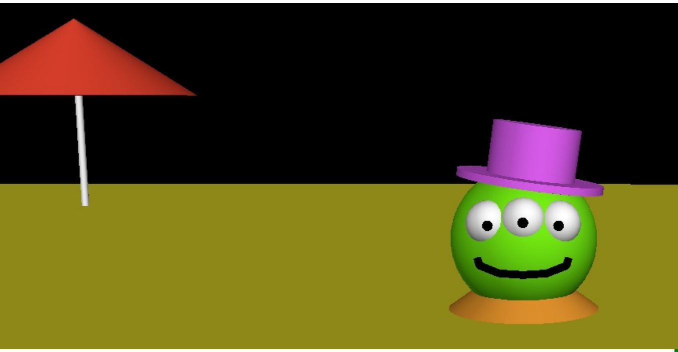
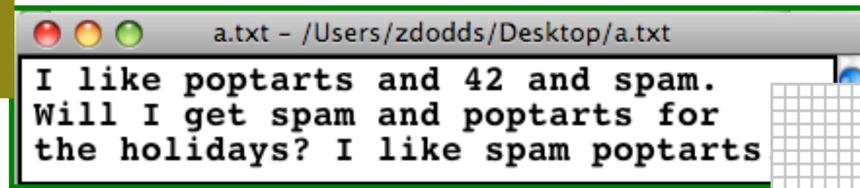
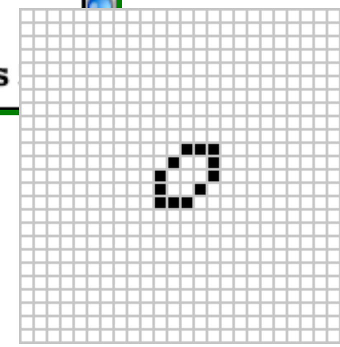
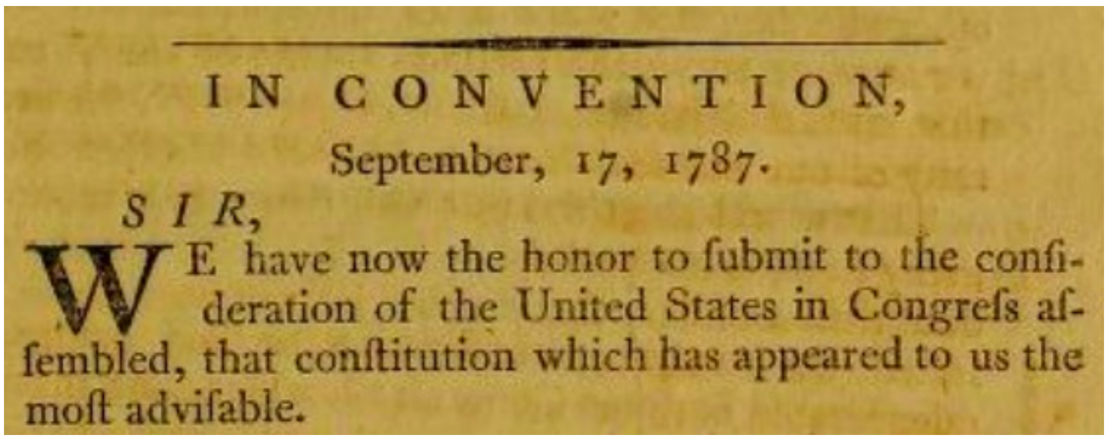


CS 5 Today







IN CONVENTION,

September, 17, 1787.

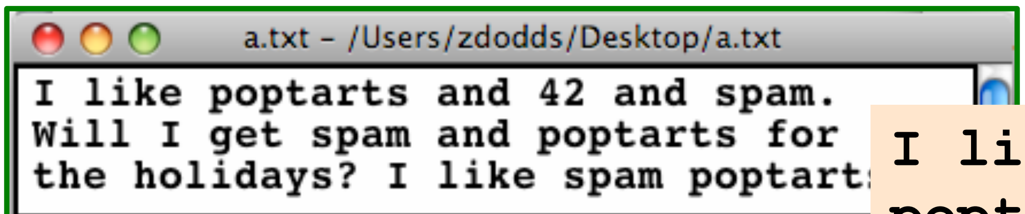
SIR,

WE have now the honor to fubmit to the confideration of the United States in Congrefs affembled, that conftitution which has appeared to us the moft advifable.

Text generation, Dictionaries, and *the final countdown!*

Markov Models

Techniques for modeling *any* sequence of **natural data** ← speech, text, sensor data...




```
I like spam and 42 and
poptarts and poptarts and
poptarts and 42 and spam.
```

Each item depends ***only*** on the <u>one</u> immediately before it .

1st-order Markov Model (defining property)

Markov Models

Techniques

sequence of …

```
a.txt – /Users/z
I like poptarts an
Will I get spam an
the holidays? I li
```

spam.

Let's revisit an old **class**y friend!

(a helpful <u>data structure</u>)

Each item depends ***only*** on the <u>one</u> immediately before it.

1st-order Markov Model (defining property)

Lists are *sequential* containers:

element

L = [42, 5, 47, 42]

0 1 2 3

index

elements are looked up by their **location**, or **index**, starting from 0

Lists are *sequential* containers:

element

```
L = [ 42, 5, 47, 42 ]
```

0 1 2 3 **index**

elements are looked up by their **location**, or **index**, starting from 0

Dictionaries are *arbitrary* containers:

"associative"

```
d = {  5: 1,   42: 3 }
```

key value **key** value

elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

Lists are *sequential* containers:

element

```
L = [ 42, 5, 47, 42 ]
```

0 1 2 3 **index**

elements are looked up by their **location**, or **index**, starting from 0

Dictionaries are *arbitrary* containers:

"associative"

```
d2 = {'b': 2, 42:'yay'}
```

key value **key** value

elements (or *values*) are looked up by a **key** starting anywhere you want! **Keys** don't have to be ints!

an example dictionary: `NL`

```
NL = { 'a':'b', 'b':'c', 'c':'d', 'd':'e', 'e':'f',
       'f':'g', 'g':'h', 'h':'i', 'i':'j', 'j':'k',
       'k':'l', 'l':'m', 'm':'n', 'n':'o', 'o':'p',
       'p':'q', 'q':'r', 'r':'s', 's':'t', 't':'u',
       'u':'v', 'v':'w', 'w':'x', 'x':'y', 'y':'z',
       'z':'a' }
```

Dictionaries
have value!

... it just *looks up* the next letter!

Dictionaries are *lookup tables!*

```
zd = {'monkey':2004, 'goat':2003}
```

key — value — key — value

zd_small

elements (***values***) are looked up by a **key** – which can be anything needed!

Keys don't have to be ints!

What's **zd**'s data here?

Now I see the **key** to dictionaries' **value**...

Dictionaries are *lookup tables!*

```
zd = {'monkey':2004, 'goat':2003}
```

key value key value

elements (***values***) are looked up by a **key** – which can be anything needed!

Dragon	Feb 05 **2000**–Jan 23 2001
Snake	Jan 24 **2001**–Feb 11 2002
Horse	Feb 12 **2002**–Jan 31 2003
Goat	Feb 01 **2003**–Jan 21 2004
Monkey	Jan 22 **2004**–Feb 08 2005
Rooster	Feb 09 **2005**–Jan 28 2006
Dog	Jan 29 **2006**–Feb 17 2007
Pig	Feb 18 **2007**–Feb 06 2008
Rat	Feb 07 **2008**–Jan 25 2009
Ox	Jan 26 **2009**–Feb 13 2010
Tiger	Feb 14 **2010**–Feb 02 2011
Rabbit	Feb 03 **2011**–Jan 22 2012

Keys don't have to be ints!

12-year zodiac!

Now I see the **key** to dictionaries' **value**...

Dictionaries are *lookup tables!*

zi

```
zy = {'goat':[2003,1991,1979, … ],
      'monkey':[2004,1992,1980, … ],
      'rooster':[2005,1993, … ], … }
```

What type are the ***keys***?

What type are the ***values***?

`z.keys()`

`z.values()`

these seem key to dictionaries' value

`z.items()`

Dictionaries are `in`:


```
zy = {'goat':[2003,1991,1979, … ],
      'monkey':[2004,1992,1980, … ],
      'rooster':[2005,1993, … ], … }
```

Is `'rooster'` a key in **z**?

```
if 'rooster' in z
```

???

Is `'alien'` a key in **z**?

```
if 'alien' in z
```

???

What?
How do I get **`in`**?!

Dictionaries are `in`:

zi

```
zy = {'goat':[2003,1991,1979, … ],
      'monkey':[2004,1992,1980, … ],
      'rooster':[2005,1993, … ], … }
```

Is `'rooster'` a key in **z**?

```
if 'rooster' in z
```

True

Is `'alien'` a key in **z**?

```
if 'alien' in z
```

False

What?
How do I get **`in`**?!

Given these two dictionaries: *What are these expressions?*

```
NL = {'a':'b', 'b':'c',
      'c':'d', 'd':'e',
  # imagine they're all here...  ... ,
      'y':'z', 'z':'a' }
```

```
dc = { 42     : 'answer',
       'cs'   : 5,
       'seis' : 6,
       'a'    : 'o',
       'e'    : 'g',
       5      : NL     } # uh oh
```

```
NL['a'] == 'b'
NL['v'] == 'w'

len(NL) == 26
len(dc) == 6

5 in dc   (True or False?)
6 in dc   (True or False?)

dc[NL['z']] == 'o'
dc[dc['cs']][dc['e']] == 'h'
```

Given this list + algorithm: *What is the resulting dictionary?!*

```
LoW = [ 'spam', 'spam',
        'poptarts', 'spam' ]

d = {}
for w in LoW:
  if w not in d:
    d[w] = 1
  else:
    d[w] += 1
```

```
d = {

      }
```

Hint! There will be only TWO keys in **d**!!

Name(s) ______________

Given these two dictionaries: *What are these expressions?*

```
NL = {'a':'b', 'b':'c',
      'c':'d', 'd':'e',
      'e':'f', 'f':'g',
      'g':'h', 'h':'i',
      'c':'d', 'd':'e',
      'c':'d', 'd':'e',
      'c':'d', 'd':'e',
      'c':'d', 'd':'e',
      'c':'d', 'd':'e',
      'y':'z', 'z':'a' }
```

```
dc = { 46     : 'CMC',
       'cs'   : 5,
       'seis' : 6,
       'a'    : 'o',
       'e'    : 'g',
       5      : NL    }
```

```
NL['a'] ==
NL['z'] ==
NL['v'] ==

len(NL) ==
len(dc) ==
len(dc)  ==

'g' in NL  (True or False?)
'Z' in NL  (True or False?)

5 in dc  (True or False?)
6 in dc  (True or False?)

dc[NL['z']] ==
dc[dc['cs']][dc['e']] ==
```

Given these two dictionaries:

```
NL = {'a':'b', 'b':'c',
      'c':'d', 'd':'e',
  # imagine they're all here...   ... ,
      'y':'z', ...
```

```
dc
```

```
  ...   : 'g',
5       : NL      } # uh oh
```

What are these expressions?

```
NL['a'] ==
NL['y'] ==
```

```
)
... dc  (True or False?)

dc[NL['z']] ==
dc[dc['cs']][dc['e']] ==
```

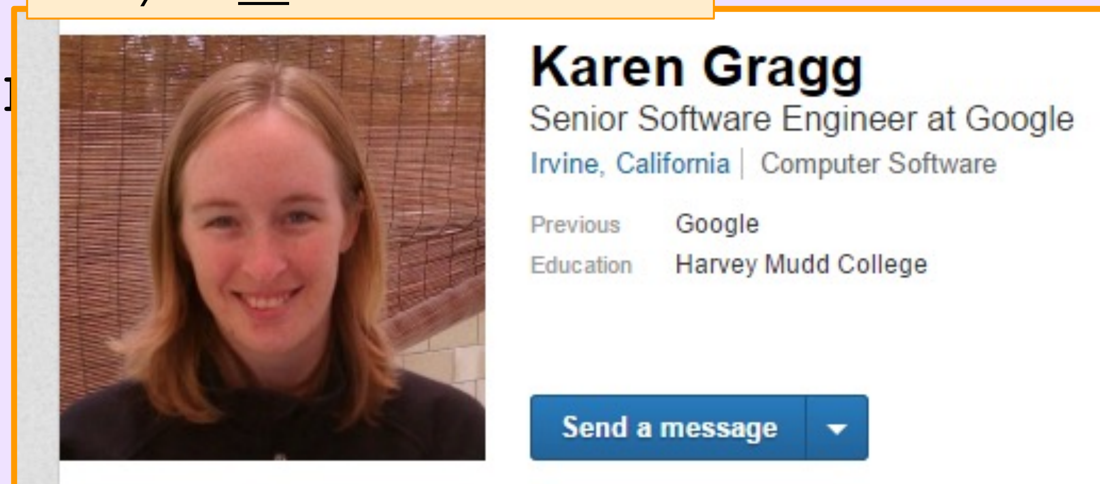
dictionaries are one of Python's built-in ***classes***

I can't tell you any of the questions -- but I can tell you all the solutions!

Given this list + algorithm:

```
LoW = [ 'spam', 'spam',
        'poptarts', 'spam' ]

d = {}
for w in LoW:
   if w not in d:
      d[w] = 1
   else:
      d[w] += 1
```

...ctionary?!



Given these two dictionaries:

What are these expressions?

```
NL = {'a':'b', 'b':'c',
      'c':'d', 'd':'e',
  # imagine they're all here...   ... ,
      'y':'z', ...
```

```
NL['a']
```

```
dc
```

Pass those Mountainward!

```
...alse?)
...]] ==
...[dc['cs']][dc['e']] ==
```

I can't tell you _any_ of the questions -- but I can tell you _all_ the solutions!

Giv... ...algorithm:

...ctionary?!

```
LoW = [ 'spam', 'spam',
        'poptarts', 'spam' ]

d = {}
for w in LoW:
  if w not in d:
    d[w] = 1
  else:
    d[w] += 1
```

Karen Gragg

Senior Software Engineer at Google

Irvine, California | Computer Software

Previous Google

Education Harvey Mudd College

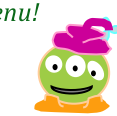
Send a message

"The algorithm..."

```
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]
```

w

Hochsgiving menu!



d starts...

{}

```
d = {}

for w in LoW:

   if w not in d:

      d[w] = 1

   else:

      d[w] += 1
```

w is...

next, **d** is

{'spam' : 1 }

then, **d** is

{'spam': 2 }

then, **d** is

{'spam': 2 , 'poptarts' : 1 }

{'poptarts':1, 'spam':3}

final **d**

vc_print(LoW)

vc_print("a.txt")

"The algorithm that counts!"

```
LoW = [ 'spam', 'spam', 'poptarts', 'spam' ]
```

The Hoch's menu!

```
d = {}

for w in LoW:

   if w not in d:

      d[w] = 1

   else:

      d[w] += 1
```

w is...

