dice Documentation

Release 0.0.1

Hao Liu

January 16, 2017

Contents

Contents:

CHAPTER 1

About DICE

DICE is a black-box random testing framework. It aims to help testers random testing a project while exploring its feature by writing some constraints. This project also tries to standardize the activity of random testing.

DICE is currently in experimental stage.

1.1 Goals and Objectives

- Standardize software random testing activity.
- Provides professional random testing.
- Apply to your project with minimal effort.
- Human friendly configuration process.
- Universally applicable to many aspect of software testing.

CHAPTER 2

Getting Started

2.1 Installing DICE

DICE is currently in experimental stage and not ready for release yet. So easy-install or pip way of installation is not available now. The only way to install DICE is from the source code.

2.1.1 Install from Git Source

To install DICE from git repository, clone the source code to local first:

```
git clone https://github.com/code-dice/dice
cd dice
```

Then install dependencies from pip:

```
sudo pip install -r requirements.txt
```

Install DICE:

```
sudo python setup.py install
```

2.2 Using DICE

2.2.1 Example Project

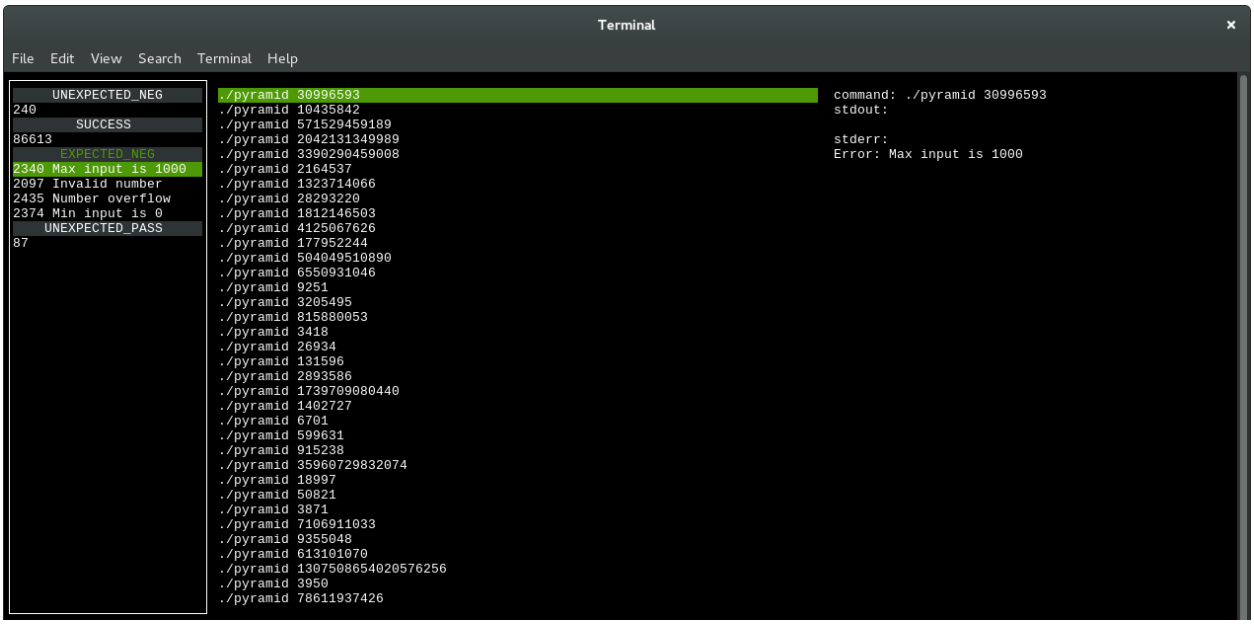
Build the example binary from source:

```
cd examples/pyramid
gcc pyramid.c -o pyramid
```

Run DICE on the example project:

```
dice
```

This will open a ncurses TUI shows the statistics of results by generating the option randomly.



The left panel is a **stat panel** shows the stat of error message patterns categorized by the exit status and whether error message matches expectation defined in the constraint file.

The central panel is a **list panel** lists recent called command lines matches the error message pattern selecting in the **stat panel**.

The right panel is a **detail panel** show the detailed information of the selected command in **list panel** includes command line, standard output and standard error.

The follow key press allowing navigation through the panels.

Key	Function
TAB	Toggle current working panel
Q	Exit DICE
P	Pause/Resume execution
J	Select next item
K	Select previous item
M	Merge stat by regex pattern
^W	Save current input
^D	Cancel current input

2.2.2 Creating a custom Project (Implementing)

1. Generate a skeleton file structure by fill the required information promptly:

```
dice --start-project
```

2. Change `item.py` to call custom command or API.
3. Add constraints depicts expected result.
4. Run dice and check result. If something need fix, goto step 2.
5. Run dice continuously until bug found.

