached a target location (circular area)
# It uses the State sensor to get the position of the vehicle
from shapely.geometry import Point

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
```

Using the Oracle based on Sensors

```
# 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 (see 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)

Ego Car, a.k.a. theTest 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., steer, 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 have to drive till the end of 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')
```

Driving the ego-car from within the 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** (steer, brake, throttle)
- Simple setup, but **requires installing the driving agent dependencies** (e.g., tensorflow, keras)

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)
```

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

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

beamer.py

```
def start_controller(self):
    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)
```

drivers.py

```
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()
```

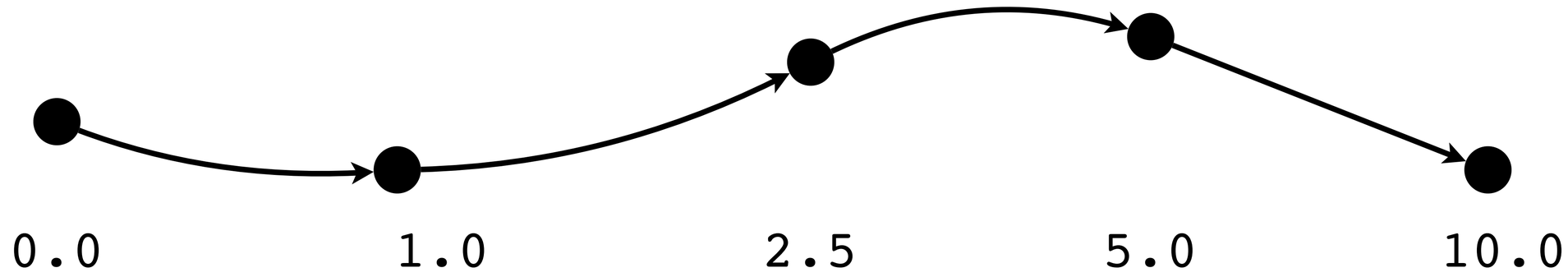
https://github.com/alessiogambi/AsFault/tree/asfault-deap/src/asfault

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 the available BeamNG driving AI and
  - **Setting the checkpoints** and the other configurations (e.g., speed, 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 the trajectory
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

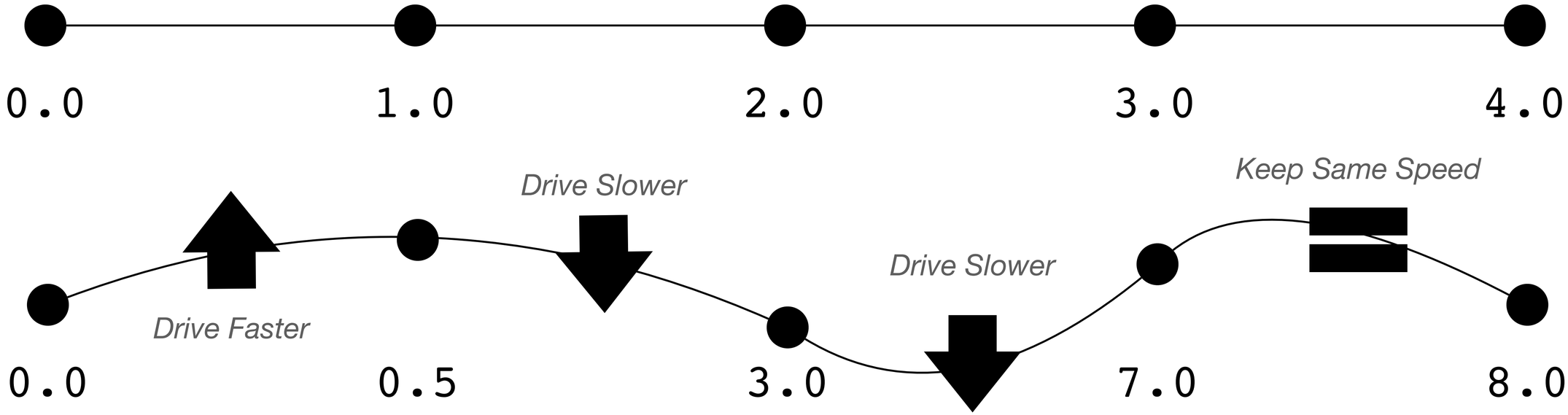
Automated test generation

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)

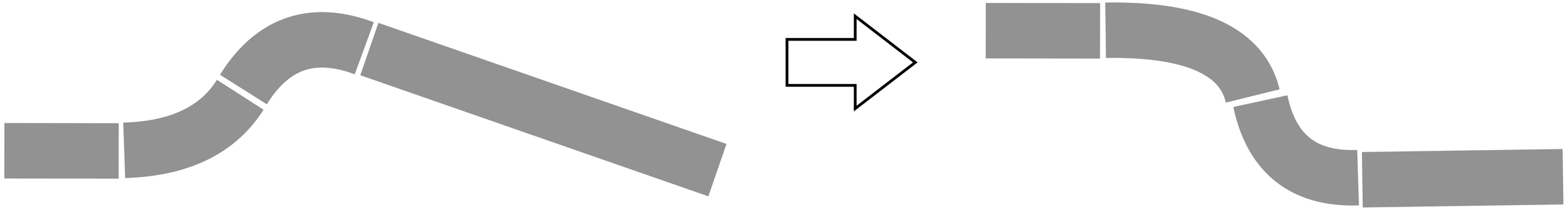
Follow the Drunk Driver

- We start with the ego-car behind a NPC car and configure the NPC to follow a straight trajectory at a constant speed
- We implement a basic **EA 1+1** (pop size = 1, only mutation)
- 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)



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



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

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