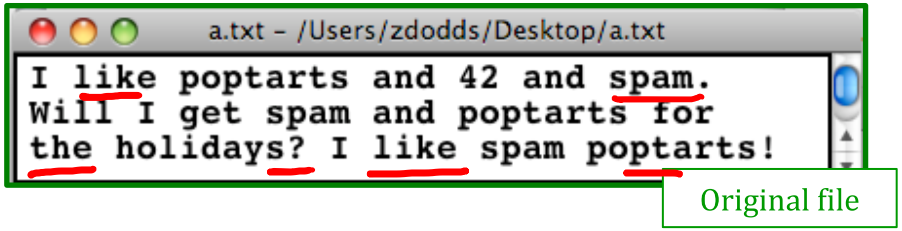
d is...

w is...	d is...
	{}
w ='spam'	{'spam':1} ← next, **d** is
w ='spam'	{'spam':2} ← then, **d** is
w ='poptarts'	{'poptarts':1, 'spam':2} ← then, **d** is
w ='spam'	{'poptarts':1, 'spam':3} ← final **d**

What do you think `len(d)` is?

```
vc_print(LoW)

vc_print("a.txt")
```

A *counting* model...



Counting Model

dictionary

keys | values

```
{ '$':        3,
  'I':        3,
  'like':     2,
  'poptarts':2,
  'and':      3,
  '42':       1,
  'Will':     1,
  'the':      1,
  'spam':     2,
  'get':      1,
  'for':      1            }
```

dictionary's end

- What types are the **keys**?
- What types are the *values*?
- What are the *missing* values?
- What is the '$'?
- Why is the key `holidays` missing?

A *Markov* Model

```
I like poptarts and 42 and spam.
Will I get spam and poptarts for
the holidays? I like spam poptarts!
```

a.txt – /Users/zdodds/Desktop/a.txt

Original file

Markov Model

dictionary

keys / values

```
{ '$':        ['I', 'Will', 'I'],
  'I':        ['like', 'get', 'like'],
  'like':     ['poptarts', 'spam']
  'poptarts': ['and', 'for'],
  'and':      ['42', 'spam.', 'poptarts'],
  '42':       ['and'],
  'Will':     ['I'],
  'the':      ['holidays!']
  'spam':     ['and', 'poptarts!'],
  'get':      ['spam'],
  'for':      ['the']                }
```

dictionary's end

- What types are the **keys**?
- What types are the *values*?
- What are the *missing* values?
- What is the '$'?
- Why is the key `holidays` missing?

A *Markov* Model

solutions

a.txt – /Users/zdodds/Desktop/a.txt

```
I like poptarts and 42 and spam.
Will I get spam and poptarts for
the holidays? I like spam poptarts!
```

Original file

keys values

```
{ '$':        ['I', 'Will', 'I'],
  'I':        ['like', 'get', 'like'],
  'like':     ['poptarts', 'spam'],
  'poptarts': ['and', 'for'],
  'and':      ['42', 'spam.', 'poptarts'],
  '42':       ['and'],
  'Will':     ['I'],
  'the':      ['holidays?'],
  'spam':     ['and', 'poptarts!'],
  'get':      ['spam'],
  'for':      ['the']                  }
```

Markov Model

A dictionary!

dictionary's end

What types are **keys**? **strings**

What types are *values*? **lists** (of strings that follow!)

What are the *missing* values? filled in: ['poptarts', 'spam'] and ['holidays?']

What is the '$'? **start-of-sentence symbol**

Why is the key `holidays` missing? **punctuation counts!**

Markov's algorithm

LoW → ['I','like','spam.','I','eat','poptarts!']

pw → $ | 'I' | 'like' | ~~'spam.'~~ '$' |

nw → 'I' | 'like' | 'spam.' | 'I' |

cdi_print(PT2)

cdi_print("a.txt")

```
d = {}
pw = '$'        pw ~ previous word

                nw ~ next word
for nw in LoW:
    if pw not in d:
        d[pw]  =  [nw]
    else:
        d[pw] += [nw]

    pw = nw
```

d in creation
(with unquoted strings)

$:
I :
like :
eat :

d's final form
(with unquoted strings)

$: [I, I]
I : [like, eat]
like : [spam.]
eat : [poptarts!]

Markov's *algorithm ...*

solutions & starting point

LoW ['I','like','spam.','I','eat','poptarts!']

pw →	$	I	like	$	I	eat
nw →	I	like	spam.	I	eat	poptarts!

```
d = {}
pw = '$'        pw ~ previous word

                nw ~ next word
for nw in LoW:
    if pw not in d:
        d[pw]  = [nw]
    else:
        d[pw] += [nw]

    pw = nw
```

for hw10pr3: check if **pw** ends with punctuation and, if so, set to '$'

d's final form
(with unquoted strings)

$: [I, I]
I : [like, eat]
like : [spam.]
eat : [poptarts!]

Markov's *algorithm ...*

solutions & starting point

LoW ['... arts!']

cdi_print(PT2)

cdi_print("a.txt")

pw → $

nw → I ... ts!

```
d = {}
pw = '$'        pw ~ prev

for nw in LoW:
    if pw not i
        d[pw]   =
    else:
        d[pw] += [nw]

    pw = nw
```

But where do we get all these "words" to

1. create new models ...
2. generate new texts ...

d's final form

strings)

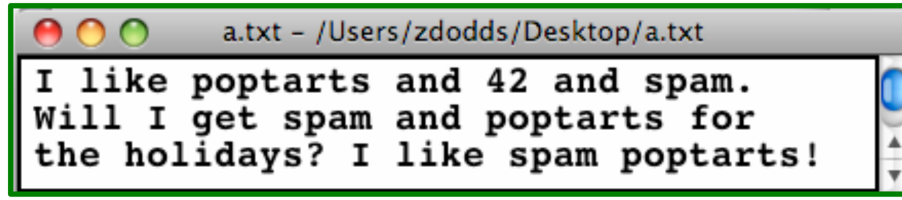
for hw10
ends wit
and, if so, set to '$'

eat : [poptarts!]

FILES !

Files...

In Python reading files is smooth…




```
f = open( 'a.txt' )
```

opens the file and calls it **f**

```
text = f.read()
```

reads the whole file into the string **text**

```
f.close()
```

closes the file (optional)

```
text
'I like poptarts and 42 and spam.\nWill I
```

```
LoW = text.split()
[ 'I', 'like', 'poptarts', ... ]
```

text.split() returns a list of each "word"

```python
def get_text( filename ):
    """
        return all text from
        the file, filename
    """

    f = open( filename, "r" )
    text = f.read()
    f.close()

    return text
```

file handling

This function is provided in hw10pr3.py ... *try it!*

```python
def word_count( text ):

    LoW = text.split()
    result = len(LoW)
    print("There are",result,"words")

    return result
```

string handling

What if we wanted the number of ***different*** words in the file?

This would be the author's ***vocabulary count***, instead of the total word count.

Vocabulary counting...

```
def vocab_count( text ):

    LoW = text.split()
    d = {}

    for w in LoW:
        if w not in d:
            d[w] = 1
        else:
            d[w] += 1

    print("There are", len(d), "distinct wds.")

    return d
```

list of words

the dictionary, d

"the algorithm"

Our counting model, as before...

return for later use ...

most/least common?

Vocabulary!

Shakespeare used ***31,534 <u>different</u> words*** -- and a grand total of 884,647 words across all his works....

gust
besmirch
unreal
superscript
watchdog
swagger

successful

affined
rooky
attasked
out-villained

unsuccessful

Shakespearean coinages

http://www.pathguy.com/shakeswo.htm
http://www.shakespeare-online.com/biography/wordsinvented.html
http://www-math.cudenver.edu/~wbriggs/qr/shakespeare.html

Your CS-Essay… !

Find a file, could be your own ~ or one you find online…
~ then ~
Copy its text into VSCode and save under a new name

Your CS-Essay... !

Find a file, could be your own ~ or one you find online...
~ then ~
Copy its text into VSCode and save under a new name

Create a **Markov Model**, perhaps named `d`

Generate a 500-word CS-Essay using your model!

Your CS-Essay... !

Find a file, could be your own ~ or one you find online...
~ then ~
Copy its text into VSCode and save under a new name

Create a **Markov Model**, perhaps named `d`

Generate a 500-word CS-Essay using your model!

Share the whole essay you generate, *plus*...

... 2-3 of your favorite Markov-generated insights!

Generating prose? Academic Opportunity!

Some of last year's highlights:

She can hear the noise of fresh ink.
- Elena A. (via a novel by A. Doerr)

I'll be chill but divine intervention I laughed.
- Yazmin Meza (via Jason Mraz)

During her summer internship, she was the Roberts Environmental Center.
- May McD. (own writing)

... 2-3 of your favorite Markov-generated insights!

WMSCI 2005

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

http://pdos.csail.mit.edu/scigen/

Markov-generated submission accepted to WMSCI '05

Not a first-order, but a ***third-order****, model*

Rooter: A Methodology for the Typical Unification of Access Points and Redundancy

Jeremy Stribling, Daniel Aguayo and Maxwell Krohn

ABSTRACT

Many physicists would agree that, had it not been for congestion control, the evaluation of web browsers might never have occurred. In fact, few hackers worldwide would disagree with the essential unification of voice-over-IP and public-private key pair. In order to solve this riddle, we confirm that SMPs can be made stochastic, cacheable, and interposable.

I. INTRODUCTION

Many scholars would agree that, had it not been for active networks, the simulation of Lamport clocks might never have occurred. The notion that end-users synchronize with the investigation of Markov models is rarely outdated. A theoretical grand challenge in theory is the important unification of virtual machines and real-time theory. To what extent can web browsers be constructed to achieve this purpose?

Certainly, the usual methods for the emulation of Smalltalk that paved the way for the investigation of rasterization do not apply in this area. In the opinions of many, despite the fact that conventional wisdom states that this grand challenge is continuously answered by the study of access points, we

The rest of this paper is organized as follows. For starters, we motivate the need for fiber-optic cables. We place our work in context with the prior work in this area. To address this obstacle, we disprove that even though the much-tauted autonomous algorithm for the construction of digital-to-analog converters by Jones [10] is NP-complete, object-oriented languages can be made signed, decentralized, and signed. Along these same lines, to accomplish this mission, we concentrate our efforts on showing that the famous ubiquitous algorithm for the exploration of robots by Sato et al. runs in $\Omega((n + \log n))$ time [22]. In the end, we conclude.

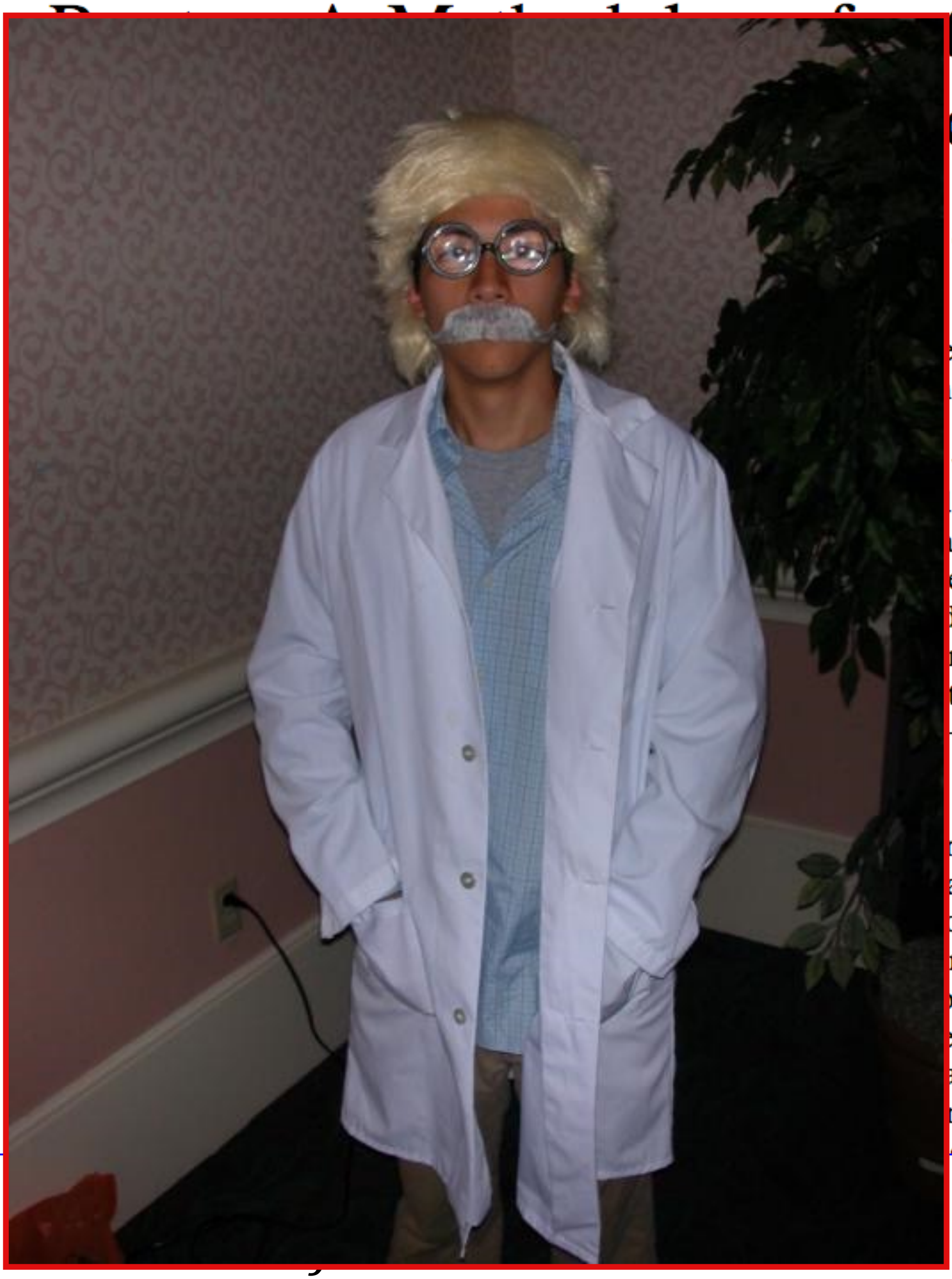
II. ARCHITECTURE

Our research is principled. Consider the early methodology by Martin and Smith; our model is similar, but will actually overcome this grand challenge. Despite the fact that such a claim at first glance seems unexpected, it is buffetted by previous work in the field. Any significant development of secure theory will clearly require that the acclaimed real-time algorithm for the refinement of write-ahead logging by Edward Feigenbaum et al. [15] is impossible; our application is no different. This may or may not actually hold in reality.

