Mon., 12/3 - *hw12+milestone due*

Fri., 12/14 - 8pm: *final proj. due*

*final project*

CS 5 black final exam 12/17 @ 2pm
'tis the season for final projects…

Today is about the CS 5 final projects + it's a sales pitch for some possible options:

- vPool
- Arcade
- TextID
- Picobot

I've got my eyes on some of these projects!

Eye'll bet!
Final projects

Final CS hw

- open-ended
- comprehensive
- (mostly) same projects for black/gold
- three choices…

Working in teams of 1-3 is

Teams need to work together - *in the same place* - and need to share the work equally…

Black/gold pairs are welcome…

**Teaming is extra-encouraged** on the final project!
Final **lab** days…

No lab meetings *this* week…

Labs **will** meet the first 2 weeks of Dec.

- these lab meetings are *extra-optional*
- **final projects**: get started and assistance
- there are *theocomp* problems, too (**hw12**)…

Now we're in a state!
3d graphics-based game using VPython

I'll take your cue.

Let's play!

... it's not really very constrained at all!

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
The vPool project

• **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)

To now, VPython has *eventually* worked for everyone. *See us for help!*

⇒ Phunky Physics is welcome!
Zeroth approximation:

Stop \( q \).

Undo any overlap.

Make \( r.\text{vel} = q.\text{vel} \).

Spherical collisions

Reality is just three eyes away!
vPool – physics?


Therefore, the classical calculation only holds true when the speed of both colliding bodies is much lower than the speed of light (about 300 million m/s).

Two- and three-dimensional

For the case of two colliding bodies in two-dimensions, the overall velocity of each body must be split into two perpendicular velocities: one tangent to the common normal surfaces of the colliding bodies at the point of contact, the other along the line of collision. Since the collision only imparts force along the line of collision, the velocities that are tangent to the point of collision do not change. The velocities along the line of collision can then be used in the same equations as a one-dimensional collision. The final velocities can then be calculated from the two new component velocities and will depend on the point of collision. Studies of two-dimensional collisions are conducted for many bodies in the framework of a two-dimensional gas.

Two-dimensional elastic collision

equations below...
Vector functions

The following functions are available for working with vectors:

\[ \text{mag}(A) = A\_\text{mag} = |A|, \text{ the magnitude of a vector} \]

\[ \text{mag2}(A) = A\_\text{mag2} = |A|^2, \text{ the vector's magnitude squared} \]

\[ \text{norm}(A) = A\_\text{norm()} = A/|A|, \text{ a unit vector in the direction of the vector} \]

\[ \text{dot}(A,B) = A\_\text{dot}(B) = A \cdot B, \text{ the scalar dot product between two vectors} \]

\[ \text{cross}(A,B) = A\_\text{cross}(B), \text{ the vector cross product between two vectors} \]

\[ \text{proj}(A,B) = A\_\text{proj}(B) = \text{dot}(A, \text{norm}(B)) \times \text{norm}(B), \text{ the vector projection of A along B} \]

\[ \text{comp}(A,B) = A\_\text{comp}(B) = \text{dot}(A, \text{norm}(B)), \text{ the scalar projection of A along B} \]

\[ \text{diff\_angle}(A,B) = A\_\text{diff\_angle}(B), \text{ the angle between two vectors, in radians} \]

\[ \text{rotate}(A,\theta,B) = A\_\text{rotate}(\theta,B) = \text{rotate(vector=A, angle=\theta, axis=B)}, \text{ result of rotating A through \theta around B} \]

\[ \text{astuple}(A) = A\_\text{astuple()}, \text{ convert this vector to a tuple, much faster than using tuple(A)} \]

This is how I like my vectors: well done!
(or, at least, already done...!)
Collisions, but in 2D…

```
Magnolia:Dungeon Game 45 $ ls images/
archer/
  character.png
  chest_closed.png
  chest_opened.png
  coin.png
  controls.png
  dead.png
arrow.png
demon.png
demon_die_1.png
demon_die_2.png
demon_slash.png
fireball.png
pt1.png
arrow_pack.png
tile_1.png
tile_2.png
tile_3.png
tile_4.png
box.png
```
## Color: RGB(A)

