narratr

a language for text adventures

THE TEAM

PROJECT MANAGER	Shloka Kini
LANGUAGE GURU	Nivvedan Senthamil Selvan
SYSTEM ARCHITECT	Yelin Hong
SYSTEM INTEGRATOR	Jonah Smith
TESTY PERSON	Cecilia Watt

--->> move right

narratr

-->> tell me more

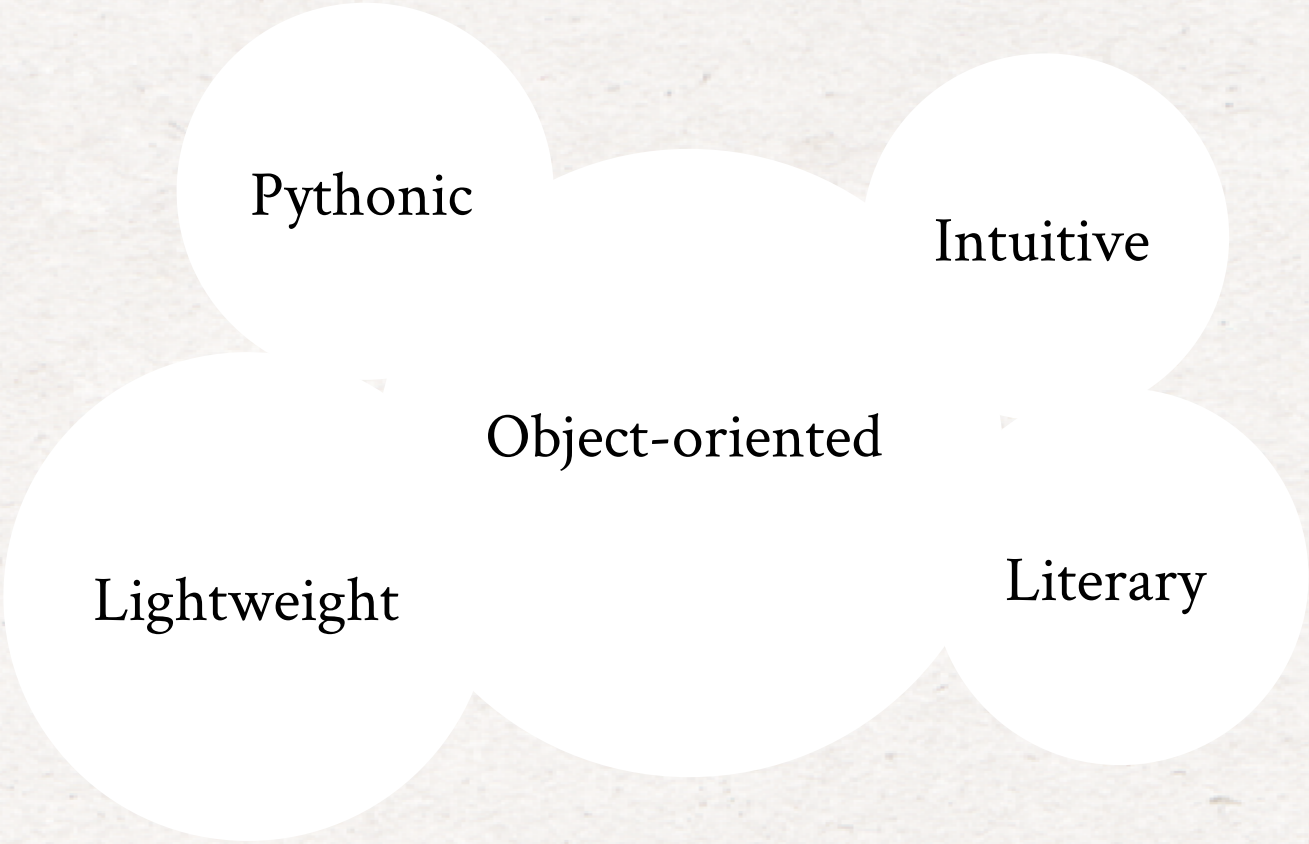
WHAT WE WANTED TO DO

Text-based adventure games are brilliant. We love them, and so should you.

A language that makes it easy to create such games can help in the revival of the glorious days.

A structured boilerplate for text-based games building upon a general purpose programming language.

BUZZWORDS


Pythonic
Intuitive
Object-oriented
Lightweight
Literary

WHAT WE WANTED TO DO

What is a generalizable quality of text adventures?