Not a first-order model … but a ***third-order*** *model*

the Typical Unification
d Redundancy

and Maxwell Krohn



rest of this paper is organized as follows. For starters, otivate the need for fiber-optic cables. We place our in context with the prior work in this area. To ad- this obstacle, we disprove that even though the much- autonomous algorithm for the construction of digital- log converters by Jones [10] is NP-complete, object- ed languages can be made signed, decentralized, and l. Along these same lines, to accomplish this mission, we ntrate our efforts on showing that the famous ubiquitous thm for the exploration of robots by Sato et al. runs in $+ \log n))$ time [22]. In the end, we conclude.

II. ARCHITECTURE

r research is principled. Consider the early methodology artin and Smith; our model is similar, but will actually ome this grand challenge. Despite the fact that such m at first glance seems unexpected, it is buffetted by us work in the field. Any significant development of theory will clearly require that the acclaimed real- algorithm for the refinement of write-ahead logging by rd Feigenbaum et al. [15] is impossible; our application different. This may or may not actually hold in reality.

the third-order wardrobe?

Your CS-Essay...

Find a file, could be your own ~ or one you find online...
~ and/or ~
Copy its text into VSCode and save it under a new .txt filename (!)

Create a **Markov Model**, perhaps named `d`

Generate a 500-word CS-Essay using your model!

Share the whole essay you generate, *plus*...

... 2-3 of your favorite Markov-generated insights!

Setting our homework timeline…

SUN	MON	TUE	WED	THU	FRI	SAT
3/31	4/1	4/2	4/3	4/4	4/5	4/6
4/7	4/8	4/9	4/10	4/11	4/12	4/13
4/14	4/15	4/16	4/17	4/18	4/19	4/20
4/21	4/22	4/23	4/24	4/25	4/26	4/27
4/28	4/29	4/30	5/1	5/2	5/3	5/4
5/5	5/6	5/7	5/8	5/9	5/10	5/11

hw10 4/9

hw11 4/16

hw12 4/23

Setting our lab timeline…

Lab 11 4/12
is the last
required lab

SUN	MON	TUE	WED	THU	FRI	SAT
3/31	4/1	4/2	4/3	4/4	4/5	4/6
4/7	4/8	4/9	4/10	4/11	4/12	4/13
4/14	4/15	4/16	4/17	4/18	4/19	4/20
4/21	4/22	4/23	4/24	4/25	4/26	4/27
4/28	4/29	4/30	5/1	5/2	5/3	5/4
5/5	5/6	5/7	5/8	5/9	5/10	5/11

Lab time 4/19 & 4/26
optional final project
and homework help

Setting our final project timeline…

Starter
4/17

SUN	MON	TUE	WED	THU	FRI	SAT
3/31	4/1	4/2	4/3	4/4	4/5	4/6
4/7	4/8	4/9	4/10	4/11	4/12	4/13
4/14	4/15	4/16	4/17	4/18	4/19	4/20
4/21	4/22	4/23	4/24	4/25	4/26	4/27
4/28	4/29	4/30	5/1	5/2	5/3	5/4
5/5	5/6	5/7	5/8	5/9	5/10	5/11

Final project
4/26 5 PM

Milestone
4/23

Setting our final exam timeline…

SUN	MON	TUE	WED	THU	FRI	SAT
3/31	4/1	4/2	4/3	4/4	4/5	4/6
4/7	4/8	4/9	4/10	4/11	4/12	4/13
4/14	4/15	4/16	4/17	4/18	4/19	4/20
4/21	4/22	4/23	4/24	4/25	4/26	4/27
4/28	4/29	4/30	5/1	5/2	5/3	5/4
5/5	5/6	5/7	5/8	5/9	5/10	5/11

Final review
5/5 7-9 PM
(optional)

Final exam
5/9 2-5 PM

Senior exam options
5/2-5/3

Setting our final timeline…

Starter
4/17

SUN	MON	TUE	WED	THU	FRI	SAT
3/31	4/1	4/2	4/3	4/4	4/5	4/6
4/7	4/8	4/9	4/10	4/11	4/12	4/13
4/14	4/15	4/16	4/17	4/18	4/19	4/20
4/21	4/22	4/23	4/24	4/25	4/26	4/27
4/28	4/29	4/30	5/1	5/2	5/3	5/4
5/5	5/6	5/7	5/8	5/9	5/10	5/11

hw10
4/9

hw11
4/16

hw12
4/23

Final review
5/5 7-9 PM
(optional)

Milestone
4/23

Final exam
5/9 2-5 PM

Senior exam options
5/2-5/3

Final project
4/26 5 PM

Setting our final timeline…

Starter
4/17

SUN	MON	TUE	WED	THU	FRI	SAT
3/31						
4/7						
4/14						
4/21						
4/28						
5/5						

hw10
4/9

hw11
4/16

hw12
4/23

Final review
5/5 7-9 PM
(optional)

4/23

ject

CS 5 Final projects

Overview

The lion's share of the effort in our last two weeks of CS5 goes into the course's final project. The project is larger in scope than a regular assignment and so is worth 200 points, spread across three deliverables: the *starter*, the *milestone*, and the *final version*.

We have five possible final-project themes in CS5. All provide the chance to build a significant software application in Python—and all of them also provide opportunities for creative expansion of the basic themes. Here are the project descriptions and links:

- Text-based board-game project, with AI (CS 5 Gold only)
- TextID: *Stylometry!* Text-style classification
- Powering Picobot via Genetic algorithms
- *More Life!* Variations on Conway's *Game of Life* (CS 5 Gold only)
- vPython: a 3D graphics simulation (or game)

Final projects

Final CS hw:

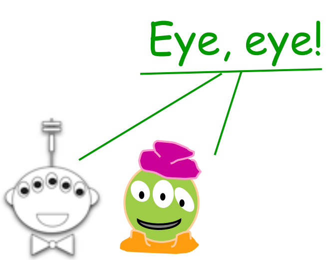
- open-ended
- comprehensive
- same projects across sections
- several choices…

Working in teams of 1-3 is OK

Teams need to work *together and at the same time,* and need to share the work equally…

Teams of 1, 2, or 3 are welcome.

Teaming is extra-encouraged on the final project!



Final-project options...

Choices of final project:

- *Picobot*
- *MoreLife*
- *TextID*
- *TextGame*

we've already covered the background you'll need!

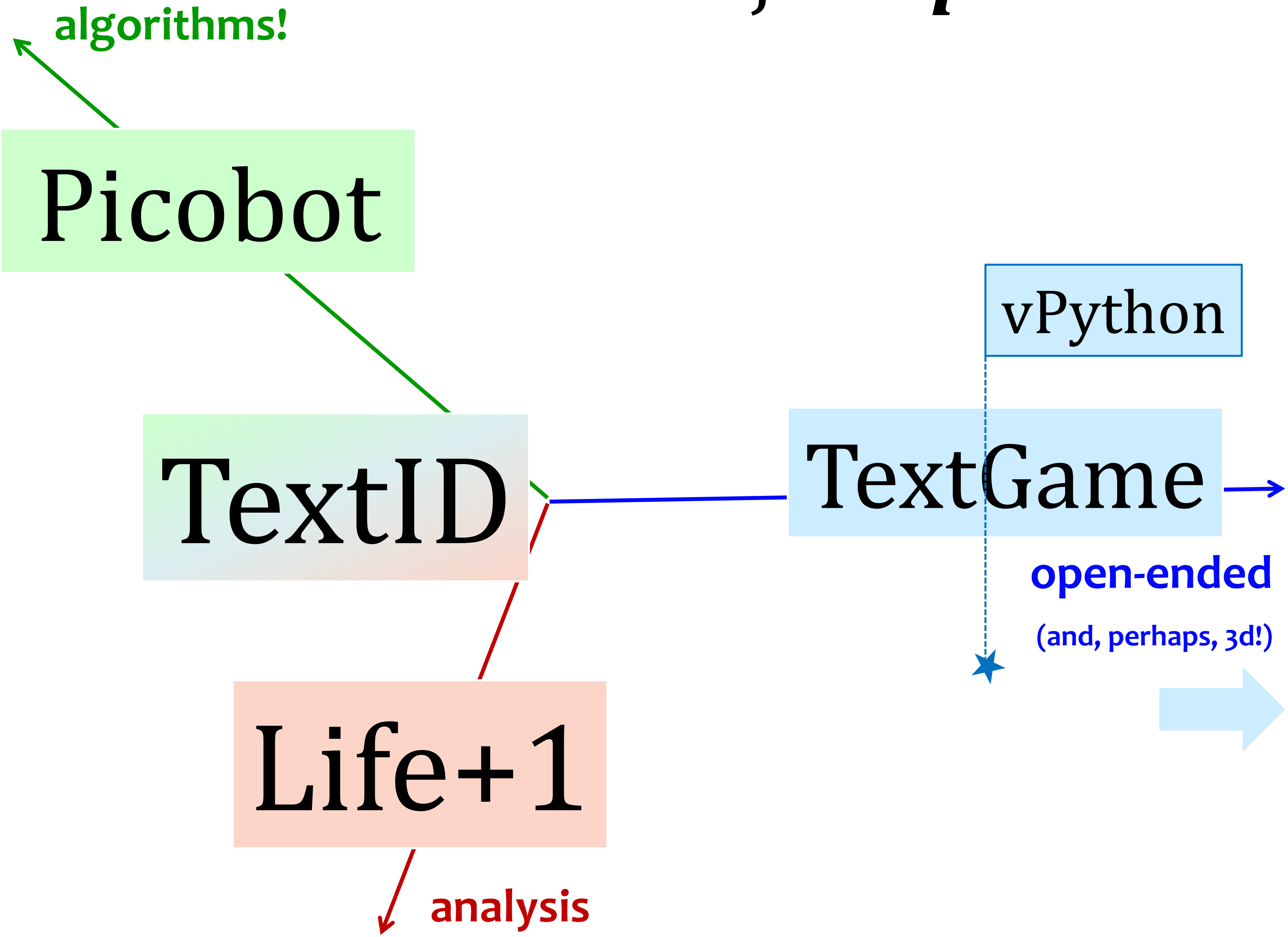
- *vPython*

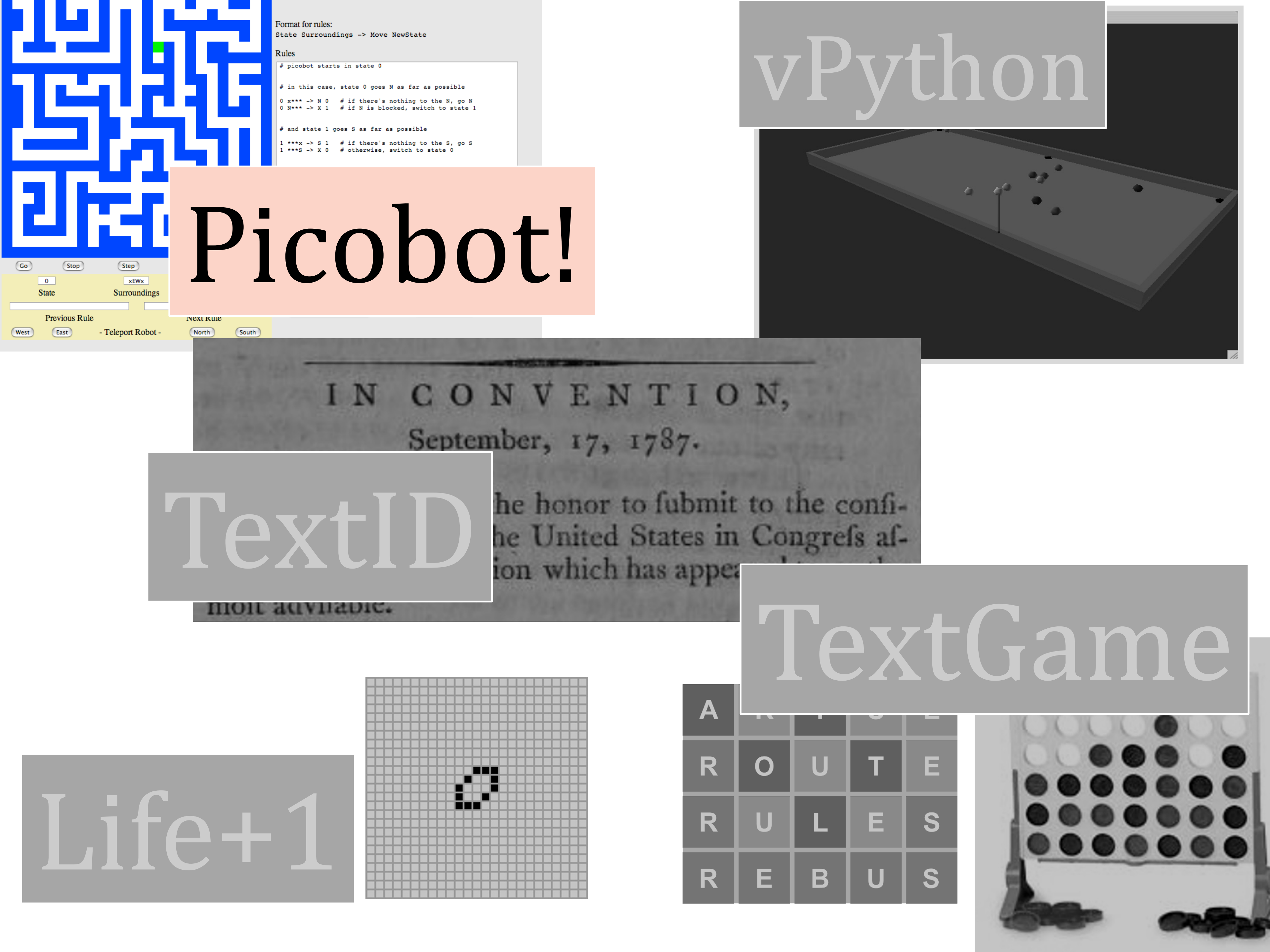
you may want to wait to see lab/homework 11

Labs <u>do</u> meet after lab 11 *(they're <u>extra</u>-optional)*

Picobot!
Format for rules:
State Surroundings -> Move NewState
Rules
picobot starts in state 0
in this case, state 0 goes N as far as possible
0 x*** -> N 0 # if there's nothing to the N, go N
0 N*** -> X 1 # if N is blocked, switch to state 1
and state 1 goes S as far as possible
1 ***x -> S 1 # if there's nothing to the S, go S
1 ***S -> X 0 # otherwise, switch to state 0
Go
Stop
Step
State
Surroundings
Previous Rule
West
East
- Teleport Robot -
North
South
vPython
IN CONVENTION,
September, 17, 1787.
TextID
TextGame
A
R O U T E
R U L E S
R E B U S
Life+1

Project *space*...