<table>
<thead>
<tr>
<th>Red</th>
<th>Green</th>
<th>Blue</th>
<th>(Alpha)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-255</td>
<td>0-255</td>
<td>0-255</td>
<td>0-1</td>
</tr>
</tbody>
</table>

- (147,128,255,0.83)
- (54,217,40,1)
- (54,217,40,0.5)
- (187,60,60,0.32)
Games!
Warning: This is new this year

Try it out for us … and help future CS5-ers!
the TextID project

Big ideas:

(1) Build *lexical* models of bodies of text…

(2) Use a similarity score to measure

*Rowlingness* vs. *Shakepearity*

*NYTimes-iness* vs. *WSJournalicity*

*Big Bang Theory* vs. *Modern Family*

**even better:** *your own choice of two or more comparisons*...
I like poptarts and 42 and spam. Spamful poptarts are like poptartful spams -- and are liked by all!

Will _Thanksgiving_ bring spam poptarts?

class TextModel

... contains at least five Python dictionaries, e.g.,

```python
{'and': 3, 'poptartful': 1, 'liked': 1, 'spamful': 1, 'like': 2, '(': 1, 'spam': 2, 'i': 1, '42': 1, 'all': 1, 'thanksgiving': 1, 'will': 1, 'bring': 1, 'poptarts': 3, 'spams': 1, 'by': 1, 'are': 2}
```

```python
{0: 1, 1: 1, 2: 2, 3: 6, 4: 5, 5: 3, 7: 1, 8: 3, 10: 1, 12: 1}
```

```python
{'and': 3, '(': 1, 'all': 1, 'like': 3, 'thanksgiv': 1, 'spam': 4, 'i': 1, '42': 1, 'by': 1, 'will': 1, 'bring': 1, 'ar': 2, 'poptart': 4}
```

```python
{12: 1, 5: 1, 7: 1}
```

```python
{':': 1, '-': 2, '?': 1, '_': 2, '.': 1}
```

What are these four other dictionaries counting?!
TextID's building blocks...

(0) Get text from file...
(1) Split up the text into words (first pass)
(2) Model punctuation marks (optional)
(3) Model sentence lengths (using ' . ! ? ')
(4) Take a breather...
(5) Clean up the words (second pass)
(6) Model words and word lengths
(7) Stem words and model those stems
(8) You're ready to score against your model!
7.1. string — Common string operations

5.6.1. String Methods

**str.replace(old, new[, count])**
Return a copy of the string with all occurrences of substring `old` replaced by `new`.

**str.lower()**
Return a copy of the string with all the cased characters [4] converted to lowercase.

**str.split([sep[, maxsplit]])**
Return a list of the words in the string, using `sep` as the delimiter string. If `maxsplit` is given, at most `maxsplit` splits are done (thus, the list will have at most `maxsplit+1` elements). If `maxsplit` is not specified or `-1`, then there is no limit on the number of splits (all possible splits are made).

If `sep` is given, consecutive delimiters are not grouped together and are deemed to delimit empty strings (for example, `'1,,2'.split(',')) returns ['1', '', '2']). The `sep` argument may consist of multiple characters (for example, `'1<>2<>3'.split('<>') returns ['1', '2', '3']). Splitting an empty string with a specified separator returns `['']`. 
Stemming

An algorithm that outputs the **root** of the input word.

