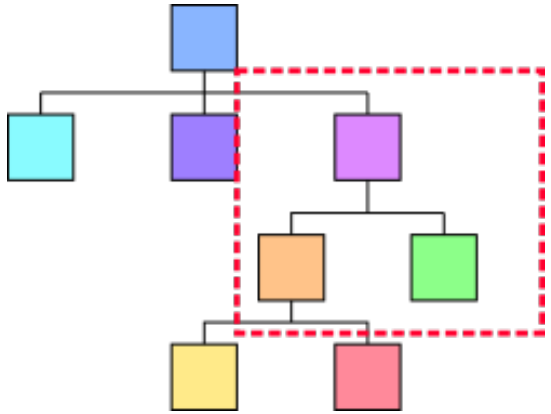
Tree matchings with Behavior Trees

FOSDEM 2019

February 3, 2019

How to recognize a complex subtree in a big tree



lionel@lse.epita.fr FOSDEM - Python Devroom 2019

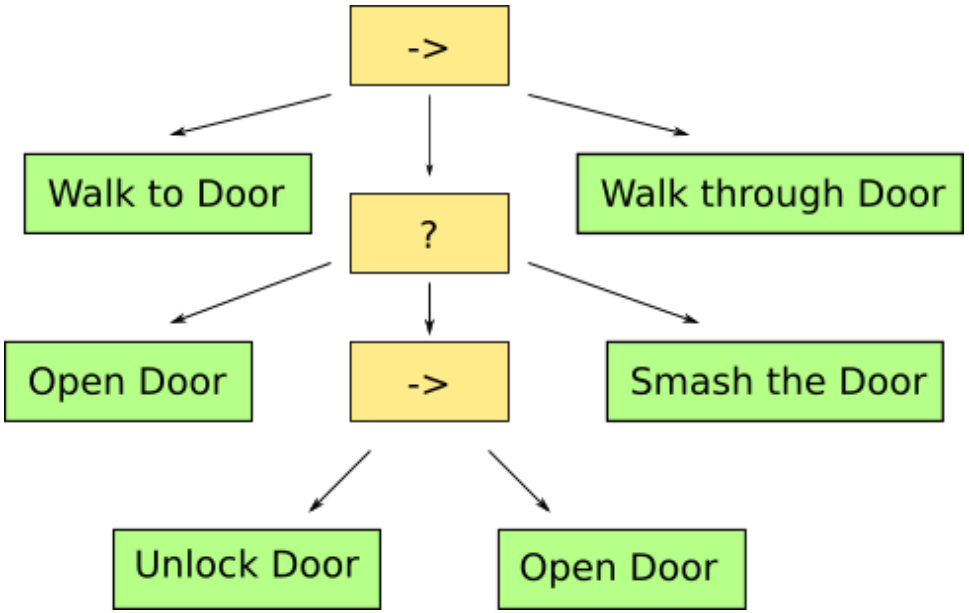
Quick summary

- About Behavior Tree
- About Tree Matching...
- ...in python

About Behavior Tree

A powefull abstraction to defined Process.

Common in Video Game to implement BOT AI.

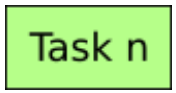


About Behavior Tree

3 components:

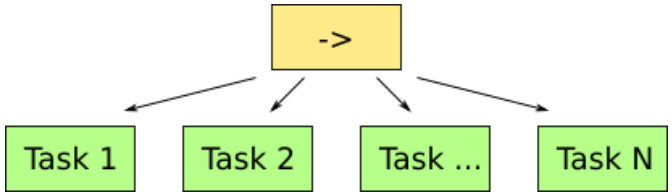
- Task

  Do a simple thing

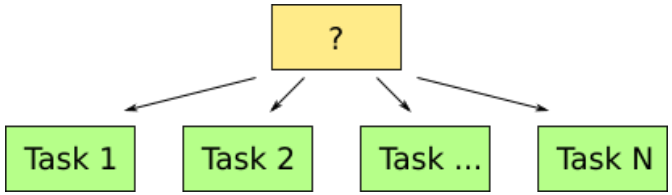
- Sequence

  Do the next if the previous succeed

- Selector

  Do the next if the previous failed

About Behavior Tree

Task

```
from enum import IntEnum

Status = IntEnum('Status', [
            'Failure',
            'Success',
            'Running'
    ])

class Task:
    def tick(self, udata) -> Status:
        return Status.Success

class Tree:
    def __init__(self, *child):
        self.child = child
```

