Mongoose

```
+ ---------------------------------- +
|                                    |
|                         .--.       |
|                        ( _  \      |
|                        / /: :\     |
| ,=.                  ^ /   . |    |
| ---'._.--._            \ :   /    |
|     \      '.__=-.      \  .' ,-.  |
|     (          ' e\      | : / _ \ |
|      \  , ' . .-'-^  ,   |  \ / \ \ |
|       |; \!_ \|;.    ;/._, \  '--'  ) |
|       ^^=     ^=      '.__,''-----'  |
|                                    |
+ ---------------------------------- +
```


Bo Xu *Tester*
Chris Haueter *System Integrator*
Etan Zapinsky *System Architect*
Mark Fischer *Language Guru*
Michael Rubin *Project Manager*

Overview

- agent-based, discrete-event simulations
- clean syntax, object-oriented
- minimal language boilerplate
- post-simulation analysis
- innovative syntactical features

Project Management

- fluid roles
- Google Hangout meetings
- ~~Gantt charts~~
- private mailing list + Github issues
- *dictatorial fiat* + *whips ;)*

Syntax

```
1   agent A {
2       ~int x
3       create(){
4           print("On create A")
5           x = 0
6       }
7       action {
8           x = x + 1
9       }
10      destroy {
11      }
12  }
13
14  agent B {
15      ~int x
16      create(){
17          print("On create B")
18          x = 0
19      }
20      action {
21          x = one.x + 1
22      }
23      destroy {
24      }
25  }
26
27  environment {
28
29      populate {
30          ~A one = A()
31          ~B two = B()
32      }
33
34      action {
35          print("one.x:", one.x)
36          print("two.x:", two.x)
37      }
38  }
39
40  terminate {
41      (two.x >= 4) {
42          print("done")
43      }
44  }
45
46  analysis {
47  }
```

```
terminate{
    2:(x > 3) {
        print("I should print on turn 4")
    }
    (false) {
        print("I should never print")
    }
}
```

Probabilistic features

```
pif(.2){
    ~int x = 3
    print("x:", x)
} pelif (.5) {
    pif (0.6) {
        ~string s = "dependent probs!"
        print(s)
    }
    ~int y = 7
    print("y:", y)
} pelse {
    ~boolean b = false
    print("b:", b)
}
```

```
~int x = ( 1:5 | 3:4 | 7:8 )
print(x)

if( ( 1:true | 1:false ) ){
    print(1)
} else {
    print(0)
}
```

```
string rand(~string x, ~string y) {
    return ( 1:x | 1:y )
}
print(rand('hi', 'bye'))
```

```
( 1:( 1:'a' | 1:'b' ) | 1:'c' )
```