CHAPTER 3

Writing DICE Tests

3.1 Anatomy of a DICE Project

The file structure of a basic DICE project likes:

```
project_root
|-- oracles
|   `-- pyramid.yaml
`-- utils
    `-- item.py
```

- An `item.py` is a python script defines how a single test item is run and pass the result to DICE for analysis.
- The `oracles` directory contains one or more YAML files defines the expected results for different conditions.
- The optional `utils` directory contains python helper modules to assist specific tests.

3.2 Writing Test Runner

`item.py` contains a class `Item` inherits from `dice.item` class from DICE's core API:

```
import os

from dice.core import item
from dice import utils


class Item(item.ItemBase):
    def run(self):
        cmdline = os.path.join(self.provider.path, 'pyramid')
        cmdline += ' %s' % utils.escape(str(self.get('option')))
        self.res = utils.run(cmdline)
```

3.3 Writing Oracle

An example oracle YAML file likes:

```
- name: option
  oracle: |
      if option is Integer:
          if option < 0:
              return FAIL('Min input is 0')
          elif option > 9223372036854775808:
              return FAIL('Number overflow')
          elif option > 1000:
              return FAIL('Max input is 1000')
          else:
              return SUCCESS()
      else:
          return FAIL('Invalid number')
```

Every oracle YAML file contains a list of oracle objects. For each oracle, there is some predefined properties.

- `name` is the identifier of a specific oracle. It is recommended in CamelCase style to be differentiated from other variables.
- `target` is where this oracle is applied to for the test item.
- `tree` is a python style code snippet shows the expected result for a given conditions.

CHAPTER 4

How DICE Works

DICE is composed of:

- Libraries help parsing constraints, generating random parameters for a tests, and run different types of tests.
- A curses-bases TUI to help testers monitor the results and statistics of called tests.

CHAPTER 5

Contributing Guidelines

5.1 Submit a Patch

DICE use github and github pull requests model to track issue and accept commits. You can find the detailed workflow and guide here.

5.2 Travis CI

DICE use Travis CI to check unit tests, coding style and documentation. You need to fix the patch for the CI builds pass before code goes into the repo.

CHAPTER 6

API References

6.1 dice package

6.1.1 Subpackages

dice.client package

Submodules

dice.client.panel module

class dice.client.panel.**InputPanel**(*screen*, *height*, *width*, *write_callback*, *cancel_callback*, *x=0*, *y=0*)

Bases: dice.client.panel._PanelBase

Curses panel allows get input from keyboard.

draw(*active=False*)

Draw the list panel.

Parameters **active** – If set to true, draw the panel with surrounding box.

on_keypress(*key*)

Event handler when keypress event received by the panel.

Parameters **key** – Key being pressed.

class dice.client.panel.**ListPanel**(*screen*, *height*, *width*, *x=0*, *y=0*, *format_str=''*)

Bases: dice.client.panel._PanelBase

Curses panel contains list of entries.

add_item(*bundle*, *catalog=''*)

Add an item the list panel.

Parameters

- **bundle** – Content of added item.
- **catalog** – Catalog of added item.

clear()

Clear panel content.

draw(*active=False*)
Draw the list panel.

Parameters **active** – If set to true, draw the panel with surrounding box.

on_keypress(*key*)
Event handler when keypress event received by the panel.

Parameters **key** – Key being pressed.

select(*cat_key=None*, *item_key=None*)
Select specified item.

Parameters

- **cat_key** – Catalog name of the item for selection.
- **item_key** – Item name of the item for selection.

set_select_callback(*callback*)
Set callback function triggered when an list item is selected.

Parameters **callback** – Select callback function.

class dice.client.panel.**TextPanel**(*screen*, *height*, *width*, *x=0*, *y=0*)
Bases: dice.client.panel._PanelBase

Curses panel contains only block of text.

clear()
Clear panel content.

draw(*active=False*)
Draw the text panel.

Parameters **active** – If set to true, draw the panel with surrounding box.

on_keypress(*key*)
Event handler when keypress event received by the panel.

Parameters **key** – Key being pressed.

set_content(*bundle*)
Set panel content.

Parameters **bundle** – Content to be set.

dice.client.window module

class dice.client.window.**Window**(*app*)
Bases: object

Class for a whole curses window of DICE client.

destroy()
Destroy the curses window.

draw()
Draw all the panels in the curses window.

get_input()
Show and focus to an input panel and return the content.

Returns A string of input content.

update()
Get events and update the window.

Module contents

class dice.client.**DiceApp**
Bases: object

Curses-based DICE client application.

run()
Main loop to run tests, update screen and send tests results.

run_tests()
Iteratively run tests.

update_window()
Update the content of curses window and refresh it.

dice.core package

Submodules

dice.core.constraint module

class dice.core.constraint.**Constraint**(*name*, *provider*, *depends_on=None*, *require=None*, *oracle=None*)
Bases: object

Class for a constraint on specific option of test item.

apply(*item*)
Apply this constraint to an item.

