pystk Documentation

Release 1.0

Philipp Krähenbühl

Sep 22, 2021

THE BASICS

This is a heavily modified version of the free SuperTuxKart racing game for sensorimotor control experiments.

Many parts that make SuperTuxKart fun and entertaining to play were removed, and replaced with a highly efficient and customizable python interface to the game. The python interface supports the full rendering engine and all assets of the original game, but no longer contains any sound playback, networking, or user interface.

If you find a bug in this version of supertuxkart please do not report it to the original project, this project significantly diverged from the original intention of the video game.

CHAPTER

ONE

HARDWARE REQUIREMENTS

To run SuperTuxKart, make sure that your computer's specifications are equal or higher than the following specifications:

- A graphics card capable of 3D rendering - NVIDIA GeForce 8 series and newer (GeForce 8100 or newer), AMD/ATI Radeon HD 4000 series and newer, Intel HD Graphics 3000 and newer. OpenGL >= 3.3
- You should have a CPU that's running at 1 GHz or faster.
- You'll need at least 512 MB of free VRAM (video memory).
- Minimum disk space: 800 MB

CHAPTER

TWO

LICENSE

The software is released under the GNU General Public License (GPL). Information about the licenses for the artwork is contained in `data/licenses`.

2.1 Installation

2.1.1 Using pip

```
pip install PySuperTuxKart
```

2.1.2 From source

```
python setup.py build
python setup.py install
```

2.1.3 Development

Clone the repository https://github.com/philkr/pystk. For easier development, it is recommended to install pystk directly through `cmake`.

```
mkdir build
cd build
cmake ..
make
```

CMake will place a copy of the library in the top level directly, with allows any examples to run from that directory. Make sure the fetch the game assets, if they don't already exist.

```
python setup.py fetch_data
```

2.1.4 Documentation

```
cd docs
make html
```

PySTK does not compile on readthedocs.org due to some missing dependencies. This means that autodoc does not work there. In order to circumvent this, the autodoc parts of the documentation are split off and can be built using

```
make rst
```

Make sure to build the html again after.

2.2 Quick start

Let's walk through a simple example on how to use pystk. You'll first need to setup the rendering engine. SuperTuxKart uses a lot of global memory objects, some of them should only be initilized once. Hence, you should only setup the rendering engine *once* per process.

```
config = pystk.GraphicsConfig.hd()
config.screen_width = 800
config.screen_height = 600
pystk.init(config)
```

This setup uses the high-definition graphics preset and sets the resolution to 800 x 600.

Now we're ready to start the race. You may play as many races as you want, but you can only run *one* race per process. If you try to start (or setup) a second race before completing the first, the wrapper will raise an exception and eventually crash.

```
config = pystk.RaceConfig()
config.num_kart = 2 # Total number of karts
config.players[0].controller = pystk.PlayerConfig.Controller.AI_CONTROL

config.track = 'lighthouse'

race = pystk.Race(config)
```

This race configuration plays the `lighthouse` track with a total of 2 karts, one of which is player controlled. By default there is only one player `len(config.players)==1` and all other karts are non-controllable AI karts.

Next, let's start the race and play for a 100 steps.

```
race.start()
for n in range(100):
    race_ended = race.step()
    # Optionally display race.render_data
```

See *Race* for a full documentation of the race object and the render_data.

Finally, delete the current race object before exiting or starting a new race.

```
race.stop()
del race
```

2.3 Graphics Setup

Before you can use pystk you need to setup the OpenGL rendering engine and graphics settings. There are three default settings `GraphicsConfig::ld` (lowest), `GraphicsConfig::sd` (medium), `GraphicsConfig::hd` (high). Depending on your graphics hardware each setting might perform slightly differently (`ld` fastest, `hd` slowest). To setup pystk call:

```
pystk.init(pystk.GraphicsConfig.hd())
# Only call init once per process
... # use pystk
pystk.clean() # Optional, will be called atexit
# Do not call pystk after clean
```

class pystk.**GraphicsConfig**

SuperTuxKart graphics configuration.

static hd() → *pystk.GraphicsConfig*

High-definitaiton graphics settings

static ld() → *pystk.GraphicsConfig*

Low-definition graphics settings

static none() → *pystk.GraphicsConfig*

Disable graphics and rendering

static sd() → *pystk.GraphicsConfig*

Standard-definition graphics settings

property animated_characters → bool

Animate characters

property bloom → bool

Enable the bloom effect

property degraded_IBL → bool

Disable specular IBL

property display_adapter → int

GPU to use (Linux only)

property dof → bool

Depth of field effect

property dynamic_lights → bool

Enable dynamic lighting

property glow → bool

Enable glow around pickup objects

property high_definition_textures → int

Enable high definition textures 0 / 2

property light_shaft → bool

Enable light shafts

property mlaa → bool

Enable anti-aliasing

property motionblur → bool

Enable motion blur

property particles_effects → int
Particle effect 0 (none) to 2 (full)

property render → bool
Is rendering enabled?

property screen_height → int
Height of the rendering surface

property screen_width → int
Width of the rendering surface

property ssao → bool
Enable screen space ambient occlusion

property texture_compression → bool
Use texture compression

pystk.**init**(*config:* pystk.GraphicsConfig) → None
Initialize Python SuperTuxKart. Only call this function once per process. Calling it twice will cause a crash.

pystk.**clean**() → None
Free Python SuperTuxKart, call this once at exit (optional). Will be called atexit otherwise.