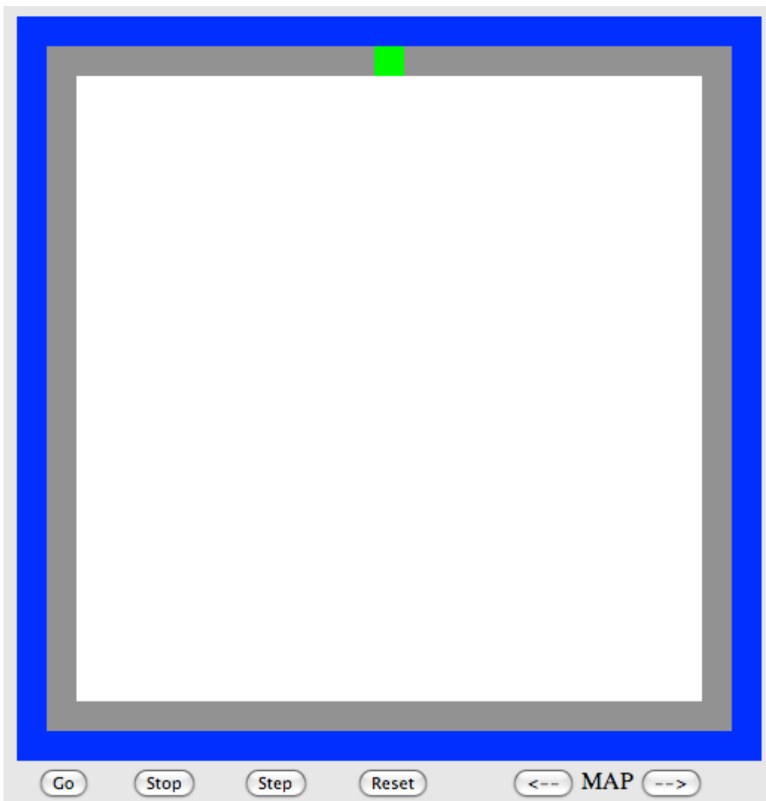

Picobot!
Format for rules:
State Surroundings -> Move NewState
Rules
picobot starts in state 0
in this case, state 0 goes N as far as possible
0 x*** -> N 0 # if there's nothing to the N, go N
0 N*** -> X 1 # if N is blocked, switch to state 1
and state 1 goes S as far as possible
1 ***x -> S 1 # if there's nothing to the S, go S
1 ***S -> X 0 # otherwise, switch to state 0
Go
Stop
Step
State
Surroundings
Previous Rule
West
East
- Teleport Robot -
North
South
vPython
IN CONVENTION,
September, 17, 1787.
TextID
TextGame
Life+1
A
R O U T E
R U L E S
R E B U S

The Picobot project

Big idea

(1) Implement Picobot in Python

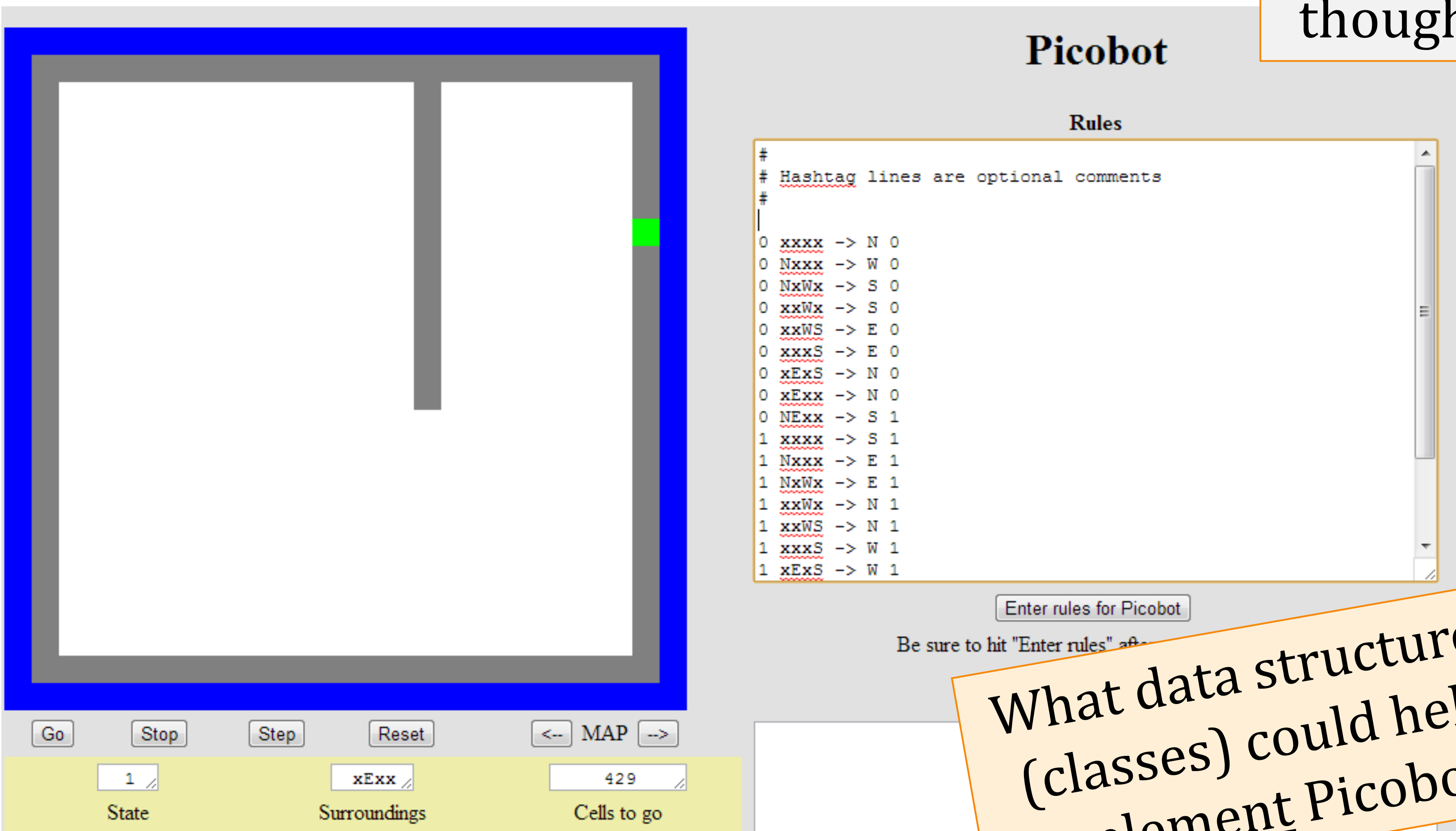
(2) *Train Python to write successful Picobot programs!*



talk about going *full circle...*

Picobot, *behind the curtain...*

design thoughts?



What data structures (classes) could help implement Picobot?

class Program:

Picobot's classes?

class World:

```
+++++++++++++++++++++++++
+oooooPooooooooooooooooo+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+ooooooooooooooooooooooo+
+++++++++++++++++++++++++
```

```
0 xxxx -> N 0
0 Nxxx -> W 0
0 NxWx -> S 0
0 xxWx -> S 0
0 xxWS -> E 0
0 xxxS -> E 0
0 xExS -> N 0
0 xExx -> N 0
0 NExx -> S 1
1 xxxx -> S 1
1 Nxxx -> E 1
1 NxWx -> E 1
1 xxWx -> N 1
1 xxWS -> N 1
1 xxxS -> W 1
1 xExS -> W 1
1 xExx -> S 1
1 NExx -> W 0
```

what **classes** could we adapt for these two?

-- ones we've already used! --

Picobot's classes

```
class Program:
```

How in Python could we most usefully hold all of these ***rules***?

What type should **self.rules** be?

a Python **dictionary**

```
0 xxxx -> N 0
0 Nxxx -> W 0
0 NxWx -> S 0
0 xxWx -> S 0
0 xxWS -> E 0
0 xxxS -> E 0
0 xExS -> N 0
0 xExx -> N 0
0 NExx -> S 1
1 xxxx -> S 1
1 Nxxx -> E 1
1 NxWx -> E 1
1 xxWx -> N 1
1 xxWS -> N 1
1 xxxS -> W 1
1 xExS -> W 1
1 xExx -> S 1
1 NExx -> W 0
```

Picobot's classes

class Program:

```
0 xxxx -> N 0
0 Nxxx -> W 0
0 NxWx -> S 0
0 xxWx -> S 0
0 xxWS -> E 0
0 xxxS -> E 0
0 xExS -> N 0
0 xExx -> N 0
0 NExx -> S 1
1 xxxx -> S 1
1 Nxxx -> E 1
1 NxWx -> E 1
1 xxWx -> N 1
1 xxWS -> N 1
1 xxxS -> W 1
1 xExS -> W 1
1 xExx -> S 1
1 NExx -> W 0
```

How in Python could we most usefully hold all of these *rules*?

What type should `self.rules` be?

a Python **dictionary**

both *tuples*

key **value**

```
self.rules[ (1,"NExx") ] = ("W",0)
```

Picobot's classes

What type in Python could most usefully hold the ***environment***?

```
class World:
```

```
+++++++++++++++++++++++++
+oooooPooooooooooooooooo+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o             o       o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+o                     o+
+ooooooooooooooooooooooo+
+++++++++++++++++++++++++
```

What class we've already written will be similar to Picobot's **`World`**?

What will **`self.room`** be?

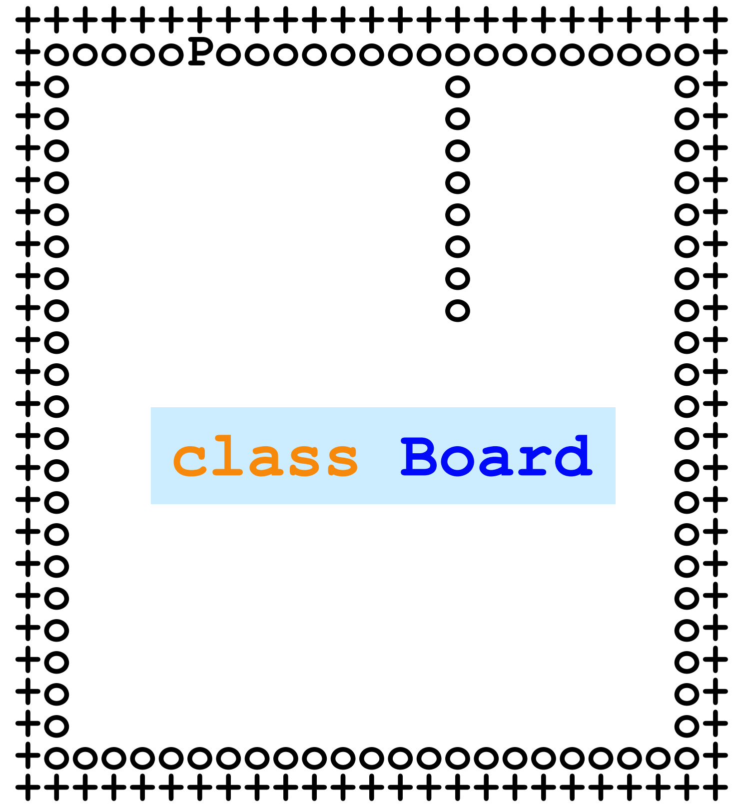
a Connect-Four **Board**

Wall:	`'+'`
Visited:	`'o'`
Picobot:	`'P'`
Empty:	`' '`

Picobot's classes

What type in Python could most usefully hold the ***environment***?

class World:



What class that you've already written will be most similar to Picobot's **World**?

What will **self.room** be?

Wall: '+'

Visited: 'o'

Picobot: 'P'

Empty: ' '

The same as the Connect-Four board's **self.data**!

a list-of-lists-of-one-character-strings….

Picobot's project

First, build an
ASCII simulation

Picobot started here...

```
+++++++++++
+o+++o+o+++
+oooooo  ++
++++o++   +
+oooo++++++
++++o     +
+oooo+++  +
++++o++++++
+Pooo     +
+++++++++++
```

and is now here...

Current State: 1

Current Rule: 1 N*W* -> X 2

then, ***evolve*** it...

http://rednuht.org/genetic_cars_2/ or http://boxcar2d.com/
Box2d: https://www.youtube.com/watch?v=uxourrlPlf8

Your terminal graphics *may* be more monochromatic...

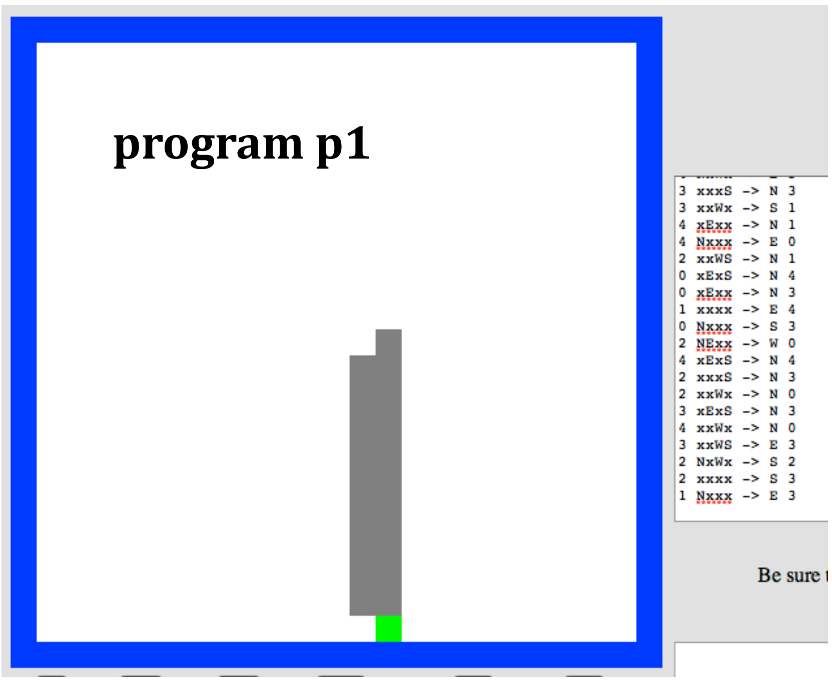
Genetic algorithms ~ program ***evolution***

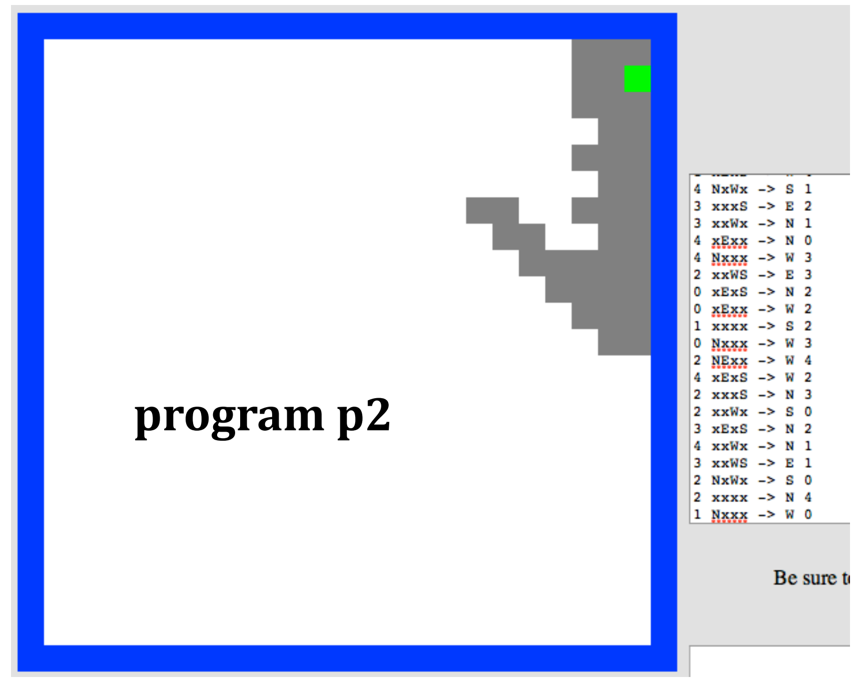
An example of ***genetic algorithms***, which are used for optimizing *hard-to-describe* functions with *easily-splittable* solutions.