- Multiple Scenes
- Interaction with scenes
- Scene transitions on player input

Remind you of something?

WITHOUT ANY FURTHER ADO

```
% Here's the Hello, World! program
scene $1 {
   setup:
      say "Hello, World!"
      win
   action:
   cleanup:
}
start: $1 % Optional
```

WITHOUT ANY FURTHER ADO

```
% Here's the Hello, World! program
scene $1 {
    setup:
        say "Hello, World!"
        win
    action:
    cleanup:
}
start: $1 % Optional
```

Any general programming language construct can go here.

WITHOUT ANY FURTHER ADO

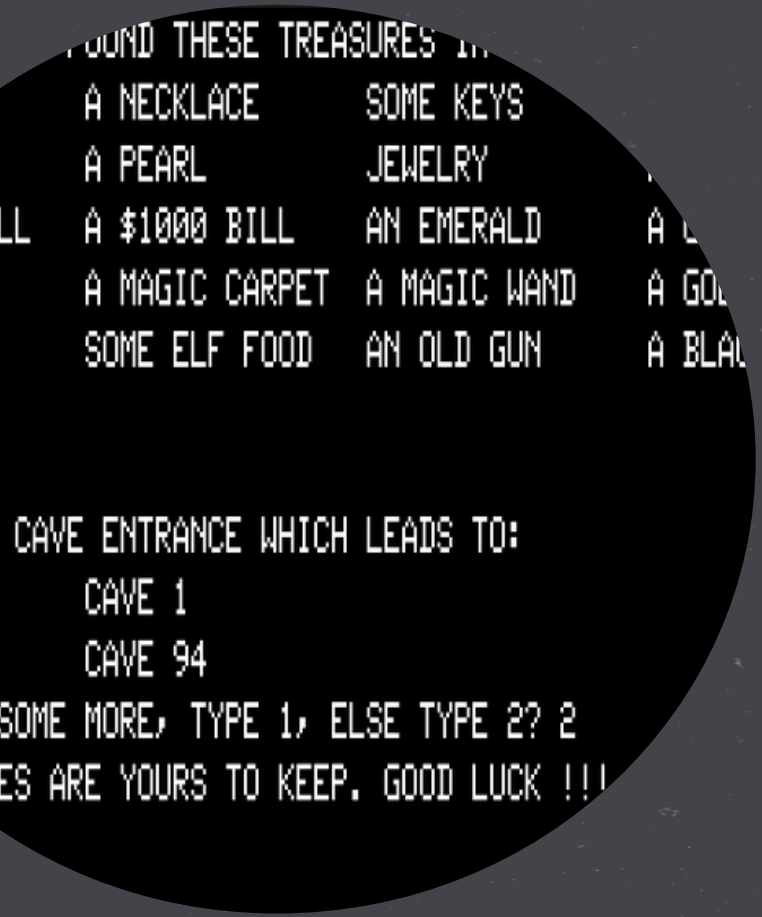
```
% Here's the Hello, World! program
scene $1 {
    setup:
        if not false == true:
            say "Hello, World!"
        win
    action:
    cleanup:
}
start: $1 % Optional
```

See?
Looks like Python,
doesn't it?

NARRATR

a language designed to build text-based adventure games

INTERACTIVE FICTION



The first text-adventure game was written in 1975 and was distributed through ARPANET. In the late 70s and early 80s, when most home computers had limited graphics capabilities, text-based games reached their peak popularity.