About Behavior Tree

Sequence

```
class Sequence(Tree):
    def tick(self, udata) -> Status:
        for child in self.child:
            childstatus = child.tick(udata)
            if childstatus == Status.Running:
                return Status.Running
            elif childstatus == Status.Failure:
                return Status.Failure
        return Status.Success
```

About Behavior Tree

Selector

```
class Selector(Tree):
    def tick(self, udata) -> Status:
        for child in self.child:
            childstatus = child.tick(udata)
            if childstatus == Status.Running:
                return Status.Running
            elif childstatus == Status.Success:
                return Status.Success
        return Status.Failure
```

About Behavior Tree

Base on these few principes, you could easlisy create your own abstraction

About Behavior Tree

Concurrent

```
class Concurrent(Tree):
    def tick(self, udata) -> Status:
        for idx, child in enumerate(self.child):
            if self.status[idx] == Status.Running:
                self.status[idx] = child.tick(udata)
        if (not sum(map(lambda _: _ == Status.Running,
             self.status))):
            if (sum(map(lambda _: _ == Status.Success,
                 self.status))):
                return Status.Success
            return Status.Failure
        return Status.Running
```

About Tree Matching

Use cases:

- Data Validation
- Data Transformation
- Data Generation

These are part of Compiler:

- Semantic check
- AST Handling
- Code generation

About Tree Matching

Tree Handling:

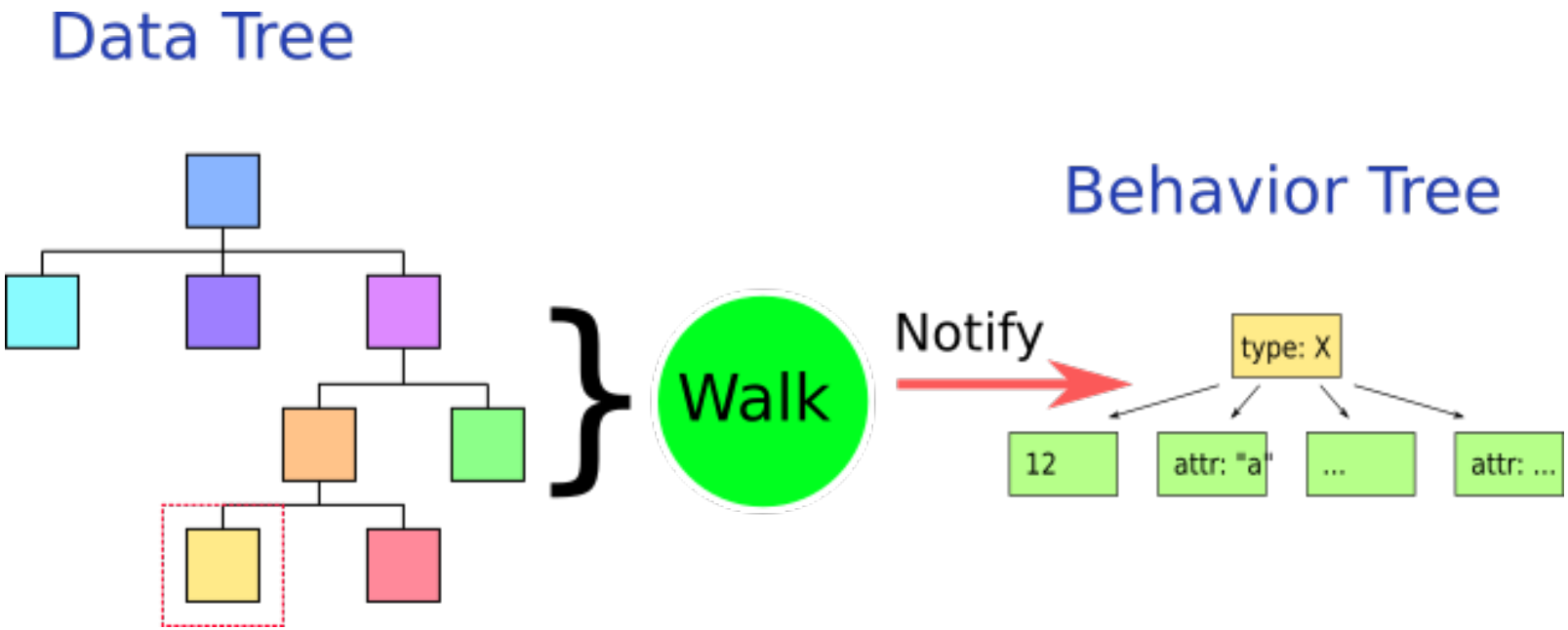
- Descent recursive function (Tree Walking)
- Pattern Matching:
  - Identify nodes
  - Deconstruct it and do something
- Top-down

Top-down pattern matching is it enough?