2.4 Race

To start a race create a new Race object. You can configure your race using the `RaceConfig` object, see *Configuration*. You need to set the graphics settings before starting a race, see *Graphics Setup*.

```
pystk.init(pystk.GraphicsConfig.hd())

config = pystk.RaceConfig(track='lighthouse', num_kart=2)
config.players[0].controller = pystk.PlayerConfig.Controller.AI_CONTROL
race = pystk.Race(config)
race.start()

n_steps = 100
for step in range(n_steps):
    race.step() # Use an optional action and set controller to pystk.PlayerConfig.
↪Controller.PLAYER_CONTROL
    # Use race.render_data[0].image
    # Use race.render_data[0].depth
    # Use race.render_data[0].instance
race.stop()
del race
# You may start a new race after you delete the old race object
pystk.clean()
```

class pystk.**Race**
The SuperTuxKart race instance

__init__(*self:* pystk.Race, *config:* pystk.RaceConfig)

restart(*self:* pystk.Race) → None
Restart the current track. Use this function if the race config does not change, instead of creating a new SuperTuxKart object

start(*self:* pystk.Race) → None
start the race

step(**args*, ***kwargs*)
Overloaded function.

- step(self: pystk.Race, action: List[pystk.Action]) -> bool

Take a step with an action per agent

- step(self: pystk.Race, action: pystk.Action) -> bool

Take a step with an action for agent 0

- step(self: pystk.Race) -> bool

Take a step without changing the action

stop(*self:* pystk.Race) → None
Stop the race

property config → *pystk.RaceConfig*
The current race configuration

property render_data → List[*pystk.RenderData*]
rendering data from the last step

SuperTuxKart uses several global variables and thus only allows one game instance to run per process. To check if there is already a race running use the `is_running` function.

pystk.**is_running**() → bool
Is a race running?

2.4.1 Configuration

The player configuration is used to add agents to the race. Each agent can be an AI or player controlled, and produces a separate `render_data` output.

class pystk.**PlayerConfig**
SuperTuxKart player configuration

class Controller

AI_CONTROL = 1

PLAYER_CONTROL = 0

property name → str

property value → int

property controller → *pystk.PlayerConfig.Controller*
Let the player (PLAYER_CONTROL) or AI (AI_CONTROL) drive. The AI ignores actions in step(action).

property kart → str
Kart type, see list_karts for a list of kart types

property team → int
Team of the player 0 or 1

The main race configuration specified everything from the track to use, the race type, number of agents and additional AI agents.

class pystk.**RaceConfig**
SuperTuxKart race configuration.

class **RaceMode**

CAPTURE_THE_FLAG = 5

FOLLOW_LEADER = 2

FREE_FOR_ALL = 4

NORMAL_RACE = 0

SOCCER = 6

THREE_STRIKES = 3

TIME_TRIAL = 1

property **name** → str

property **value** → int

property **difficulty** → int
Skill of AI players 0..2

property **laps** → int
Number of laps the race runs for

property **mode** → *pystk.RaceConfig.RaceMode*
Specify the type of race

property **num_kart** → int
Total number of karts, fill the race with num_kart - len(players) AI karts

property **players** → pystk.VectorPlayerConfig
List of all agent players

property **reverse** → bool
Reverse the track

property **seed** → int
Random seed

property **step_size** → float
Game time between different step calls

property **track** → str
Track name

pystk.**list_tracks**() → List[str]
Return a list of track names (possible values for RaceConfig.track)

pystk.**list_karts**() → List[str]
Return a list of karts to play as (possible values for PlayerConfig.kart

2.4.2 Action

The *Race.step* function takes an optional action or list of actions as an input.

class pystk.**Action**
SuperTuxKart action

property acceleration → float
Acceleration, normalize to 0..1

property brake → bool
Hit the brakes. Zero acceleration and brake=True uses reverse gear.

property drift → bool
Drift while turning

property fire → bool
Fire the current pickup item

property nitro → bool
Use nitro

property rescue → bool
Call the rescue bird

property steer → float
Steering angle, normalize to -1..1

2.4.3 Data

class pystk.**RenderData**
SuperTuxKart rendering output

property depth → numpy.ndarray
Depth image of the kart (memoryview[float] screen_height x screen_width)

property image → numpy.ndarray
Color image of the kart (memoryview[uint8] screen_height x screen_width x 3)

property instance → numpy.ndarray
Instance labels (memoryview[uint32] screen_height x screen_width)

Each instance label is spit into an `ObjectType` and instance label. Right shift (>>) the instance label by `ObjectType.object_type_shift` to retrieve the object type.

class pystk.**ObjectType**

object_type_shift
Number of bits for the instance label (shift of the object type)

