Moire Documentation

Release 0.1.0

Andrey Zamaraev

May 05, 2018

Contents

Multi-Objective Interactive Runtime Environment

The official GUI for Artipixoids! project.

It is integrating natively with Xentica framework, but you may also use it to visualize your own step-by-step simulations.

The engine is dealing nothing with rendering, it's relying fully on frames rendered with your simulation. The main Moire purpose is to automatically build GUI that gives you control over simulation's parameters, change them at runtime, save/load experiments, shoot videos, etc.

> **Warning:** Current version is a work-in-progress, it works to some degree, but please do not expect something beneficial from it. As planned, really useful stuff would be available only starting from version 0.3.

CHAPTER 1

User Guide

If you brave enough to ignore the warning above, dive right into this guide. Hopefully, you will manage to install Moire on your system and at least run some examples.

1.1 Installation Instructions

Moire is built upon Kivy framework, it is the main dependency for its core functionality. Also, you need NumPy in order to run examples and tests.

> **Warning:** This page *may* give you some insights on how to set up your environment and run Moire examples, but without any guarantees. More detailed instructions are coming soon.

1.1.1 Prerequisites

- Python 3.5+
- Kivy

Possible solution for Debian-like systems:

```
$ sudo apt-get install python3
$ sudo pip3 install Cython==0.23
$ sudo pip3 install kivy
```

1.1.2 Run Moire examples

In order to run Noise TV environment with Moire:

1. Clone Moire repository.

2. Put it on Python path.
3. Install NumPy.
4. Run `moire/examples/noisetv.py` with Python 3 interpreter.

Possible solution for Debian-like systems:

```
$ mkdir moire
$ cd moire
$ git clone https://github.com/a5kin/moire.git
$ PYTHONPATH="$(pwd)/moire/:$PYTHONPATH" python3 ./moire/examples/noisetv.py
```

1.2 Tutorial

Tutorial is coming soon. Meanwhile, you may check the official Noise TV example.

```
"""
The example showing how to build an environment runnable with Moire.

We will set up Noise TV environment, which is generating a
preudorandom array each step, then render it to a viewport as
necessary.

"""

import numpy as np

import moire


class NoiseTV:
    """
    Welcome to the Noise TV.

    You will find nothing but a pseudorandom noise here, 24/7. Just
    relax, stare at the screen for some time, and your imagination
    could start playing games with you.

    """

    def __init__(self):
        """
        Initialize the environment.

        Some attributes are mandatory for this class, since Moire is
        relying on them:

        ``timestep``
            A positive integer, holding current time step number. You
            must properly increment it in ``step()`` method.
        ``speed``
            An integer >= 1, representing the simulation speed, or
            exactly the number of steps per visualization frame.
        ``paused``
            A flag showing if simulation is paused or running.
        ``bridge``
```

(continues on next page)

(continued from previous page)

```
            A bridge class between Moire and your environment.

        """
        self._size = 3
        self._screen = np.zeros((self._size, ), dtype=np.uint8)
        self.timestep = 0
        self.speed = 1
        self.paused = False
        self.bridge = MoireBridge

    def set_viewport(self, size):
        """
        Set viewport (camera) size and initialize array for it.

        :param size: tuple with width and height in pixels.

        """
        self._size = size[0] * size[1] * 3
        self._screen = np.zeros((self._size, ), dtype=np.uint8)

    def apply_speed(self, dval):
        """
        Change the simulation speed.

        :param dval: Delta by which speed is changed.

        """
        self.speed = max(1, (self.speed + dval))

    def toggle_pause(self):
        """
        Toggle ``paused`` flag.

        When paused, the ``step()`` method does nothing.

        """
        self.paused = not self.paused

    def step(self):
        """
        Perform a single simulation step.

        ``timestep`` attribute will hold the current step number.

        """
        if self.paused:
            return
        self._screen = np.random.randint(0, 255, (self._size, ),
                                         dtype=np.uint8)
        self.timestep += 1

    def render(self):
        """
        Render the field at the current timestep.

        You must call :meth:`set_viewport` before do any rendering.
```

(continues on next page)