*stem('parties')*  $\rightarrow$ 'parti'

ends in ies

*stem('love')*  $\rightarrow$ 'lov'

ends in vow + cons + e

*stem('swerving')*  $\rightarrow$ 'swerv'

ends in vow + cons + ing

*stem('quickly')*  $\rightarrow$ 'quick'

ends in vow + cons + ly

*stem('slowest')*  $\rightarrow$ 'slow'

ends in vow + cons + est

*stem('stems')*  $\rightarrow$ 'stem'

ends in cons + s

*stem('stemming')*  $\rightarrow$ 'stem'

ends in vow + 2*cons + ing

these don't have to be words, just **stems**...

but, they *can* also be words

either way, they *all* have exceptions!
Exceptional English exceptions to each of these stemming patterns:

- **pies**
  - `stem('parties')` → 'parti'
    - ends in ies
- **kite**
  - `stem('love')` → 'lov'
    - ends in vow + cons + e
- **nothing**
  - `stem('swerving')` → 'swerv'
    - ends in vow + cons + ing
- **ally**
  - `stem('quickly')` → 'quick'
    - ends in vow + cons + ly
- **stem('slowest')` → 'slow'
    - ends in vow + cons + est
- **lemming**
  - `stem('stemming')` → 'stem'
    - ends in vow + 2*cons + ing
- **under**
  - `stem(____undo____)` → _do_
    - create your own rule for stemming
Model *matching*

Suppose we have two trained models:

WS: 

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

JKR: 

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

Unknown text: 

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

These must have been some really avant-garde texts!
Model matching

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

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

Suppose we have two normalized models:

how do we compare the models with an unknown text?

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

What's the likelihood of the new model arising from each?

There's probably a way to do this!
Model matching

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

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

how do we compare the models with an unknown text?

Let's just pretend that the words are all independent of each other…

What's the likelihood of the new model arising from each?

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

There's probably a way to do this!
Model **matching**

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**

```
love  love  love  thou  potter  potter  spam  spam  spam  spam

0.50 0.50 0.50 _?_   ?  ?  ?  ?  ?  ?  ?  ?
```

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

I've got near-zero ideas on this one!
Model matching

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 \}

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

-29.48

-33.89

the (much) better match...
Naïve Bayes classification

Bayesian spam filtering
From Wikipedia, the free encyclopedia

Bayesian spam filtering (/ˈberzɪən/ 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 naïve Bayes model. The naïve Bayes classifier combines this model with a decision rule. One popular decision rule is 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 classified:

\[
\text{classify}(f_1, \ldots, f_n) = \arg\max_c p(C = c) \prod_{i=1}^{n} p(F_i = f_i | C = c).
\]
The Picobot project

Big ideas

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

talk about going *full circle*...
class Program:

What type should `self.rules` be?

What in Python could most usefully hold all of these `rules`?
Picobot's classes

```python
class Program:

    What in Python could most usefully hold all of these `rules`?

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

self.rules[ (1,"NExx") ] = ("W",0)
Picobot's classes

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

```python
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
The Picobot project

Current State: 1
Current Rule: \[1 \text{ } N^*W^* \rightarrow X \text{ } 2\]

First, build an ASCII simulation

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-splittable solutions.

Start with a population of, say, ~200 random Picobot programs...
Genetic Algorithms...!

the current "fitness" leader

the current "generation"

fitesses!

Rafael Matsunaga: rednuht.org/
Repeat this "survival of the fittest" process for many generations…

… and by the end, your Python code should/will have evolved a much more capable Picobot program!

Solving hw0!
Extra: Picobot graphics...

Choice of graphical packages

- 2d: graphics.py from C4
- 3d: VPython

2d version...

2d should be enough for anybody

What? All the best stuff is 3d! Think coffee, poptarts, etc.

Mine's going to be in 5d!
Picobot!

Project space...

Arcade!

VPool

TextID

algorithmic

experimental

open-ended (and 3d!)

practical + checkable
What's due?

Mon. 12/4

• The "milestone"

• project-specific tasks ~ to ensure that everyone has started!

(\textit{Note: we can't grade these in time})

Fri. 12/8

• Final-project due date

• A zip file with everything needed to run your final project and a .\texttt{txt} file describing how to do so!
Enjoy the projects!

An unusual variation on VPool...

...the graders certainly do!
Enjoy the projects!

Have a relaxing long weekend!

An unusual variation on VPool

... the graders certainly do!