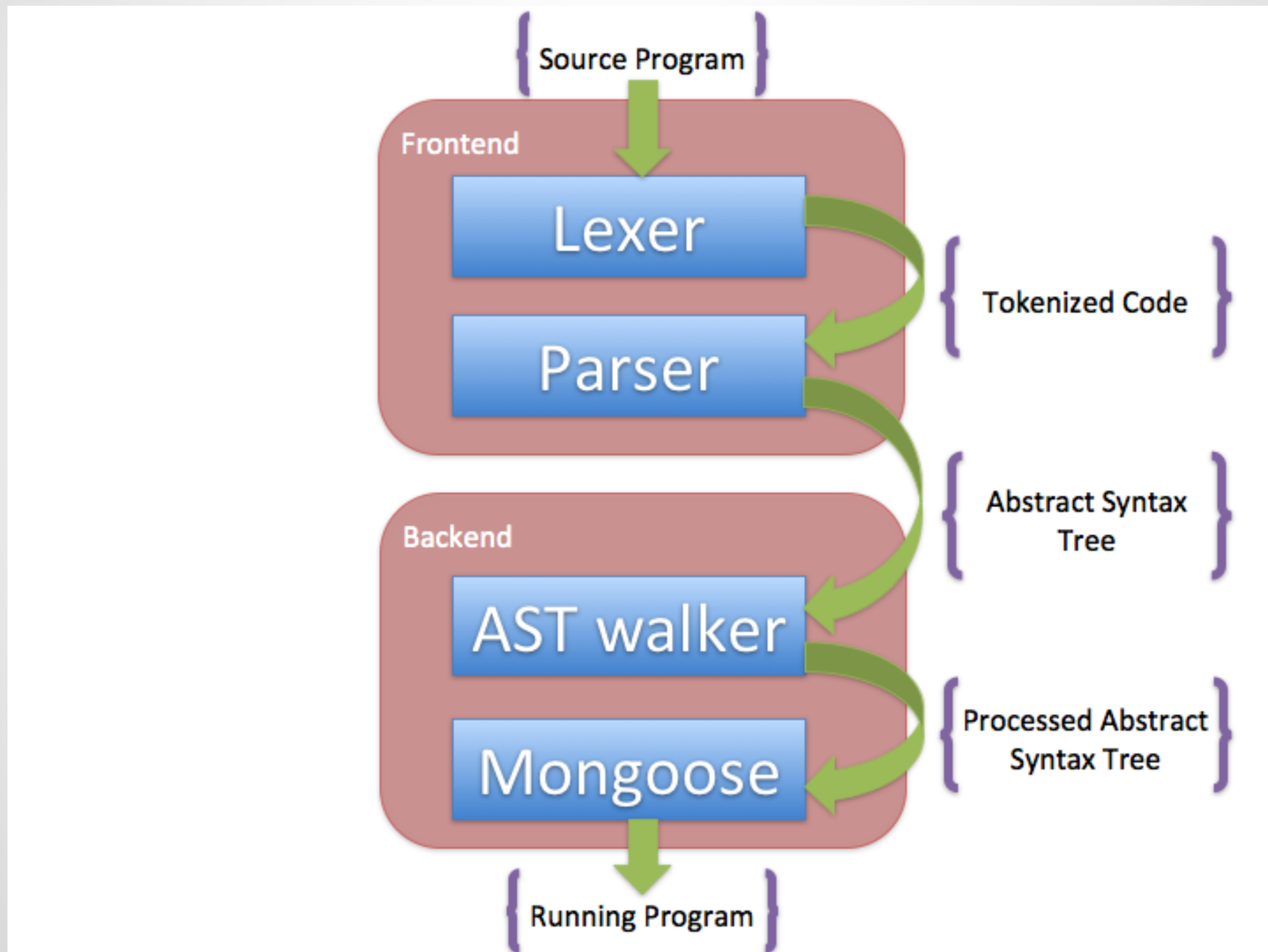
Compiler Tools

- PLY (Python Lex-YACC)

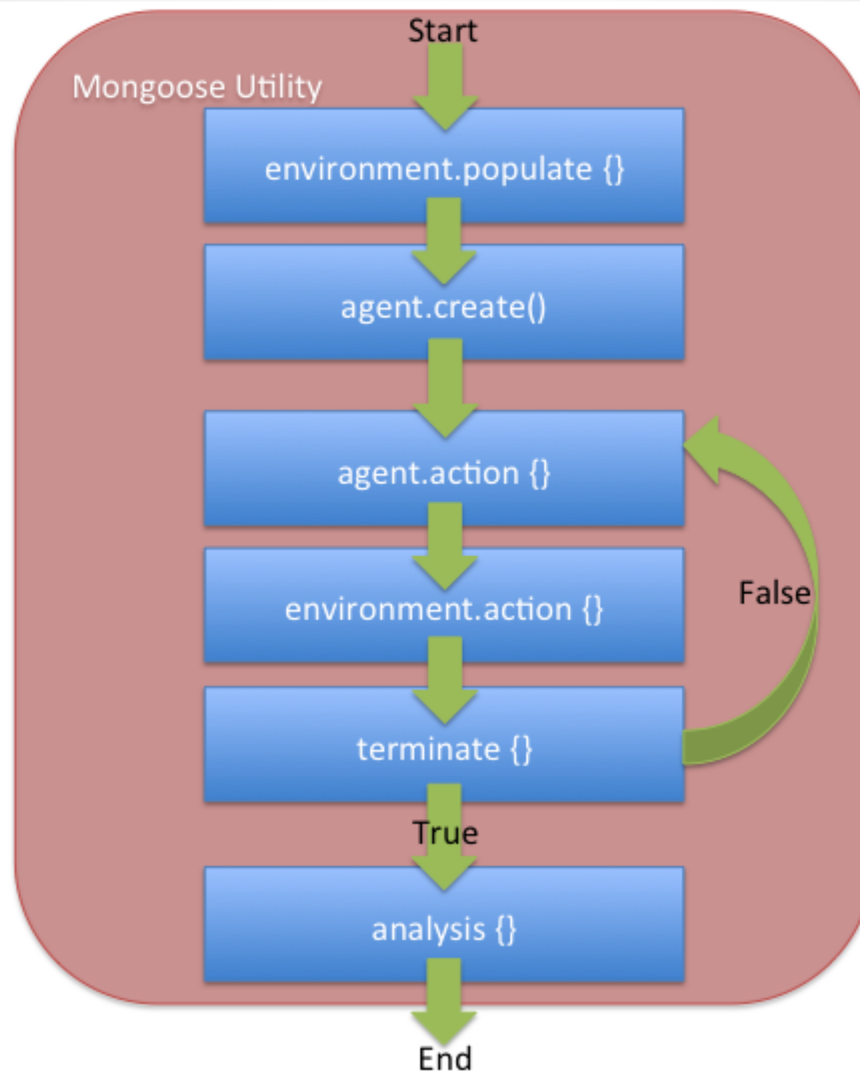
```
t_VINTEGER = r'[0-9]+'
t_VFLOAT = r'[0-9]*\.[0-9]+'
t_EQ = r'\=\='
t_NEQ = r'\!\='
t_LEQ = r'\<\='
t_GEQ = r'\>\='

def t_VSTRING(self, t):
    r'("[^"]*")|(\'[^\']*\')'
    # gets rid of the quotes
    t.value = t.value[1:-1]
    return t

def t_COMMENT(self, t):
   r'\#.*'
   pass
#t_ignore_COMMENT = r'\#.*'

def t_VBOOLEAN(self, t):
    r'true|false'
```

```
def p_agent_list_wrapper(p):
    ''' agent_list_wrapper : NEWLINE agent_list NEWLINE
                          | NEWLINE agent_list
                          | agent_list NEWLINE
                          | agent_list
    '''
    if len(p) == 3:
        p[0] = p[2]
    elif len(p) == 2:
        if type(p[1]) is str:
            p[0] = p[2]
        else:
            p[0] = p[1]
    else: #len(p) == 4
        p[0] = p[2]
```