Suppose we start with 200 ***random*** Picobot programs…

Genetic algorithms ~ program *evolution*

An example of ***genetic algorithms***, which are used for optimizing *hard-to-describe* functions with *easily-splittable* solutions.





Suppose we start with 200 ***random*** Picobot programs...

(1) How might we measure each program's **"fitness"**?

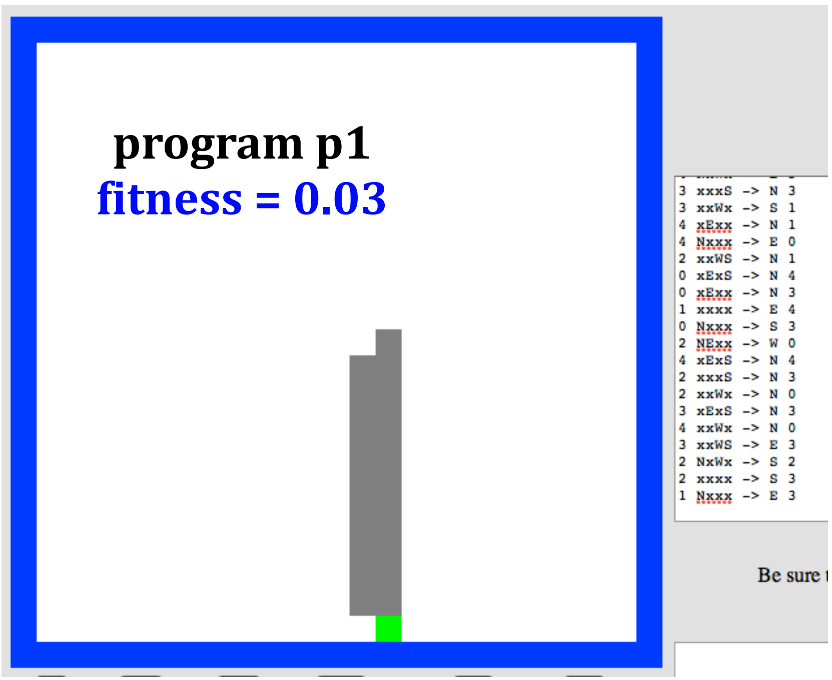
(2) How might we **"mutate"** a program?

(3) How might we **"mate"** two programs, to create a new, "child" program?

(*) What else should we worry about?!!

Program *evolution*

An example of ***genetic algorithms***, which are used for optimizing hard-to-describe functions with easily-splittable solutions.

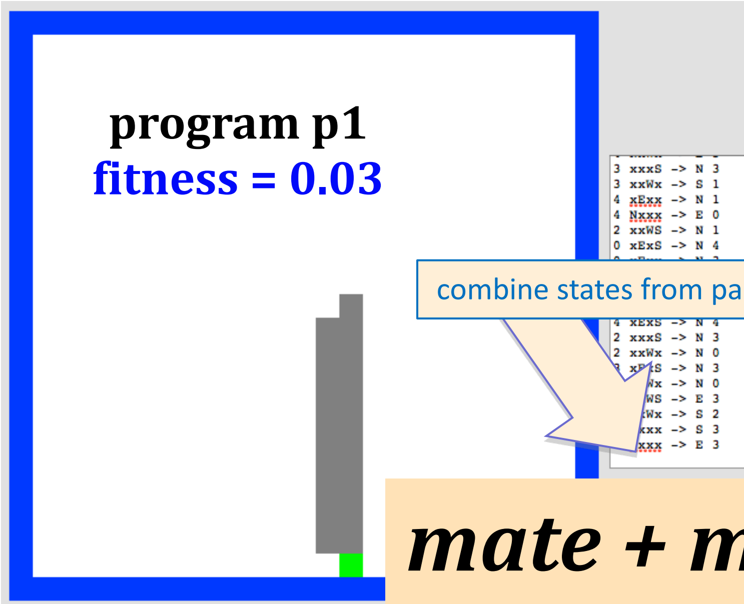


Coverage–as–**fitness**!

... using several starting points

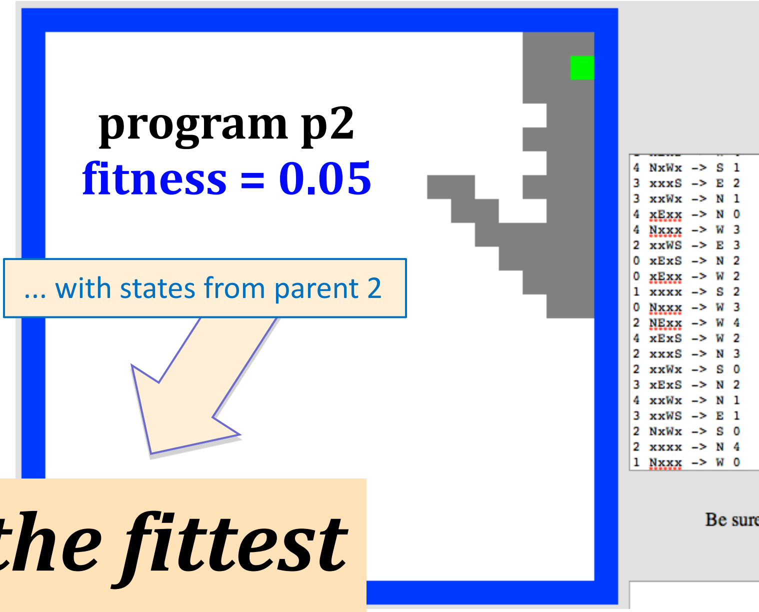
Measure?
How??

program p1
fitness = 0.03



combine states from parent 1 ...

program p2
fitness = 0.05



... with states from parent 2

mate + mutate the fittest
10-20% of programs

... plus, change some rules randomly!

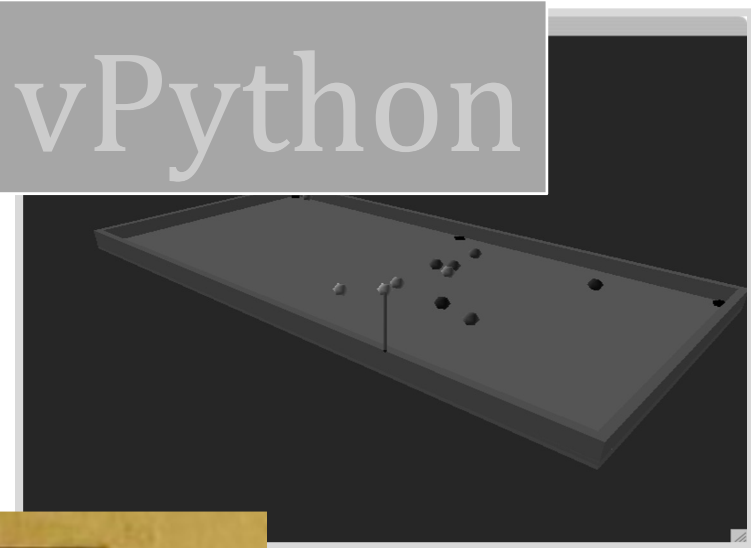
to create a *new generation*
of ~200 programs...

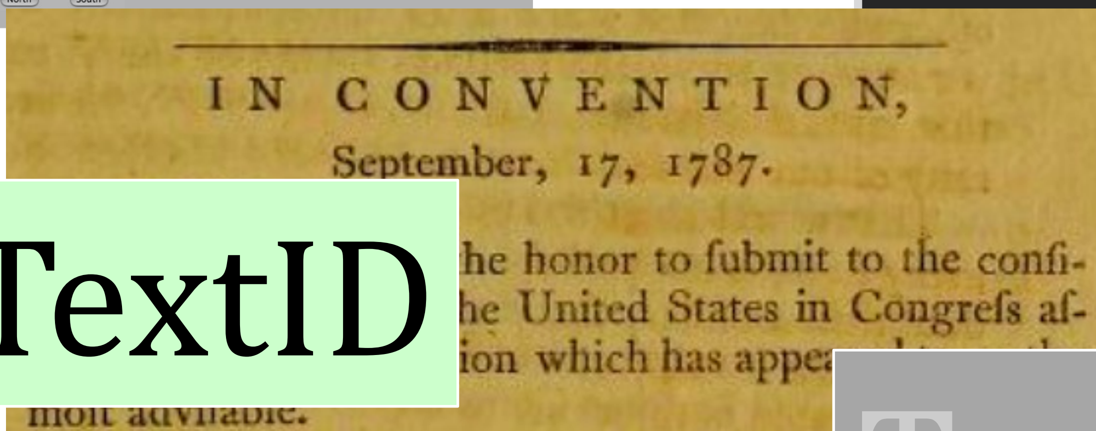
program c1
fitness = 0.19

What the goal?

Picobot!

vPython

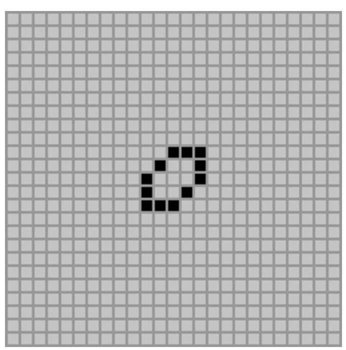


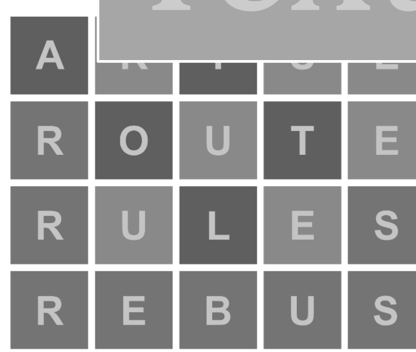


TextID

TextGame

Life+1





Text ID algorithms...

PNAS
Proceedings of the National Academy of Sciences of the United States of America

RESEARCH ARTICLE

Check for updates

Narrative structure of *A Song of Ice and Fire* creates a fictional world with realistic measures of social complexity

Thomas Gessey-Jones, Colm Connaughton, Robin Dunbar, Ralph Kenna, Pádraig MacCarron, Cathal O'Conchobhair, and Joseph Yose

PNAS first published November 2, 2020; https://doi.org/10.1073/pnas.2006465117

Edited by Kenneth W. Wachter, University of California, Berkeley, CA, and approved September 15, 2020 (received for review April 6, 2020)

Significance

We use mathematical and statistical methods to probe how a sprawling, dynamic, complex narrative of massive scale achieved broad accessibility and acclaim without surrendering to the need for reductionist simplifications. Subtle narrational tricks such as how natural social networks are mirrored and how significant events are scheduled are unveiled. The narrative network matches evolved cognitive abilities to enable complex messages be conveyed in accessible ways while story time and discourse time are carefully distinguished in ways matching theories of narratology. This marriage of science and humanities opens avenues to comparative literary studies. It provides quantitative support, for example, for the widespread view that deaths appear to be randomly distributed throughout the narrative even though, in fact, they are not.

Authorship *analysis*

Did Hamilton compose the "Constitution's cover letter" ? – Part 2

April 11, 2020 admin Leave a comment

The Hamilton Authorship Thesis

Recognizing the lion by his claw

The Constitution was introduced to the world with a little known cover letter signed by George Washington, as the President of the Constitutional Convention. The cover letter "was read once throughout, and afterwards agreed to by paragraphs," making it a unique official communication of the Constitutional

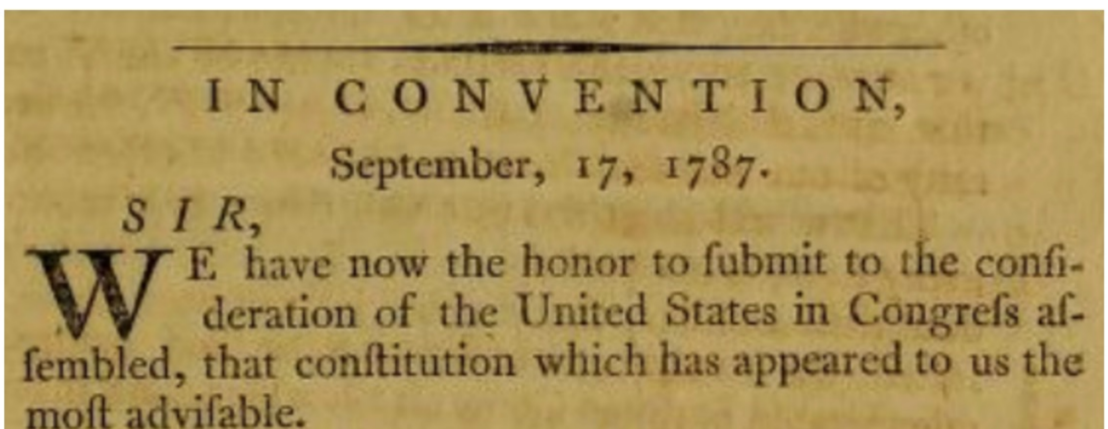
April 11, 2020 admin Leave a comment

The Hamilton Authorship Thesis

Recognizing the lion by his claw

The Constitution was introduced to the world with a little known cover letter signed by George Washington, as the President of the Constitutional Convention. The cover letter "was read once throughout, and afterwards agreed to by paragraphs," making it a unique official communication of the Constitutional Convention. But who wrote it?

For far too long, historians have assumed that the nearly forgotten cover letter was written by Gouverneur Morris, the so-called "penman of the Constitution." Others have attributed the letter to Washington who signed it. This post will attempt to demonstrate that overwhelming evidence supports the conclusion that Alexander Hamilton was the author of the Constitution's cover letter.



IN CONVENTION,

September, 17, 1787.