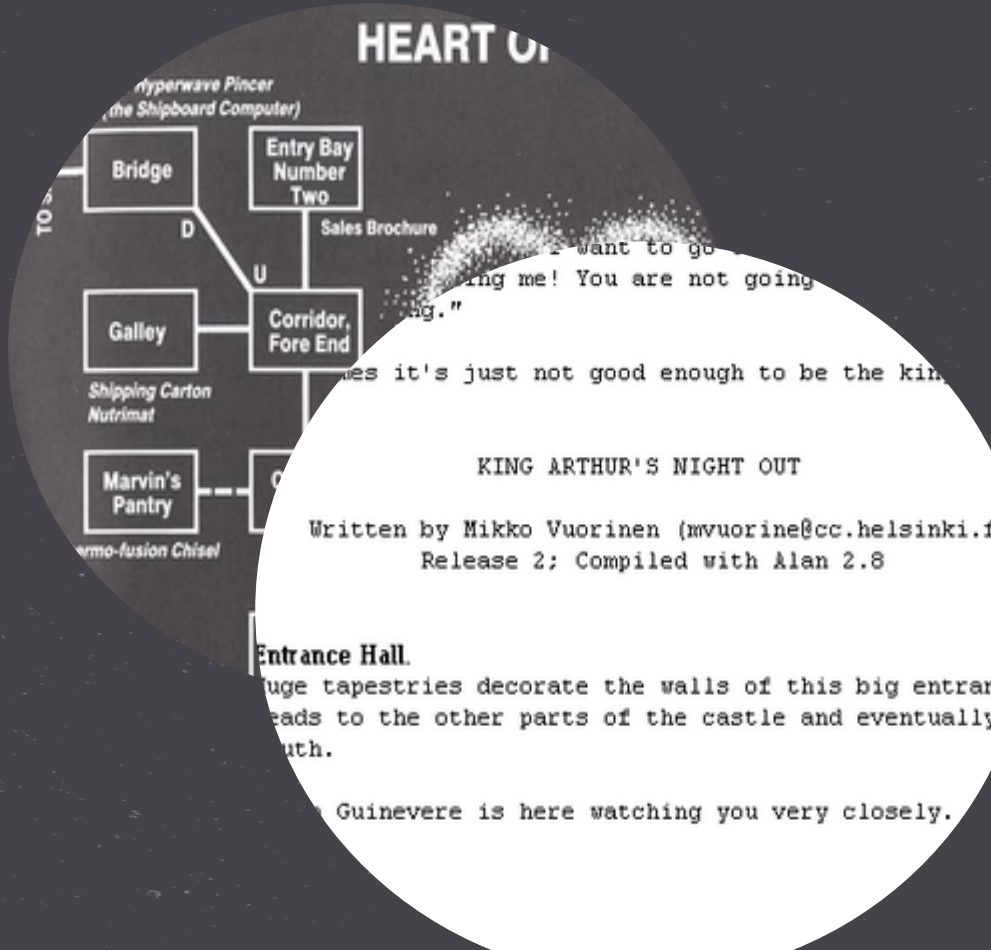
By the 90s, it was an art form in decline.

INTERACTIVE FICTION

The success of a text-based game hinges almost entirely on the strength of the game's storytelling.

Interactive fiction demands readers to take an active role in the telling of a story.

Text forces readers to exercise the imagination.



1.

SYNTACTIC CONSTRUCTS

WHAT A GAME IS MADE OF

scenes

items

RUNNING EXAMPLE

```
scene $1{
    setup:
      exposition "You are in a room. It has
a key."
      moves right($2)
    action:
      if response == "pick up key":
           pocket.add("key", key(1))
      else:
          "There's a key on the floor. What
do you want to do with it?"
    cleanup:
}

item key(keyid){
    id is keyid
}
```

```
scene $2{
    setup:
      exposition "Now you are in a new room.
In the corner, you see a locked door."
    action:
      if response == "open door":
           if pocket.has("key"):
                say "You unlocked the door."
                win "You won!"
           else:
                say "You don't have the key."
    cleanup:
}
```

SCENES

- They are numbered
- Have three components sub-blocks to them
  - Setup
  - Action
  - Cleanup
- All or any of them can be empty.
- Action block executes in an REPL
- Player can transition using the **move** command
- Programmer can transition with a **moveto** statement

ITEMS

- They're like classes (or should we say structs?)
- Can create objects of items
- Can set and access attribute values
- Can be carried around in your **pocket**

```
item key (keyid) {
   id is keyid
}
```

POCKET

- Global container for the player's inventory
- Can add, remove and update items in **pocket**
- The items in **pocket** can be accessed and modified by all scenes
- Pocket can be used to simulate function calls