N = 10

background = 3

bomb = 6

kart = 1

property name → str

nitro = 5

object = 7

pickup = 4

projectile = 8

track = 2

unknown = 9

property value → int

2.4.4 Game state

PySTK also exposes the internal state of the game.

class pystk.**WorldState**

static set_ball_location(*position: float3*, *velocity: float3 = [0.0, 0.0, 0.0]*, *angular_velocity: float3 = [0.0, 0.0, 0.0]*) → None
Specify the soccer ball / hockey puck position (SOCCER mode only).

static set_kart_location(*kart_id: int*, *position: float3*, *rotation: Quaternion = [0.0, 0.0, 0.0, 1.0]*, *speed: float = 0*) → None
Move a kart to a specific location.

update(*self:* pystk.WorldState) → None
Update this object with the current world state

property ffa → pystk.FFA
Free for all match info

property items → List[*pystk.Item*]
State of items

property karts → List[*pystk.Kart*]
State of karts

property players → List[*pystk.Player*]
State of active players

property soccer → *pystk.Soccer*
Soccer match info

property time → float
Game time

class pystk.**Track**

update(*self:* pystk.Track) → None

property length → float
length of the track

property path_distance → numpy.ndarray[numpy.float32]
Distance down the track of each line segment (float N x 2)

property path_nodes → numpy.ndarray[numpy.float32]
Center line of the drivable area as line segments of 3d coordinates (float N x 2 x 3)

property path_width → numpy.ndarray[numpy.float32]
Width of the path segment (float N)

class pystk.**Player**

property camera → *pystk.Camera*
Camera parameters of the player

property kart → *pystk.Kart*
Kart of the player

class pystk.**Camera**

class Mode

CLOSEUP = 1

FALLING = 4

LEADER_MODE = 3

NORMAL = 0

REVERSE = 2

property name → str

property value → int

property aspect → float
Aspect ratio

property fov → float
Field of view

property mode → *pystk.Camera.Mode*
Camera mode

property projection → readonly_memoryview
Projection matrix (float 4x4)

property view → readonly_memoryview
View matrix (float 4x4)

class pystk.**Item**

class Type

BANANA = 1

BONUS_BOX = 0

BUBBLEGUM = 4

EASTER_EGG = 6

NITRO_BIG = 2

NITRO_SMALL = 3

property name → str

property value → int

property id → int
Item id compatible with instance data

property location → float3
3D world location of the item

property size → float
Size of the object

property type → *pystk.Item.Type*
Item type

class pystk.**Kart**

property attachment → *pystk.Attachment*
Attachment of kart

property distance_down_track → float
Distance traveled on current lap

property finish_time → float
Time to complete race

property finished_laps → int
Number of laps completed

property front → float3
Front direction of kart 1/2 kart length forward from location

property id → int
Kart id compatible with instance labels

property jumping → bool
Is the kart jumping?

property lap_time → float
Time to completion for last lap

property lives → int
Lives in three strikes battle

property location → float3
3D world location of the kart

property max_steer_angle → float
Maximum steering angle

property name → str
Player name

property overall_distance → float
Overall distance traveled

property player_id → int
Player id

property powerup → *pystk.Powerup*
Powerup collected

property race_result → bool
Did the kart finish the race?

property rotation → Quaternion
Quaternion rotation of the kart

property shield_time → float
Second the shield is up for

property size → float3
Width, height and length of kart

property velocity → float3
Velocity of kart

property wheel_base → float
Wheel base

class pystk.**Powerup**

class Type

ANVIL = 10

BOWLING = 3

BUBBLEGUM = 1

CAKE = 2

NOTHING = 0

PARACHUTE = 9

PLUNGER = 5

RUBBERBALL = 8

SWATTER = 7

SWITCH = 6

ZIPPER = 4

property name → str

property value → int

property num → int
Number of powerups

property type → *pystk.Powerup.Type*
Powerup type

class pystk.**Attachment**

class Type

ANVIL = 1

BOMB = 2

BUBBLEGUM_SHIELD = 6

NOTHING = 9

PARACHUTE = 0

SWATTER = 3

property name → str

property value → int

property time_left → float
Seconds until attachment detaches/explodes

property type → *pystk.Attachment.Type*
Attachment type

class pystk.**Soccer**

property ball → *pystk.SoccerBall*
Soccer ball information

property goal_line → List[List[float3[2]][2]]
Start and end of the goal line for each team

property score → int[2]
Score of the soccer match

class pystk.**SoccerBall**

property id → int
Object id of the soccer ball

property location → float3
3D world location of the item

property size → float
Size of the ball

2.5 Logging

PySTK uses a global logging mechanism. You can select one of the log levels below.

class pystk.**LogLevel**
Global logging level

debug = 0

error = 4

fatal = 5

info = 2

property name → str

property value → int

verbose = 1

warn = 3

pystk.**set_log_level**(*arg0: int*) → None
Set the global log level

You may also set the log level through an environment variable PYSTK_LOG_LEVEL using a string corresponding to the log level.

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

Z