SIR,

WE have now the honor to fubmit to the confideration of the United States in Congrefs affembled, that conftitution which has appeared to us the moft advifable.

Copied above is a screen shot of the first paragraph of the cover letter, which was printed below the Constitution in the Acts of the First Congress

https://www.youtube.com/watch?v=PTziNb-otPE&ab_channel=VG @ 11:30 ish

April 11, 2020 admin Leave a comment

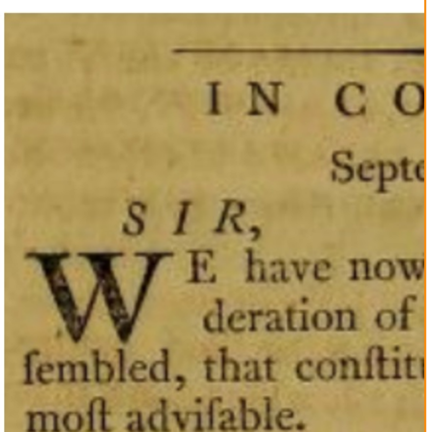
The Hamilton Authorship Thesis

Recognizing the lion by his claw

"Stylometry"

The Constitution was introduced
signed by George Washington, a
The cover letter "was read once
paragraphs," making it a unique
Convention. But who wrote it?

For far too long, historians have
written by Gouverneur Morris, t
have attributed the letter to Was
demonstrate that overwhelming
Hamilton was the author of the C


IN CO
Septe
SIR,
WE have now
deration of
sembled, that constitu
most advisable.

Copied above is a screen shot of the
the Constitut

Hamilton's June 18 notes	Cover Letter
Importance of the occasion	present occasion
the public mind	deeply impressed on our minds
complete sovereignty	independent sovereignty to each
Its practicability to be examined; if not impracticable	it is obviously impracticable
local ci...umstances	situation a... circumstance
Entrus... he great interests ... he nation	the greate... nterest of every true American
habit... ense of obligation	habits
Particular & general *interests*	particular interests
necessity	necessity of a different organization.
necessary consequence	the consequences
powers too great must be given to a single branch; Entrusts the great interests of the nation to hands incapable of managing them	the impropriety of delegating such extensive trust to one body of men is evident — Hence results the necessity of a different organization.
hopes and fears	we hope and believe
will sacrifice	the magnitude of the sacrifice
true interest ...	every true American
	secure all righ...
...government; the means will not be equal to the object	the ...ect to be...
The formi... new government to ...rvade th... whole with decisive powers ... short w... complete sovereignty; power (13x); peace (3x); war (5); treaty (2x); money (3x); commerce (4x)	power of making war, peace, and treaties, that of levying money and regulating commerce
Eac... incipl... ught to exist in full force, or i... ill not ... wer its end	should be fully ... nd effectually vested

textual features

being compared

April 11, 2020 admin Leave a comment

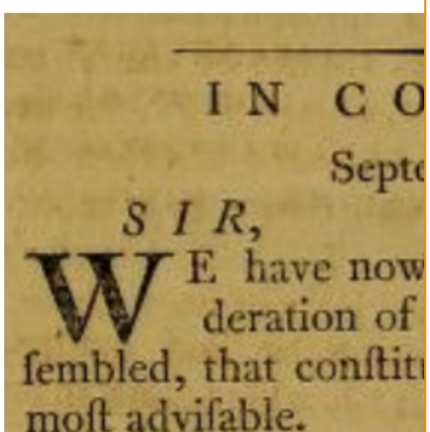
The Hamilton Authorship Thesis

Recognizing the lion by his claw

"Stylometry"

The Constitution was introduced
signed by George Washington, a
The cover letter "was read once
paragraphs," making it a unique
Convention. But who wrote it?

For far too long, historians have
written by Gouverneur Morris, t
have attributed the letter to Was
demonstrate that overwhelming
Hamilton was the author of the



Copied above is a screen shot of the
the Constitut

Hamilton's June 18 notes	Cover Letter
Importance of the occasion	present occasion
the public mind	deeply impressed on our minds
complete sovereignty	independent sovereignty to each
Its practicability to be examined; if not impracticab	mpracticable
local ci n	cumstance
Entrus he	est of every true American
habit en	
Particular &	
necessity	
necessary c	
powers too single branc of the natio managing th	sive of men is evident e necessity of a different
hopes and fears	we hope and believe
will sacrifice	the magnitude of the sacrifice
true interest	every true American
	secure all righ
ernment; the means will not be equal to the object	the ect to be
he formin new government to rvade th whole with decisive powers short w complete sovereignty; power (13x); peace (3x); war (5); treaty (2x); money (3x); commerce (4x)	power of making war, peace, and treaties, that of levying money and regulating commerce
Eac incipl ught to exist in full force, or ill not wer its end	should be fully nd effectually vested

word-frequencies
stem-frequencies
word-lengths
sentence-lengths
[punctuation use]

5 feature models

textual features

being compared

Algorithmic Intuition...

Dictionary-comparing

Here are two word-count models from *known authors*, Alexander Hamilton + Lin-Manuel Miranda. An *unknown author* created the middle model.

All of the models have been made into Python dictionaries:

LMM

```
{ "shot": 50,
  "Burr": 8,
  "story": 42 }
```

model for Lin-Manuel Miranda

~?~

```
{ "shot": 3,
  "story": 1,
  "money": 2,
  "spam": 4 }
```

word-count model for an unknown author

AH

```
{ "shot": 25,
  "Burr": 275,
  "money": 700 }
```

word-count model for A. Hamilton

Which is the *better match* for the *unknown-author* model?

Algorithm: *Bayesian classification*

Bayesian spam filtering

From Wikipedia, the free encyclopedia

Bayesian spam filtering (/ˈbeɪziən/ *BAY-zee-ən*; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naive Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Model *scale*

Suppose we have two text ***models***:

LMM: { "shot": 50,
"Burr": 8,
"story": 42 }

AH : { "shot": 25,
"Burr": 275,
"money": 700 }

aargh! the totals are different...

Unknown-author text:

{ "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

These must have been some *really* avant-garde texts!

Step 1: adjust our word counts to be non-zero

LMM: { "shot": 50,
"Burr": 8,
"story": 42 }

AH : { "shot": 25,
"Burr": 275,
"money": 700 }

Add 1 to each word in the shared vocabulary for each model

LMM: { "shot": 51,
"Burr": 9,
"story": 43,
"money": 1,
"spam": 1 }

AH : { "shot": 26,
"Burr": 276,
"money": 701,
"story": 1,
"spam": 1 }

Unknown-author text: { "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

These must have been some *really* avant-garde texts!

Step 2: normalize our counts to sum to 1

LMM: { "shot": 51,
"Burr": 9,
"story": 43,
"money": 1,
"spam": 1 }

AH : { "shot": 26,
"Burr": 276,
"money": 701,
"story": 1,
"spam": 1 }

Divide by the total # of words in each

LMM: { "shot": 0.4857,
"Burr": 0.0857,
"money": 0.0095,
"story": 0.4095,
"spam": 0.0095 }

AH : { "shot": 0.0259,
"Burr": 0.2746,
"money": 0.6975,
"story": 0.0010,
"spam": 0.0010 }

Unknown-author text: { "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

These must have been some *really* avant-garde texts!

Step 3: estimate probability for each known author

LMM: { "shot": 0.4857,
"Burr": 0.0857,
"money": 0.0095,
"story": 0.4095,
"spam": 0.0095 }

AH : { "shot": 0.0259,
"Burr": 0.2746,
"money": 0.6975,
"story": 0.0010,
"spam": 0.0010 }

Unknown-author text:

{ "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

pretend the words are all independent

What's the ***likelihood*** of each author making this text?

? · ? · ? · ? · ? · ? · ? · ? · ? · ? = ???

shot shot shot story money money spam spam spam spam

Step 3: estimate probability for each known author

LMM: { "shot": 0.4857,
"Burr": 0.0857,
"money": 0.0095,
"story": 0.4095,
"spam": 0.0095 }

Unknown-author text:

{ "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

What's the ***likelihood*** of each author making this text?

$$\underbrace{.49 \cdot .49 \cdot .49}_{\text{shot, shot, shot}} \cdot \underbrace{.41}_{\text{story}} \cdot \underbrace{.01 \cdot .01}_{\text{money, money}} \cdot \underbrace{.01 \cdot .01 \cdot .01 \cdot .01}_{\text{spam, spam, spam, spam}} = \sim 4.82 \times 10^{-12}$$

Step 3: estimate probability for each known author

LMM: { "shot": 0.4857,
"Burr": 0.0857,
"money": 0.0095,
"story": 0.4095,
"spam": 0.0095 }

Unknown-author text:

{ "shot": 3,
"story": 1,
"money": 2,
"spam": 4 }

What's the ***likelihood*** of each author making this text?

.49 • .49 • .49 • .41 • .01 • .01 • .01 • .01 • .01 • .01 = ~4.82x10^{-12}

shot shot shot story money money spam spam spam spam

3*log(.49) + log(.41) + 2*log(.01) + 4*log(.01) = -37.59

shot shot shot story money money spam spam spam spam

3 1 2 4

OK!

take the $\log_2$ of everything!

Model *matching*

from two ***normalized models***:

LMM: { "shot": 0.4857,
"Burr": 0.0857,
"money": 0.0095,
"story": 0.4095,
"spam": 0.0095 }

AH : { "shot": 0.0259,
"Burr": 0.2746,
"money": 0.6975,
"story": 0.0010,
"spam": 0.0010 }

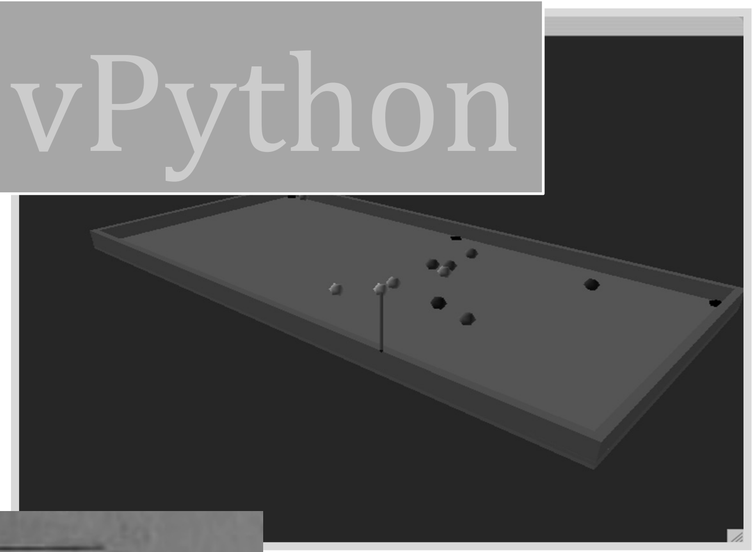
Unknown text:
{ "shot": 3, "money": 2,
"story": 1, "spam": 4 }

-37.59

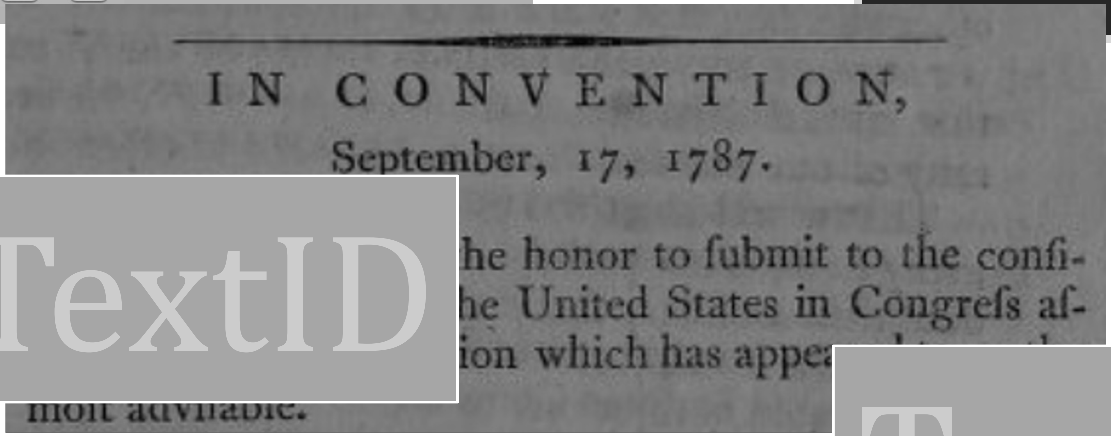
the (much) better match…

-66.68

Picobot!



vPython



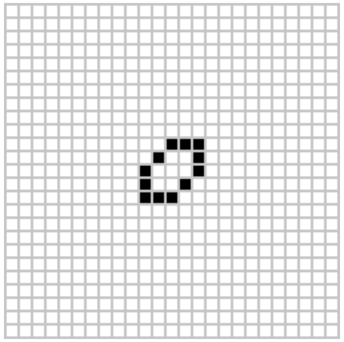
IN CONVENTION,

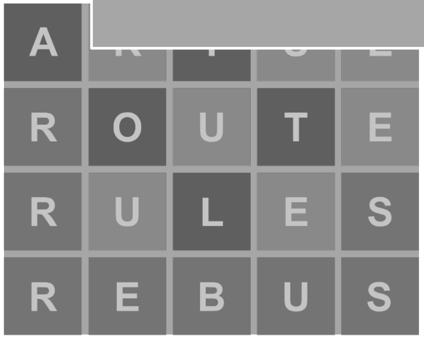
September, 17, 1787.

...he honor to fubmit to the confi-
...he United States in Congrefs af-
...ion which has appea...
moft advifable.

TextID

TextGame





Life+1

Life+1

Building from *Week 9*'s Lab...

[1] Should create a **`Life`** class: similar to C4's **`Board`**

enable methods for analysis + data members for data-storage

and, you need to visualize your code with the ***Pyglet*** 2d library

[2] Should allow *any* "Life-like" rules

Python dictionaries, e.g., **{ 'B'**: [3], **'S'**: [2,3] **}** # B3/S23 Life!

Notation for rules [edit]

In the notation used by the Golly open-source cellular automaton package and in the RLE format for storing cellular automaton patterns, a rule is written in the form By/Sx where x and y are the same as in the MCell notation. Thus, in this notation, Conway's Game of Life is denoted B3/S23. The "B" in this format stands for "birth" and the "S" stands for "survival".[4]

Life+1

Building from Week 9's Lab...

[1] Should create a **`Life`** class: similar to C4's **`Board`**

enable methods for analysis + data members for data-storage

and, you need to visualize your code with the ***Pyglet*** 2d library

[2] Should allow *any* "Life-like" rules

Python dictionaries, e.g., `{ 'B': [3], 'S': [2,3] }` # B3/S23 Life!