LANGUAGE OVERVIEW

say / exposition statements

win / lose statements

`if` statments

`while` loops

`moves` declaration

`moveto` statement

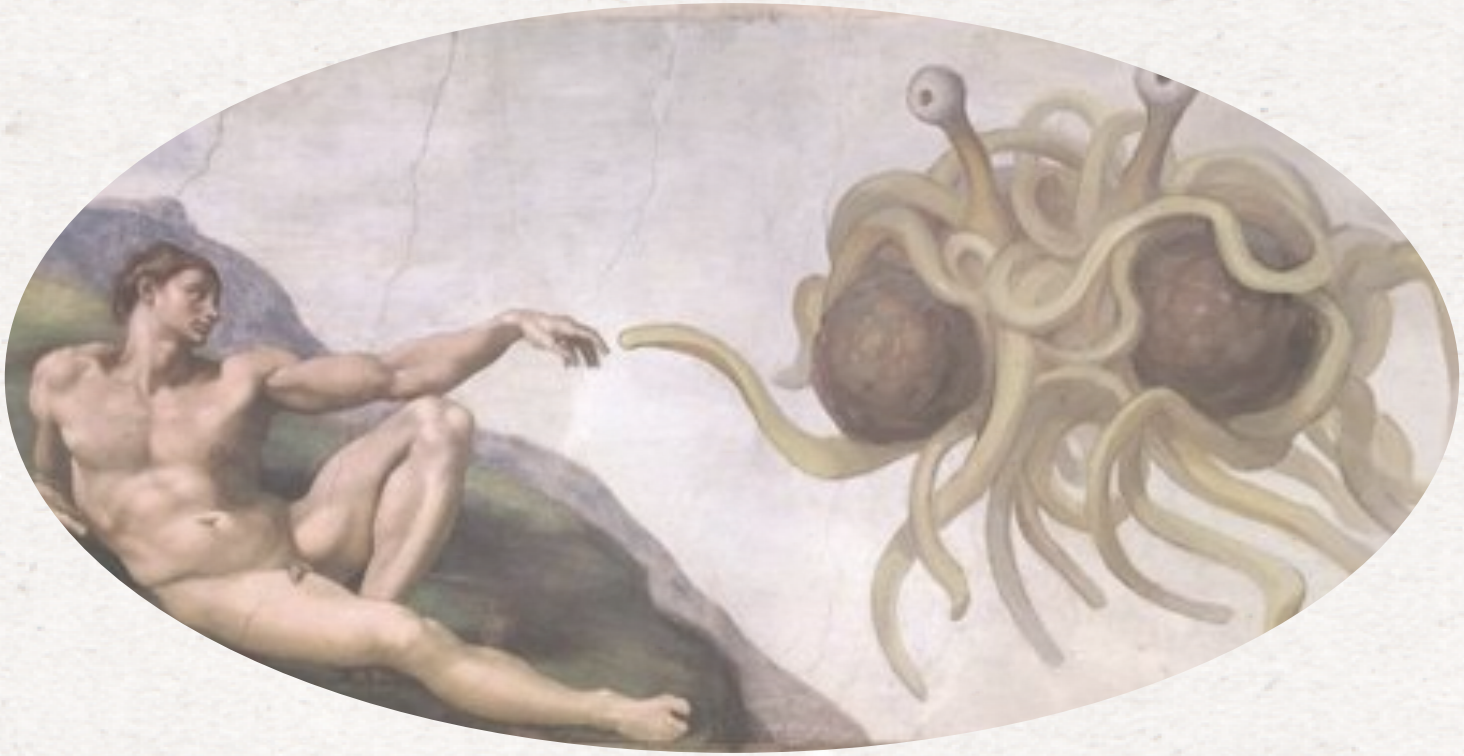
LANGUAGE OVERVIEW

assignment statements

- **`is`** operator
- **`god`** modifier
- creating item objects

```
k is key(1)
```

GOD



Variables persist in scenes and cannot be reinitialized.

2.

ARCHITECTURE

HOW TO COMPILE AND RUN

- `python narratr.py helloworld.ntr`
  - You can add -t after narratr.py if you want to print out the AST.
  - This instruction produces helloworld.ntr.py
- `python helloworld.ntr.py`
  - This executes the compiled output.

SYSTEM ARCHITECTURE

helloworld.ntr

```
scene $1 {
    setup:
        say "Hello, World!"
        win
    action:
    cleanup:
}
```

scanner and parser

SYSTEM ARCHITECTURE

AST of helloworld.ntr:

code generator

helloworld.ntr.py

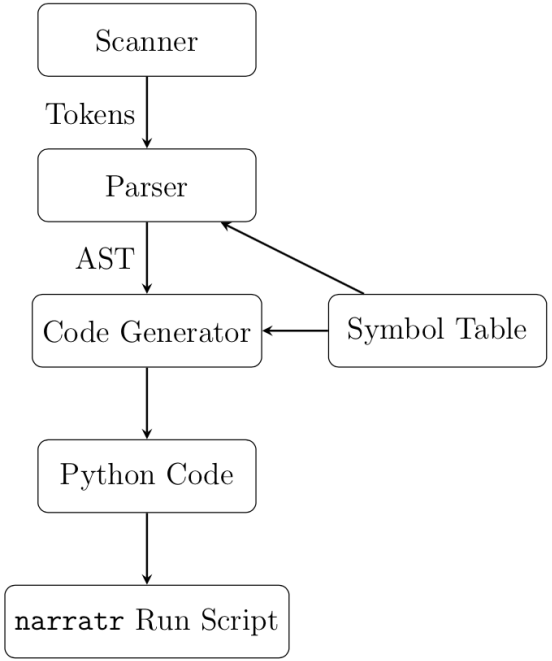
```
class s_1:
    def __init__(self):
        self.__namespace = {}
        self.directions = {}

    def setup(self):

        print 'Hello, World!'
        exit(0)
        return self.action()

    def action(self):
        response = ""
        while True:
            response = get_response(self.directions)
            if isinstance(response, list):
                self.cleanup()
                return response[0]

    def cleanup(self):
        self.__namespace = {}
```

SYSTEM ARCHITECTURE

80 lines of Python code are generated from 9 lines of narratr code

UNDER THE HOOD

Architecture Diagram



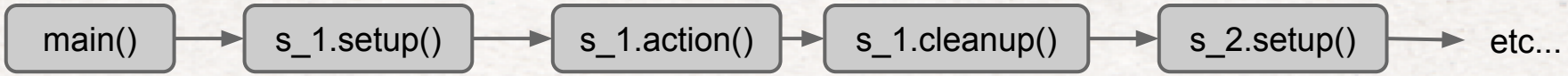
3.

ENVIRONMENT

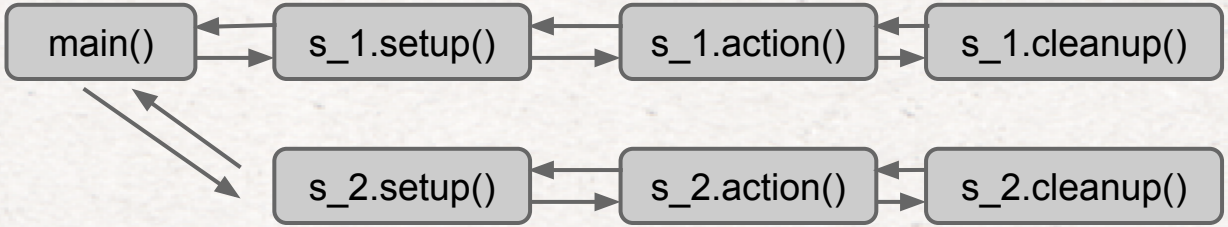
RUNTIME ENVIRONMENT

- Python 2.7+ (not Python 3) interpreter
- Scenes, Items → Python classes
- User interface challenges
  - Response normalization
    - `MOVE: left →  move left`

runtime activation tree



vs.



etc...

DEVELOPMENT ENVIRONMENT

- Local systems (Mac OS X)
- Sublime Text and TextMate
- Python 2.7.9
- Git and GitHub
- Testing tools (`examine.py`, `narratr.py -vti`, `nosetests`)

4.

MANAGEMENT

PROJECT MANAGEMENT

- Written project plan
  - Week-by-week
  - Individual tasks
  - Buffer time
  - Weekly Meetings

5.

COMPILER TOOLS

Python Lex-Yacc (PLY)

- Quite easy to use and well documented with examples.
- Integrates seamlessly with Python
- Dummy/pseudo tokens were not straightforward
- But ... was possible to look at the lex source and design a workaround.

6.

TESTING & VALIDATION

TESTING

We used an automated test suite, built with Python's unittest framework as well as nose.

We tested that programs would compile and print appropriate output.

We tested that faulty code would have errors.

STATISTICS

69 tests

2031 lines of test code

5074 total lines

40% of our code is tests

coverage

100%	narratr.py
91%	lexer.py
88%	parser.py
90%	node.py
80%	codegen.py
81%	symtab.py

7.

A DEMONSTRATION

LESSONS LEARNED

- When in doubt, always look at the grammar.
- There's value in coding together in the same room.
- Parallelize work when possible.
- Everyone should participate in writing tests.
- Be confident and trust your prior self.

```
-->> You win!
```