Parameters **item** – The item to be applied on.

Returns Expected result of constraint item.

classmethod from_dict(*provider*, *data*)
Generate a constraint instance from a dictionary

path_prefix = 'DPATH'

exception dice.core.constraint.**ConstraintError**
Bases: exceptions.Exception

Constraint module specified exception.

class dice.core.constraint.**ConstraintManager**(*provider*)
Bases: object

Manager class contains and manipulates all constraints.

constrain(*item*)
Apply constraints to an item.

Parameters **item** – Item for constraints to apply on.

dice.core.item module

class dice.core.item.**ItemBase**(*provider*)
Bases: object

Base class for an item. This should be overridden in the providers item.py.

get(*path*)
Get value for specific item option.

Parameters **path** – An XPath-like string for the getting target.

Returns Option value got.

run()
Run the item. Must be overridden in the providers.

set(*path*, *value*)
Set value for specific item option.

Parameters

- **path** – An XPath-like string for the setting target.
- **value** – Option value to be set.

exception dice.core.item.**ItemError**
Bases: exceptions.Exception

Class for Item specific exceptions.

dice.core.provider module

class dice.core.provider.**Provider**(*path*)
Bases: object

Class for a dice test provider.

generate()
Generate a new constrained test item.

Returns Constrained item.

exception dice.core.provider.**ProviderError**
Bases: exceptions.Exception

Class for provider specific exceptions.

dice.core.symbol module

class dice.core.symbol.**Bytes**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: *dice.core.symbol.SymbolBase*

Symbol class for a string contains random bytes (1~255).

generate()
Generate a random bytes string.

class dice.core.symbol.**Integer**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: *dice.core.symbol.SymbolBase*

Symbol class for a random integer.

generate()
Generate a random integer.

class dice.core.symbol.**NonEmptyBytes**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: *dice.core.symbol.Bytes*

Symbol class for a random byte(1-255) string except empty string.

generate()
Generate a random non-empty bytes string.

class dice.core.symbol.**String**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: *dice.core.symbol.Bytes*

Symbol class for a random printable string.

generate()
Generate a random printable string.

class dice.core.symbol.**StringList**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: *dice.core.symbol.SymbolBase*

Symbol class for a list of random printable strings.

generate()
Generate a random printable strings.

model()
Generate a random-numbered list contains random printable strings.

class dice.core.symbol.**SymbolBase**(*scope=None*, *excs=None*, *exc_types=None*)
Bases: object

Base class for a symbol object represent a catalog of data to be randomized.

generate()
Generate a random instance of this symbol without considering scope, excs or exc_types. Must be overridden.

model()
Generate a random instance of this symbol.

dice.core.trace module

class dice.core.trace.**Trace**(*provider*, *trace_list*)
Bases: object

Class represent a condition trace in constraint oracle code. It contains a list of commands, including comparisons, operations and ends with a return command.

solve(*item*)
Generate a satisfiable random option according to this trace. :param item: Item to which generated option applies. :return: Generated random option.

exception dice.core.trace.**TraceError**
Bases: exceptions.Exception

Class for trace specific exceptions.

Module contents

dice.utils package

Submodules

dice.utils.data_dir module

dice.utils.rnd module

dice.utils.rnd.**count**(*min_inc=0*, *max_inc=None*, *lambd=0.1*)

dice.utils.rnd.**cpuset**(*min_inc=0*, *max_inc=100*, *max_len=1000*, *used_vcpu=None*)

dice.utils.rnd.**int_exp**(*min_inc=0*, *max_inc=None*, *lambd=0.01*)
A non accurate exponentially distributed integer generator.

dice.utils.rnd.**integer**(*min_inc=0*, *max_inc=10*)

dice.utils.rnd.**regex**(*re_str*)
Generate a random string matches given regular expression.

dice.utils.rnd.**text**(*min_len=5*, *max_len=10*, *charset=None*, *excludes=None*)
Generate a randomized string.

Module contents

class dice.utils.**CmdResult**(*cmdline*)
Bases: object

A class representing the result of a system call.

pprint()
Print the command result in a pretty and colorful way.

dice.utils.**escape**(*org_str*)

dice.utils.**kernel_pids**()

dice.utils.**pids**()

dice.utils.**run**(*cmdline*, *timeout=10*)
Run the command line and return the result with a CmdResult object.

Parameters

- **cmdline** (*str.*) – The command line to run.
- **timeout** (*float.*) – After which the calling processing is killed.

Returns CmdResult – the command result.

Raises

dice.utils.**weighted_choice**(*choices*)

6.1.2 Module contents

6.2 dice

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

d

O

P

R

S

T

U

W