*Cells are **B**orn, if there are 3 living neighbors*

*Cells **S**urvive, if there are 2 or 3 living neighbors*

A selection of Life-like rules [edit]

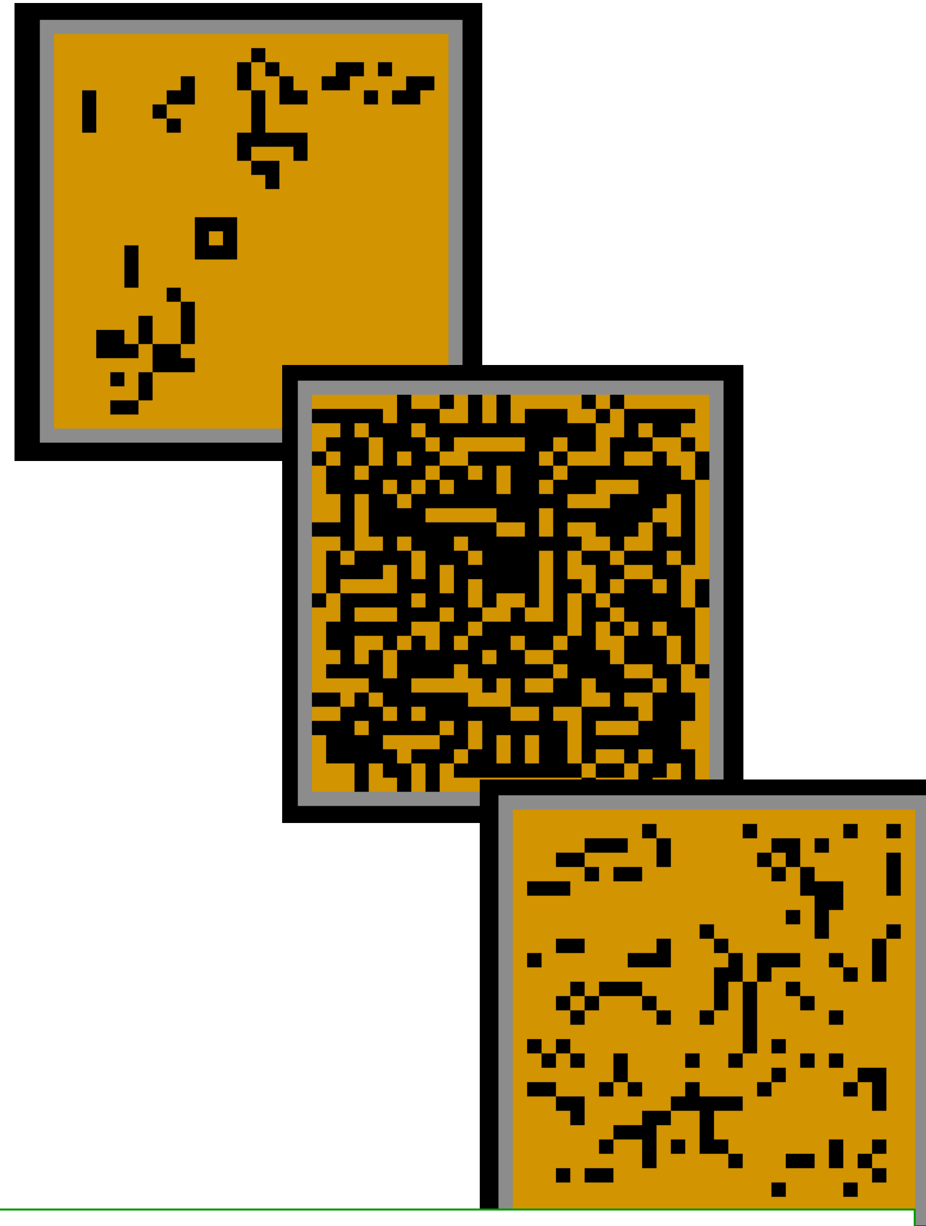
There are 2^{18} = 262,144 possible Life-like rules, only a small fraction of which have been studied in any detail.

Rules ~ Behavior?

```
{'B': [2],
 'S': []}
```

```
{'B': [3],
 'S': [2,3]}
```

```
{'B': [3],
 'S': list(range(9))}
```



A selection of Life-like rules [edit]

There are 2^{18} = 262,144 possible Life-like rules, only a small fraction of which have been studied in any detail.

Life+1

Building from Week 9's Lab...

[1] Should create a **`Life`** class: similar to C4's **`Board`**

enable methods for analysis + data members for data-storage
and, you need to visualize your code with the ***Pyglet*** 2d library

[2] Should allow *any* "Life-like" rules

Python dictionaries, e.g., { **'B'**: [3], **'S'**: [2,3] } # B3/S23 Life!

[3] Should track generations' *evolution*

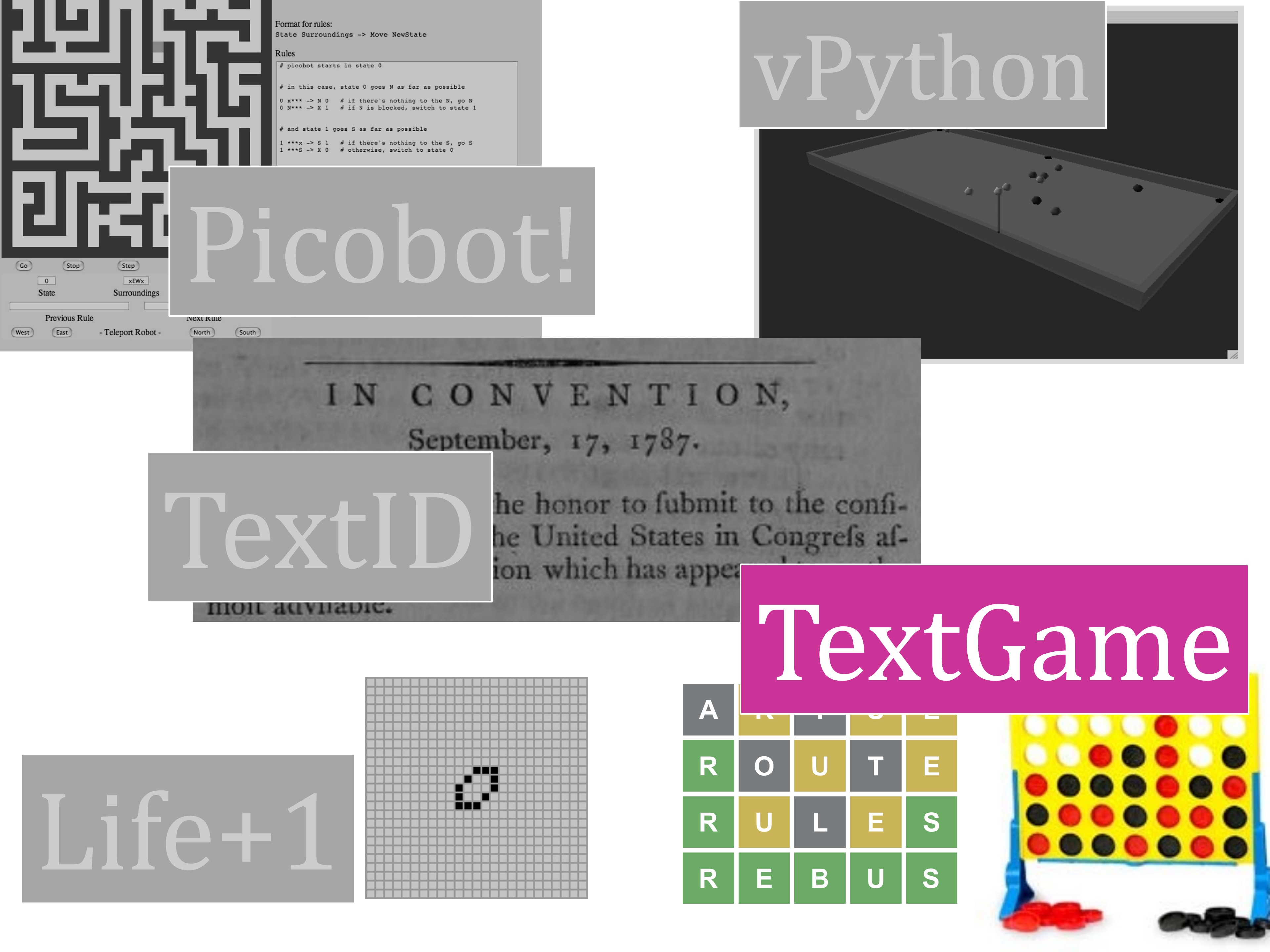
Grow? Fade? What % of the world is alive?!

[4] Should create + explore your own variation(s)

Can follow more Birth/Survived rulesets, add more states,
or something completely different...

Demo!

hw9pr1_...



Format for rules:
State Surroundings -> Move NewState
Rules
picobot starts in state 0
in this case, state 0 goes N as far as possible
0 x*** -> N 0   # if there's nothing to the N, go N
0 N*** -> X 1   # if N is blocked, switch to state 1
and state 1 goes S as far as possible
1 ***x -> S 1   # if there's nothing to the S, go S
1 ***S -> X 0   # otherwise, switch to state 0
Go
Stop
Step
0
State
xEWx
Surroundings
Previous Rule
Next Rule
West
East
- Teleport Robot -
North
South
Picobot!
vPython
IN CONVENTION,
September, 17, 1787.
TextID
he honor to fubmit to the confi-
he United States in Congrefs af-
ion which has appea
moft advifable.
TextGame
Life+1
A
R O U T E
R U L E S
R E B U S

TextGame

Varying based on hw11pr2...

[1] Should have a "Board": *some visible game-state*

doesn't really need to be a board: Jotto, Wordle, Nim, Hangman are all ok...

[2] Should have multiple turns (per game)

Jotto, Nim, Hangman all fit this, but RPS does not (that's the starter code)

[3] Should track the human/machine rivalry

A starting point for this is provided that you can modify...
...for some games (e.g. Wordle) you may have to figure out how to add an AI "player"...

[4] *Should have an AI of some sort*

The "I" does *not* have to be lookahead

conversational AI
random AI
Misère AI
or ...

Keys:

You should be able to play vs. the machine

The machine should be able to play vs. the machine!

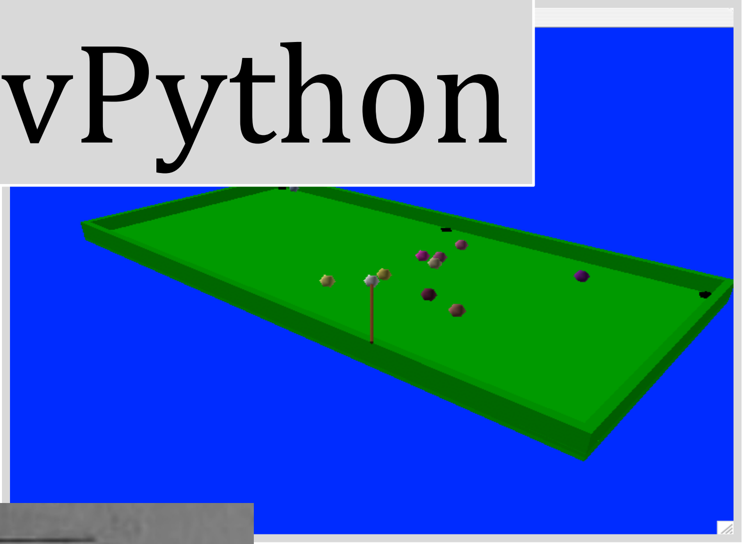
Examples
beyond C4

```
Choose a letter! z
you guessed wrong!  You have

-------------------

  ___________
      |    |
      _    |
           |
           |
           |
           |
           |
           |
___________

Choose a letter! x
you guessed wrong!  You have

-------------------

  ___________
      |    |
      _    |
    (*_*)  |
           |
           |
           |
           |
           |
___________

Choose a letter!
```

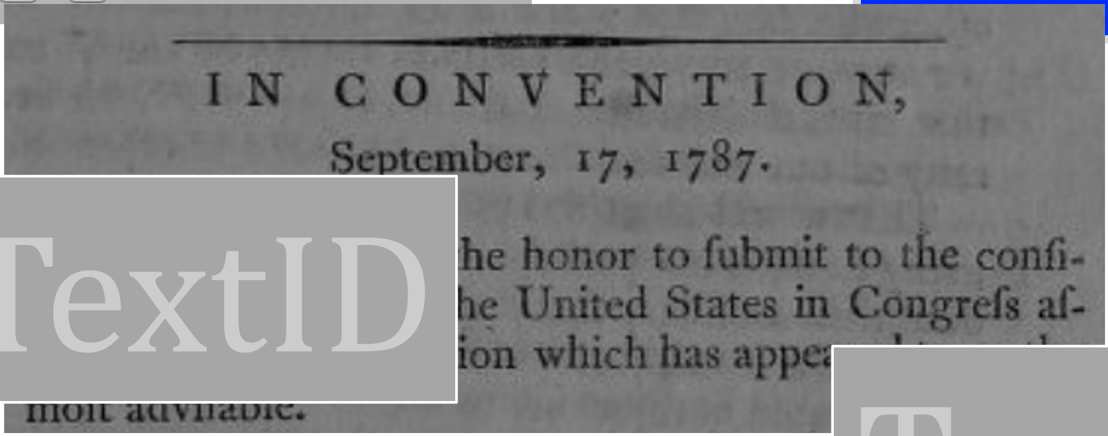
```
In [115]: run mancala.py


+ Current tally +
    My marbles: 0
  Your marbles: 0

Menu:
  (1) Start Mancala
  (2) Load our game
  (3) Reset the board
  (4) Save our game
  (5) Rules of Mancala
  (8) Quit

Your choice: 1

Welcome to Mancala!
You have 4 marbles, and six pits,
numbered from left to right (1 - 6):

------------------------------------------


Comp:    [ 4 ][ 4 ][ 4 ][ 4 ][ 4 ][ 4 ]

User:    [ 4 ][ 4 ][ 4 ][ 4 ][ 4 ][ 4 ]

+ Current tally +
Comp:    [ 0 ]
User:    [ 0 ]
------------------------------------------
```

Picobot!

vPython

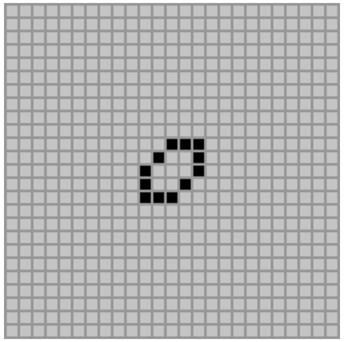


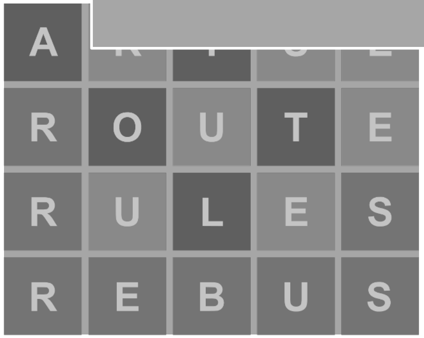
TextID


IN CONVENTION,
September, 17, 1787.
...he honor to fubmit to the confi-
...he United States in Congrefs af-
...ion which has appea...
moft advifable.