Translator Architecture



Running a Mongoose program



COMS W1001

Assignment #5

Due April 25th

PIG: Pig is a two player game played with a single die. The object of the game is to accumulate 100 points. The player to roll first is randomly determined by coin flip. During each turn a player repeatedly rolls the die earning points equal to the face value on each roll until:

- A one is rolled, in which case the turn is over and the player forfeits all points earned so far on that turn.
- The player holds, takes the points earned so far and hands the dice over to the opponent.

Your job is to write a Python application that allows two computer players to play Pig. Your computer player should have two different possible modes of play. The beginner level computer strategy is simply to always hold after 3 rolls. The advanced level computer strategy is to halt after attaining 20 points on the turn.

(demos)

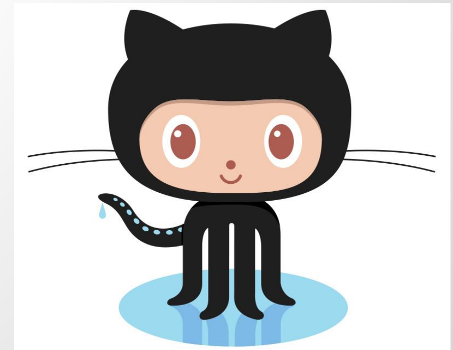
Mongoose Runtime Environment

- Python 2.7.2 (CPython)
- Unix only (for now)
- manage.sh

```
if [ $# -eq 0 ]
then
    which bpython >/dev/null 2>&1
    bpython_status=$?
    which ipython >/dev/null 2>&1
    ipython_status=$?
    if [ $bpython_status -eq 0 ]
    then
        PYTHONPATH='.' bpython $*
    elif [ $ipython_status -eq 0 ]
    then
        PYTHONPATH='.' ipython $*
    else
        PYTHONPATH='.' python $*
    fi
elif [ $1 = 'tests' ]
then
    PYTHONPATH='.' nosetests $*
else
    PYTHONPATH='.' python $*
fi
```

Development Environment

- pip + virtualenv + virtualenvwrapper
- git + github
- emacs / vim / sublime text 2
- cactus.py (markdown → HTML)



Future (v1.1) Features

- parallelizing (iid)
- state saving (stop/start programs)
- built-in aggregation decorators (@average)
- fix current bugs

Test Plan

```
1. sniffer (python)
(plt)➜ mongoose git:(master) ✗ sniffer
Starting watch...
(<file_validator py_files>,)
Using scent:
...................................................................
----------------------------------------------------------------------
Ran 67 tests in 7.863s

OK
In good standing
```

- unit tests (assertions for doc and test)
- highly descriptive test names
- integration tests
- nose (automation)
- sniffer

```
(plt)➜ mongoose git:(master) ✗ ./manage.sh tests -v
test_boolean_op_and (mongoose.tests.test_backend.BackendTests) ... ok
test_boolean_op_and_no_args (mongoose.tests.test_backend.BackendTests) ... ok
test_boolean_op_and_one_arg (mongoose.tests.test_backend.BackendTests) ... ok
test_boolean_op_not (mongoose.tests.test_backend.BackendTests) ... ok
test_boolean_op_or (mongoose.tests.test_backend.BackendTests) ... ok
test_float_bool_incomparable_equal (mongoose.tests.test_backend.BackendTests) ... ok
test_float_string_incomparable_equal (mongoose.tests.test_backend.BackendTests) ... ok
test_int_add (mongoose.tests.test_backend.BackendTests) ... ok
test_int_divide (mongoose.tests.test_backend.BackendTests) ... ok
test_int_float_incomparable_equal (mongoose.tests.test_backend.BackendTests) ... ok
test_int_greater_than (mongoose.tests.test_backend.BackendTests) ... ok
test_int_greater_than_equal (mongoose.tests.test_backend.BackendTests) ... ok
test_int_less_than (mongoose.tests.test_backend.BackendTests) ... ok
```

Conclusions

- rely heavily on git
- consistent velocity → work gets done
- *state* is a bitch
- frontend ambiguity → backend complexity
- grow by accretion & integrate earlier

Special thanks to:

- Karan
- Professor Aho