This week's **classes**

Three-eyed? This week, we're *3d*’ed!

**VPython**
3D Programming for Ordinary Mortals

---

**Connect 4, Part 2**

<p>| | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>O</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
<td>X</td>
</tr>
<tr>
<td>O</td>
<td>O</td>
<td>O</td>
<td>X</td>
<td>O</td>
<td>X</td>
<td>X</td>
</tr>
</tbody>
</table>

---

**hw11pr2.py**

```python
b.colsToWin( 'O' )
b.colsToWin( 'X' )
```

```python
b.aiMove( 'O' )
b.aiMove( 'X' )
```

```python
hostGame( self )
```

---

**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 - *in the same place* - and need to share the work equally...

Pairs/trios are welcome (larger should split)

**Teaming is extra-encouraged** on the final project!

---

**4/22**

- "Start" ~ part of hw11

**When?!**

**Mon. 4/29**

- "Milestone" ~ part of hw12
- project-specific tasks to help w/progress...

**Fri. 5/3**

- Final project & short reflection on how to run it and how it went.
- due at 8pm
- Euros ok; grutoring tapers.

---

**Eye, eye!**

Notice that the value of (dimension + eyes) is conserved ~ at 5!

Notice that it’s black’s move or red’s, they’re eye-ing the same column!

---

**Nice milestone!**

It’s a kilometer stone, actually!
The project page...

Model matching

Suppose we have two trained models:

WS: { "love": 50, "spell": 8, "thou": 42 }

JKR: { "love": 25, "spell": 275, "potter": 700 }

how do we handle different-sized texts?

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

These must have been some really avant-garde texts!

Suppose we have two normalized models:

WS: { "love": 0.50, "spell": 0.08, "thou": 0.42 }

JKR: { "love": 0.025, "spell": 0.275, "potter": 0.700 }

the WS-based probability of each word in Unknown text

Unknown text: { "love": 3, "thou": 1, "potter": 2, "spam": 4 }

I've got some semi-idea on this one!
Naïve Bayes classification

Bayesian spam filtering

Bayesian spam filtering (IPA: /ˈbeɪzjən/ bay-ZEE-ən; after Rev. Thomas Bayes) is a statistical technique of e-mail filtering. In its basic form, it makes use of a naïve Bayes classifier on bag of words features to identify spam e-mail, an approach commonly used in text classification.

Constructing a classifier from the probability model

The discussion so far has derived the independent feature model, that is, the discussion so far has derived the classifier from the probability model. The naïve Bayes classifier combines this model with a decision rule for selecting the hypothesis that is most probable; this is known as the maximum a posteriori (MAP) decision rule. The corresponding classifier, a Bayes classifier, is the function
classify($f_1, \ldots, f_n$) = argmax$_c p(C = c) \prod_{i=1}^n p(F_i = f_i | C = c)$.

Don't take these formulas too seriously…

Big idea

(1) Implement Picobot in Python
(2) Train Python to write successful Picobot programs!

Picobot's classes

class Program:

What type should self.rules be?

0 xxxx -> N 0
0 NxWx -> W 0
0 NxWx -> S 0
0 xxWx -> S 0
0 xxWx -> E 0
0 xxxx -> E 0
0 xExS -> N 0
0 xExS -> N 0
0 NExx -> S 1
1 xxxx -> S 1
1 xxxx -> E 1
1 NxWx -> E 1
1 xxWx -> N 1
1 xxWx -> N 1
1 xxxx -> S 1
1 xxxx -> W 1
1 xExS -> W 1
1 xExS -> S 1
1 NExx -> W 0

Picobot's classes

class World:

What type in Python could most usefully hold the environment?

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

What will self.room be?

Wall: +
Visited: ◯
Picobot: P
**Picobot's project**

*Current State*:

- 1

*Current Rule*:

- \( N^* W^* \rightarrow X 2 \)

---

**First, build an ASCII simulation**

Picobot started here...

P + + + + + + + + +
+ o o o o o o o +
+ + + + + + + + +
+ + + + + + + + +
+ + P + + + + + + +
+ + + + + + + + +

and is now here...