TextGame

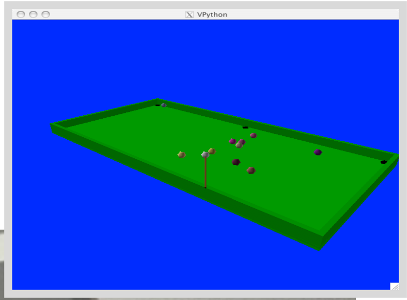
Life+1



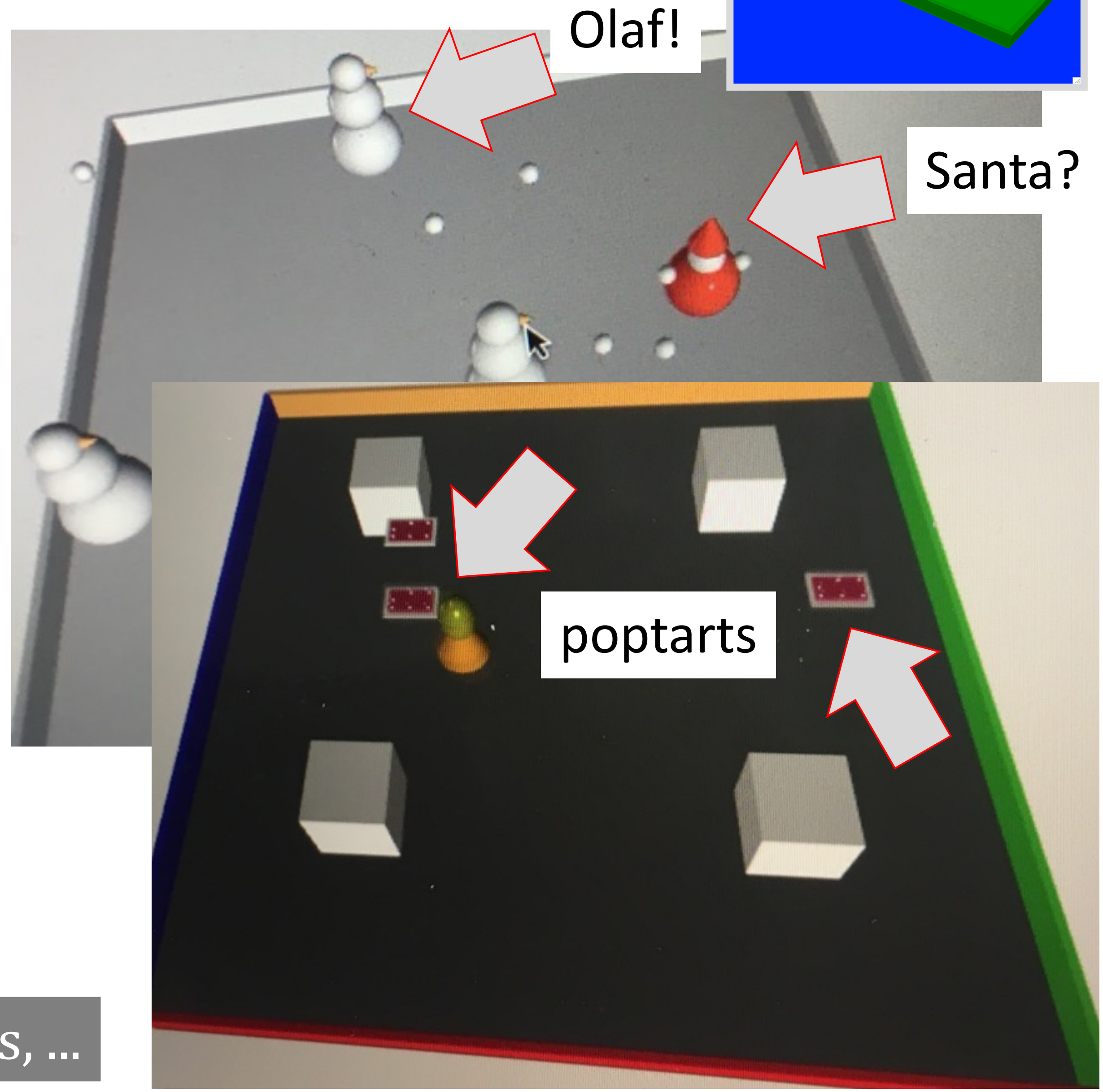


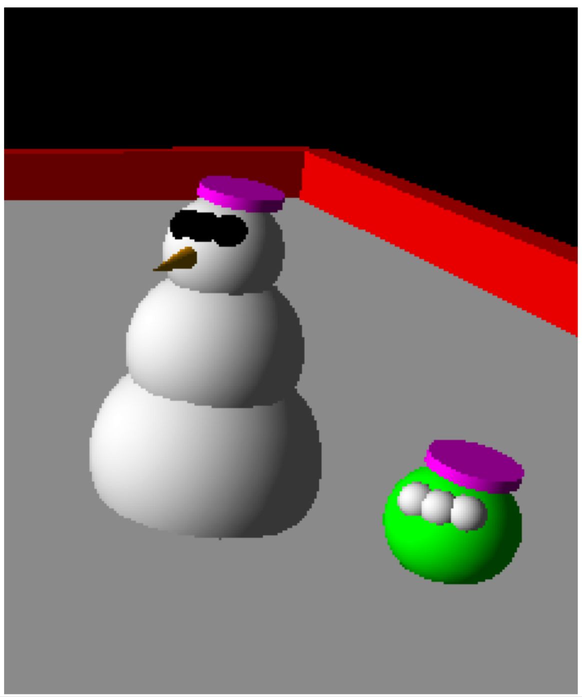
vPython

From Lab 11



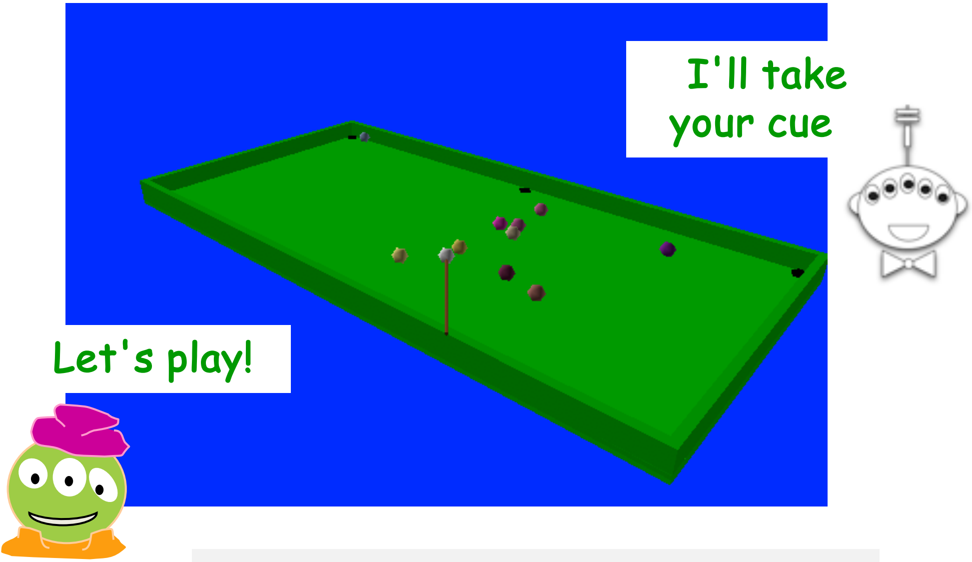
Past examples...





add features, characters, ...

More vPython?



A few constraints…

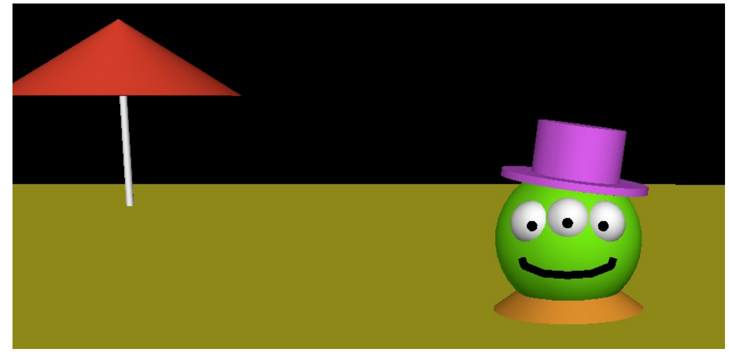
need ≥4 physically interacting objects

allow the user to direct 1+ objects, either by keyboard or mouse or both

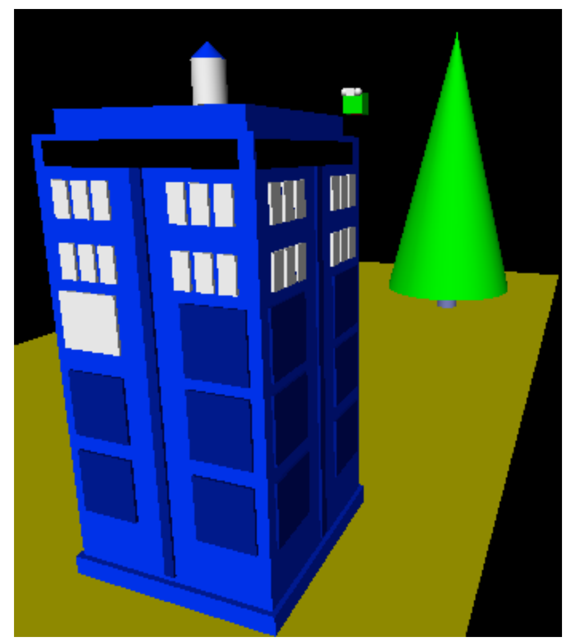
needs a game goal + be winnable!

must detect **some** "linear" and some "spherical" collisions and implement their results on the motion of the objects

Physics engine…



… it's not really very constrained at all!



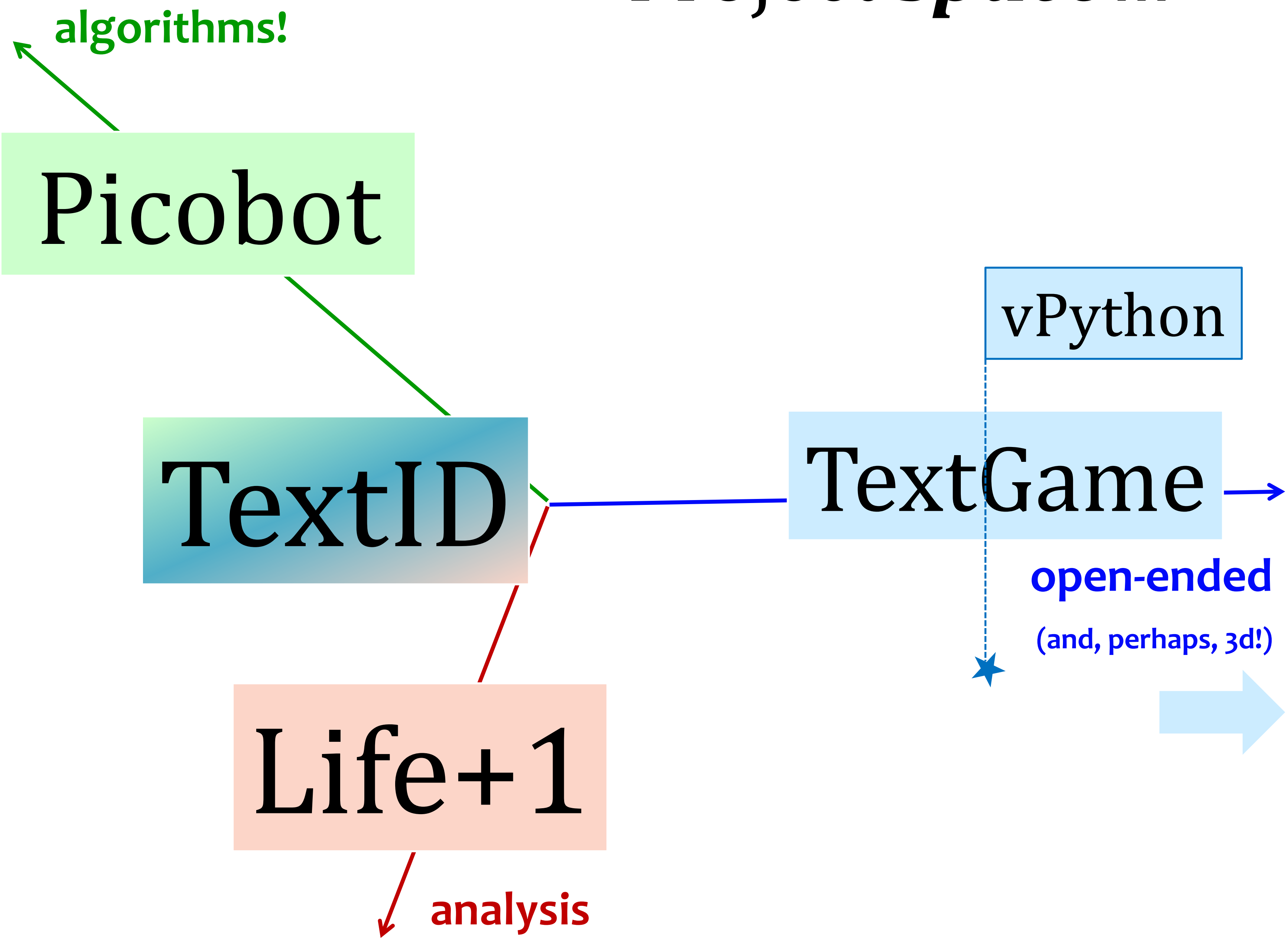
More vPython details...

VPython was designed to make 3d physics simulations simpler to program – as a result, the **library itself is physics-free**!

"surreal physics" is welcome...

- ***Linear collisions*** should be somewhere ("walls")
- ***Spherical collisions*** should be somewhere ("points")
- You need "pockets" – *or some other game objective*
- You need **user control** of at least one object (mouse/kbd)

Project *space*...



Surreal? Sounds like *Spec.Rel.* to me!

Tips across projects:

- **Think about your plan!** This is the ongoing "design" part of the project.
- ***Test your code with every change you make.*** Making a large number of changes at once is where things could be going wrong.
- **Use good documentation practices:**
  - A docstring for *every* function and method that you write.
  - Comments to explain tricky pieces of code.
  - Descriptive variable names for nontrivial values (avoiding "magic" values)
- **Make the basic version work first.**
  - Build your game out of entirely spheres/ASCII characters
  - Start with a less-than-intelligent AI