(continued from previous page)

```
        :returns:
            NumPy array of ``np.uint8`` values, ``width * height * 3``
            size. The RGB values are consecutive.

        """
        return self._screen


class Bridge:
    """Main bridge class containing basic functions."""

    @staticmethod
    def exit_app(_env, gui):
        """Exit GUI application."""
        gui.exit_app()

    @staticmethod
    def speed(dspeed):
        """Change simulation speed."""
        def func(env, _gui):
            """Wrap speed applying."""
            env.apply_speed(dspeed)
        return func

    @staticmethod
    def toggle_pause(env, _gui):
        """Pause/unpause simulation."""
        env.toggle_pause()

    @staticmethod
    def toggle_sysinfo(_env, gui):
        """Turn system info panel on/off."""
        gui.sysinfo.toggle()


class MoireBridge:
    """Class incapsulating the actions for Moire UI."""

    key_actions = {
        "[": Bridge.speed(-1),
        "]": Bridge.speed(1),
        "spacebar": Bridge.toggle_pause,
        "f12": Bridge.toggle_sysinfo,
        "escape": Bridge.exit_app,
    }


def main():
    """Run the whole environment with Moire."""
    environment = NoiseTV()
    gui = moire.GUI(runnable=environment)
    gui.run()


if __name__ == "__main__":
    main()
```

CHAPTER 2

API Reference

2.1 Moire

The official GUI for Artipixoids! project.

It is integrating natively with Xentica framework, but you may also use it to visualize your own step-by-step simulations.

The engine is dealing nothing with rendering, it's relying fully on frames rendered with your simulation. So, you should manually implement the stuff like zooming or scrolling, and bind the actions to keyboard/mouse/etc events.

The only thing Moire doing right now, is keeping the rate of your rendered frames, speeding simulation up/down as necessary. Also, system info dialog is included, showing current timestep, frames/steps per second and simulation speed.

2.1.1 Main App (moire.main)

Main module containing the base class for Moire GUI.

All Moire apps should be ran using *GUI* class. See the example below.

class moire.main.**GUI**(***kwargs*)

Bases: object

Main class for moire GUI.

Moire apps should be ran with it in following way:

```
runnable = YourRunnable()
gui = moire.GUI(runnable=runnable)
gui.run()
```

The engine is fully relying on your custom runnable class. It should implement a number of features like perform a simulation step, render frame etc. See the detailed example in the official NoiseTV example.

Parameters **runnable** – Runnable class, implementing all necessary features.

build()
Prepare GUI for running.

class moire.main.**MainEngine**(*app*, **args*, ***kwargs*)
Bases: object

Class incapsulating main Moire engine functionality.

Parameters **app** – *GUI* class instance.

background = <ObjectProperty object>
Kivy property containing Rectangle with rendered frames.

exit_app()
Exit the app in convenient way.

key_down(*key*)
Emulate key down.

prepare()
Do main preparations before app run.

runnable = <ObjectProperty object>
Kivy property containing runnable class.

update(*_dt*)
Update the whole app, while keeping the frame rate.

update_sysinfo()
Update System Info widget with actual data.

viewport = <ObjectProperty object>
Kivy property containing rendered frames as Texture.

class moire.main.**ObjectProperty**
Bases: object

2.2 Widgets (moire.widgets)

The package with the widgets and UI components.

You may use widgets package as a shortcut to the following classes.

- widgets.PanelWidget → *moire.widgets.panel.PanelWidget*
- widgets.SystemInfoWidget → *moire.widgets.sysinfo.SystemInfoWidget*
- widgets.DescriptiveList → *moire.widgets.dlist.DescriptiveList*

2.2.1 Descriptive List (moire.widgets.dlist)

The module with a two-column param-value list.

class moire.widgets.dlist.**DescriptiveList**(*fields*, *item_width*, *item_height=25*, ***kwargs*)
Bases: object

The widget representing key-value descriptive list.

Initially, values are empty, but could be changed at runtime like:

```
my_list = DescriptiveList(['Foo', 'Bar'], 200, 200)
# ...sometimes later
my_list['Foo'] = '235'
```

Parameters

- **fields** – A list of field names, to be shown at the left column in bold font. This will also be keys for values access.
- **item_width** – The width of the item in inner units.
- **item_height** – The height of the item in inner units.

2.2.2 Panel Base (moire.widgets.panel)

Module with the base `Panel` class.

class moire.widgets.panel.**PanelWidget**(***kwargs*)
Bases: object

The implementation of the base panel widget.

Other widgets could be inherited from `Panel` in order to get the standard visual decoration and show/hide transitions.

redraw(**_args*)
Redraw panel on the canvas.

toggle()
Toggle show/hide the panel with its contents.

2.2.3 System Info (moire.widgets.sysinfo)

Module for widget with system info.

class moire.widgets.sysinfo.**NumericProperty**
Bases: object

class moire.widgets.sysinfo.**SystemInfoWidget**(***kwargs*)
Bases: *moire.widgets.panel.PanelWidget*

Build-in widget showing system info.

The fields are:

FPS Current frames per second speed (for GUI).

SPS Current steps per second speed (for simulation).

Speed How many steps are skipped each frame.

Time Global timestep counter.

fps = <NumericProperty object>
Desired frame rate

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index