---

Your actual ASCII is likely to be more monochromatic!

---

**Program evolution**

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

---

**3d graphics-based game using VPython**

Let's play!

---

**Physics engine...**

*It’s not really very constrained at all!*

---

**Python features, motivated by VPython...**

**Tuples** are similar to lists, but they’re parenthesized:

\[
T = (4, 2) \quad x = (1, 0, 0)
\]

example of a two-element tuple named \( T \) and a three-element tuple named \( x \)

---

```
def f(x=3, y=17):
    return 10*x + y
```

examples of *default and named inputs* in a function definition
Default – and named – inputs!

Functions can have default input values and can take named inputs.

```python
def f(x=3, y=17):
    return 10*x + y
```

Example of default input values for x and y:

```python
f(x=4, y=2)
```

Example of named input values for x and y:

```python
f(x=4, y=2)
```

Function inputs look like tuples, but they’re not quite the same…

Using GlowScript / vPython...

Vectors

```python
b.vel = vector(1, 0, 0)
b.pos = vector(0, 0, 0)
b.pos = b.pos + b.vel*0.2
```

b.pos, b.vel,… are vectors

Components

```python
vel.x
vel.y
vel.z
```

Scalar multiplication

```
b.pos = b.pos + b.vel*0.2
```

Component-by-component addition

Let’s compare with tuples…

Named inputs

Input your name(s) = _______________

```
def f(x=2, y=11):
    return x + 3*y
```

What will these function calls to f return?

- f(3, 1)
- f(3)
- f(y=4, x=2)

What call to f returns the string 'Lalalalala'?

What is the shortest call to f returning 42?

Extra… what does this return?

```
y = 60; x = -6; f(y=x, x=y)
```
vPython: Linear + Spherical collisions...

At least some of the game needs to be about detecting collisions and changing velocities

Collisions...

# if the ball hits wallA
if ball.pos.z < wallA.pos.z:  # hit - check for z
    ball.pos.z = wallA.pos.z  # bring back into bounds
    ball.vel.z *= -1.0        # reverse the z velocity

# if the ball hits wallB
if ball.pos.x < wallB.pos.x:  # hit - check for x
    ball.pos.x = wallB.pos.x  # bring back into bounds
    ball.vel.x *= -1.0        # reverse the x velocity

# if the ball collides with the alien, give a vertical velocity
if mag(ball.pos - alien.pos) < 1.0:
    print("To infinity and beyond!")
    alien.vel = vector(0,1,0)

Spherical collisions

0 Zeroth approximation:
Stop q. Undo any overlap.
Make r.vel = q.vel.

1 First approximation:
Stop q. Undo any overlap.
Compute d = r.pos - q.pos
Make r.vel = d

2 Second approximation:
Same as first, but
Make q.vel = d ⊥, at 90° from d

point-to-line collisions

point-to-point collisions

compound

The compound object lets you group objects together and manage them as though they were one object, by specifying the usual way pos, color, size (and length, width, height), axis, up, opacity, shininess, emissive, and texture. Moreover, the display of a complicated compound object is faster than displaying the individual objects one at a time. (In GlowScript version 2.1 the details were somewhat different.)

The object shown above is a compound of a cylinder and a box:

handle = cylinder(size=vec(1,2,2),
color=color(0.72,0.42,0.2))

head = box(size=vec(2,6,2),
pos=vec(1,0,0),
color=color.gray(0.6))

hammer = compound([handle, head])
hammer.axis = vec(1,1,0)

The size of the object: After creating the compound named "hammer", hammer.size represents the size of the bounding box of the object.
Named inputs

```
def f(x=2, y=11):
    return x + 3*y
```

What will these function calls to `f` return?

- `f(3,1)`
- `f()`
- `f(3)`
- `f(y=4,x=2)`

None of the above are 42!

What call to `f` returns the string `'Lalalalala'`?

What is `f([], (1,0))`?

What is the `shortest` call to `f` returning 42?

It's only four characters, too!

Extra... what does this return? `y = 60; x = -6; f(y=x,x=y)`

These are tuples – they work like lists!