Energy Documentation

Release 0.1.0

Heungsub Lee

September 06, 2012

CONTENTS

CHAPTER

ONE

FOR SOCIAL GAMES

1.1 What is energy?

Oh, you are kidding. You already know what energy is. But, just to clarify, energy is a concept that is consumable and recoverable in social games. It limits how far players can advance in each session.

Players use energy to perform actions such as farming, housing, or social interactions. Then consumed energy will be recovered after certain amount of time designed by the developer. Recovery is the essence of energy system. It will make players to come back to the game periodically.

Popular social games such as FarmVille , Zoo Invasion or The Sims Social are benefited from the system in high retention rate.

1.2 How to use?

Install via PyPI first:

```
$ easy_install energy
```

Or check out developement version:

```
$ git clone git://github.com/sublee/energy.git
```

What you need to implement energy system is only `Energy` object. Maximum energy and recovery interval have to be set in seconds before use:

```
from energy import Energy
energy = Energy(max=10, recovery_interval=300)
```

The example `Energy` object has 10 of maximum and will recover in every 5 minutes. When a player performs a action that requires energy just call `Energy.use()` method:

```
>>> print energy
<Energy 10/10>
>>> energy.use()
>>> print energy
<Energy 9/10 recover in 05:00>
```

If the player has not enough energy, it throws `ValueError`:

```
>>> print energy
<Energy 9/10 recover in 04:12>
```

```
>>> energy.use(10)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "energy.py", line 104, in use
    raise ValueError('Not enough energy')
ValueError: Not enough energy
```

You may want to save `Energy` object within a specific player's data in database.

An `Energy` object is serializable by Pickle. If you have a key-value storage, you can save an `Energy` object with a player easily. Or, you should prepare some columns for `Energy.used` and `Energy.used_at` in your database to save them. Here's an example of save/load an `Energy` object:

```
>>> MAX_ENERGY, ENERGY_RECOVERY_INTERVAL = 10, 300
>>> energy = Energy(MAX_ENERGY, ENERGY_RECOVERY_INTERVAL)
>>> saved_used, saved_used_at = energy.used, energy.used_at
>>> loaded_energy = Energy(MAX_ENERGY, ENERGY_RECOVERY_INTERVAL,
...                        used=saved_used, used_at=saved_used_at)
>>> loaded_energy == energy
True
```

1.3 API

class `energy.`**`Energy`**(*max*, *recovery_interval*, *recovery_quantity=1*, *used=0*, *used_at=None*)

A consumable and recoverable stuff in social gamers. Think over reasonable energy parameters for your own game. Energy may decide return period of your players.

Parameters

- **max** – maximum energy
- **recovery_interval** – an interval in seconds to recover energy. It should be *int*, *float* or *timedelta*.
- **recovery_quantity** – a quantity of once energy recovery. Defaults to `1`.

`config`(*max=None*, *recovery_interval=None*, *time=None*)

Updates `max` or `recovery_interval`.

Parameters

- **max** – quantity of maximum energy to be set
- **time** – the time when setting the energy. Defaults to the present time in UTC.

`current`(*time=None*)

Calculates the current energy. This equivalents to casting to *int* but can work with specified time.

```
>>> energy = Energy(10, 300)
>>> energy.use()
>>> energy.current()
9
>>> int(energy)
9
```

Parameters **time** – the time when checking the energy. Defaults to the present time in UTC.

`max`

The maximum energy.

passed(*time=None*)
Calculates the seconds passed from using the energy first.

Parameters **time** – the time when checking the energy. Defaults to the present time in UTC.

recover_in(*time=None*)
Calculates seconds to the next energy recovery. If the energy is full, this returns `None`.

Parameters **time** – the time when checking the energy. Defaults to the present time in UTC.

recovered(*time=None*)
Calculates the recovered energy from the player used energy first.

Parameters **time** – the time when checking the energy. Defaults to the present time in UTC.

recovery_interval = None
The interval in seconds to recover energy.

recovery_quantity = None
The quantity of once energy recovery.

reset(*time=None*)
Makes the energy to be full. Most social games reset energy when the player reaches higher level.

Parameters **time** – the time when setting the energy. Defaults to the present time in UTC.

set(*quantity*, *time=None*)
Sets the energy to the fixed quantity.

```
>>> energy = Energy(10, 300)
>>> print energy
<Energy 10/10>
>>> energy.set(3)
>>> print energy
<Energy 3/10 recover in 05:00>
```

You can also set over the maximum when give bonus energy.

```
>>> energy.set(15)
>>> print energy
<Energy 15/10>
```

Parameters

- **quantity** – quantity of energy to be set
- **time** – the time when setting the energy. Defaults to the present time in UTC.

use(*quantity=1*, *time=None*)
Consumes the energy.

Parameters

- **quantity** – quantity of energy to be used. Defaults to `1`.
- **time** – the time when using the energy. Defaults to the present time in UTC.

used = 0
Quantity of used energy.

used_at = None
A time when using the energy first.

1.4 Licensing and Author

This project licensed with BSD. See LICENSE for the details.

I'm Heungsub Lee, a game developer. Any regarding questions or patches will be welcomed.

PYTHON MODULE INDEX

e