- Tree Reconstruction (update tree during walking) need Bottom-Up matching
- Ancestors and Siblings

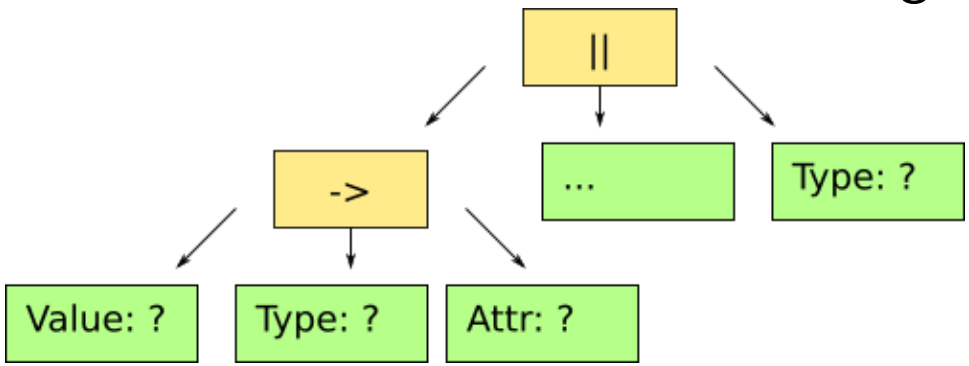
About Tree Matching

Intuition: Using BT to match subtree?



About Tree Matching

- Behavior Tree could mimic matching subtree



- Versatile (Top-Down and/or Bottom-Up)

Handle Python tree

Get a generic python data tree

```
class A:
    def __init__(self, **kw):
        self.__dict__.update(kw)

class B(dict):
    def __init__(self, d, **kw):
        self.update(d)
        self.__dict__.update(kw)

class C(list):
    def __init__(self, l, **kw):
        self.extend(l)
        self.__dict__.update(kw)
```

Handle Python tree

```
data_tree = A(foo=42,
        bar=C([
                1, 2, 3,
                B(h={'low': 1.2, 'high': 3.4}),
                4, 5, 6
            ],
            meh=3
            )
    )
```

Handle Python tree

To handle it

- vars
- getattr
- collections.Mapping
- collections.Iterable
- yield
- yield from

Handle Python tree

A generic walking function

```
def walk(tree):
    if isinstance(tree, c.Mapping):
        lsk = list(sorted(tree.keys()))
        for k in lsk:
            yield from walk(tree[k])
            ...
    elif (isinstance(tree, c.Iterable) \
          and type(tree) not in {str, bytes}):
        ls = enumerate(tree)
        for idx, it in ls:
            yield from walk(it)
            ...
    if hasattr(tree, '__dict__'):
        attrs = vars(tree)
        for attr in sorted(attrs.keys()):
            yield from walk(attrs[attr])
            ...
```

Handle Python tree

Notification

```
('value', tree)
('type', tree)
('attr', attr)
('key', k)
('idx', idx)
```

Handle Python tree

Behavior Tree Item to handle patterns:

- Value: a specific value or any
- Type: Concurrent Attributes/List/Dict
- List: Sequence of value/idx
- Dict: Sequence of value/key
- Attributes: Sequence of value/attr

Handle Python tree

Matching not a single **tree** but a **forest**:

- 1 pattern = 1 BT, N concurrent pattern = N BT
- 1 walking notification = 1 tick on each BT
- Don't store matching state IN the BT:
  - self.status[I] become udata.status[I]
- cleaning

Handle Python tree

See module **treematching** on https://github.com/LionelAuroux/treematching

```
bt = AnyType(
            List(
                AnyIdx(Type(str, Value('lala'))),
                AnyIdx(Type(int, Value(666))),
            ),
            Attrs(
                Attr('foo', Type(int, AnyValue()))
            )
    )
e = MatchingBTree(bt)
...
match = e.match(data_tree)
```

Handle Python tree

Features of **treematching** module:

- Value / AnyValue
- Type / KindOf / AnyType
- List / AnyList
- Dict / AnyDict
- Attrs strictly or not
- Ancestor and Sibling
- Nodes capturing
- Callback function

Status: Cleaning, bug hunting (Integration Testing)
Documentation: WIP

Conclusion

Q/A!

- **slides**
- `https://github.com/LionelAuroux/treematching`